

Building a secure Condor[®] pool in an open academic environment

Bruce Beckles

University of Cambridge Computing Service, New Museums Site, Pembroke Street, Cambridge CB2 3QH, UK

Abstract

The Condor[®] system ([1], [2]) is a widely used “cycle scavenging batch scheduler” which is increasingly being used to utilise the “idle time” of workstations in so-called “campus grid” installations across the UK, yet its security architecture is not well suited to such environments. In this paper we describe how a campus grid Condor installation at the University of Cambridge has addressed the security and other related challenges to build a stable, secure Condor service. We present the system architecture we have formulated, which we hope will be of interest to organisations engaged in similar projects.

1. Introduction

The Condor[®] system ([1]) is a “specialized workload management system for compute-intensive jobs” [2] which is widely used for utilising the “idle-time” of workstations. In the UK an increasing number of academic institutions are deploying large Condor pools ([3], [4], [5]), often as part of so-called “campus grid” installations.

Such academic environments are usually quite “open”, in the sense that there are often few security boundaries (firewalls, etc.) segregating the network, and, in any case, in many areas of the network there will be a mix of trusted (e.g. staff) and untrusted users (e.g. students), often sharing the same resources (public access workstations, etc). In such an environment the security concerns have to be carefully managed if the environment is to remain secure.

The Condor project started in 1988 and the Condor system has been in continuous development ever since. An unfortunate consequence of this seems to be that it has inherited a security model predicated on a high degree of trust – a degree of trust unusual in the open academic environments described above. Some of the security concerns regarding the Condor system are detailed in [6].

In this paper we describe a University of Cambridge Computing Service (UCS) project to deploy a large Condor pool in our open academic environment, similar in scale to those mentioned above, that addresses many of the security concerns of running the Condor system in such environments. First we analyse and discuss the requirements for this type of Condor service and then we describe how we have addressed the associated challenges.

2. Architectural requirements for the Condor service

Large Condor installations such as the one described here and those mentioned above have a number of characteristics in common, most notably:

- their size (typically hundreds of workstations, e.g. about 940 PCs at UCL)
- the workstations used are centrally managed in some manner,
- the primary purpose of the workstations used is *not* for running Condor jobs, and
- the workstations used are typically public access machines, i.e. they are available for general use to all or most members of the institution.

As the primary purpose of the workstations across which Condor will be deployed is not for running Condor jobs, the Condor service must not unduly affect this primary purpose. This gives rise to two fundamental requirements for the Condor service: **stability**, and that it has a **low impact** upon the normal functioning of the workstations.

For similar reasons, the Condor service must not make the environment any less secure (as a security incident on this scale would have an extremely negative impact on the public access workstation service whilst the incident was being resolved). In addition, a successfully compromised Condor service might give the attacker the ability to run arbitrary executable code on hundreds of machines across the institution potentially bringing all those machines under their control. Indeed, the attacker might gain access to all the workstations in the Condor deployment (which may be all the public access workstations in the institution). As a worst-case scenario, this is likely to be unacceptable to many institutions.

Thus another fundamental requirement of the Condor service is **security**.

Clearly, there are many other requirements that might be reasonably ascribed to a Condor service in such environments, but most of those are more specific and/or arise from a more detailed analysis of the Condor service's and/or institution's purpose. The three requirements given here (discussed in turn in each of the following sections) are believed to be of a sufficiently general nature to be applicable to almost any large Condor installation in an open academic environment.

Note: the public access workstation service provided by the UCS is known as the Personal Workstation Facility (PWF). The UCS also provided a similar service using the same infrastructure for Departments or Colleges of the University; this service is known as the Managed Cluster Service (MCS).

2.1. Stability

In order to address stability concerns, it was decided that – unless this was in conflict with other requirements – only the most recent version of the stable release series of Condor (the 6.6 series at the time of writing) would be deployed – see [7] for an explanation of the Condor version numbering scheme. In addition, prior to deployment it would be extensively tested on a small number of workstations that were configured identically to ordinary PWF workstations.

