

File and Memory Security Analysis for Grid Systems

Unnati Thakore and Lorie M. Liebrock

Department of Computer Science, New Mexico Institute of Mining and Technology, USA
{unnati, liebrock}@cs.nmt.edu

Abstract. The grid security architecture today does not prevent certain unauthorized access to the files associated with a job executing on a remote machine. Programs and data are transferred to a remote machine for completing staging jobs. Footprints may remain on the machine after the programs and data are deleted from the remote machine. Anyone with super user privileges can access these footprints. In this paper, we explore the problem of unauthorized access to programs and data, supporting it with experiments carried out using the Globus toolkit. After showing that programs and data are at risk, we also discuss ways for secure deletion of data and programs on the remote machine.

1 Introduction

One of the goals of any security architecture is confidentiality, which ensures access to computer resources must be authorized. Grid security research and development currently revolves around developing better solutions to address the following requirements: Authentication, Secure Communication, Effective Security Policies, Authorization, and Access Control [11]. For example, Globus Grid Security Infrastructure (GSI) provides X.509 certificates with the use of TLS (Transport Layer Security) protocol for mutual authentication and delegation. GSI by default provides unencrypted communication but can be configured to use encryption for communication [2, 3, 13].

The policies for access control and authorization as applied to grid computing today affect access to resources by authenticated grid users abiding by authorization policies on the resource [11]. Other policies define the resource usage and security provided by the resource providers in the VO (Virtual Organization) [12]. Access to grid resources, which are spread across geographical and political boundaries, is based on a trust relationship between the resource providers and the users. Thus both have to stick to the security policies agreed upon by both parties. Before a grid user accesses any grid resource, the user needs assurance that the machine is not compromised and his data and programs will not be stolen [10]. However, the users and providers may not be aware of the system being compromised or having internal breaches based on some users having administrator privileges. User data is at risk if there is any significant compromise of any system on the grid.

To our knowledge, no grid security architecture deals with securing a grid user's data and programs on the remote machine from any user that has super user privileges; there are no policies governing this type of confidentiality requirement. In other words, we have found no grid security policies that deal with confidentiality of user data and code on the grid system with respect to super users. Although super user related risks include insider threat, these risks also include risks associated with any

hacker that compromises a component of the grid system to an extent that provides super user access. In this paper, we discuss this confidentiality issue and show how the programs and data of a grid user are in jeopardy on a remote machine.

An experimental grid was set up to search for the footprints of a grid user's data and program on the remote machine. Experiments were carried out using the Globus tools for job submission on a grid implemented with the Globus toolkit version 4.0.3. Physical memory and secondary storage on the remote machine were examined for footprints using standard digital forensics tools. The results indicate when a grid user's program and data on the remote machine can be at risk. We then discuss some ways to prevent this unauthorized access in certain scenarios or to at least limit the time frame during which such access could occur.

The paper begins in Section 2 with explanation of what can be extracted by reading secondary storage and physical memory on a grid system. Then the paper describes details of our experimental configuration in Section 3. Section 4 describes the experiments carried out to find footprints of the grid user's data and programs in secondary storage and physical memory. Section 5 explains how a grid user's data and programs can be deleted from secondary storage and outlines a plan for how to remove them from physical memory. The paper finally concludes after a brief discussion of future work.

2 Memory and Storage Analysis

Data Lifecycle Management (DLM) is an important security requirement for Grids [8, 13]. DLM is the process of managing data throughout its lifecycle from generation to its disposal across different storage media for the span of the entire life of the data. The data lifecycle is the time from data creation to its indefinite storage or deletion. There is a need for a security system that protects the grid user's data from all other users, even the super user. In order to determine whether current grid systems provide this protection, we need to analyze the memory and storage on grid systems. Ideally, backups and other storage media should be considered. However, here we focus on the standard memory and storage on grid systems. We recommend encryption of data when it is in storage, so that all backups will also be more secure. Next, we show how such analysis for memory and secondary storage is performed.

2.1 Secondary Storage Analysis

Secondary storage of a machine on a grid system can be analyzed to locate data and programs. A super user can read any grid user's data and programs as the current security systems don't restrict the super user's access. Based on the method used for deleting data and programs (<fileCleanUp> attribute of job description language or the UNIX rm command), files are just deleted from the file system. Deletion of files typically entails just removing the metadata and adding the storage space to the available space list. This deletion is not a secure deletion of files on the remote system. These files could be temporary files, executables, program files, data files, output files, stdout, or stderr.

