hiding within the trees

by Glenn M. Brunette, Jr.

Glenn Brunette is the Chief Security Architect for Sun Professional Services in the United States and the co-founder of the Solaris Security Toolkit (a.k.a. JASS). He is focused primarily on the development of recommended practices, methodology, training, and tools to improve the quality and security of customer environments.

glenn.brunette@sun.com

Editor's Note: This article is somewhat specific to Solaris 9 but seemed of general interest to all those interested in filesystem technology.

"Come out, come out, wherever you are!" – Recall the popular refrain that brings back memories of childhood games such as "hide and seek." It was so much simpler then. Together, you and your friends defined the boundaries of play and then simply had fun. You could easily define what was in and out of bounds. It would have been an entirely different game if you or your friends could become invisible by hiding within the trees.

Hiding in the Filesystem

The purpose of this article is to highlight some of the methods that can be used to hide programs and data in a filesystem. This article focuses primarily on how extended file attributes, introduced in the Solaris™ Operating System ("Solaris OS"), version 9, could be abused for this purpose.

Hiding programs and data within the Solaris OS or any other operating system is not a new concept. In fact, variants of the UNIX® operating system encourage the use of "hidden" files to store users' preferences, configuration settings, and other attributes. These "hidden" files, known as "dot files," are not really hidden. They are simply not displayed using the ls(1) command unless the –a option is given. While any user can create "dot files" by creating a file that begins with the character ".", they are also easily detectable using the ls or find(1) commands as well as with filesystem integrity tools such as Tripwire or AIDE (assuming a baseline had previously been created that could be used for comparison).

Similarly, attackers have attempted to "hide" their programs and data by using naming conventions that map to similar but unused names on the filesystems. In the past, this has led to a plethora of names such as /dev/… and /usr/ccs/alpha. These files, just as with the "dot files" above, can only be created under directories to which the user has write access. Often, to the untrained or unfocused eye file names such as these look legitimate. As a result, they have been quite successfully used to hide programs and data on systems. The detection methods for files of this type are similar to those for "dot files."

In a similar vein, some developers have even been known to embed entire programs within existing software packages. Often these Easter Eggs, as they are called, are used to hide a game or some feature of the software. Unless discovered, these features will lie dormant on the system. A malicious developer could use this method to steal resources or information or even launch denial-of-service or other attacks. Once identified, however, a fingerprint of the affected software can often be developed to aid in the detection of the Easter Eggs. You must always be careful whom you decide to trust.

Another, more sophisticated method for hiding programs and data involves the use of file slack space. Slack space is the amount of space left over when the file does not end at a block boundary. Typically, small files tend to leave a significant amount of slack space on a filesystem, which can be used to store other information. Tools have been developed, such as bmap for the Linux operating system, to store and retrieve data from a file's slack space. The Solaris OS does not support the ability to write to slack space by default, but it is possible to hide information in slack space by writing directly to the disk device. Similarly, detection involves reading from the disk device.

Lastly, loadable kernel modules can be used to intercept system calls in order to hide programs and data according to some set of rules. These modules must be loaded into the kernel at each system boot, but they can provide a quick and easy method for an attacker to hide from an administrator. SLKM is an example of a tool that implements these basic file-hiding capabilities. Loadable kernel modules can be difficult to develop, but once written are easily used. Loadable kernel modules can be very difficult to detect and require offline analysis even when used in conjunction with filesystem integrity tools. The good (or possibly very bad) news is that only users with administrative privileges, such as root in the Solaris OS, can use the modload(1M) command to load a kernel module. If you find such a module running on your system, you can safely assume that a user or process with those privileges loaded it.

The Solaris 9 OS provides a new capability called extended file attributes which can permit any user to "hide" programs and data. This method is similar to the use of slack space in that the programs are not actually stored in the viewable filesystem. However, a user does not need any special tools to create or use extended file attributes. New methods (commands, options, etc.) are required for administrators to detect the use and existence of extended file attributes.

Introduction to Solaris 9 OE Extended File Attributes

Starting in the Solaris OS version 9, the UFS, NFS, and TMPFS filesystems were enhanced to include extended file attributes, enabling application developers to associate specific attributes with a file. For example, a developer of a file management application for a windowing system might choose to associate a display icon with a file. In the past, this has been done using application logic that bound a particular file type or name to a specific icon.

