Intrusion Detection using Solaris' Basic Security Module

by David Endler

last updated Friday, July 14, 2000

Re

Su
the
ID
Lis
Se

Sc
Se
Su
Mi

Int
De
SA

Introduction

In our existing online world, intrusion detection has become a necessary expense. Not only does intrusion detection validate the effectiveness of border access controls (e.g., firewalls, screening routers, etc.), but it also helps combat the persistence of insider abuse and corporate espionage. For this reason, intrusion detection systems (IDSs) have become an essential component in creating any comprehensive network infrastructure.

Intrusion detection systems rely on network traffic and/or system audit data as their main input sources. It is evident that an IDS can be only as powerful as the detail of the audit information fueling it. For instance, a host-based IDS monitoring only the `syslog` audit trail will be much less capable, than say, one that also examines `/var/log/messages` and the `wtmp` logs.

One of the most information rich sources of audit information is the UNIX kernel. Kernel auditing is available in many flavors of UNIX, although some administrators shy away from enabling it. Three common reasons for hesitation are fears of cpu performance drain, not enough disk space, and not enough time available to comb through copious amounts of audit output. This article explains the value of kernel audit sources in host and hybrid-based intrusion detection systems. As an example, I demonstrate the built-in C2 kernel auditing features of the Solaris Basic Security Module, and present a few ideas for automating real-time misuse detection with it.

What is C2?

Most professionals in the Information Security world are well acquainted with the reality that until recently, UNIX was not designed with security as a priority. Over the years, the UNIX user base has shifted from "trusted" university and research environments to the commercial spectrum. In this shifting market, data security has taken on extreme importance.

In the mid-eighties, the NSA NCSC published the Rainbow Series, a set of criteria used for evaluating the security features of computer systems. Security levels are defined in the "Orange Book" and are ranked from A1 (most trustworthy) to D (least). Stock UNIX's were roughly C1 (discretionary security protection) and could be upgraded with few headaches to C2 (added auditing capability and increasing access control).

Operating system vendors at the time relied significantly more on government contracts than in today's dot.com market. For that reason, there was an effort to ensure that products were evaluated against the C2 criteria, so as not to preclude potentially lucrative government business. As a result, vendors tended to allocate as few resources as they could toward developing auditing capability while still giving the appearance of C2 compliance. Today, most of these original developers have moved on to other projects, and in many cases, to new companies.

This Isn't Your Mother's C2

Kernel auditing has gradually been integrated as a feature into various flavors of UNIX over the past decade. Kernel audit trails differ from other types of audit sources in that they are generated internally by the kernel and are often stored in some sort of binary format, consequently making them tedious to tamper with. Most other audit sources (e.g., `syslog`, `sulog`, etc.) are the outputs of specific applications and are stored in plain ASCII format, making them more susceptible to alteration by an intruder in order to hide malicious behavior.

Sun first distributed C2 compliant kernel auditing capabilities with a patch for SunOS 4.1.2 called C2Conv. This shell script enabled features that met all government C2 criteria, but was difficult to use. Under development at the time was Solaris (SunOS 5.x) which had a much better audit subsystem. Sun put out another patch for SunOS 4.1.3 that used this yet-to-be-released Solaris kernel auditing. When it was finally released, Solaris included the improved Basic Security Module (BSM) which had yet another C2 audit subsystem.

As a result, there are three Sun auditing variants that meet C2 criteria: 2 patches (SunOS 4.1.2 BSM and SunOS 4.1.3 BSM) and the built in kernel auditing under the Solaris BSM. The look and feel of the Solaris BSM has not changed since it first debuted almost 10 years ago.

The Solaris Basic Security Module

The Basic Security Module (BSM) has been integrated into all versions of Solaris, and contributes just two things to the OS: kernel auditing and a device allocation mechanism. These two features when properly used in conjunction with the standard built-in security of Solaris, meet C2 level criteria. What does this mean to you? Probably nothing. The point is that BSM's kernel auditing feature is a powerful standalone tool that can serve as the data source to a variety of host and hybrid-based IDSs.

Many system administrators are unaware of this powerful auditing capability in Solaris. Many others are turned off by the lack of helpful BSM documentation, manuals which have not changed much since they were first published almost 10 years ago. BSM includes some built-in utilities (e.g., `praudit`, `auditreduce`, etc.) for converting the raw binary audit trails into human-readable format, although Sun's Solaris Answerbook provides little guidance for monitoring intrusive behavior, except to say:

Y*ou can accomplish more sophisticated display and reporting by postprocessing the output from* praudit *(with* sed *or awk, for instance) or by writing programs that interpret and process the binary audit records.*

The following steps will help you enable and configure BSM for use as input to any number of homegrown or commercial IDSs.

