Integrating BART and the Solaris™ Fingerprint Database in the Solaris 10 Operating System

Glenn Brunette, Client Solutions

Sun BluePrints™ OnLine—April 2005

A Sun BluePrints Cookbook

Please
Recycle

Adobe PostScript

Integrating BART and the Solaris™ Fingerprint Database in the Solaris 10 Operating System

In a previous Sun BluePrints™ Cookbook (*Automating Centralized File Integrity Checks in the Solaris™ 10 Operating System*), we discussed how to centralize and automate the collection of file integrity information using the Solaris™ 10 Operating System (Solaris 10 OS) Basic Audit and Reporting Tool (BART), Solaris 10 OS Process Privileges, Role-based Access-Control, and Secure Shell.

This Sun BluePrints Cookbook describes how to quickly and easily authenticate BART manifests using the Solaris Fingerprint Database (sfpDB). Using this process, you can determine whether there exist any files within the BART manifest that have been modified from the way in which they were shipped by Sun. This information is crucial when deciding how much trust can be placed in the validity of the files at the time the BART manifest was generated.

About BART and sfpDB

Before you begin the process of integrating BART and sfpDB, it's important to understand more about them—what they are, how they differ, and how they can be used to complement one another.

Basic Audit and Reporting Tool

BART is a tool that collects and compares a variety of attributes of filesystem objects installed on a system. For example, with BART, you can detect file ownership, permissions, and content changes. While this kind of functionality is clearly useful

for security incident detection, it is also often used as part of a larger change management process to validate approved changes and to detect those that may have occurred outside of an approved process.

Collecting Filesystem Object Attributes

BART collects filesystem object attributes, such as name, size, permissions, access control lists, UID, GID, and so on. The exact attributes collected depend on the type of object being evaluated. Each time that BART is run, it captures point in time information (*snapshot*) about the filesystem. For more information about the attributes collected, see bart(1M).

Comparing System Snapshots

You can also use BART to compare any two independent BART snapshots to determine whether there have been any changes made to the objects being assessed. For example, with BART, you can quickly and easily answer the question: "Has this file changed since yesterday?"

Using cron(1M), you can generate BART manifests every minute, hour, day, week or, month. How often you actually run BART to generate new manifests should be based on how critical your need is to detect change. Because one size does not fit all, you might even want to consider having different BART rules or policies. For example, you might have:

- a smaller policy, targeting a few key files, that runs every minute or hour, and
- a larger policy, collecting information across the entire system, that runs only once a day or week.

Use common sense when deciding how often you will generate manifests, balancing your detection priorities with the I/O load that will be generated on the system by the collection process.

Determining Whether an Object is Genuine

Regardless of the selected time intervals, however, BART is still not able to definitively answer the question: "Is this a genuine object that Sun shipped?" This is because you need to manually perform your data collection using BART *after* the system has been installed.

There is always the possibility (however slight) that someone or something could have changed a Sun-provided file from its default before you performed your first BART snapshot. As a result, it is important to be able to determine whether the files contained in your BART manifest are genuine. This is especially true for a *control*

manifest that will be used as the basic of comparison for future BART runs. Remember that, if your *control* manifest is somehow corrupt, then all of the later comparisons against it will be suspect.

Solaris Fingerprint Database (sfpDB)

The *Solaris Fingerprint Database* (*sfpDB*) is a tool that you can use to help solve this problem. The sfpDB is accessible via a free SunSolve Online[SM] Web service that enables users to verify the integrity of files distributed with the Solaris Operating Environment. You can read more about the Solaris Fingerprint Database in the following Sun BluePrint: *The Solaris Fingerprint Database—A Security Tool for Solaris Operating Environment Files*.

How the sfpDB Tool Works

The sfpDB tool:

- takes as input a list of MD5 file fingerprints
- evaluates each fingerprint against a master list developed and maintained internally by Sun
- returns information about any fingerprints found that match known files shipped by Sun, and flags any file fingerprints for which no match was found in the database

About MD5

Message Digest Algorithm Number 5 (*MD5*) is a secure, one-way hashing algorithm that transforms a data string of any length into a 128-bit "fingerprint" or "message digest." For more information on MD5, see RFC 1321 (ftp://ftp.rfc-editor.org/in-notes/rfc1321.txt).

