

## Automating Solaris 10 File Integrity Checks

by: Glenn M. Brunette, Jr., 10/22/2004

<http://www.securitydocs.com/library/2649>

Well, after a short hiatus, I am back with what I hope is a very interesting and useful tip for our Solaris 10 user community. Today's tip talks about how you can automate the collection of file integrity information using Solaris Secure Shell, Role-based Access Control, Process Privileges and the new Solaris 10 Basic Auditing and Reporting Tool. This sounds like quite a bit of work, but I can assure that it is simple. Allow me to demonstrate...

New to the [Solaris 10 OS](#) is a feature called the [Basic Auditing and Reporting Tool](#) or just "BART" for short. BART is intended to provide you with a quick and easy way to collect information on filesystem objects and their attributes so that at a later point in time you can determine if there have been any changes.

BART has two primary modes of operation: *create* and *compare*. When run in *create* mode, BART will collect filesystem object information from a system. You can just collect everything on the system, everything under a specified root directory, just a subset of files or you can specify a more granular policy in what is called a [rules](#) file) that can be customized to meet your organization's requirements. What's more is that you can specify the rules to be read from a file or from standard input. The output of a BART run is called a [manifest](#) and it is directed to standard output (at which you can redirect it to a file, another program, etc.) The ability to read rules from standard input and produce a *manifest* on standard output are important points as they factor heavily in the tip that I describe below.

To use BART in *compare* mode, you need two BART manifests and optionally a *rules* file that will be used to determine how the comparison is made. The first manifest, called the *control*, is used as your baseline. The second manifest, called the *test*, is then compared against the *control* (in accordance with a set of rules if supplied). In many cases, it is likely that you will use a *rules* file to help eliminate any noise from your reports so that you can better focus your efforts.

So now that I have given you some background on the [Basic Auditing and Reporting Tool](#), let's dive into the specifics of this tip. For customers with both large and small Solaris deployments, there is a growing need to manage cost and complexity. The goal of this tip is to highlight how the collection of filesystem information using BART can be securely automated across any number of systems (with any number of zones). Through the use of a centralized collection authority, we will be able to collect BART manifests across a network of Solaris 10 systems using strong authentication, least privilege and encryption over the wire. This allows you to then store and compare these manifests on a well-protected system (or zone) to which access can be significantly limited.

I will now describe the steps that you will need to take to implement this tip. Brief descriptions or guidance will be provided as appropriate. As a matter of convention, I will refer to the two systems in this example as *client* and *manager*. The *client* system is the one being examined by BART. The *manager* is where all of the BART rules and manifests will be stored and from where all connections to *client* will be made.

The first thing that we will do is create a new user on *client* whose only purpose is to collect filesystem information and create BART manifests. While for this example, I am only focused on a single *client* system, this same type of approach could be applied for a network of systems.

```
# mkdir -p /export/home
# useradd -d /export/home/bartadm -m -s /bin/pfsh bartadm
64 blocks
# passwd -N bartadm
passwd: password information changed for bartadm
```

Notice that I created the *bartadm* account as [non-login](#) account. This means that this account does not have a Unix login password, but is otherwise able to access the system (by using other authentication mechanisms or through the use of delayed execution mechanisms such as `cron(1M)`). This is necessary since the default behavior of `useradd(1)` is to create an account that is locked. You may also notice that this account was created with a profile shell, `/bin/pfsh`. This was done to allow commands executed by this user to be evaluated by the Solaris [Role-based Access Control](#) ("RBAC") facility to determine if the command will run with altered privileges.

Now that the account has been created, we will create a Secure Shell public key that will be used to access the account. Note that this does not need to be done on the system where you created the user. In fact, I would recommend that you generate the key on *manager*. This way, you will not need to transfer the private key over any network.

```
$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/export/home/bartadm/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /export/home/bartadm/.ssh/id_dsa.
Your public key has been saved in /export/home/bartadm/.ssh/id_dsa.pub.
The key fingerprint is:
42:ca:d7:fa:ab:1c:f8:c0:5b:2c:7b:56:28:85:dc:65 bartadm@manager
```