Also, features of Condor that were not required by any of the service's stakeholders, or that could not be adequately supported by the deployment team, would be disabled where possible. It was also decided that features that it was felt might have an adverse impact on the environment would be disabled (if possible), although to date this decision has not required any features of Condor to be disabled that were not already being disabled for other reasons.

2.2. Low impact

To minimise impact on the service provided by the PWF, usage statistics were gathered for each computer room in which Condor would be deployed. The Condor service was only allowed to run in a particular computer room at times when usage of that room would normally be low. Naturally, usage patterns vary throughout the week, and between term-time and vacation and this was taken into account in deciding the times at which the Condor service would run.

Also, Condor was configured not to run jobs if a user was logged in, and to evict jobs if a user logged in to the machine or touched the keyboard or mouse. Under Linux, Condor can

currently only detect USB keyboard or mouse activity if patches are applied to the Linux kernel [8]. At present none of the machines across which Condor is deployed have USB keyboards or mice, so this is not yet an issue.

Condor's does not automatically include in its machine ClassAds [9] information about the number of users logged in to a machine, which is necessary if it is to make policy decisions (whether or not to run a job, etc.) based on the number of users logged in. To cater for this, Condor's *cron* capabilities [10] were used to run a shell script once a minute that reported the number of logged in users in a form that Condor could include in its machine ClassAd.

An additional concern was to ensure that Condor jobs did not leave the machine in a state where it was unusable or where the service offered to users interactively logging in was degraded (e.g. by filling up the workstation's hard disk or leaving badly behaved processes running after the Condor job had supposedly completed). This concern was addressed by controlling the job's execution environment and sterilising that environment after job completion (see "Controlling the execution environment" in Section 3.2).

2.3. Security

Unsurprisingly, given the nature of the Condor system, security proved the most difficult of these requirements to address. One of our first decisions was to disable (where possible) those features of Condor that were felt to have unacceptable security implications. And although the Condor system has a number of features that were useful in addressing some of our security concerns, a considerable amount of work was necessary to use these features effectively, and it was necessary to re-engineer the Condor security model to address other issues. Before discussing how our security concerns were addressed, it will be useful to examine those concerns in more detail, and this is done in the following section.

3. Security concerns

Before examining the security concerns of this environment it is worth contextualising these concerns by discussing the threat landscape, and how this threat landscape differs from that of other common environments (e.g. in commercial or industrial settings) in which Condor is deployed. Following this discussion we discuss specific security concerns of this environment and how they have been addressed in our Condor deployment.

3.1. Threat landscape

Many commercial or industrial organisations often employ firewalls with restrictive security policies as an important part of their security strategy. It is not surprising, therefore, that in many industrial or commercial institutions the security situation is contained by placing all the machines that are part of the Condor deployment behind a restrictive firewall. The viability of this strategy depends on a threat landscape in which the primary threats are seen as being *external* to the organisation, i.e. it relies on the users of the network in the organisation being *trusted* users (note that there may be degrees of trust).

In many academic environments however, serious threats arise from both within the institution as well as from outside it – the typical user population of such institutions is a mix of trusted and *untrusted* users. In a large university such as the University of Cambridge the majority of the user population – the student population – consists, in fact, of untrusted users. To make matters worse, the machines across which Condor is deployed in the scenario discussed here are public access workstations (i.e. workstations accessible to any current member of the University) that are not solely dedicated to Condor. Thus there can be no clean separation between trusted and untrusted users of the resources across which Condor is deployed.

Further, in most academic environments it is not unusual for legitimate use of the network to require access to a wide range of network resources (which may well have serious implications for the configuration of any firewalls in the environment). It is also not unusual for users to run legitimate Condor jobs that require access to network resources across a variety of networks, port numbers and network protocols. This contrasts with the situation in commercial or industrial institutions where often the jobs that would be run in a Condor pool would only need to access resources within that institution.

In these academic environments firewalls are of limited usefulness in containing the security situation – indeed, sometimes they will be of no use, at least in security terms. Similarly, private networks (i.e. networks of machines whose IP addresses are in the range specified in RFC1918 [11]), are also of little or no benefit in containing the security situation, particularly where those machines can be accessed from other machines on the institution's network that are also on the public Internet. Indeed, at the University of Cambridge, machines on private networks

within the University have been successfully compromised by attackers who are believed to have launched their attacks from outside the University network.