How BART Complements the sfpDB

This is where BART comes into play, because BART collects the MD5 fingerprints of the files that it processes. You can extract these fingerprints from the generated BART manifest and, with some minor manipulation, prepare them for submission to the sfpDB. You can submit fingerprints directly (using the Web site interface) or using the Solaris Fingerprint Database Companion tool (a freely available Perl script that automates the submission process).

Leveraging the sfpDB for the First BART Manifest

Leveraging the Solaris Fingerprint Database is very useful when you build your first BART manifest, also known as a *control* manifest. Because this could have been done at system installation—or perhaps days, weeks, or months later—it is important for you to determine whether you are creating a baseline from known good and genuine files. In addition, this same technique can be re-applied to help resolve any conflicts that might arise from the installation of patches or other software. That way, you will know whether a conflict was generated from a genuine, newly-installed Sun file, or from some other type of change.

Note – Using the sfpDB with BART is most useful when validating binaries, libraries, kernel modules, or other software that does not change from the way in which Sun provided it (other than when patches are applied). The validation techniques described in this Sun BluePrints Cookbook will not help when attempting to validate files that have been changed after the software was delivered by Sun. For example, configuration files will often not match a known Sun fingerprint because they were modified by users during or after system installation.

Steps to Integrate BART and sfpDB

This section describes how to integrate BART and the sfpDB in order to determine whether objects collected by BART are genuine Sun files. In the instructions that follow, you will use a very simple BART rules file to collect file fingerprints for items in the /usr/lib/nis directory. Although any directory or set of directories could have been used as an example, the /usr/lib/nis directory was chosen arbitrarily for its relatively small size.

Step 1: Create a File Manifest in BART

To instruct BART to create a manifest for the files in /usr/lib/nis, enter the following command:

```
# find /usr/lib/nis | bart create -I
! Version 1.0
! Saturday, March 12, 2005 (21:21:21)
# 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/lib/nis D 512 40755 user::rwx,group::r-x,mask:r-x,other:r-x
42322d0f 0 2
/usr/lib/nis/nisaddent F 63996 100555
user::r-x,group::r-x,mask:r-x,other:r-x 41f2faff 0 2
c5cc75004f529ec0cb1bcbc4758ae85e
/usr/lib/nis/nisauthconf F 10432 100555
user::r-x,group::r-x,mask:r-x,other:r-x 41f2faff 0 2
c0f39d7a88ac826ab5cc48b16ef11ae9
/usr/lib/nis/nisclient F 38782 100755
user::rwx,group::r-x,mask:r-x,other:r-x 41f19222 0 2
0dad0b8dbf58a779088ef3463c8f55ed
/usr/lib/nis/nisctl F 10144 100555
user::r-x,group::r-x,mask:r-x,other:r-x 41f2faff 0 2
991252265bb68197e81333955293e354
/usr/lib/nis/nisopaccess F 5545 100755
user::rwx,group::r-x,mask:r-x,other:r-x 41f19223 0 2
00f19fcd403283f0510efadeafac2e66
/usr/lib/nis/nisping F 10228 100555
user::r-x,group::r-x,mask:r-x,other:r-x 41f2faff 0 2
13444988b12cc91f64dfdee22660643c
/usr/lib/nis/nispopulate F 33943 100755
user::rwx,group::r-x,mask:r-x,other:r-x 41f19222 0 2
5af2053863e687464285d4892bcc4b11
/usr/lib/nis/nisserver F 38902 100755
user::rwx,group::r-x,mask:r-x,other:r-x 41f19222 0 2
9e9bbcfd2beef6ff0f7e640da1fcbf18
/usr/lib/nis/nissetup F 10927 100755
user::rwx,group::r-x,mask:r-x,other:r-x 41f19222 0 2
8bdc76f327b86edee2c1dcafef289cb6
/usr/lib/nis/nisshowcache F 9980 100555
user::r-x,group::r-x,mask:r-x,other:r-x 41f2faff 0 2
```

```
14129756758acc54e0bfba55c1f82771
/usr/lib/nis/nisstat F 11380 100555
user::r-x,group::r-x,mask:r-x,other:r-x 41f2faff 0 2
625e0fe64af8fbe7860ce9f88afeefa0
/usr/lib/nis/nisupdkeys F 18496 100555
user::r-x,group::r-x,mask:r-x,other:r-x 41f2faff 0 2
b68df0fefe594d5d86462db16d6eeee5
```

