

Condor Local File System Sandbox

Requirements Document

1 Table of Contents

1	Table of Contents	2
2	Introduction	3
3	Describing the problem space	3
4	Defining the project's scope	4
5	Desired Architecture	5
5.1	The Policy Files	7
5.1.1	Description	7
5.1.2	Policy File Types	7
5.1.3	Policy File Structure	8
5.1.4	Applicability of a Policy Files	9
5.2	The SandBox Service	9
5.2.1	Description	9
5.2.2	An example of a bad Condor dependency	10
5.2.3	Functional Requirements	10
5.3	The SandBox Driver	11
5.3.1	Description	11
5.3.2	Functional Requirements	11
5.4	Integration – the changes in Condor	13
5.4.1	Description	13
5.4.2	Functional Requirements	13
6	General Requirements	14
6.1	System Requirements	14
7	Dependencies and assumptions	14

2 Introduction

This is the Requirements Document of the Condor File System Sandbox project.

This document is based on several Team meetings that focused on describing the problem space, the project's scope and other significant requirements of the project.

A great deal of the analysis took place in the team meeting that was held on 18/03/04, therefore this document will base its rationale on the discussion that took place during that meeting. (Though some concepts described there were abandoned during later team meetings)

Since after every meeting a report is compiled, one can refer to those reports for more information.

Never the less, this document is self contained and there is no obligation on reading the previous meetings' reports in order to understand this document.

Since this project can be branded in the category of highly technical projects, writing the requirements without peeking into the design is practically impossible. The requirements nearly enforce a specific design and therefore it seems to be wise to consider that specific design while writing the Requirements Document.

One can think of this document as the second iteration of the "Waterfall" design approach.

3 Describing the problem space

A little background of the Condor project

Condor is a wide spread distributed infrastructure that enables computing-resources sharing.

One can add a "job" to Condor's queue and that job, when its turn comes, will be executed on one of the Condor-registered computers.

Consider the following example as an introduction to the problem that arises from such architecture.

Say, a Condor user named Mat wants to run several CPU demanding jobs, He registers himself with the distributed Condor environment, and adds his jobs to the Condor job queue.

Mat believes that since other users share their CPUs with him the total running time of all his tasks will be shorter.

Condor starts dispatching Mat's jobs across the registered machines, and one of these tasks start to run on Alice's machine. Alice, up until today, considered Condor as a harmless environment that allows her to share her resources when she does not need/use them.

However, unfortunately, Mat's intentions were not that naïve. His code, besides doing the expected computations, reads the password file on all the machines that run it, specifically Alice's machine.

Now Mat receives all the user passwords that belong to the people who let him use their computer.

This is exactly where the Condor Sandbox comes into the picture. If Condor had run its distributed jobs under the surveillance of a SandBox, none of this would have happened.

Mat's process could not have had any access to the password file, since the Sandbox would have not allowed it to reach that file-system location.

4 Defining the project's scope

Sandboxing is a very wide goal.

The project's goal is to implement a sandbox of the local file system only. No network traffic monitoring, no CPU limitations, No network-mapped drives and not any other restriction.

Further more, this project's target running environment is based on Microsoft Windows 2000 Pro.

This sandbox should be integrated into the Condor system, though it does not need to support a non-service installation (there are two Condor flavors, a full-blown Condor client and a non-privileged service-less client)

The policies that define the restrictions should be given to the Sandbox from the Condor system and are assumed safe.

They are considered safe since the Condor System will incorporate a mechanism that inspects a process's requested access rights and intersects them with the allowed access rights. Defining, designing and implementing this mechanism are not in the scope of this project.

Further more the Condor system will have to pass these Policy files in a secure and authenticated way to the Sandbox.

Given a list of processes' IDs and restriction policies, the Sandbox will restrict the desired processes of file system access, upholding the restrictions defined by the policies.

The list of processes' IDs will be given by the Condor System, since it will be the one that spawns the executables.

The Sandbox should not be dependant of the Condor processes in order to work correctly, namely, if an executable is spawned via Condor, it shall be under surveillance even though the Condor system might crash.

A Condor-job should not get privileged access under any circumstances!

One last worthy note is that currently the Sandbox will not be able to *fully* handle surveillance of processes that are spawned from under-surveillance processes.

The Sandbox *will* assure that any process that is run under a Condor user (there might be several Condor users) will be under surveillance or at least be more strictly restricted, either if the process was spawned by the Condor system or by any other process (such as a Condor-job)

A corollary to that is that if a Condor job spawns a process under the context of a non-Condor user it will not be monitored by the Sandbox and by that, it will escape the sandbox.