In summary, in these environments:

- (a) the threat landscape for the environment is one in which threats *internal* to that environment are at least as significant as external threats, and
- (b) an essential requirement for many legitimate activities is access (often largely or wholly unrestricted) to the public Internet.

In such scenarios network security boundaries – such as firewalls or private networks – offer little or no protection, and we must rely on other security boundaries and mechanisms.

3.2. Specific security concerns

In order to have any confidence in the security of the Condor service we wished to offer there were a number of specific security concerns that needed to be addressed. The principal ones were:

- *Strong authentication of individual machines:*

By default, Condor bases its identification of machines on their IP addresses, and restriction by IP address is the most commonly used mechanism for restricting access to machines in a Condor pool. Whilst this may be appropriate on a trusted network with the appropriate logical structure, it is wholly unsatisfactory in any environment where the integrity of IP addresses cannot be guaranteed (i.e. IP and/or ethernet addresses can be “spoofed”). Although in most academic institutions it is difficult or impossible to “spoof” an IP address belonging to a machine on that institution's network from *outside* that network, this is little comfort if there are malicious individuals operating *within* the network boundary (as discussed in Section 3.1). Thus it was necessary to have a better method of identifying machines and restricting access to machines in the Condor pool.

Under the Linux environment Condor currently supports two so-called “strong authentication” methods: Kerberos [12] and GSI [13] authentication. In principle either of these could have been used to give us strong assurance of the identity of machines in Condor daemon-daemon communication between machines. The decision to use Kerberos was based on a number of factors:

1. Using GSI authentication in Condor requires the Condor daemons to run with root privilege, which violates the security model we wished to impose upon Condor (see “Privilege separation” below).

2. There are a number of serious usability issues with GSI ([14], [15]) – unsurprisingly, as it is a PKI and there are known to be serious usability issues with PKIs in general ([16], [17], [18]). As well as these considerations, our own experience of supporting the infrastructure necessary for inter-institutional authentication using GSI has not been entirely positive, which increased our reluctance to deploy it internally.
3. By using dedicated Kerberos Domain Controllers (KDCs), against which all Condor daemons authenticated when they made network connections, we had a separate set of log files, independent of Condor’s logging facilities, that we could use as an audit trail. Using GSI authentication would not have provided us with a similar independent audit trail.
4. We already planned to develop a Kerberos infrastructure for use across the University, although this was not yet in place.
5. When this deployment was begun the Condor Team informed us (*pers. comm.*) that they were closer to supporting Kerberos on the Windows® platform than to supporting GSI, as a mature Kerberos implementation existed on that platform whilst there were a number of issues with the Windows® GSI implementation which the Globus Alliance appeared (at least at that time) unwilling to fix. This was a factor in our decision as we wished to eventually extend our pool to include workstations running Windows® XP.

Unfortunately, there is a bug in the Condor 6.6 series that means that Kerberos authentication only works when the Condor daemons are run with root privilege (as mentioned above this violates our desired security model; see “Privilege separation” below). This bug has been fixed in the Condor 6.7 series (the current development release series) and thus we had to back-port the fix from the 6.7 series, patch the Condor 6.6 series source and re-compile Condor. Since when we started this deployment the Condor 6.6 series was built against a version of Kerberos that was known to have security vulnerabilities (although this is believed to no longer be the case), it was also necessary to patch the Kerberos libraries included with the source at that time.

The Condor documentation [19] does not give much detail on configuring Condor to use Kerberos authentication and so a certain amount of experimentation and examination of the Condor source was necessary to work out how to configure Condor to use Kerberos correctly in our environment.

Despite these problems, once Condor has been correctly configured, our experience has been that the Kerberos infrastructure works well with it. Although Kerberos is often described as being a “heavyweight” infrastructure that is difficult to set up and maintain, we have not found this to be the case, perhaps because we are only using Kerberos to authenticate Condor daemons at present (as opposed to Condor users or users of other systems in the University).