> **Note –** In this output, when the object type (column 2) is a file (type *F*), then the
> very last field is an MD5 fingerprint. For more information on the format of entries
> contained in manifest files, see `bart_manifest(4)`.

Step 2: Create a List of MD5 Fingerprints

After generating the file manifest in BART, then you simply construct a command to
create a list of MD5 fingerprints that can be passed to the sfpDB in a recognizable
format. The command basically needs to select any type *F* (file) records and capture
each corresponding fingerprint value.

To create a list of MD5 fingerprints, enter the following command:

```
# find /usr/lib/nis | bart create -I |\
  awk '$1 ~ /^\// && $2 == "F" { print $NF }'
c5cc75004f529ec0cb1bcbc4758ae85e
c0f39d7a88ac826ab5cc48b16ef11ae9
0dad0b8dbf58a779088ef3463c8f55ed
991252265bb68197e81333955293e354
00f19fcd403283f0510efadeafac2e66
13444988b12cc91f64dfdee22660643c
5af2053863e687464285d4892bcc4b11
9e9bbcfd2beef6ff0f7e640da1fcbf18
8bdc76f327b86edee2c1dcafef289cb6
14129756758acc54e0bfba55c1f82771
625e0fe64af8fbe7860ce9f88afeefa0
b68df0fefe594d5d86462db16d6eeee5
```

Step 3: Submit MD5 Fingerprints to the sfpDB

After creating a list of MD5 fingerprints, you then submit the list to the sfpDB for processing. The sfpDB accepts up to 256 individual fingerprints at a time. In our example, only 12 MD5 fingerprints were generated. However, if more than 256 had been generated (which is likely for a real BART run), then you can use the Solaris Fingerprint Database Companion tool to process them. The sfpDB Companion tool automatically breaks up the list of MD5 fingerprints into sets of 256, and processes each set in turn, until all of the fingerprints have been processed.