This is a major security hazard, since a malicious user can put his malicious code in a spawned process under a non-Condor user (if he has such user credentials) and by that, he will not be restricted by the Sandbox.

The reason this issue is not mitigated in this project is only because of the effort that might be needed to uphold this restriction, making this task out of scope for this project.

5 Desired Architecture

The requirements and the limitations that the Sandbox needs to uphold forces us to think of the following high-level architecture as an adequate solution.

This high-level design is really a requirement that needs to be fulfilled in order for the Sandbox to be able to complete its tasks. It is obvious that the Sandbox, in its core, needs to be developed as a filter driver. Since in this manner the sandboxing will be done in kernel mode and that will ensure that the sandbox cannot be easily hacked.

In the coming sub-sections, we shall further describe the requirements of each sub component that is related to the Sandbox solution. (As described in the diagram above)

5.1 The Policy Files

5.1.1 Description

This section describes one of the most important aspects of the project. The policies are a set of rules that will describe the accessing rules, of a process spawned by Condor, to the file system.

Refer to the "Process Policy" and the "System Policy" tables in Figure1.

These Policy files shall be considered as trusted since they will be generated by the Condor System or a Condor Administrator using a mechanism that will ineligibly intersect the process's requested access rights and the allowed access rights by the Condor system. Validating the correctness of these policy files is out of scope for this project. (See Section 7.5)

5.1.2 Policy File Types

Three kinds of Policy sets will be used:

1) **System Policy**

- a) These shall be an "Out-Of-The-Box" set of fundamental rules allowing access to files that every Win32 process will need to load.
- b) The creation of the System Policy is not done on-line for each executable, but rather it is done off-line and infrequently, by a Condor Administrator user, in order for the System Policy to comply with standard executables' binaries dependency tree.
- c) This file will be loaded every time the driver is loaded, and only then.

2) **Process Policy**

- a) These are the set of rules that we want to use in order to restrict the process's accesses to the file system.
- b) This Policy file will be supplied by the Condor system for each Condor-job.
- c) It is the responsibility of the Condor System to provide a Process Policy file that complies with the correct restrictions.
- d) This file will be loaded by the driver before the Condor-job executes a single code instruction of its own.

- e) This file will be evacuated from the Driver only after the process has terminated
- 3) **Default Policy** – (Optional)
- a) A set of rules that apply to any spawned process, though not critical for it to load. (i.e. "c:\temp").
 - b) This Policy file will be supplied by A Condor Administrator for each Condor client machine.
 - c) It is his responsibility to provide a Default Policy file that complies with the correct restrictions.
 - d) This file will be loaded every time the driver is loaded, and only then.

5.1.3 Policy File Structure

1. We will adopt the firewall policy rules model for the policy files.
2. A policy file (any one of those describe above), is constructed of access rules.
3. Each access rule will be placed in a separate line.
4. The rules are ordered and processed by the order they appear in the file.
5. Each line will be of the following format:
<Path> <AccessType> <Action>
 - a. Path
 - i. This will be a full path to a file system object (file or directory):
 1. i.e.:
c:\MyFolder\OtherFolder\RequestedFile.ext
 - ii. Wildcards are allowed only at the end of the file name (in order to simplify the handling of regular expressions)
 1. i.e.: c:\MyFolder\OtherFolder\Requested*
 - iii. System environment variables are also allowed
 1. i.e.: %WinDir%\System32
 2. This is not a security hazard since in any case we must rely on the system environment variables for an application to be correctly loaded into the memory.
 3. These are not standard environment variables - only an administrator can change these environment variables
 - b. AccessType
 - i. This can be one of the following {Read, Write, Execute}

1. For a more fine-grained list of access types refer to the Windows documentation
 2. Supporting the Access types above is mandatory, any other access type should be considered as a sub-access-type to one of the three above if it is not handled on its own
 3. Consider, during implementation, to create a "tree" of access types for simple yet powerful, access rule creation.
 - a. i.e.: All access types of the "Read" class can be denied except for the "notification alerts", which is a one of the "Read" access types.
 4. Sub-typing of any sort will be implemented if time permits and is an optional requirement.
 - ii. The rule specifies what kind of access it permits or restricts using this field
- c. Action
- i. This can be one of the following {Allow, Deny}
 - ii. The rule specifies if the AccessType to the Path is allowed or denied for the process

5.1.4 Applicability of a Policy Files

1. Since a specific policy file is attached to every executable, the rules will apply only on the related process and not on all the other processes.
2. An exception to the rule above is, of course, the System Policy and the Default policy that will be applied to all the executed processes
3. The origin of the Process Policy file will always be from the Condor system, and it will be assumed that it can be trusted (As stated above in the Section 5.1.1).