The normal deletion of files from secondary media is not secure, as the file contents are still stored on unallocated space of the secondary storage. The deleted

files can be read by a variety of software or utilities (e.g., `dd` command of UNIX) that can read raw images of the storage. Due to the nature of grid systems, multiple users can be mapped to a single account on local machine. Therefore, there it may be possible for the users to read footprints left by previous users.

For securely deleting files from secondary storage it is necessary to overwrite that data with some pattern on the secondary storage. The number of times the data of the file is overwritten determines whether the data from the deleted space can be extracted [5]. The required level of confidentiality of the file to be deleted determines the number of times it has to be overwritten. There are many free tools and utilities that securely delete files from secondary storage. An example is the UNIX command `shred` that deletes files securely.

Another place to look for footprints is swap space, which is used for swapping out inactive jobs when system resources are low. With the size of RAM increasing, there are fewer possibilities to find much on the swap space, unless the system is heavily loaded [9]. Virtual memory provides the same opportunity.

Slack space is another place to look on the disk, which can be used to read information, which was previously written to the disk. Slack space is part of an allocated file that has not yet been written to and thus retains the old data (this is at the end of the last block of a file).

Overwriting is one method to prevent reading deleted data using programs that read raw data. However another analysis method, magnetic force microscopy (MFM), can still read overwritten data. MFM is a technique based on imaging magnetization patterns with high resolution and minimal sample preparation [6]. Although this technique can be used in forensics, it requires access to the physical media and this limitation would protect the data from hackers, at the very least. Further, most super users do not have access to MFM, so we do not consider this a significant risk for grid users unless there is a critical forensic analysis in progress.

Thus, it appears that analysis of hard drives on a grid system can let anyone with super user privileges read data and programs or their footprints. In particular, our experiments assess how much can be found and under what conditions.

2.2 Physical Memory Analysis

Physical memory analysis can be used to extract text and executable files, detailed information about terminated and executing processes, open network connections, and passwords [1]. Some confidential information access may prove to be fatal for a grid user, as this information can be used on behalf of that user on other sites. The physical memory can be analyzed to read raw data and meta data. Raw data is the execution code or data section from a memory mapped file or temporary data structures such as the stack used by the executing processes. Meta data is the data related to memory structures of page tables, processes, etc.

Physical memory (RAM) on Linux machines can be read using `/dev/mem` or `/proc/kcore` with administrator privileges. `/dev/mem` is a device file that mirrors main memory and the byte offsets are interpreted as the memory addresses by the kernel. `/proc/kcore` is an alias to the RAM. When we read from `/proc/kcore` the kernel performs the necessary memory reads. The output of `kcore` is in Executable and Linkable Format (ELF) and can be read using `gdb`, whereas the output of `/dev/mem`

can be read using an ascii editor or examined by running strings on it. The System.map (the name depends on the kernel version) contents provide a map for addresses of important kernel symbols, which can be used to find the addresses of system calls, the address of memory attached to a process, etc. [1]. The physical memory contents are not erased after the process ends, but are stored as free memory until it is overwritten (as shown in Experiment III) or the power is turned off [9]. Thus an image of physical memory can give details related to the process and its data.

3 Experimental Grid Setup

A grid with two machines connected to each other using Globus Toolkit version 4.0.3 with default settings was configured [3]. Details of the grid environment (machines Grid0 and Grid1) are:

- 1) Before installing the globus toolkit, the prerequisites were installed on both of the machines. The basic prerequisites installed are j2sdk-1.4.2.13 and Apache Ant 1.6.5.
- 2) For consistency, Fedora Core 5 was installed on both machines. The Fedora Core 5 kernel (2.6.19) was patched to enable administrator access to /dev/mem.
- 3) Grid0 has 512 MB of system memory and a 30 GB hard drive. Grid1 has 256 MB of system memory and a 40 GB hard drive.
- 4) The minimum set of users is created on both machines. Grid0 has globus, postgres, and student in addition to root. Grid1 has globus and student1 in addition to root. The user student1@Grid1 maps to student@Grid0 when executing jobs.
- 5) The PostgreSQL database was installed on Grid0 to support the RFT (Reliable File Transfer). RFT is a web service that provides scheduling properties for data movement, for which it requires a relational database to storing states [3].
- 6) Tomcat was installed on Grid1 to support WebMDS (MDS - Monitoring and Discovery System), which is a web based interface for viewing information about grid services.
- 7) A Simple CA certificate authority was set up for signing user certificates on Grid0 to support grid operation.
- 8) Firewall settings were changed to allow tcp traffic for ports 8443 (GRAM and MDS), 2811 (GridFTP), and 5432 (PostgreSQL). A range of ports were added on both machines for temporary connections to PostgreSQL for RFT transfers.

