An Archived Oracle Technical Paper
March 2005

Automating Centralized File Integrity Checks in the Solaris 10 Operating System

Important note: this paper was originally published before the acquisition of Sun Microsystems by Oracle in 2010. The original paper is enclosed and distributed as-is. It refers to products that are no longer sold and references technologies that have since been re-named.

Automating Centralized File Integrity Checks in the Solaris™ 10 Operating System

Glenn Brunette, Client Solutions

Sun BluePrints™ OnLine—March 2005

A Sun BluePrints™ Cookbook

http://www.sun.com/blueprints

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300

Part No. 819-2259-10
Revision 1.0, 2/18/05
Edition: March 2005

Automating Centralized File Integrity Checks in the Solaris™ 10 Operating System

This Sun BluePrints™ Cookbook describes how to centralize and automate the collection of file integrity information using the following Solaris™ Operating System (Solaris OS) features:

- Secure Shell
- Role-based Access Control (RBAC)
- Process Privileges
- Basic Audit and Reporting Tool (BART)

Each of these features can be quickly and easily integrated to centralize and automate the process of collecting file fingerprints across a network of Solaris 10 systems.

Note – While Solaris Secure Shell and RBAC have been in the Solaris OS for some time, Process Privileges (discussed under the banner of Process Rights Management) and BART are new to the Solaris 10 OS.

About BART

This section provides and overview of the new Basic Audit and Reporting Tool (BART), including concepts you need to understand before proceeding with the steps to automate file integrity checking.

BART provides a quick and easy way to collect information on filesystem objects and their attributes so that, at a later time, you can determine whether there have been any changes. BART can help you detect accidental or malicious changes to files within an operating system due to either a security incident or change management incident.

BART is able to collect such information as an object's UID, GID, permissions, access control lists, modification time, size, and type. In addition, for files, BART generates an MD5 fingerprint from the contents of the file. For a full list of the attributes that can be collected, see the *bart_rules(4)* manual page.

BART has two primary modes of operation: create and compare.

Create Mode

When run in *create mode*, BART collects filesystem object information from a system. You can control the scope of collection on a system, including the entire system, under a specified root directory, or just a subset of files. You can even define a more granular policy using a rules file that can be customized to meet your organization's requirements.

When you use BART in create mode, it can read its rules file from either standard input or from a regular file—for a listing of file types supported by BART, see `bart_manifest(4)`. As BART processes individual filesystem objects, it records its results in a *manifest file*. This manifest is directed to standard output by default, although you can easily redirect the output to a file or to another process. BART's ability to read rules from standard input and produce a manifest on standard output are important for the automation of file integrity checking.

Compare Mode

To use BART in *compare mode*, you need two BART manifests and, optionally, a rules file.

- The first (and original) manifest, called the *control* manifest, is used as your baseline.
- The second manifest, called the *test* manifest, is then compared against the control (in accordance with a set of rules, if supplied).
- If a rules file is specified, then BART will use the rules it contains to determine how to make the various comparisons. One of the benefits of a rules file is that you can use it to define rules to help eliminate any false alarms in your reports, thereby allowing you to better focus your efforts on the remaining alarms.

Why Automate BART?

For customers with both large and small Solaris deployments, there is a growing need to manage cost and complexity. The goal of this BluePrints Cookbook is to highlight how the collection of filesystem information using BART can be securely automated across any number of systems (with any number of Solaris Containers).

BART automation has several benefits:

- Through the use of a centralized collection authority, you can collect BART manifests across a network of Solaris 10 systems using strong authentication, least privilege, and encryption over the wire.
- The rules and manifest files never need to be stored on the system (or Container) being evaluated—they can all be managed and protected on a central authority. Similarly, the comparison process can be performed in relative isolation because the comparison need not be done on the host being evaluated.

  This approach offers a significant security benefit over other file integrity methods in use today, where artifacts of the collection or comparison process must exist on the system being evaluated.

Steps to Automate File Integrity Checking

This section describes the steps to automate file integrity checking. As a matter of convention, these instructions 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 the system on which all of the BART rules and manifests are stored, and from which all connections to the *client* are made.

Step 1: Create a New User Account

The first step is to create a new user on *client* whose only purpose is to collect filesystem information and create BART manifests.

Note – The following example focuses on a single client system, but this same type of approach could be applied for a network of systems, for which this account could be created—either locally on each system, or in a networked naming service (such as LDAP).

To create a new user, enter the following commands.

```
# 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
```

In this example, note that:

- The *bartadm* account is created as a non-login account. This means that, while this account does not have a Unix login password, it is otherwise able to access the system, either by using other authentication mechanisms, or through the use of delayed execution mechanisms such as *cron(1M)*. This is required because the default behavior of *useradd(1)* is to create an account that is locked.
- 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 whether the command will run with altered privileges.