Enabling and Configuring BSM

To enable the Basic Security Module, go into the `/etc/security` directory and run the script `bsmconv`.

```
# cd /etc/security
# ./bsmconv
This script is used to enable the Basic Security Module (BSM).
Shall we continue with the conversion now? [y/n] y
bsmconv: INFO: checking startup file.
bsmconv: INFO: move aside /etc/rc2.d/S92volmgt.
bsmconv: INFO: turning on audit module.
bsmconv: INFO: initializing device allocation files.

The Basic Security Module is ready.
If there were any errors, please fix them now.
Configure BSM by editing files located in /etc/security.
Reboot this system now to come up with BSM enabled.
#
```

Next, let's take a look at the file `/etc/security/audit_control`:

```
#ident  @(#)audit_control.txt  1.3     97/06/20 SMI
#
dir:/var/audit
flags:all
minfree:20
naflags:lo
```

This file determines how auditing will behave. The following flags are described:

- **dir** - specifies the directory to write the binary audit trails. You can put more than one dir directive in the file, which tells BSM where to start writing audit trails once the first directory fills up.
- **flags** - specifies the types of actions to audit. The complete list of possible "audit classes" includes:

FLAG	LONG NAME	SHORT DESCRIPTION
no	no_class	null value for turning off event preselection
fr	file_read	Read of data, open for reading, etc.
fw	file_write	Write of data, open for writing, etc.
fa	file_attr_acc	Access object attributes: stat, pathconf, etc.
fm	file_attr_mod	Change object attributes: chown, flock, etc.
fc	file_creation	Creation of object
fd	file_deletion	Deletion of object
cl	file_close	close(2) system call
pc	process	Process operations: fork, exec, exit, etc.
nt	network	Network events: bind, connect, accept, etc.
ip	ipc	System V IPC operations
na	non_attrib	non-attributable events
ad	administrative	administrative actions: mount, exportfs, etc.
lo	login_logout	Login and logout events
ap	application	Application auditing
io	ioctl	ioctl(2) system call
ex	exec	exec(2) system call
ot	other	Everything else
all	all	All flags set

- minfree - specifies the minimum total percentage of free space available in the audit directories. If the threshold falls below the number indicated (in this case 20%), then the script /etc/security/audit_warn is run which should be configured to either email the system administrator, automatically zip and archive existing logs, etc.
- naflags - specifies which audit classes are to be logged when the system events can not be attributed to a specific user.

For the purposes of our example, the audit_control flags line should be edited to read:

```
flags:ex,pc
```

The file `/etc/security/audit_user` can be used to add more granularity for certain users (by default, root login information is always monitored).

Now, reboot the system. The next time Solaris comes up, C2 auditing is enabled and your audit directory should look something like this:

```
# ls /var/audit
20000601074538.not_terminated.localhost
#
```

Each time you restart the system, or stop and start the audit daemon, a new audit file is created with the timestamp in its name. For instance, the above audit trail was created on June 01, 2000 at 07:45:38 local time. The "not_terminated" signifies either the audit daemon was interrupted before it could close the file gracefully (system crash), or it is the actual real-time audit trail currently being appended. To be sure, check in `/etc/security/audit_data` for the name of the audit trail currently in use by BSM.

To monitor the audit trail in real-time, try this:

1. Bring up two shell terminals in your favorite windowing system
2. In one of the shell windows, type (substitute the name in `/etc/security/audit_data` after `tail -0f`):

   ```
   # tail -0f /var/audit/20000601074538.not_terminated.localhost | praudit
   ```