3.1 Job Submission and Execution

This project uses WS - GRAM for job submission and management. Job submission to WS-GRAM requires a valid X.509 proxy certificate. WS GRAM has many command line clients to submit jobs, of which we use globusrun-ws, the official command line client, to submit jobs. The jobs submitted are staging jobs specified in a job description file using the standard job description XML schema.

4 Analysis on the Experimental Grid Setup

The following sets of experiments were carried out to find footprints of the grid user's data and programs that reside in the secondary storage and physical memory. Each

experiment was run numerous times with results always being consistent with the observations presented below. In the tables that follow for each experiment, the approximate number of times the experiment was run is listed along with the summary of results. Any super user can access any grid user's data and programs. Such files can be accessed (until they are overwritten) even after the files are deleted by a grid user.

4.1 Experiment Set I

The experiments carried out consist of executing a simple Perl script on the remote machine. This Perl script (Appendix I) creates a temporary file and writes easily identifiable strings in the file and then deletes the file using the UNIX `rm` command. The job was submitted from Grid1 to Grid0 using the job description file (Appendix II). During the execution of that script on Grid0 a dump of the physical memory (`/dev/mem`) was taken using the standard forensic tool `dd` on an external hard drive. After the job was executed, an image of the hard disk (`/dev/hda`) on Grid0 was taken using the `dd` command. Then the dumps were searched using the UNIX command `grep` to find the strings that were written in the file.

Observations. The experiments of remote execution were performed many times to observe and verify consistent behavior depending on the size of the file created by the Perl script on the remote machine (Grid0).

Small Files. Small files (with size around 24MB) created by the Perl scripts were tested. A search on the secondary storage dump did not reveal the file created by the Perl script, although a search on the physical memory dump showed the contents of the file.

Large Files. Large files (with size greater than 40MB) created by the Perl script were tested. A search on the secondary storage as well as in physical memory showed the contents of the file.

File Risk Summary. In summary, Experiment Set I shows the following.

File size	# Experiments	Files found in secondary storage	Files found in physical memory dump
~ 24 MB	90	0%	100%
> 40MB	75	100%	100%

As an indication of the degree of security risk detected, let the risk associated with a file being found in storage be 70% risk and the risk associated with a file being found in the physical memory dump be 30%. This is an arbitrary allocation of risk, but it illustrates the point that the risk of information being recovered is higher in storage than it is in memory, as memory is more volatile. Further, when the data is located in both, methods can be used on either to recover the information so the risk

of both recovery approaches is present. Note that this discussion does not consider all security risks, e.g., it does not account for the security risk associated with transferring data across the Internet. However, with these caveats, we arrive at the risk analysis graph in Figure 1, where the percentage of risk is shown on the y-axis and risk associated with different security issues are shown on the x-axis.

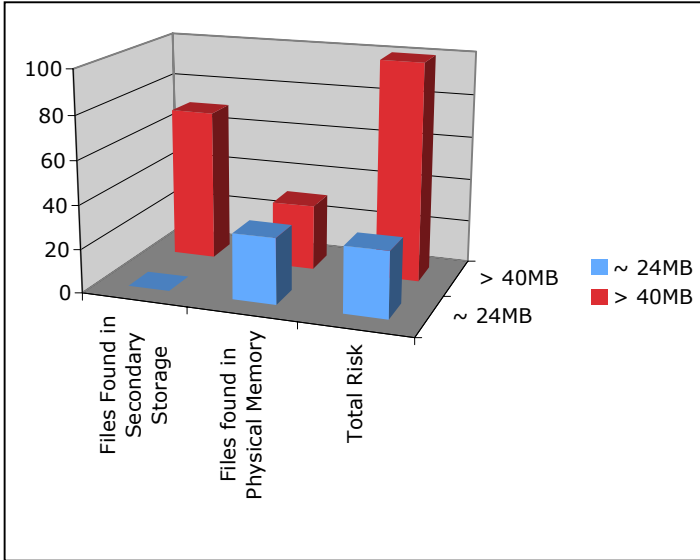


Fig. 1. Security risk using rm for file removal

Other Findings. The Perl script submitted was found in the secondary storage dump of Grid0. The contents of files written by earlier scripts were found on the physical memory and secondary storage dump.