- *Privilege separation:*

In order to run securely Condor requires all its daemons that listen to the network (except on dedicated central managers) to run with root privilege. This violates “best practice” for daemons that listen to the network and raises the “attack surface” of the machines on which those daemons are running, making them more vulnerable to serious compromises. A common solution to this type of problem is to implement privilege separation [20], and so we developed a restricted form of privilege separation for the Condor system (described in [21], [22]). In addition we use a dedicated central manager, which means that the Condor daemons that provide the functionality of the central manager do not need to run with root privilege.

- *Controlling access to the Condor service:*

Even if the Condor service were available to all current members of the University, some robust form of access control would still be necessary. A policy of *unrestricted* access would make it difficult to distinguish between legitimate and illegitimate use, and might lead to an excessive load being placed on the service. In addition, it was clear to us that if we wished to extend this service to MCS workstations owned by the Departments and Colleges of the University then sensible access control mechanisms would be a prerequisite.

In addition, given that the potential for using the Condor service for illegitimate purposes (e.g. for distributed denial-of-service (DDoS) attacks), whether intentionally or accidentally, is so great, a deliberate policy of piloting the service with a small number of well-known, (relatively) trusted users was adopted. In fact, once the pilot service has proved itself, it is hoped that a general policy of “vetting” users (e.g. by inspecting sample jobs, initially restricting the number of machines on which their jobs can run, etc.) before allowing them full access to the service will be adopted.

In the Condor system, the easiest way of controlling access is at the point of job submission. The best way of doing this is to minimise the number of submit nodes (i.e. machines that can submit jobs to the Condor system), and to centralise and tightly control

these submit nodes. Unfortunately, reducing the number of submit nodes has performance and scalability implications for the Condor system – these are discussed in Section 4.

In our initial pilot service we are using a single centralised submit node, access to which is only available via SSH. SSH authentication makes use of the same authentication infrastructure as that which is used by the PWF (which is based on Novell® eDirectory™). This means that that administration of user accounts is handled by the same user administration services that administer PWF/MCS user accounts. SSH access is controlled via an access control list (ACL) that is stored as a NetWare® group in the eDirectory.

Because we have strong assurance of the identity of machines involved in Condor network communication, we can ensure that only jobs submitted from our centralised submit nodes will be accepted by the execute nodes (i.e. machines that are configured to execute Condor jobs) in the pool. Excerpts of the Condor security configuration [23] on each type of machine in our Condor pool are given below (“Condor service machines” means the Condor central manager and the submit node(s)):

○ *On the central manager:*

```
HOSTALLOW_READ = <Condor service machines>,
<UCS administrative machines>, <PWF/MCS sub-
domains>
HOSTALLOW_WRITE = $(FULL_HOSTNAME) ,
<submit nodes>, <PWF/MCS sub-domains>
SEC_DEFAULT_AUTHENTICATION_METHODS =
KERBEROS,FS
SEC_DEFAULT_AUTHENTICATION = OPTIONAL
SEC_WRITE_AUTHENTICATION = REQUIRED
SEC_ADMINISTRATOR_AUTHENTICATION =
REQUIRED
```

○ *On the submit node(s):*

```
HOSTALLOW_READ = $(FULL_HOSTNAME)
HOSTALLOW_WRITE = $(FULL_HOSTNAME)
SEC_DEFAULT_AUTHENTICATION_METHODS =
KERBEROS,FS
SEC_DEFAULT_AUTHENTICATION = REQUIRED
SEC_CLIENT_AUTHENTICATION = OPTIONAL
SEC_WRITE_AUTHENTICATION = REQUIRED
```

○ *On execute nodes (PWF workstations):*

```
HOSTALLOW_READ = $(FULL_HOSTNAME) , <Condor
service machines>, <UCS administrative machines>
HOSTALLOW_WRITE = <Condor service machines>
SEC_DEFAULT_AUTHENTICATION_METHODS =
KERBEROS,FS
SEC_DEFAULT_AUTHENTICATION = REQUIRED
ALLOW_ADMIN_COMMANDS = NO
```

The above configuration means that there is a relatively relaxed policy to reading information from the central manager (which allows users and UCS administrative staff to query the status of the Condor pool). However only machines with the appropriate Kerberos credentials *and* DNS hostnames (although, as discussed above this does not really provide any significant additional protection) can add

themselves to the pool, submit jobs, query specific submit or execute nodes, etc.