3. In the other shell window, start typing commands such as:

```
#finger
#ls
#cd /etc
#perl -v
```

The other window will show something similar to:

```
file,Thu 01 Jun 2000 09:59:38 PM EDT, + 391003 msec,
header,111,2,execve(2),,Thu 01 Jun 2000 09:59:41 PM EDT, + 220000000 msec
path,/usr/bin/finger
attribute,100555,root,bin,26738688,74333,0
subject,root,root,other,root,other,648,281,0 0 localhost
return,success,0
header,61,2,exit(2),,Thu 01 Jun 2000 09:59:41 PM EDT, + 240000000 msec
subject,root,root,other,root,other,648,281,0 0 localhost
return,success,0
header,79,2,fork(2),,Thu 01 Jun 2000 09:59:57 PM EDT, + 860000000 msec
argument,0,0x289,child PID
subject,root,root,other,root,other,580,281,0 0 localhost
return,success,0
header,107,2,execve(2),,Thu 01 Jun 2000 09:59:57 PM EDT, + 860000000 msec
path,/usr/bin/ls
attribute,100555,root,bin,26738688,74378,0
subject,root,root,other,root,other,649,281,0 0 localhost
return,success,0
header,61,2,exit(2),,Thu 01 Jun 2000 09:59:57 PM EDT, + 880000000 msec
subject,root,root,other,root,other,649,281,0 0 localhost
return,success,0
```

```
header,100,2,chdir(2),,Thu 01 Jun 2000 10:00:07 PM EDT, + 470000000 msec
path,/etc
attribute,40755,root,sys,26738688,7425,0
subject,root,root,other,root,other,580,281,0 0 localhost
return,success,0
header,79,2,fork(2),,Thu 01 Jun 2000 10:00:23 PM EDT, + 0 msec
argument,0,0x28a,child PID
subject,root,root,other,root,other,580,281,0 0 localhost
return,success,0
header,109,2,execve(2),,Thu 01 Jun 2000 10:00:23 PM EDT, + 330000000 msec
path,/usr/bin/perl
attribute,100555,root,bin,26738688,137497,0
subject,root,root,other,root,other,650,281,0 0 localhost
return,success,0
header,61,2,exit(2),,Thu 01 Jun 2000 10:00:23 PM EDT, + 840000000 msec
subject,root,root,other,root,other,650,281,0 0 localhost
file,Thu 01 Jun 2000 10:00:41 PM EDT, + 30217 msec,
```

Anatomy of a BSM Audit Event

As you can see from the output above, the meanings of the records produced by BSM auditing are not entirely self-evident. Here is a brief run-down of how BSM audit records are generally structured.

Each user-level and kernel event record has at the very least the following tokens:

Header
Subject
Return

Each event will begin with a "header" which will be in the format:

header, record length in bytes, audit record version number, event description, event description modifier, time and date

The "subject" line is of most interest to us when monitoring for unauthorized privilege escalation. Each subject line is in the form of:

subject, user audit ID, effective user ID, effective group ID, real user ID, real group ID, process ID, session ID, and terminal ID consisting of a device and machine name

Notice the changes that occur in the audit subject lines when the user *endler* performs a buffer overflow attack using the Xsun vulnerability in Solaris x86. The local attack:

```
$ whoami
endler
$ ./a.out
Jumping to Address 0xefffcc08
# whoami
root
```

generates the following in the audit trail:

```
header,73,2,old setuid(2),,Thu 01 Jun  2000 01:59:31 AM EDT, + 860000000 msec
header,103,2,execve(2),,Thu 01 Jun  2000 01:59:41 AM EDT, + 540000000 msec
path,/tmp/a.out
attribute,100755,root,other,26738688,329,0
subject,endler,endler,other,endler,other,696,688,0 0 localhost
```

```
return,success,0
header,117,2,execve(2),,Thu 01 Jun  2000 01:59:41 AM EDT, + 550000000 msec
path,/usr/openwin/bin/Xsun
attribute,104775,root,bin,26738688,167114,0
subject,endler,root,other,endler,other,696,688,0 0 localhost
return,success,0
header,61,2,exit(2),,Thu 01 Jun  2000 01:59:41 AM EDT, + 600000000 msec
subject,endler,root,other,endler,other,696,688,0 0 localhost
return,success,0
header,79,2,fork(2),,Thu 01 Jun  2000 01:59:48 AM EDT, + 300000000 msec
argument,0,0x2b9,child PID
subject,endler,endler,other,endler,other,690,688,0 0 localhost
return,success,0
header,104,2,execve(2),,Thu 01 Jun  2000 01:59:48 AM EDT, + 300000000 msec
path,/tmp/ksh
attribute,100555,root,other,2,472011600,4294967295
subject,endler,root,other,endler,other,697,688,0 0 localhost
return,success,0
header,61,2,setpgrp(2),,Thu 01 Jun  2000 01:59:48 AM EDT, + 310000000 msec
subject,endler,root,other,endler,other,697,688,0 0 localhost
return,success,688
header,61,2,setpgrp(2),,Thu 01 Jun  2000 01:59:48 AM EDT, + 310000000 msec
subject,endler,root,other,endler,other,697,688,0 0 localhost
return,success,688
header,61,2,setpgrp(2),,Thu 01 Jun  2000 01:59:48 AM EDT, + 310000000 msec
subject,endler,root,other,endler,other,697,688,0 0 localhost
return,success,0
header,61,2,setpgrp(2),,Thu 01 Jun  2000 01:59:48 AM EDT, + 310000000 msec
subject,endler,root,other,endler,other,697,688,0 0 localhost
return,success,688
```