5.2 The SandBox Service

5.2.1 Description

One of the most significant requirements is that the Sandbox would be as self-contained and as self-dependent as possible. Further more, depending on the Condor processes is not acceptable since we would not want a crash in condor to cause

any performance reduction or even worse, a crash in the sandbox driver (which will cause a blue screen!)

5.2.2 An example of a bad Condor dependency

As described in Figure1 the Condor_Starter is the process that spawns the requested condor-job.

The Condor_Starter process will have to send the Process Policy to the driver in some way in order to advise the driver of the relevant policy rules of the new job.

Figure1 also illustrates that the driver will allocate memory for that Policy file. Therefore, it will need to be notified when the process is done so it can release the allocated memory.

If for some reason (a fault in Condor_Starter or even a DOS attack) the Condor_Starter terminates, before the job it spawned terminates, then the sandbox driver will not be notified when the job is terminated, leading to a memory leak and a possible DOS.

5.2.3 Functional Requirements

- 1) The service should be a standard Win32 Service application
- 2) The service will verify that Driver has accepted and applied the relevant data regarding a new process that needs to be spawned, before it replies to the Condor_Starter allowing it to resume the suspended process (See [Integration – the changes in Condor](#))
- 3) Basic logging will be introduced
 - a. Log message categories are:
 - i. Info – For normal behavior events (i.e. successful loading of the driver/service)
 - ii. Error – For abnormal functional behavior (i.e. failure in loading the driver/service)
 - iii. Warning – For attempts to breach the sandbox security model (i.e. a process tries to access a file conflicting with its access policy)
 - b. All log messages will be saved in a log file
 - c. Logging needs to be reasonable, and will be defined during the implementation phase.
- 4) The service should always be running in parallel to the Sandbox Driver.
- 5) The service will be the only software component that sets and gets the Sandbox Driver's settings.
- 6) The service will be in charge of spawning the Condor-job and it will monitor it till it terminates

- 7) It should notify the Sandbox Driver when a Condor-job has terminated, for it to clean up unused resources
- 8) It needs to be robust and prevent a normal process, or even an administrator, to crash it brutally. (Of course it should make it as hard as possible, if it is not feasible to eliminate the ability altogether)
- 9) It should terminate all Condor-jobs when its standard shutdown sequence is called (therefore, it'll notify the driver of their termination)
- 10) It should not accept any requests issued from non-privileged processes. Only processes run in the context of the System account user may interact with the Service.
- 11) The Service will parse the Policy files and validate their correct structure and only then it'll send the data to the Sandbox Driver

5.3 The SandBox Driver

5.3.1 Description

This software component is the only code that will run in kernel mode.

Its main goal is to restrict the file-system IRPs that are sent via the Condor-jobs according to the Policies described above.

Being a kernel-resident component will make it harder for any hacker to detour the restriction mechanism it imposes.

5.3.2 Functional Requirements

- 1) The driver will be a standard Win32 filter driver.
- 2) Basic logging will be introduced
 - a. Log message categories are:
 - i. Info – For normal behavior events (i.e. successful loading of the driver/service)
 - ii. Error – For abnormal functional behavior (i.e. failure in loading the driver/service)
 - iii. Warning – For attempts to breach the sandbox security model (i.e. a process tries to access a file conflicting with its access policy)
 - b. All log messages will be saved in a log file
 - c. Logging needs to be reasonable, and will be defined during the implementation phase.
- 3) The driver will supply two communication channels to it, one for the Service's configuration messages and one for the Condor-job's file-system access requests.