Something that needs to be borne in mind with this configuration is that it prevents execute nodes from updating the submit node with information (e.g. the job’s current image size) whilst the job is running. This is not a concern for us, however, as one of the consequences of our implementation of privilege separation (see [22] for details) is that such information would be inaccurate in any case.

• *Controlling the execution environment:*

As users’ Condor jobs can execute arbitrary executable code, it was necessary to exercise control over the execution environment to mitigate the security concerns of this scenario. In particular, it was important to be able to determine what processes the user had started (which we do via process accounting on the workstations), to ensure that when the user’s job “finished” there were no processes left behind, and to protect against privilege escalation (as much as possible).

In addition, it was important to ensure that users’ jobs ran in a consistent environment (so that they could have some confidence in their jobs behaving as expected regardless of which machine the job was executed upon), and that they did not adversely affect the workstation after completion (cf. Section 2.2), e.g. by filling up its hard disk.

To address these concerns we rely principally upon our implementation of restricted privilege separation (see [21], [22]) to ensure that the job is started in a sterile environment, running under a dedicated non-privileged user account. The job is started under this user account by a wrapper script which makes sure the environment is configured sensibly and in accordance with the way Condor would configure the environment, *insofar as it is sensible to respect Condor’s wishes* (e.g. Condor allows the user to accidentally set many environment variables which should not be set by the user at job submission time).

Our implementation of privilege separation also gives us another advantage: it provides a convenient “hook” for executing commands of our choice at *job completion*, a feature currently lacking in Condor. Since the job is started under a dedicated non-privileged user account, we can use this hook to safely forcibly terminate any processes “left behind” by the job.

Taking advantage of Condor’s USER_JOB_WRAPPER feature [24] (used by our privilege separation implementation) and its *cron* facility [10], we remove any files created by Condor jobs and left behind on the workstation’s hard disk, both immediately prior

to job execution, and periodically when the machine is idle.

Note that our implementation of privilege separation allows us to run jobs in a separate “virtual machine” of some description (e.g. User-mode Linux [25]) or with a different root directory via the `chroot` command, if desired.

- *Restricting access to the Condor commands:*

Because the trust assumptions in Condor’s security model assume a relatively high degree of trust between users of the Condor system and the system’s administrators, it has not been designed to allow fine-grained restrictions on the users’ use of the system. For instance, although individual instances of the Condor daemon that manage a job queue (the `condor_schedd` daemon) cannot cope with excessively large job queues (see the discussion of scalability in Section 4), there are no features provided to restrict the number of jobs that can be submitted to the job queue, either in absolute terms or on a per-user basis. This means that an individual user can easily disable any submit node to which they can submit jobs, and it is worth bearing in mind that their doing so might be entirely accidental.

Another example, also drawn from our own experience: if a site uses custom ClassAd attributes to control which collections of machines particular types of jobs should be matched against (or to affect job priority, etc.), Condor provides no mechanism of preventing users from using those attributes in their job submission files. A more appropriate security model would recognise that it is important to be able to restrict the use of certain ClassAd attributes to particular users under certain conditions.

Such considerations mean that, in general, users should not be allowed unrestricted access to the basic Condor commands. The ideal situation would be to design and implement a usable front-end which only exposed exactly that functionality that it was desirable for the user to have, and did not allow them direct access to the underlying Condor system.

For example, this is one of the features of the architecture of C.O.R.E. Feature Animation’s Condor deployment ([26]) that has allowed them to so effectively utilise Condor with a large number of jobs and machines. By separating the users from the Condor system itself, they have been able to design an efficient system architecture that does not need to directly concern itself with the actions of the users and can instead be optimised for its deployment environment.