The following example shows what happens when MD5 fingerprints are submitted to the database. For the sake of simplicity, this example used the sfpDB web interface, which produced the following output.

```
*Results of Last Search*


c5cc75004f529ec0cb1bcbc4758ae85e - - 1 match(es)


        * canonical-path: /usr/lib/nis/nisaddent
        * package: SUNWnisu
        * version: 11.10.0,REV=2005.01.21.15.53
        * architecture: sparc
        * source: Solaris 10/SPARC


c0f39d7a88ac826ab5cc48b16ef11ae9 - - 1 match(es)


        * canonical-path: /usr/lib/nis/nisauthconf
        * package: SUNWnisu
        * version: 11.10.0,REV=2005.01.21.15.53
        * architecture: sparc
        * source: Solaris 10/SPARC


0dad0b8dbf58a779088ef3463c8f55ed - - 2 match(es)


        * canonical-path: /usr/lib/nis/nisclient
        * package: SUNWnisu
        * version: 11.10.0,REV=2005.01.21.16.34
        * architecture: i386
        * source: Solaris 10/x86
```

```
              * canonical-path: /usr/lib/nis/nisclient
              * package: SUNWnisu
              * version: 11.10.0,REV=2005.01.21.15.53
              * architecture: sparc
              * source: Solaris 10/SPARC


      991252265bb68197e81333955293e354 - - 1 match(es)


              * canonical-path: /usr/lib/nis/nisctl
              * package: SUNWnisu
              * version: 11.10.0,REV=2005.01.21.15.53
              * architecture: sparc
              * source: Solaris 10/SPARC


      00f19fcd403283f0510efadeafac2e66 - - 13 match(es)


              * canonical-path: /usr/lib/nis/nisopaccess
              * package: SUNWnisu
              * version: 11.10.0,REV=2005.01.21.16.34
              * architecture: i386
              * source: Solaris 10/x86


              * canonical-path: /usr/lib/nis/nisopaccess
              * package: SUNWnisu
              * version: 11.8.0,REV=2000.01.08.18.17
              * architecture: i386
              * source: Solaris 8/x86


              * canonical-path: /usr/lib/nis/nisopaccess
              * package: SUNWnisu
              * version: 11.8.0,REV=2000.10.28.19.07
              * architecture: i386
              * source: Trusted Solaris 8/x86


              * canonical-path: /usr/lib/nis/nisopaccess
              * package: SUNWnisu
```

```
* version: 11.8.0,REV=2003.04.03.19.26
* architecture: i386
* source: Trusted Solaris 8/x86


* canonical-path: /usr/lib/nis/nisopaccess
* package: SUNWnisu
* version: 11.8.0,REV=2003.11.11.20.36
* architecture: i386
* source: Trusted Solaris 8/x86


* canonical-path: /usr/lib/nis/nisopaccess
* package: SUNWnisu
* version: 11.9.0,REV=2002.11.04.02.51
* architecture: i386
* source: Solaris 9/x86


* canonical-path: /usr/lib/nis/nisopaccess
* package: SUNWnisu
* version: 11.10.0,REV=2005.01.21.15.53
* architecture: sparc
* source: Solaris 10/SPARC


* canonical-path: /usr/lib/nis/nisopaccess
* package: SUNWnisu
* version: 11.8.0,REV=2000.01.08.18.12
* architecture: sparc
* source: Solaris 8/SPARC


* canonical-path: /usr/lib/nis/nisopaccess
* package: SUNWnisu
* version: 11.8.0,REV=2000.10.30.04.48
* architecture: sparc
* source: Trusted Solaris 8/SPARC


* canonical-path: /usr/lib/nis/nisopaccess
* package: SUNWnisu
```

```
                    * version: 11.8.0,REV=2001.09.25.00.12

                    * architecture: sparc

                    * source: Trusted Solaris 8 4/01


                    * canonical-path: /usr/lib/nis/nisopaccess

                    * package: SUNWnisu

                    * version: 11.8.0,REV=2003.04.03.21.27

                    * architecture: sparc

                    * source: Trusted Solaris 8/SPARC


                    * canonical-path: /usr/lib/nis/nisopaccess

                    * package: SUNWnisu

                    * version: 11.8.0,REV=2003.11.11.23.55

                    * architecture: sparc

                    * source: Trusted Solaris 8/SPARC


                    * canonical-path: /usr/lib/nis/nisopaccess

                    * package: SUNWnisu

                    * version: 11.9.0,REV=2002.04.06.15.27

                    * architecture: sparc

                    * source: Solaris 9/SPARC
```

[The rest of the content was omitted for brevity.]

Reviewing the Output

In our example above, all of the fingerprints that were evaluated were found to match at least Solaris 10/SPARC. This is great news because that was, indeed, our test platform. If any of these files did not match Solaris 10/SPARC, then there might be cause for concern. Remember that the Solaris Fingerprint Database will not have signatures for files that were installed by a third party or open source application (even when those files were installed in operating system directories such as /usr/bin or /usr/lib). So, bear in mind that, while a failure to obtain a fingerprint match for a given file is not necessarily a sign that you have been hacked, it is certainly an indication that more investigation is needed to determine why no match was found for that file.

