File System

Objective

To ensure that access to the local file system of any of the systems is protected from unauthorized creation, modification, or deletion.

Environments Affected

All.

Relevant COBIT Topics

DS11 – Manage Data – All sections should be reviewed

DS11.9 – Data processing integrity

DS11.20 – Continued integrity of stored data

Description

The file system is a fertile ground for average attackers and script kiddies alike. Attacks can be devastating for the average site, and they are often some of the easiest attacks to perform.

Best Practices

* Use “chroot” jails on Unix platforms
* Use minimal file system permissions on all platforms
* Consider the use of read-only file systems (such as CD-ROM or locked USB key) if practical

Defacement

Defacement is one of the most common attacks against web sites. An attacker uses a tool or technique to upload hostile content over the top of existing files or via configuration mistakes, new files. Defacement can be acutely embarrassing, resulting in reputation loss and loss of trust with users.

There are many defacement archives on the Internet, and most defacements occur due to poor patching of vulnerable web servers, but the next most common form of defacement occurs due to web application vulnerabilities.

How to identify if you are vulnerable

* Is your system up to date?
* Does the file system allow writing via the web user to the web content (including directories?)
* Does the application write files with user supplied file names?
* Does the application use file system calls or executes system commands (such as exec() or xp\_cmdshell()?
* Would any of execution or file system calls allow the execution of additional, unauthorized commands? See the OS Injection section (TODO) for more details.

How to protect yourself

* Ensure or recommend that the underlying operating system and web application environment are kept up to date
* Ensure the application files and resources are read-only
* Ensure the application does not take user supplied file names when saving or working on local files
* Ensure the application properly checks all user supplied input to prevent additional commands cannot be run

Path traversal

All but the most simple web applications have to include local resources, such as images, themes, other scripts, and so on. Every time a resource or file is included by the application, there is a risk that an attacker may be able to include a file or remote resource you didn’t authorize.

How to identify if you are vulnerable

* Inspect code containing file open, include, file create, file delete, and so on
* Determine if it contains unsanitized user input.

How to protect yourself

Only work with files

Insecure permissions

How to identify if you are vulnerable

How to protect yourself

Second Order Injection

If the web application creates a file that is operated on by another process, typically a batch or scheduled process, the second process may be vulnerable to attack. It is a rare application that ensures input to background processes is validated prior to first use.

How to identify if you are vulnerable

* Does the application use background / batch / scheduled processes to work on user supplied data?
* Does this program validate the user input prior to operating on it?
* Does this program communicate with other business significant processes or otherwise approve transactions?

How to protect yourself

* Ensure that all behind the scenes programs check user input prior to operating on it
* Run the application with the least privilege – in particular, the batch application should not require write privileges to any front end files, the network, or similar
* Use inbuilt language or operating system features to curtail the resources and features which the background application may use. For example, batch programs rarely if ever require network access.
* Consider the use of host based intrusion detection systems and anti-virus systems to detect unauthorized file creation.

Unmapped files

Web application frameworks will interpret only their own files to the user, and render all other content as HTML or as plain text. This may disclose secrets and configuration which an attacker may be able to use to successfully attack the application.

How to identify if you are vulnerable

Upload a file that is not normally visible, such as a configuration file such as config.xml or similar, and request it using a web browser. If the file’s contents are rendered or exposed, then the application is at risk.

How to protect yourself

* Remove or move all files that do not belong in the web root
* Rename include files to be normal extension (such as foo.inc 🡪 foo.jsp or foo.aspx)
* Map all files that need to remain, such as .xml or .cfg to an error handler or a renderer that will not disclose the file contents. This may need to be done in both the web application framework’s configuration area or the web server’s configuration.

Temporary files

Applications occasionally need to write results or reports to disk. Temporary files if exposed to unauthorized users, may expose private and confidential information, or allow an attacker to become an authorized user depending on the level of vulnerability.

How to identify if you are vulnerable

Determine if your application uses temporary files. If it does, check the following:

* Are the files within the web root? If so, can they be retrieved using just a browser? If so, can the files be retrieved without being logged on?
* Are old files exposed? Is there a garbage collector or other mechanism deleting old files?
* Does retrieval of the files expose the application’s workings, or expose private data?

The level of vulnerability is derived from the asset classification assigned to the data.

How to protect yourself

Temporary file usage is not always important to protect from unauthorized access. For medium to high-risk usage, particularly if the files expose the inner workings of your application or exposes private user data, the following controls should be considered:

* The temporary file routines could be re-written to generate the content on the fly rather than storing on the file system
* Ensure that all resources are not retrievable by unauthenticated users, and that users are authorized to retrieve only their own files
* Use a “garbage collector” to delete old temporary files, either at the end of a session or within a timeout period, such as 20 minutes
* If deployed under Unix like operating systems, use chroot jails to isolate the application from the primary operating system. On Windows, use the inbuilt ACL support to prevent the IIS users from retrieving or overwriting the files directly
* Move the files to outside the web root to prevent browser-only attacks
* Use random file names to decrease the likelihood of a brute force pharming attack

Old, unreferenced files

It is common for system administrators and developers to use editors and other tools which create temporary old files. If the file extensions or access control permissions change, an attacker may be able to read source or configuration data.

How to identify if you are vulnerable

Check the file system for:

* Temporary files (such as core, ~foo, blah.tmp, and so on) created by editors or crashed programs
* Folders called “backup” “old” or “Copy of …”
* Files with additional extensions, such as foo.php.old
* Temporary folders with intermediate results or cache templates

How to protect yourself

* Use source code control to prevent the need to keep old copies of files around
* Periodically ensure that all files in the web root are actually required
* Ensure that the application’s temporary files are not accessible from the web root

Further Reading

* Klein, A., *Insecure Indexing*  
  <http://www.webappsec.org/projects/articles/022805-clean.html>