Using Solaris 9 OE extended file attributes, a developer can more readily do this by binding the icon directly to a file, thereby providing the ability to simplify the application's logic. The extended attributes assigned to a file are arbitrary in nature and take the form of regular files that are stored within a hidden directory associated with a given file. This is referred to as the file's extended attribute namespace. By default, no files in the Solaris 9 OE have extended attributes. Note that while different in implementation, in concept this capability is similar to Microsoft NTFS Alternate Data Streams or the older Apple MacOS Resource Forks.

Using Extended File Attributes

Extended file attributes can be created using either a set of shell commands or a C API. For the purposes of this discussion, we will focus on the shell commands. For those interested in the C API, refer to the attropen(3C), fchownat(2), fsattr(5), fstatat(2), openat(2), renameat(2), and unlinkat(2) manual pages.

To manage extended file attributes for any given file, use the runat(1) command. Using this command, you can perform various operations to create, display, read, modify, or delete objects within a file's extended attribute namespace. The following sections describe some typical scenarios highlighting the creation, display, and removal of extended file attributes.

Create an Extended File Attribute

To create an extended file attribute for the sample.conf file, located in the current directory, use the following command sequence:

```
$ runat ./sample.conf cp /etc/motd ./motd
```

In this example, the content of the /etc/motd file is copied into a file called motd stored within the hidden extended file attribute directory that is associated with the file sample.conf. This same technique can be used to modify an extended file attribute by overwriting an existing attribute file with a new one containing the updated content. Note that you must be able to write to a filesystem object in order to be able to create an extended file attribute for it.

Display an Extended File Attribute

To determine if a particular file, in this case sample.conf, has extended file attributes, you can use the following command sequence:

```
$ runat ./sample.conf ls –l
total 2
-rw-r—r—    1 gbrunett staff          49 Aug 25 14:16 motd
```

In this example, the file motd, created in the step above, was displayed. No other extended attributes were found. The contents of file motd can be read using the cat(1) command, as in the following example:

```
$ runat ./sample.conf cat motd
Sun Microsystems Inc. SunOS 5.9     Generic May 2002
```

Delete an Extended File Attribute

If an extended file attribute is no longer needed, it can be disassociated from its parent file and removed from the hidden extended attribute directory. For example, to remove the motd attribute file that is associated with sample.conf, use the following command sequence:

```
$ runat ./sample.conf rm motd
```

To verify that the object has been removed, list the file's extended attributes using the method described above:

```
$ runat ./sample.conf ls –l
total 0
$
```

Limitations of Extended File Attributes

The extended file attribute functionality that exists in the Solaris 9 OE only supports a single, flat directory structure. It is not possible to create subdirectories within the extended attribute namespace. Attempts to create directories using the mkdir(1) command will fail, as in the example below:

```
$ runat ./sample.conf mkdir test
mkdir: Failed to make directory "test"; Invalid argument
```

Similarly, the creation of either symbolic or hard links is prohibited. Attempts to create links within the extended attribute namespace result in error messages similar to the following:

```
$ runat ./sample.conf ln -s ../test2 .
ln: cannot create ./test2: Invalid argument
$ runat ./sample.conf ln -s `pwd`/test2 .
ln: cannot create ./test2: Invalid argument
```

```
$ runat ./sample.conf ln `pwd`/test2 .
ln: cannot create link ./test2: Invalid argument
```

Lastly, any command executed using the runat command that relies on it knowing its current working directory is also likely to fail, as is shown in the following example:

```
$ runat ./sample.conf man ls
getcwd: Not a directory
```

Accessing the Extended File Attribute namespace

In addition to running specified commands, the runat command can provide a user shell within the extended file attribute namespace. This provides the user with an interface for manipulating extended file attributes without having to repeatedly execute runat commands. To enter a file's extended attribute namespace, simply execute the runat command with only a file argument, as in the following example:

```
$ runat ./sample.conf
```