Note also that for some of the files above, additional fingerprint matches were found. Some of the files matched Solaris 9/SPARC, Solaris 9/x86, and even Trusted Solaris 8/SPARC. When this happens, it is often an indication that the file is a shell

script, configuration file, or text file that did not depend on the underlying operating system version or hardware. In our example output, files that matched multiple operating system version and/or hardware combinations were simply not modified for Solaris 10 and, consequently, share the same fingerprint with those other versions.

Automating with the sfpDB Companion

The following example shows how you can automate sfpDB validation using the sfpDB Companion. This example looks for files that do not match any of the records in the database.

```
# find /usr/lib/nis | bart create -I |\
awk '$1 ~ /^\// && $2 == "F" { print $NF }' > ./md5.list

# cat ./md5.list
c5cc75004f529ec0cb1bcbc4758ae85e
c0f39d7a88ac826ab5cc48b16ef11ae9
0dad0b8dbf58a779088ef3463c8f55ed
991252265bb68197e81333955293e354
00f19fcd403283f0510efadeafac2e66
13444988b12cc91f64dfdee22660643c
5af2053863e687464285d4892bcc4b11
9e9bbcfd2beef6ff0f7e640da1fcbf18
8bdc76f327b86edee2c1dcafef289cb6
14129756758acc54e0bfba55c1f82771
625e0fe64af8fbe7860ce9f88afeefa0
b68df0fefe594d5d86462db16d6eeee5

# sfpC.pl ./md5.list | grep -- "0 match"
#
```

In this example, all of the fingerprints that were evaluated were found to match at least one known entry in the Solaris Fingerprint Database. This result is very positive. If a fingerprint could not be found in the Solaris Fingerprint Database, then the sfpC program would have generated output similar to the following:

```
b68df0fefe594d5d86462db16d6ffff5 -  - 0 match(es)
```

Using the BSD MD5 Format Instead

In addition to the fingerprint, it is helpful to have the filename listed so that you do not need to manually correlate them later. To do this, you simply construct the MD5 fingerprints in a slight different manner using the BSD MD5 format (which is also readable by the sfpDB), as shown in the following example.

```
# find /usr/lib/nis | bart create -I |\
awk '$1 ~ /^\// && $2 == "F" { printf "MD5 (%s) = %s\n", $1, $NF; }' > md5.list

# cat ./md5.list
MD5 (/usr/lib/nis/nisaddent) = c5cc75004f529ec0cb1bcbc4758ae85e
MD5 (/usr/lib/nis/nisauthconf) = c0f39d7a88ac826ab5cc48b16ef11ae9
MD5 (/usr/lib/nis/nisclient) = 0dad0b8dbf58a779088ef3463c8f55ed
MD5 (/usr/lib/nis/nisctl) = 991252265bb68197e81333955293e354
MD5 (/usr/lib/nis/nisopaccess) = 00f19fcd403283f0510efadeafac2e66
MD5 (/usr/lib/nis/nisping) = 13444988b12cc91f64dfdee22660643c
MD5 (/usr/lib/nis/nispopulate) = 5af2053863e687464285d4892bcc4b11
MD5 (/usr/lib/nis/nisserver) = 9e9bbcfd2beef6ff0f7e640da1fcbf18
MD5 (/usr/lib/nis/nissetup) = 8bdc76f327b86edee2c1dcafef289cb6
MD5 (/usr/lib/nis/nisshowcache) = 14129756758acc54e0bfba55c1f82771
MD5 (/usr/lib/nis/nisstat) = 625e0fe64af8fbe7860ce9f88afeefa0
MD5 (/usr/lib/nis/nisupdkeys) = b68df0fefe594d5d86462db16d6eeee5

# sfpC.pl ./md5.list | grep "match(es)"
  c5cc75004f529ec0cb1bcbc4758ae85e - (/usr/lib/nis/nisaddent) - 1 match(es)
  c0f39d7a88ac826ab5cc48b16ef11ae9 - (/usr/lib/nis/nisauthconf) - 1 match(es)
  0dad0b8dbf58a779088ef3463c8f55ed - (/usr/lib/nis/nisclient) - 2 match(es)
  991252265bb68197e81333955293e354 - (/usr/lib/nis/nisctl) - 1 match(es)
  00f19fcd403283f0510efadeafac2e66 - (/usr/lib/nis/nisopaccess) - 13 match(es)
  13444988b12cc91f64dfdee22660643c - (/usr/lib/nis/nisping) - 1 match(es)
  5af2053863e687464285d4892bcc4b11 - (/usr/lib/nis/nispopulate) - 4 match(es)
  9e9bbcfd2beef6ff0f7e640da1fcbf18 - (/usr/lib/nis/nisserver) - 2 match(es)
  8bdc76f327b86edee2c1dcafef289cb6 - (/usr/lib/nis/nissetup) - 2 match(es)
  14129756758acc54e0bfba55c1f82771 - (/usr/lib/nis/nisshowcache) - 1 match(es)
  625e0fe64af8fbe7860ce9f88afeefa0 - (/usr/lib/nis/nisstat) - 1 match(es)
  b68df0fefe594d5d86462db16d6eeee5 - (/usr/lib/nis/nisupdkeys) - 1 match(es)
```