4.2 Experiment Set II

The experiments carried out consist of executing a modified version of the Perl script used in Experiment Set I. The modified Perl script replaces the UNIX rm command with the UNIX shred command to securely delete the file it creates on the remote machine. Note that, as a reviewer pointed out, the secure deletion of files is based on the operating system. Here we are using a built in UNIX command. If the operating system does not have a command in the standard distribution, then we would recommend an operating system extension or add on.

During the execution of that script on Grid0, a dump of the physical memory (/dev/mem) was taken using the dd command on an external hard drive. After the job was executed, the image of the hard disk (/dev/hda) on Grid0 was taken using the dd command. Then the dumps were searched using UNIX command grep to find the strings that were written in the file.

Observations. A search of the secondary storage dump did not show the file created by the Perl script but a search on the dump of physical memory did find the contents of the file. A search on the secondary storage dump showed the existence of the Perl script that was submitted. The execution of the Perl script takes a long time because of overwriting the storage space many times (25 times as a default number) with the shred command. For many users, the number of overwrite iterations could be significantly lower and still provide adequate security.

File Risk Summary. In summary, Experiment Set II shows the following.

File size	# Experiments	Files found in secondary storage	Files found in physical memory dump
~ 24 MB	60	0%	100%
> 40MB	45	0%	100%

Using the same indications of the degree of security risk detected, we arrive at the risk analysis graph in Figure 2.

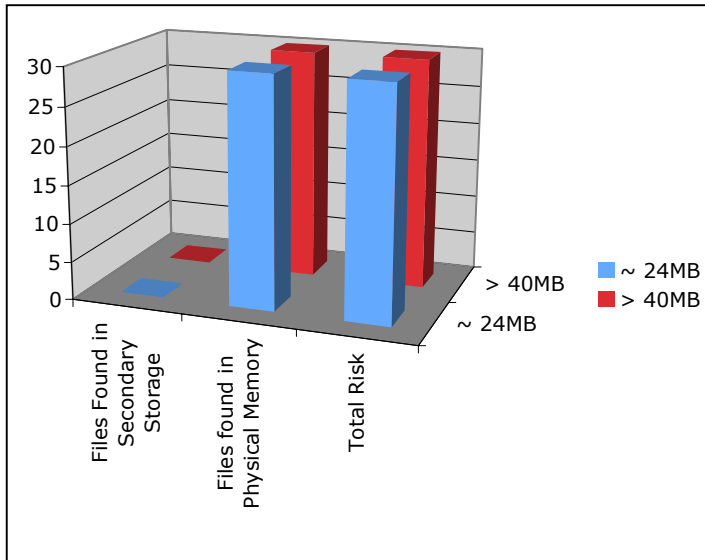


Fig. 2. Security risk using shred for file removal

4.3 Experiment Set III

The experiment carried out consisted of adding the following script [9] to the Perl script used in Experiment Set I and creating a new script with just the following lines

of code. This script attempts to overwrite all accessible user memory. These files were submitted for execution on Grid0. After the job was finished, the dump of primary memory was taken on the external hard drive. Then the dump was searched using the UNIX command grep to attempt to find the strings that were written in the original file.

```
for (;;) {
    $buffer[$n++] = '\000' x 4096;
}
```

Observations. In both cases (the combined scripts and the separate scripts), the process was killed after some time but the physical memory dump showed the memory was overwritten with \000. Thus the strings, which were originally written in the file, were overwritten by \000 preventing the original strings from being shown.

File Risk Summary. In summary, Experiment Set III shows the following.