Step 2: Create a Secure Shell Key-Pair

After the new user account has been created on *client*, you next create a Secure Shell key-pair that will be used to access the account. Remember that, because *bartadm* is a non-login account, the only way to access it over the network is to use public key authentication with Secure Shell.

Note – This does not need to be done on the system where you created the user. In fact, we recommend that you generate the key on *manager* so that you will not need to transfer the private key over any network.

> **Warning –** This recommendation is based on a default Solaris 10 OS installation. If other authentication mechanisms are enabled by default, however, there might be other ways in which the *bartadm* user can be accessed across the network. We recommend that you verify your */etc/pam.conf* settings to be certain.

To create a Secure Shell key-pair, enter the following commands.

```
$ 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
```

Step 3: Copy the Secure Shell Key-Pair

After the new Secure Shell key-pair has been created, you copy it (*id_dsa.pub*) from *manager* to *client*. When copying, be sure to rename the file *id_dsa.pub* to *authorized_keys* if the file does not already exist. If the *authorized_keys* file exists on *client*, then simply append the contents of *id_dsa.pub* to the *authorized_keys* file. Once copied, you should have something similar to the following examples.

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        600 Oct  1 09:14 authorized_keys
```

Step 4: Configure Secure Shell

Next, on *client*, you must configure Secure Shell to run only a specific command when this public key is used. When this public key is used (which is, by default, the only remote access method), then the *bartadm* user will be able to run only the command that you specify. A remote user accessing the *bartadm* account will not be able to run any other commands. To do this, you use the Secure Shell *command* directive. For more information, see the "*authorized_keys* File Format" section of *sshd(1M)*.

To configure the Secure Shell to run the command, edit the *authorized_keys*, adding the following prefix to the existing public key:

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

Making this change causes BART to be run in create mode, taking a rules file from standard input, which allows you to specify different BART rules files (as needed) without having to change the configuration of *client*.

The result will look something like the following example (with a different public key):

```
command="/usr/bin/bart create -r -" ssh-dss AAAAB3NzaC1kc3MAAACBAJ6zG8SJtQVi/Et
OugyktNssLVofLmUepqsh712+D1AObTwRWZwjSH4hE423U3AcfY99u9ZxsdJ0sEpqnnvXmKaym7pMgk
NxMCPoPcnf4mAIcx9IQkpotAiCbCQ+My5lFD4iW4Nxjqh6KwIecEaABcpg2x5nhaX8Bsx0XURO/f+jA
AAAFQCD6dOAM1JunvUeCWNpXoB6tLyLewAAAIAXya1UPijNFIjymsJ0gjQXyCgll8/tORHy2vrloH7v
gh9RJ9YNRWSZZjyRvLlKTd4KFIfcjT43WlVWJKa/A7l14DGntoTS+dRh4MohJXdUjYMvV+OODc1j8V2
```

```
p+JWbbHlqDxa+zAuFEskoWNPmBrTnbLNzamIPnQ7ZaqWsbWuePQAAAIEAmqlCaMfuFYWlvDHeak79Fm
xHJjRLqmvRwlPPtkW8XDuF8wn8lj/+glWWY6/VJVtbfgteZLweotdM2wvdfXNqROiU9vvlylOdv29iA
DxsSlPGSrjXkbkNGQXMHTgPQmfbDhmtpnM6occl2R+J8dpDT59zWV7+egNZ0TTV8GNnmng=
gmb@manager
```

Step 5: Create an RBAC Rights Profile

Next, you will create an 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 important to prevent the *bartadm* command from running as the *root* account.

Note – Remember that, to successfully access this account, you will also need possession of the *bartadm* private key (which should be stored on the protected, centralized authority) as well as the passphrase to unlock the private key. Further, once you successfully access the account, you will be able to run only the *bart* command, as configured above, with the privileges that are described below. Each of these controls serves to reinforce the security of the overall solution.

To create an RBAC rights profile that will be associated with BART and assigned to the *bartadm* user, you need to add the following lines to the */etc/security/prof_attr* and */etc/security/exec_attr* files:

Note – When entering the following commands, be sure to omit the line breaks, which are included here for readability only.

```
# 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
```

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, and so on) as implemented in the Solaris operating system. A description of these two privileges can be found using the *ppriv(1)* command, as shown in the following example.

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

Step 6: Assign the Profile to the bartadm User

Finally, you need to assign the new *File Integrity* rights profile to the *bartadm* user.

To assign the rights profile, 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
```

Step 7: Optional Tasks

You have completed the basic steps to automate file integrity checking with BART. However, you can perform optional tasks to enhance security, including:

- 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)* by adding the "bartadm" account to the `/etc/cron.d/cron.deny` file

  There might be other security controls that you will want to evaluate and implement based on your individual security policies and requirements. Take care to identify and understand any residual risk in your environment and act accordingly.

Step 8: Verify the Setup

The final task is to verify that everything works as expected from the *manager* system.

Create a Sample Rules File

To verify the setup, you first create a small and simple example BART rules file on *manager* to verify that the functionality works. You will use this rules file as input to BART on *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 you can redirect this to a file for later comparison.