Once your key-pair has been created, you should copy the public key (*id\_dsa.pub*) from *manager* to *client*. As part of this copy, be sure to rename the file *id\_dsa.pub* to *authorized\_keys*. Once complete, you should have something like this:

[on manager]

```
# pwd
/export/home/bartadm/.ssh
# ls -l
total 6
-rw-----  1 bartadm  other          736 Sep 30 23:03 id_dsa
-rw-r--r--  1 bartadm  other          600 Sep 30 23:03 id_dsa.pub
```

[on client]

```
# pwd
/export/home/bartadm/.ssh
# ls -l
total 6
-rw-r--r--  1 bartadm  other          640 Oct  1 09:14 authorized_keys
```

Now, while on *client*, we will need to make one more change. We must configure Secure Shell to only

run a specific command when this public key is used. To do this, we use the *command* directive. For more information on this capability, see the "authorized\_keys File Format" section on [sshd\(1M\)](#). So, at this point, edit the *authorized\_keys* adding the following prefix to the existing public key:

```
command="/usr/bin/bart create -r -"
```

This will cause BART to be run in *create* mode taking a *rules* file from standard input. This will allow you to specify different BART *rules* files (as needed) without having to change the configuration of *client*.

The end result will look something like this (with a different public key):

```
command="/usr/bin/bart create -r -" ssh-dss AAAAB3NzaC1kc3MAAACBAJ6zG8SJtQVi/Et
OugyktNssLVofLmUepqsh712+D1AObTwRWZwjSH4hE423U3AcfY99u9ZxsdJ0sEppqnnvXmKaym7p
NxMCPoPcnf4mAicx9IQkpotAiCbCQ+My5IFD4iW4Nxjqh6KwIecEaABcpg2x5nhaX8Bsx0XURO/f+j
AAAFQCD6dOAM1JunvUeCWNpXoB6tLyLewAAAIAXya1UPijNFIjymsJ0gjQXyCgll8/tORHy2vrlo
gh9RJ9YNRWSZZjyRvLlKtd4KFIfcjT43W1VWJKa/A7114DGntoTS+dRh4MohJXdUjYmVv+OODc1
p+JWbbHlqDxa+zAuFEskoWNPmBrTnbLNzamIPnQ7ZaqWsbWuePQAAAIEAmqlCaMfuFYWlvDHc
xHJjRLqmvRwIPpTkW8XDuf8wn8lj/+glWWY6/VJVtbfgteZLweotdM2wvdfXNqROiU9vvlylOdv29iA
DxsSIPGSrjXkbnkNGQXMHTgPQmfbDhmtpnM6occl2R+J8dpDT59zWV7+egNZ0TTV8GNnmng=
gmb@manager
```

We are in the home stretch now. Next, we will create a RBAC rights profile on *client* that will allow the *bartadm* user to run BART with sufficient privileges to collect files across the filesystem. This is done to prevent the *bartadm* command from running as *root*. Remember that this account is configured as non-login and its Secure Shell public key is configured to only run the BART command. Even then, you will need possession of the *bartadm* private key and passphrase (which is stored on the heavily protected *manager* system). To do this, you will need to add the following lines to the [/etc/security/prof\\_attr](#) and [/etc/security/exec\\_attr](#) files:

```
# grep "^File Integrity:" /etc/security/prof_attr
File Integrity:::File Integrity Management:
# grep "^File Integrity:" /etc/security/exec_attr
File Integrity:solaris:cmd:::/usr/bin/bart:privs=file_dac_read,file_dac_search
```

Notice that the *File Integrity* rights profile grants the *file\_dac\_read* and *file\_dac\_search* privileges. These privileges are needed so that the *bartadm* user can search directories and read files that normally would not be permitted due to discretionary access controls (Unix permissions, ACLs, etc.) A description of these two privileges can be found using the [ppriv\(1\)](#) command:

```
# ppriv -l -v file_dac_read
file_dac_read
    Allows a process to read a file or directory whose permission
    bits or ACL do not allow the process read permission.
# ppriv -l -v file_dac_search
file_dac_search
    Allows a process to search a directory whose permission bits or
    ACL do not allow the process search permission.
```

Lastly, you need to grant the new *File Integrity* rights profile to the *bartadm* user. To complete this task,

use the following command:

```
# usermod -P "File Integrity" bartadm
```

This command will add the following line to the [/etc/user\\_attr](#) file:

```
# grep "^bartadm:" /etc/user_attr
bartadm:::type=normal;profiles=File Integrity
```

That was not too bad, right? There are certainly other things that you can do such as limiting access to the *bartadm* public key by hostname or IP address (for example only allowing access from *manager*), restricting *bartadm* access to cron(1M), etc. but at least you now have the basics that you need to get this working. So, let's now verify that everything works as expected from the *manager* system.