By using the BSD MD5 format, you can more easily determine the names of files for which a match may not have been found in the Solaris Fingerprint Database. Because all of the files in our example are valid Solaris 10/SPARC files, each of them has a fingerprint for which there is at least one match.

Caveats with BART and sfpDB

BART can tell you whether a file has changed between two sets of snapshots. The sfpDB can tell you whether a file was shipped by Sun. However, you should be aware of certain caveats when using these tools.

Upgrade and Downgrade Attacks using Sun Binaries

The sfpDB does *not* indicate whether the binary is appropriate for the system. Due to Solaris OS binary compatibility guarantees, it is possible for an attacker to replace a valid Solaris binary or library with another *from an older (or unpatched) version* of the operating system, such as a old binary that is vulnerable in some way. Therefore, you must be alert to file fingerprint entries that match *only* versions of the operating system that are not the same as the one installed. If a fingerprint matches a version of the operating system that is either newer or older than the one installed, it is possible that the binary has been replaced.

Consider an example in which you are running the Solaris 9 OS but you find that your `/bin/login` matches only a version that existed on the Solaris 2.5.1 OS. You might want to consider that program suspect because your `/bin/login` should at least match the Solaris 9 OS. In an attempt to avoid detection, an attacker could have "downgraded" your `/bin/login` to a vulnerable version that existed in the Solaris 2.5.1 OS (see Sun Alert 41987). Therefore, while the `/bin/login` file will appear to be genuine, you must still be aware that the file is out of place because it was not delivered with the version of the operating system that you are currently running.

Note – Remember that a given fingerprint can match multiple operating system versions. If this is the case, be certain that at least one of the entries returned matches the version of the operating system that you are running.

Limitations of the MD5 Fingerprint Approach

The sfpDB verifies the content of a file using the MD5 hash algorithm, which does *not* detect changes to certain file attributes, such as UID, GID, permissions, or access control lists. Because this information is collected by BART and used as a basis of comparison with future runs, it is important to be able to trust this information.

To address this issue, you can limit the window of exposure by creating your BART *control* manifest immediately following the installation of the operating system and any patches. You can also compare these file attributes against the Solaris package database or even the Sun-supplied software package and patch distributions. For example, prior to creating your BART control manifest, you might want to verify files using the pkgchk(1M) command, as shown in the following example:

```
# pkgchk -v -n -P /usr/lib/nis
/usr/lib/nis
/usr/lib/nis/nisaddent
/usr/lib/nis/nisauthconf
/usr/lib/nis/nisclient
/usr/lib/nis/nisctl
/usr/lib/nis/nisopaccess
/usr/lib/nis/nisping
/usr/lib/nis/nispopulate
/usr/lib/nis/nisserver
/usr/lib/nis/nissetup
/usr/lib/nis/nisshowcache
/usr/lib/nis/nisstat
/usr/lib/nis/nisupdkeys
```