This causes a new user shell to be spawned within the extended file attribute namespace. From here, attribute creation, modification, and removal operations can proceed without having to prefix each command with runat <filename>. For example:

```
$ runat ./sample.conf
$ pwd
cannot access parent directories
$ cp /etc/motd .
$ ls -l
total 2
-rw-r—r—    1 gbrunett staff         49 Aug 25 16:54 motd
$ exit
```

To exit the user shell, simply type exit. To select a different shell, you can specify the shell name as the command to be executed, as in the following example:

```
$ runat ./sample.conf /bin/csh
```

Security Implications of Extended File Attributes

While the original intent behind the development of extended file attributes was good, they offer a significant opportunities for misuse. Many of the security implications of extended file attributes stem from three primary concerns:

- Extended file attributes cannot be disabled on the system.
- Extended file attributes are not readily visible to administrators.
- Commands may be executed in the extended file attribute namespace.

Each of these points will be addressed in more detail in the following sections. It is important to understand these problems more completely so that you can develop an appropriate policy on the use of extended file attributes in your environment.

Extended File Attributes Cannot Be Disabled on the System

It is an often-recommended administration practice to disable any service or feature that you do not need. Extended file attributes, however, are integrated with the Solaris OS and cannot be disabled. This presents a problem for an organization wishing to control access to this functionality.

In lieu of preventing the use of this functionality, an administrator is forced to be on the defensive and attempt to detect the creation, modification, or removal of extended file attributes.

Extended File Attributes Are Not Readily Visible To Administrators

Since an administrator cannot configure the Solaris 9 OE to prohibit the use of extended file attributes, methods for detection must be addressed. Extended file attributes provide a great opportunity for those wishing to conceal information or data. In particular, extended file attributes could be used to hide hacking tools or root kits, circumvent site security policies by concealing illegal or illicit material, or even used as an additional file repository (for filesystems that do not enforce quotas).

Caution: As of the publication of this paper, filesystem integrity tools such as Tripwire and AIDE do not check for the existence of or changes to extended filesystem attributes. As a result, it is possible that changes made to systems in this manner could go undetected.

Solaris OS commands such as find *will typically not return results for those potential matches that occur as extended file attributes. There is no mechanism for including extended file attributes in the results returned except through the mechanisms described below.*

DETECTION USING THE ls(1) COMMAND

The first method for detecting the use of extended file attributes is the new -@ option to the ls command. Without this option, the ls command is not able to detect or show the use of extended file attributes:

```
$ ls -@
total 2
-rw-r—r—@  1 gbrunett staff          0 Aug 25 14:16 test
-rw-r—r—    2 gbrunett staff          0 Aug 25 14:28 test2
```

In the above example, the ls *command was able to detect the presence of extended file attributes associated with the file test. Note that no extended file attributes were found for the file test2. The presence of attributes is indicated by the display of the @ symbol following the object's permissions.*

Note: Do not use the -@ option in combination with the -l option, otherwise extended file attribute information will not be displayed. Also, when the -@ option is used, access control list information associated with the object will not be displayed.

This process can be further automated across a filesystem or group of filesystems using the find *command using the* –xattr *option:*

```
$ find / -xattr –exec –ls
12891479    0 -rw-r—r—   1 gbrunett staff      0 Sep 2 12:00 /tmp/test
```

Note: Commands that check for the size of filesystem objects will continue to report only the objects' actual size and not that of the extended attributes. As a result, objects such as /tmp/test in the above example will show a size of 0 even though the size of its extended attributes could be considerable.

DETECTION USING THE runat(1) COMMAND.

Another method for detecting the presence of extended file attributes is to use the runat command itself. As noted above, extended file attributes can be listed by using the ls command in conjunction with the runat command, as in the following example:

```
$ runat ./sample.conf ls –al
total 4
drwxr-xr-x   2 gbrunett staff      512 Aug 25 15:11 .
-rw-r—r—     1 gbrunett staff        0 Aug 25 14:16 ..
-rw-r—r—     1 gbrunett staff        0 Aug 25 15:11 .abc
-rw-r—r—     1 gbrunett staff       49 Aug 25 15:08 motd
```

In this example, two extended file attributes were found, motd and .abc. It is recommended that you use the -a option to the ls command in order to find any extended attributes that take the form of "dot files."