File size	# Experiments	Files found in secondary storage	Files found in physical memory dump
~ 24 MB	10	0%	0%
> 40MB	10	0%	0%

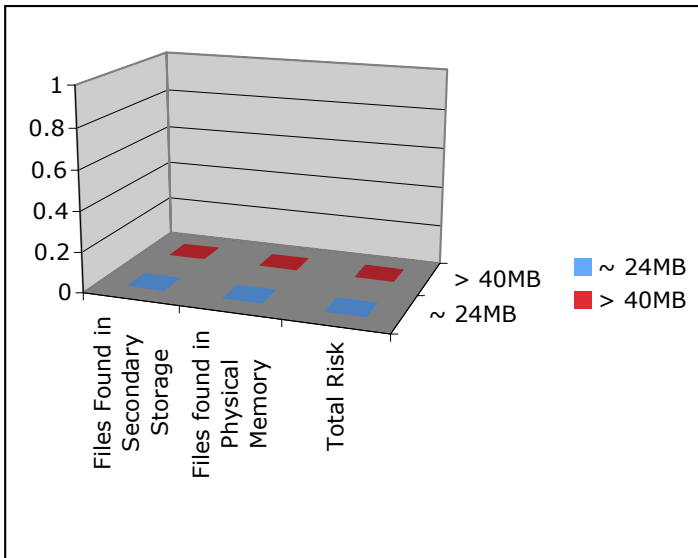


Fig. 3. Security risk using shred and the buffer erasing script for file removal

Using the same indications of the degree of security risk detected, we arrive at the risk analysis graph in Figure 3.

5 Secure Deletion

In order to overcome some of the security risks that have been illustrated in this paper, we now present some simple deletion methods that reduce these dangers.

5.1 Secondary Storage

Secure deletion methods depend on the operating system and file system on the remote machine. Securely deleting the code and data from the hard drive on a Linux machine with a default Linux file system can be done using the `shred` command (as shown in the experiments). In the case of Windows machines, there are no such included commands that securely delete files. Thus, in case of remote machines having Windows, the grid user should make sure there is secure file deletion software installed. If no such software is installed, the user should in addition to sending the files for the job, send an executable to be used to securely delete files that were transferred or created. There are free software and utility programs (e.g., `sdelete`, `eraser`, `cipher`, etc.) available for Windows that can be used to securely delete the data and code. Each of these scripts works in the Windows operating system framework, but extends the standard capabilities of the operating system.

5.2 Physical Memory

Removing the data files from physical memory can be dependent on the memory management system of the underlying operating system. One naive way to erase the data/code from memory is shown in Experiment Set III. Even repeated runs of such programs as root only changes about 3/4 of the main memory. Such programs do not overwrite any of anonymous memory, kernel, or file caches [9]. The best way to eradicate everything from physical memory is to reboot the system, but that is not possible for a normal user in a grid environment. The naive method shown tries to overwrite as much memory as a process can access and when as it exceeds its limit the process is killed by the operating system. This reduces the likelihood of finding the data and programs of the grid user in the physical memory. Since the data was originally written in the user space, we should be overwriting that same memory with this deletion process. However, more experiments are needed to verify this assumption and to ensure that data and code are not sometimes stored in anonymous memory or file caches.

Another approach that may be considered is to overwrite all variables in the program before termination of the simulation or solution code. This, however, is more complicated and we have not experimented with this approach yet.

5.3 Other Storage

Note that we have not addressed any backups or offline storage systems. If the data were encrypted, when not in actual use, then with high probability, the data in other

storage systems would be encrypted. The caveat of “with high probability” is needed for the case where the data must be decrypted and temporarily put in storage. If that occurs, then it would be possible, but not probable, that a backup would occur while the data was stored in an unencrypted form.

If the unencrypted data is only in memory, and never put in user storage, then there should not be a backup created as systems typically only create backups of files in storage, not files in memory.

In summary, the presence of other storage systems and backups creates an added level of security risk for access to the data. If the data is not encrypted and securely deleted, it can be recovered from any of the media on which it is stored.

6 Future Work

This work was a side effect of our beginning work on Grid Forensics. In the future, we intend to carry out more detailed memory analysis and extend attribution methods for use in Grid Forensics. In particular, we will extend our experiments on secure file and memory deletion to provide better guidelines for grid user data and code protection. We will also explore the encryption of data while in untrusted file systems.

In addition to memory analysis, other vulnerabilities of the grid systems will be studied to inform grid users of the risks to their data and code, as well as for use in Grid Forensics. For example, network forensics will be applied to grids in order to find other means to collect code and/or data belonging to grid users.