The detection of an attack in this sequence is subtle. The subject line in question (outlined in red) illustrates a buffer overrun attack because the effective user ID (third field) becomes root. The significance of this line, as compared to the subject lines outlined in blue, is that the program that was executed (in this case `/tmp/ksh`) is not a setuid program so there is no reason that the user *endler* should have root permissions while running it. We can determine the default permissions on the file `/tmp/ksh` by translating the attribute field (outlined in purple) to

```
-r-xr-xr-x  user:root   group:other
```

In contrast, the Xsun program *is* a root owned setuid program which can be extrapolated from the attribute token (outlined in green). The fields in the green attribute line describe the file attributes of the Xsun executable and the appropriate permissions block can also be translated to (notice the setuid bit):

```
-rwsrwxr-x      user:root   group:other
```

This means that a transition to root privilege as shown in the effective user ID field (blue lines) is normal when a user runs this program. Many root-owned setuid programs (e.g. `ps`, `quota`, etc.) will appear to show the same behavior as Xsun and must be factored into detection scripts so false positives do not occur.

A nice perl script that will automatically do all of this detection for you is St. Jude, by Tim Lawless. A similar set of perl scripts by the author can be found here that go along with the article **Detecting Illegal Root Transition in Solaris** in *Sys Admin: The Journal for UNIX Systems Administrators*, 7(8), p. 29-32, 34--35, August 1998.

Other Ideas for Detecting Abuse

Monitoring the effective user ID field in the BSM audit trail is only one method for detecting potential system abuse. Using the configuration listed above, you can also look for other types of warning signs that a machine may have been compromised or that an insider is misusing his privileges.

Certain system directories (e.g., */etc/security*, */var/log*, etc.) may offer signs of malicious intent when someone tries to access them. Even when users fail to access a directory, it will still show up in the audit trail. For instance, the following is a snippet from a user changing directories to */var/log*.

```
header,97,2,chdir(2),,Thu 01 Jun  2000 03:26:34 AM EDT, + 430000000 msec
path,/var/log
attribute,40755,root,root,26738688,2,0
subject,endler,endler,other,endler,other,754,752,0 0 localhost
return,success,0
```

The "path" token shown above can also be quite useful in monitoring executables. There have been many papers written on the behavior patterns of online miscreants in relation to the types of programs they are likely to execute. For instance, an attacker will most likely use the `finger` or `w` command to ensure no administrators are logged in. He will then probably scan the process table for any suspicious logging mechanisms (such as BSM) that may retain evidence of his actions. The possible scenarios are endless, although simple sequences of commands can be easily scanned for in the audit trail in real-time (see USTAT by Koral Ilgun).

Even single instances of execution may signal future attacks, such as a user trying to execute commands such as `ufsrestore` or `sudo`.

```
header,108,2,execve(2),,Thu 01 Jun  2000 03:26:34 AM EDT, + 450000000 msec
path, /usr/sbin/ufsrestore
attribute,100555,root,root,26738688,74360,0
subject,endler,endler,other,endler,other,754,752,0 0 localhost
return,success,0
```

Conclusion

UNIX kernel audit data is one of the most overlooked sources for intrusion detection systems. Understanding the audit subsystem on your UNIX machine should be one of the very first priorities of any security or system administrator. Simple host and hybrid-based IDSs can incorporate a kernel audit trail once an effective preprocessing method has been established (e.g., perl, c-shell, etc.).

In addition to Solaris, other UNIX flavors that support kernel auditing include HP-UX 10.x, AIX, and DG-UX 4.12T. Unfortunately, there is not much homogeneity to these audit trail formats. There is however a Basic Security Module patch for Linux that behaves nearly the same as its Solaris C2 counterpart. Until the Common Intrusion Detection Framework (CIDF) becomes more accepted among OS vendors and security professionals, there seems to be no easy fix in the near future for integrating all of these different audit sources into IDSs.

David Endler is a Senior Security Engineer for iDEFENSE, Inc., a company specializing in cyber-threat intelligence. He holds a B.S. and M.S. in Computer Science, and enjoys participating in research on intrusion detection systems that incorporate artificial intelligence.