- 4) The Service is the only means of communication with the driver; the driver will not accept any requests from any other software application but the Sandbox Service.
- 5) The Condor Driver will hold some kind of independent representation of the Policy files' rules that will be passed to it by the Sandbox Service.
- 6) For each process under surveillance, the Driver will have a separate Process Policy that it will enforce on the file system accesses issued by that process
 - a. If a process that was spawned in the context of a Condor user attempts to access the file system and the Driver cannot find a Process Policy for that process it shall enforce the implicit "Deny All" rule on all such accesses.
 - i. Such a process might be one that was spawned by another process that is under surveillance
 - ii. There might be several Condor users known to the Driver.
- 7) The driver will have a "built-in" System Policy that will allow the loading of applications
- 8) If a file-system access is issued by a Condor-job, and there is no rule that permits this access request, then the implicit rule that denies all requests will be triggered.
- 9) When a Condor-job has terminated the Driver can safely free the resources consumed for the task of restricting that process (Such as, the memory that was allocated for the Process Policy)
- 10) During the Sandbox shutdown sequence, the driver will be the only software component that will not be unloaded, though it will be neutralized, namely it shall not apply any policies on any file system access attempts.
 - a. All attempts of this sort will continue to pass through the driver, since it is not unloaded
 - b. All attempts will be sent directly to the next driver in the chain (At least there will be the FS driver), since the Sandbox is disabled.
 - i. This should be done with minimal performance impact on the file system access requests

5.4 Integration – the changes in Condor

5.4.1 Description

The Sandbox needs to be secure from attacks that try to relax the configuration (Policies) for a given process.

In order to achieve such a goal we must allow only trusted software components to be allowed to feed the Sandbox with the Policy files.

The main assumption is that the Condor Code that is run in the context of a System user account can be considered a trusted software component.

This assumption is based on the fact that only a privileged user can run an executable in such a user context.

Therefore, the only reasonable solution is to alter the Condor_Starter's code so that it will notify the Sandbox of a newly spawned process.

Condor_Starter is run under the context of a System user account and therefore will guaranty that the Service can, and will, trust it. Since we would like to make as little changes to the Condor code-base as possible, the change that will be described here should be considered as a requirement.

Instead of regularly spawning the process, the Condor_Starter will spawn it in suspended mode. Thereafter it will pass its process ID and its Process Policy to the Sandbox Service. Only after the Service acknowledges the Condor_Starter that the Driver correctly incorporated the new Policy, will the Condor_Starter resume the suspended process.

5.4.2 Functional Requirements

- 1) The change will be done in the Condor_Starter code only
- 2) If the Sandbox will not be installed or enabled the change will allow the Condor System to behave as it has before the change
- 3) The Condor_Starter will still own the task of spawning the Condor jobs
- 4) The Condor_Starter will inform the Sandbox Service when it needs to execute a job before a single line of code is executed in the Condor job.
- 5) The Condor_Starter will wait for the Sandbox Service's acknowledgement that will be sent only after the Sandbox Driver incorporates the new Process Policy

6 General Requirements

6.1 System Requirements

The Sandbox should be installed on a Windows 2000 Pro machine.

7 Dependencies and assumptions

- 1) An installation process should install and configure the Sandbox
- 2) An uninstallation process should leave the machine as if the sandbox was never installed (may require a reboot)
- 3) The Condor System will be altered, as described in Section 5.4.
This change, which will be made in the Condor_Starter, will notify the Sandbox Service of a new process in case the sandboxing is enabled.
- 4) The Condor system will have to notify the Sandbox of a new process it wishes to spawn before the process executed a single code instruction
- 5) The Condor system will supply the Sandbox of the relevant Process Policies.
 - a. Generally, these policies will be the intersection between the process declared requested resources and the resources the Condor System is willing to allow access.
 - b. Therefore they will be considered as trusted policies if and only if they will be passed to the Sandbox by a trusted Condor software component
 - i. A trusted Condor software component will be identified as a process or service that runs in the context of a system account user
 - c. Policy files, if saved to the local disk, will be secured with strict ACLs to prevent alteration by a malicious user.
- 6) The Sandbox will restrict all processes that the Condor system will advise it to.
 - a. Any process that is spawned from the restricted processes will be handled as follows:
 - i. If the spawned process is run under a Condor user's context, it will be restricted by the implicit "Deny All" access rule.

- ii. If the spawned process is run under a non Condor user's context, no restrictions will be enforced on this process – it will escape the Sandbox
 - b. The Condor system should not spawn its jobs using the machine-owner's user, since this will cause the Sandbox to restrict all applications run by that user.
- 7) It is assumed that the Condor system's installation process correctly sets the ACLs on the Condor binaries denying all non administrative users any access to them
 - a. This assumption is needed for the Sandbox to be able to trust the Condor binaries
- 8) The Condor system will be installed in the Service flavor, namely the Condor daemons will be executed as Services (in the context of the, trustworthy, system account user)