7 Conclusion

Currently, it is the responsibility of the grid user to determine whether they wish to trust their code and data on a remote file / computer system. Here we examined what risks are associated with the decision to extend such trust.

In particular, we analyzed the problem of unauthorized access to deleted data and program files on a grid system. We have shown that both data and code can be found in secondary storage and memory by anyone with super user access. Grid users must be made aware of such security loopholes and provided with methods to avoid them. We have also illustrated how such unauthorized access can be avoided or limited in certain cases.

References

1. Burdach, M.: Digital Forensics of the Physical Memory (2005), http://forensic.secure.net/pdf/mburdach_digital_forensics_of_physical_memory.pdf
2. Welch, V., Foster, I., Kesselman, C., Mulmo, O., Pearlman, L., Tuecke, S., Gawor, J., Meder, S., Siebenlist, F.: X.509 Proxy Certificates for Dynamic Delegation. In: 3rd Annual PKI R&D Workshop, pp. 42–58 (2004)
3. Globus Toolkit 4.0 Release Manuals (2007), <http://www.globus.org/toolkit/docs/4.0/>

4. Foster, T., Kesselman, C., Tsudik, G., Tuecke, S.: A Security Architecture for Computational Grids. In: Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83–92 (1998)
5. Mallery, J.R.: Secure File Deletion, Fact or Fiction (2006), <http://www.sans.org/rr/papers/27/631.pdf>
6. Gutmann, P.: Secure Deletion of Data from Magnetic and Solid-State Memory. In: Sixth USENIX Security Symposium Proceedings, San Jose, California, pp. 77–89 (1996)
7. Foster, I.: A Globus primer (2005), www.globus.org/primer
8. Naqvi, S., Arenas, A., Massonet, P.: Scope of Forensics in Grid Computing - Vision and Perspectives. In: Proceedings of the International Workshop on Information Security and Digital Forensics. LNCS, pp. 964–970 (2006)
9. Farmer, D., Venema, W.: Forensic Discovery. Addison Wesley Professional, Boston (2005)
10. Humphrey, M., Thompson, M.R.: Security Implications of Typical Grid Computing Usage Scenarios. In: Proceedings of HPDC, pp. 95–103 (2001)
11. Humphrey, M., Thompson, M.R., Jackson, K.R.: Security for Grids. Proceedings of the IEEE 93(3), 644–652 (2005)
12. Verma, D., Sahu, S., Calo, S., Beigi, M., Chang, I.: A Policy Service for GRID Computing. In: Proceedings of the Third International Workshop on Grid Computing (2002)
13. Sotomayor, B., Childers, L.: Globus Toolkit 4: Programming Java Services. Morgan Kaufmann, San Francisco (2005)

Appendix I: Perl Script for File Creation

```
#!/usr/bin/perl
open ( TMPFILE , ">outputFile" ) || die "cannot open
outputFile for writing : $!" ;
for ( $i =0 ; $i < 10000 ; $i++ ) {
    print TMPFILE "5555aaaa5555aaaa5555aaaa5555aaaa\n" ;
}
close ( TMPFILE ) ;
`rm -rf outputFile` ;
```

Appendix II: Job Description File

```
<job>
  <executable>/usr/bin/perl</executable>
  <directory>${GLOBUS_USER_HOME}</directory>
  <argument>my_echo</argument>
  <stdout>${GLOBUS_USER_HOME}/stdout</stdout>
  <stderr>${GLOBUS_USER_HOME}/stderr</stderr>
  <fileStageIn>
    <transfer>

<sourceUrl>gsiftp://student1@Grid0:2811/tmp/second.pl</so
urceUrl>
  <destinationUrl>
```

```
        file:///${GLOBUS_USER_HOME}/my_echo
    </destinationUrl>
</transfer>
</fileStageIn>
<fileStageOut>
    <transfer>

<sourceUrl>file:///${GLOBUS_USER_HOME}/stdout</sourceUrl>
    <destinationUrl>
        gsiftp://student1@Grid0:2811/tmp/stdout
    </destinationUrl>
    </transfer>
</fileStageOut>
<fileCleanUp>
    <deletion>

<file>file:///${GLOBUS_USER_HOME}/my_echo</file>
    </deletion>
</fileCleanUp>
</job>
```