For our initial pilot deployment we have not had the resources to develop a fully functional usable front-end for our users; instead we have wrapped many of the Condor commands and daemons in ways that have allowed us to better control the use of the environment. Of course, in some cases it is possible for users to circumvent these measures (which is why strict separation between the Condor system and the users is so desirable), but this is of less concern to us in the current pilot as we have chosen to pilot the service to relatively trusted users.

4. Other concerns and problems

There are a number of other concerns and problems that we needed to address, some of which were a result of the ways in which we had satisfied the fundamental requirements outlined in Section 2, and, in particular, our security concerns (Section 3.2). Some of these are discussed below:

- *Scalability and performance:*

The current design of the Condor system is not well suited to a deployment architecture in which the number of submit nodes is minimised. This means that in such scenarios the Condor system suffers from certain performance and scalability issues. One of the fundamental problems is to do with the `condor_schedd` daemon, which can quickly become a performance bottleneck under load.

The central problem with this daemon that causes it to be a bottleneck is that it is single-threaded and is responsible for too many tasks. Amongst its other tasks, it manages the job queue (accepting job submissions, etc.), handles user queries of the job queue, is responsible for initiating the shadow processes that communicate with jobs on execute nodes, is responsible for communicating job queue information to the central manager and processing its responses. A further problem is that many network connections made by this daemon are *blocking* connections, which exacerbates the problem (this has been improved as of Condor 6.7.5, but these improvements have not yet appeared in a version of a stable release series of Condor).

Thus a common scenario under heavy load is that job handling and/or throughput degrade, and this is noticeable to users, who then query the `condor_schedd` daemon to find out why their jobs are not being processed as quickly as they should. This increases the load on the already struggling daemon, exacerbating the problem. In addition, this daemon cannot handle excessively large queues (in our tests, queue sizes of 10,000 jobs were too large for it to cope with).

Although careful tuning of the operating system and the Condor configuration can help somewhat, fundamental improvements are needed in the design of the Condor system, and the `condor_schedd` daemon in particular. Until this happens, either multiple instances of this daemon need to be available to users (which may well mean an increase in the number of submit nodes), or users need to be educated to behave sensibly with regard to these limitations.

For the pilot of our Condor service we have undertaken a twin strategy of:

- (a) user education (which works because the users of this pilot service are relatively trustworthy), and
- (b) wrapping or replacing certain Condor commands to encourage sensible behaviour and alleviate some of these problems.

We have provided alternative commands that query the job queue by directly inspecting the job queue file rather than making a request of the `condor_schedd` daemon. We have also wrapped the `condor_submit` command so that it will not submit jobs to the job queue if the queue already contains 4,000 or more jobs.

When the pilot service is expanded we plan to run additional instances of the `condor_schedd` daemon on our submit node (which is a machine with a symmetric multiprocessing (SMP) architecture whose processors support Hyper-Threading Technology) and provide a usable front-end that will handle load balancing job submissions between the separate job queues, and allow users to easily query all queues, manage their jobs, etc.

- *Restricting Condor's functionality:*

Mainly because we have only undertaken a restricted implementation of privilege separation in the Condor system – some of those restrictions arise from the design requirement that this implementation did not modify the Condor source (see [22]) – some of Condor's more advanced features are not available in our Condor pool. Also, as discussed in Section 2.1, features considered extraneous to our stakeholders were also disabled, where possible.

This means that our pool only supports jobs that run in the Vanilla and Java universes [27], and does not support some of the more advanced features of those universes, such as the `ExitBySignal` ClassAd attribute [28] and `Chirp I/O` [29] – for details of features that are not supported due to our implementation of privilege separation, see [22].

Prior to the design of our Condor service we undertook to consult with as many potential users of the service as possible, and their requirements were taken into consideration in the design and implementation of the pilot

service. Thus far, the current restrictions have not yet been a problem for users of our pilot service. Since undertaking this user requirements gathering exercise a small number of potential users have begun using Condor's Standard universe [27] in other Condor pools. We believe that we should be able to provide restricted support for the Standard universe in future versions of our pilot service (the main problems to be solved are to do with our implementation of privilege separation; see [22] for details).