Create the following sample BART rules file on *manager*:

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

This example limits information collection to files under */usr/sbin*. When used in compare mode, all of the collected attributes are checked. Once your setup is verified, you can develop more sophisticated policies based on your organization's needs.

Run the Command to Verify

To verify the setup (from *manager*), enter the following command.

```
$ 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...]

Compare Manifest Files

After verification, save 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

$
```

You should get no comparison errors in this example, which indicates that your files have not changed relative to the baseline—*client.manifest.1*. In contrast, here is an example in which the comparison detected two differences:

```
$ 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 changed) and had its access control list modified—adding write access to `world`, which is certainly a bad thing!

Conclusion

In this BluePrints Cookbook, we have described a method for centralizing and automating file integrity checks across a network of Solaris 10 systems. This method uses strong authentication, least privilege, and encryption over the wire to provide a secure and scalable mechanism for the collection and transport of file fingerprints from clients to a centralized authority. While providing strong security, this solution is also flexible in that it allows an unlimited number of BART rules files to be used. Rules files can be developed per system, per application, per data center, or based on any other customer requirements.

In addition, the use of this mechanism does not require that the central authority itself be a system. It can be implemented within a Solaris Container in the Solaris 10 OS to further offer greater security isolation. While this does not improve the security of BART processing per se, it does offer greater protection for BART rules, manifests, and related user-developed scripts. By using a Solaris Container as a BART central authority, you can reap the security benefits that have been designed into them, including spare-root configurations (read-only, loopback-mounted filesystems), reduced process privilege sets, namespace isolation, resource management and global-zone observability, and so on.

For example, you could have a Solaris Container that has no listening services and that houses all of the rules and manifest files for an entire network of systems. No other services running on that same system (perhaps other security monitoring tools) could access the BART data. Further, by using Solaris Containers, you can monitor all of your BART rules and manifests from the isolated global zone (using BART, of course) to ensure that they have not been altered.

How you configure the BART management container is up to you, but one thing is certain—by leveraging the Solaris 10 OS and, more specifically, Solaris Containers, you will have the opportunity to build your BART central authority upon a very strong security foundation.

References and Related Sources

Publications

- Dasan, Vasanthan; Noordergraaf, Alex, and Ordorica, Lou. “The Solaris Fingerprint Database - A Security Tool for Solaris Operating Environment Files,” Sun BluePrints OnLine, May 2001,

  http://www.sun.com/solutions/blueprints/0501/Fingerprint.pdf

- Sun Microsystems, Inc. “Basic Audit and Reporting Tool,” Sun Solaris 10 OS Product Documentation,

  http://docs.sun.com/db/doc/816-4557/6maosrjds?a=view

- Sun Microsystems, Inc. “Roles, Rights Profiles and Privileges”, Sun Solaris 10 OS Product Documentation,

  http://docs.sun.com/app/docs/doc/816-4557/6maosrjfe?a=view

- Sun Microsystems, Inc. “Using Solaris Secure Shell”, Sun Solaris 10 OS Product Documentation,

  http://docs.sun.com/app/docs/doc/816-4557/6maosrjj8?a=view

- Sun Microsystems, Inc., “BART Manifest, Rules File and Reporting”, Sun Solaris 10 OS Product Documentation,

  http://docs.sun.com/app/docs/doc/816-4557/6maosrje9?a=view

Web Sites

- Glenn Brunette’s Solaris 10 OS Security Weblog:

  http://blogs.sun.com/gbrunett?catname=Solaris%2010%20Security

- Brunette, Glenn. “Managing Solaris 10 Non-Login and Locked Accounts”,

  http://blogs.sun.com/gbrunett/20040921#managing_non_login_and_locked

About the Author

Glenn Brunette is a Sun Distinguished Engineer with nearly 15 years experience in information security. Glenn works in the Client Solutions division as the Chief Security Architect for the Global Data Center Practice. In this role, Glenn is responsible for global security strategy as well as improving the quality and security of consulting solutions delivered to Sun's customers.

Glenn is the co-founder of the Solaris Security Toolkit software and a frequent author and contributor to the Sun BluePrints program. Glenn works closely with teams across Sun on the development of security strategy, products, services, methodologies, best practices, training, certifications and tools.

Externally, Glenn is currently the Vice-Chair of the Enterprise Grid Alliance Security Working Group and has served as Champion for the Common Configurations Working Group of the National Cyber Security Partnership's Technical Standards and Common Criteria Task Force. Glenn is also an active contributor to the Center for Internet Security's Unix Benchmark team.

Glenn is a Certified Information Systems Security Professional (CISSP) and has been trained in the National Security Agency's INFOSEC Assessment Methodology (IAM).

Ordering Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com/`

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html`

Automating Centralized File Integrity Checks in the Solaris 10 Operating System
March 2005

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment

Hardware and Software, Engineered to Work Together