This will verify attributes, such as user, group, file type, file size, and expected modification time. It will even calculate and compare a file checksum based on the sum(1) command. Note that this example uses the Solaris package database to determine which files should exist in that directory. If you want to take a closer look, you can use the following command to examine each file under a directory tree:

```
# find /usr/lib/nis -exec pkgchk -n -v -p {} \;
/usr/lib/nis
/usr/lib/nis/nisaddent
/usr/lib/nis/nisauthconf
/usr/lib/nis/nisclient
/usr/lib/nis/nisctl
/usr/lib/nis/nisopaccess
```

```
/usr/lib/nis/nisping
/usr/lib/nis/nispopulate
/usr/lib/nis/nisserver
/usr/lib/nis/nissetup
/usr/lib/nis/nisshowcache
/usr/lib/nis/nisstat
/usr/lib/nis/nisupdkeys
WARNING: no information associated with pathname </usr/lib/nis/nisstat2>
WARNING: no information associated with pathname </usr/lib/nis/nisstat2>
WARNING: no information associated with pathname </usr/lib/nis/nisstat3>
WARNING: no information associated with pathname </usr/lib/nis/nisstat3>
```

As you can see, our first example did not detect the existence of the nisstat2 and nisstat3 objects in /usr/lib/nis because it used only the package database to obtain a list of objects to check. Depending on your own needs, however, either approach might suffice. Note that this second method is much slower than the first one simply because the Solaris package database must be scanned anew each time pkgchk(1M) is run, whereas this is not the case in the first example.

Note – Remember to update your BART *control* manifest whenever you install new software or patches on the system. It is important that you are able to maintain a valid snapshot of the current state of the system for future comparison.

Valid But Incorrect Sun Binary

An attacker can conceivably replace one valid Sun binary with another. In this case, however, the name reported by the Solaris Fingerprint Database *canonical-path* field will differ from the name passed via the MD5 file fingerprint.

Consider the following example, in which the sfpDB is used on a program called */tmp/myprog*. The results of the database search show that this program is really the Solaris OS */usr/lib/nis/nispopulate* program that has simply been renamed.

```
5af2053863e687464285d4892bcc4b11 - (/tmp/myprog) - 4 match(es)

            * canonical-path: /usr/lib/nis/nispopulate
            * package: SUNWnisu
            * version: 11.10.0,REV=2005.01.21.16.34
            * architecture: i386
            * source: Solaris 10/x86
```

```
* canonical-path: /usr/lib/nis/nispopulate
* package: SUNWnisu
* version: 11.9.0,REV=2002.11.04.02.51
* architecture: i386
* source: Solaris 9/x86
```

[The rest of the content was omitted for brevity.]

Best Practices

To address these caveats, use the following best practices:

- When you do detect file conflicts using BART and check them out using the sfpDB, be sure to validate file ownership and permissions as well as review the canonical path, package, version, and other fields to ensure that they are appropriate for your system.
- Always review the fingerprints associated with any patches you apply.

Regardless, you should not just ignore these conflicts, as they could be an indication of a security incident—or at least a change control problem.

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. "Automating Centralized File Integrity Checks in the Solaris™ 10 Operating System," Sun BluePrints Online, March 2005

  http://www.sun.com/blueprints/0305/819-2259.pdf

- Sun Microsystems, Inc. "bart(1M) Manual Page", Sun Solaris 10 OS Product Documentation

  http://docs.sun.com/db/doc/816-5166/6mbb1kpuc?a=view

Web Sites

- Solaris Fingerprint Database

  http://sunsolve.sun.com/pub-cgi/fileFingerprints.pl

- Solaris Fingerprint Database SideKick and Companion Tools

  http://www.sun.com/blueprints/tools/

- Glenn Brunette's Solaris 10 OS Security Weblog

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

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 for improving the quality and security of consulting solutions delivered to Sun's customers.

Glenn is the co-founder of the Sun 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).

Acknowledgements

Ordering Sun Documents

The SunDocs^{SM} 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`