- *Partitioning the pool:*

Although our Condor service consists of a single Condor pool, we needed to be able to partition it, i.e. ensure that certain workstations would only run certain classes of jobs, e.g. a Department might wish to ensure that only jobs from its users were run on its MCS workstations. The normal way to do with would be with custom ClassAd attributes and/or Condor's user-based authorisation facilities [30]. Unfortunately, unless users are prevented from directly submitting jobs to the Condor pool, they can circumvent ClassAd-based "restrictions", and Condor's user-based authorisation does not scale particularly well.

Thus, the approach adopted was to use a very small number of custom ClassAd attributes, in conjunction with a wrapper around the `condor_submit` command. Since currently this can be circumvented, we enforced this policy by making use of our existing LDAP infrastructure (which is based on Novell eDirectory). ACLs were stored in eDirectory and a wrapper script on the execute nodes makes an LDAP query for these ACLs. This ensures that if a user adds a custom ClassAd attribute to their job's ClassAd to try to get their job to run on a machine that they are not authorised to use, the job will be rejected.

5. Architectural overview

In summary, a brief overview of the architecture of our Condor service is as follows:

- A large number of centrally managed public access workstations running Linux as execute nodes,
- Jobs are run only when the machines do not have users logged into them,
- Centralised submit nodes (currently only one), access to which is via SSH,
- Condor commands wrapped or replaced as the first stage in providing a separate usable front-end,
- Restricted (but still useful) subset of Condor's features, and
- An improved security model: privilege separation on the execute nodes, tight

control of the job execution environment, strongly authenticated communication between Condor daemons, etc.

6. Conclusion

Although the Condor system is not designed to be deployed in an environment in which the major threats are internal to the security boundary, it is possible to deploy it in such environments provided an appropriate security model is adopted and implemented in the Condor deployment. Whilst this is certainly possible, it requires a significant amount of development work as Condor currently lacks many features that would be helpful in this regard, and its system architecture does not support centralisation particularly well.

Nevertheless, the deployment of our pilot Condor service demonstrates that the problems are not insurmountable and that the resulting service more than adequately fulfils our users' requirements. Finally, it cannot be stressed highly enough that deployments of this nature would simply be not be possible without the supporting infrastructure of a stable, centrally managed public access workstation service that provides a consistent environment across all workstations of a given operating system.

Acknowledgements

The author would like to thank the University of Cambridge Computing Service, and in particular the Unix Systems Division, without whom none of the work described here would have been possible.