This process can be further automated across a filesystem or group of filesystems using the find *command in conjunction with the* runat *command, as in the following example:*

```
$ find . -xattr -print -exec runat {} ls -al \;
/tmp/test
total 16
-rw-r—r—     1 gbrunett staff         49 Sep 2 12:00 test2
```

DETECTION USING SOLARIS AUDITING

The Solaris Auditing subsystem, also known as the Solaris Basic Security Module ("BSM") is a fine-grained kernel auditing facility. As such, it is able to audit the use of those system calls involved in the creation, modification, or destruction of extended file attributes.

The specific audit events that are relevant to extended file attributes are shown in the following table:

Audit Event	System Call
AUE_FCHOWNAT	fchownat(2)
AUE_FSTATAT	fstatat(2)
AUE_OPENAT_*	openat(2)
AUE_RENAMEAT	renameat(2)
AUE_UNLINKAT	unlinkat(2)

The Solaris Auditing subsystem must be enabled on the system, using the bsmconv(1M) command, and configured to log these particular events. For more information on configuring and using the Solaris Auditing facility, see the Solaris 9 OE product documentation as well as the Sun BluePrints™ article titled "Auditing in the Solaris 8 Operating Environment." While this paper was originally written about the Solaris 8 OE, all of the concepts and commands remain relevant for the Solaris 9 OE.

Commands May Be Excecuted In the Extended File Attribute Namespace

Another concern with the implementation and use of extended file attributes is the ability to execute commands from within the extended attribute namespace. This can be a significant issue when attempting to find commands that may be running on the

REFERENCES

Publications

What's New in the Solaris 9 Operating Environment?
http://docs.sun.com/db/doc/806-5202/6je7shk4h?a=view

Solaris 9 Operating Environment Product Documentation
http://docs.sun.com/prod/solaris.9

Auditing in the Solaris 8 Operating Environment
http://www.sun.com/solutions/blueprints/0201/audit_config.pdf

Linux Data Hiding and Recovery
http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html

Tools

AIDE
http://www.cs.tut.fi/~rammer/aide.html

bmap
ftp://ftp.scyld.com/pub/forensic_computing/bmap/

runat(1) Manual Page
http://docs.sun.com/db/doc/816-0210/6m6nb7mjs?a=view

Solaris Loadable Kernel Modules (SLKM)
http://packetstormsecurity.nl/groups/thc/slkm-1.0.html

Tripwire
http://www.tripwire.com/

system. For example, let's consider a scenario where we attempt to find information on a running process called nap.

For our scenario, we will first create an extended attribute for the file sample.conf by copying the sleep(1) program and renaming it to nap:

```
$ runat sample.conf cp /usr/bin/sleep ./nap
$ runat sample.conf ls –l
total 10
-r-xr-xr-x       1 gbrunett staff         4856 Aug 25 15:22 nap
```

Next, we will execute the nap program from within the sample.conf file's extended attribute namespace using the following command:

```
$ runat sample.conf ./nap 30000 &
[1957]
```

We can verify that the nap program is running by using the ps(1) command:

```
$ ps -aef | grep nap | grep -v grep
gbrunett  1958     1957     0 15:23:39 pts/17   0:00 ./nap 30000
gbrunett  1957     1633     0 15:23:39 pts/17   0:00 /bin/sh -c ./nap 30000
```

Remember, as noted above, you cannot get the current working directory of extended file attributes. As a result, the pwdx(1) command fails:

```
$ pwdx 1958
pwdx: cannot resolve cwd for 1958: Not a directory
```

A different response is returned when a program is launched from within a directory that is later removed. In this case, the result of the pwdx command is:

```
pwdx: cannot resolve cwd for 8248: No such file or directory
```

Using this distinction, you may be able to determine whether a program was executed from within an extended attribute namespace.

Summary

Solaris 9 OE extended file attributes are not a cause for immediate alarm and panic, but their existence and use must be clearly understood. As with any new capability, there is an opportunity for someone to misuse it to gain some kind of advantage. By understanding how extended file attributes are created, managed, and detected, you will be able to better defend your systems from attack as well as detect forms of misuse or abuse of this capability.