Before we begin, back on *manager*, I will create a small example BART *rules* file. This will be used as input to BART on the *client* passed over a Secure Shell channel that uses public-key authentication to execute a specific command. The output of BART will be displayed to standard output so we can redirect this to a file for later comparison. Here is the sample *rules* file:

```
/usr/sbin
CHECK all
```

This example will only collect information for files under */usr/sbin*. When used in comparison mode, all of the collected attributes will be checked. This is just a small and simple example to verify that the functionality works. Once this is achieved, you can then begin to develop more sophisticated policies based on your organization's needs. So, let's try it out (from *manager*)...

```
$ cat ./client.rules | ssh -T -l bartadm client
! Version 1.0
! Friday, October 01, 2004 (10:46:56)
# Format:
#fname D size mode acl dirmtime uid gid
#fname P size mode acl mtime uid gid
#fname S size mode acl mtime uid gid
#fname F size mode acl mtime uid gid contents
#fname L size mode acl lnmtime uid gid dest
#fname B size mode acl mtime uid gid devnode
#fname C size mode acl mtime uid gid devnode
/usr/sbin D 4608 40755 user::rwx,group::r-x,mask:r-x,other:r-x 415c6c1d 0 2
/usr/sbin/6to4relay F 9888 100555 user::r-x,group::r-x,mask:r-x,other:r-x 414f3ef2 0
2 5dbc53336307f5caf965e4451abde647
/usr/sbin/acctadm F 28356 100555 user::r-x,group::r-x,mask:r-x,other:r-x 414f3bb4 0
2 ece9d92d00b0c13ed2d56580e3856df7
/usr/sbin/add_drv F 44244 100555 user::r-x,group::r-x,mask:r-x,other:r-x 414f3cda 0
2 10f542c2c228c2a0efdc16bc543d96d6
/usr/sbin/allocate F 18764 104755 user::rwx,group::r-x,mask:r-x,other:r-x 414f3e96 0
2 2e98bb2d02c4e87b875885dfb3838932
/usr/sbin/arp F 9912 100555 user::r-x,group::r-x,mask:r-x,other:r-x 414f3ef2 0
2 203a43e71abc9c3b9ba2a1c38647b285
/usr/sbin/audit F 10140 100555 user::r-x,group::r-x,mask:r-x,other:r-x 414f3e85 0
2 26b6e6241c6a21aab5fc1bebb816f8fc
```

[... content edited for brevity...]

Looks great, so let's save a two copies to illustrate how to use the *compare* feature:

```
$ cat ./client.rules | ssh -T -l bartadm client > ./client.manifest.1
$ cat ./client.rules | ssh -T -l bartadm client > ./client.manifest.2
$ bart compare -r ./client.rules ./client.manifest.1 ./client.manifest.2
$
```

It should come as no surprise that we get no comparison errors. While this is not overly interesting, it is a very good thing since you know your files have not changed (relative to the baseline - *client.manifest.1*). As an example, here is a case where the comparison detected two problems:

```
$ bart compare -r ./client.rules ./client.manifest.1 ./client.manifest.2
/usr/sbin/auditd:
  acl  control:user::r-x,group::r-x,mask:r-x,other:r-x
       test:user::r-x,group::r-x,mask:r-x,other:rwx
  contents  control:28dd3a3af2fcc103f422993de5b162f3
            test:28893a3af2fcc103f422993de5b162f3
```

In this case, the [/usr/sbin/auditd](#) program was modified (contents change) and had its access control list modified - adding write access to "world". (Not a very good thing.)

Well, that's all for today. I hope you enjoyed this article and find its content useful in helping you and your organizations better leverage Solaris 10.