References

- [1] The Condor[®] Project Homepage: <http://www.cs.wisc.edu/condor/>
- [2] What is Condor?: <http://www.cs.wisc.edu/condor/description.html>
- [3] UCL Research Computing Condor: <http://grid.ucl.ac.uk/Condor.html>
- [4] Condor – Cardiff University: <http://www.cardiff.ac.uk/schoolsanddivisions/divisions/instrv/forresearchers/condor/index.html>
- [5] University of Southampton Windows Condor Pilot Service: <http://www.iss.soton.ac.uk/research/e-science/condor/>
- [6] Beckles, B. Security concerns with Condor: A brief overview: http://www.nesc.ac.uk/esi/events/438/security_concerns_overview.pdf
- [7] Condor[®] Version 6.6.10 Manual, Section 8.1: http://www.cs.wisc.edu/condor/manual/v6.6.10/8_1Introduction_Condor.html
- [8] Condor's Linux Kernel Patch: <http://www.cs.wisc.edu/condor/kernel.patch.html>
- [9] Condor[®] Version 6.6.10 Manual, Section 2.3: http://www.cs.wisc.edu/condor/manual/v6.6.10/2_3Matching_with.html
- [10] Condor[®] Version 6.6.10 Manual, Section 3.3: http://www.cs.wisc.edu/condor/manual/v6.6.10/3_3Configuration.html#8753
- [11] RFC 1918 (Address Allocation for Private Internets): <http://www.ietf.org/rfc/rfc1918.txt>
- [12] Kerberos: The Network Authentication Protocol: <http://web.mit.edu/kerberos/www/>
- [13] Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S. A security architecture for computational grids (1998). *Proceedings of the 5th ACM conference on Computer and communications security*, San Francisco, CA, 1998, pp.83-92: <http://portal.acm.org/citation.cfm?id=288111>
- [14] Beckles, B., Brostoff, S., and Ballard, B. A first attempt: initial steps toward determining scientific users' requirements and appropriate security paradigms for computational grids (2004). *Proceedings of the Workshop on Requirements Capture for Collaboration in e-Science*, Edinburgh, UK, 14-15 January 2004, pp. 17-43: http://www.escience.cam.ac.uk/papers/req_analysis/first_attempt.pdf
- [15] Beckles, B., Welch, V. and Basney, J. Mechanisms for increasing the usability of grid security (2005). *Int. J. Human-Computer Studies*, in preparation (July 2005).
- [16] Ellison, C.M. The nature of a usable PKI (1999). *Computer Networks* 31 (8) pp. 823-830.
- [17] Ellison, C., Schneier, B. Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure (2000). *Computer Security Journal* Volume XVI, Number 1 (2000) pp. 1-7: <http://www.schneier.com/paper-pki.pdf>
- [18] Gutmann, P. PKI: It's Not Dead, Just Resting. Extended version of an article with the same name that originally appeared in *IEEE Computer*, Volume 35, Number 8 (August 2002), pp.41-49: <http://www.cs.auckland.ac.nz/~pgut001/pubs/notdead.pdf>
- [19] Condor[®] Version 6.6.10 Manual, Section 3.7.4.2: http://www.cs.wisc.edu/condor/manual/v6.6.10/3_7Security_In.html#SECTION00474200000000000000
- [20] Provos, N., Friedl, M. and Honeyman, P. Preventing Privilege Escalation (2003). *12th USENIX Security Symposium*, Washington, DC, USA, 2003: http://www.usenix.org/publications/library/proceedings/sec03/tech/provos_et_al.html
- [21] Beckles, B. Privilege Separation in Condor. Presentation at Condor Week 2005, Madison, WI, USA, 14-16 March 2005: http://www.cs.wisc.edu/condor/CondorWeek2005/presentations/beckles_privilege_separation.ppt
- [22] Beckles, B. Implementing privilege separation in the Condor[®] system (2005). *Proceedings of the UK e-Science All Hands Meeting 2005*, Nottingham, UK, 19-22 September 2005, *forthcoming*.
- [23] Condor[®] Version 6.6.10 Manual, Section 3.7.3: http://www.cs.wisc.edu/condor/manual/v6.6.10/3_7Security_In.html#SECTION00473000000000000000
- [24] Condor[®] Version 6.6.10 Manual, Section 3.3: http://www.cs.wisc.edu/condor/manual/v6.6.10/3_3Configuration.html#param:UserJobWrapper
- [25] The User-mode Linux Kernel Home Page: <http://user-mode-linux.sourceforge.net/>
- [26] Stowe, J. Large, Fast, and Out of Control: Tuning Condor for Film Production. Presentation at Condor Week 2005, Madison, WI, USA, 14-16 March 2005: http://www.cs.wisc.edu/condor/CondorWeek2005/presentations/stowe_core.ppt
- [27] Condor[®] Version 6.6.10 Manual, Section 2.4.1: http://www.cs.wisc.edu/condor/manual/v6.6.10/2_4Road_map_Running.html#SECTION00341000000000000000
- [28] Condor[®] Version 6.6.10 Manual, Section 2.5.2.2: http://www.cs.wisc.edu/condor/manual/v6.6.10/2_5Submitting_Job.html#1843
- [29] Condor[®] Version 6.6.10 Manual, Section 2.8.2: http://www.cs.wisc.edu/condor/manual/v6.6.10/2_8Java_Applications.html#SECTION00382000000000000000
- [30] Condor[®] Version 6.6.10 Manual, Section 3.7.5.1: http://www.cs.wisc.edu/condor/manual/v6.6.10/3_7Security_In.html#SECTION00475100000000000000