



Interested in learning
more about security?

SANS Institute InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

Solaris C2 Auditing with BSM

This article is intended to introduce the C2 auditing capabilities of Solaris BSM. It will also discuss the process of setting up BSM for common auditing policies and the process of doing the analysis of the resulted audit trail. An example of setting up BSM to track users activity, is also discussed along with problems arises in other tools such as the normal process accounting.

Copyright SANS Institute
Author Retains Full Rights

AD

Build your business'
breach action plan.

START NOW

 **LifeLock**
BUSINESS SOLUTIONS

No one can prevent all identity theft. © 2016
LifeLock, Inc. All rights reserved. LifeLock
and the LockMan logo are registered
trademarks of LifeLock, Inc.

Solaris C2 Auditing with BSM

Introduction

Many **Solaris** system administrators are unaware of the auditing capabilities inside the **Solaris Basic Security Module (BSM)** which is said by **SUN Microsystems**, to conform to the requirement laid forth in the **Trusted Computer System Evaluation Criteria (TCSEC) C2**.

This article is intended to introduce the **C2 auditing** capabilities of **Solaris BSM**. It will also discuss the process of setting up **BSM** for common auditing policies and the process of doing the analysis of the resulted audit trail. An example of setting up BSM to track users activity, which was the reason I started deploying BSM is also discussed along with problems arises in other tools such as the normal **process accounting**.

What is C2 Auditing

C2 is a security rating originally defined in the Trusted Computer System Evaluation Criteria (TCSEC), published by the United States National Computer Security Center (NSCC) commonly referred to as the Orange Book.

TCSEC provides four level of security:

- A : Verified Protection
- B : Mandatory Protection
- C : Discretionary Protection
- D : Minimal Protection

The Complete Set of Ratings are D, C1, C2, B1, B2, B3, A1

Most commercial OSES are certified at C2 which generally regarded as the lowest acceptable level of security. A C2 system must be able to:

- provide system level audit trail
- audit the use of identification and authentication mechanisms
- audit file access (open, close, read, write, create) and program initiation
- audit file/object deletion
- audit administrative actions

What is BSM

As opposed to syslog which is an application based, **Basic Security Module (BSM)** is a kernel based auditing mechanism.

It does two things to the Solaris operating system :

- Kernel Auditing
- Device Allocation Mechanism

which enables Solaris to meet C2 level criteria.

This C2 auditing capability is already available in Sun operating system since SunOS 4.1.2 or Solaris 1.1.2 via a patch to the kernel and has been migrated to loadable kernel module when Sun moved to Solaris 2.X as Basic Security Module. It has been a standard part of Solaris since Solaris 2.3.

Installing BSM

Before we start with the installation, we need to verify that the following packages : SUNWcar, SUNWcsr, SUNWcsu, SUNWhea and SUNWman are installed. This can be verified by using “pkginfo” utility as shown below :

```
soltest# pkginfo | egrep 'SUNWcar|SUNWcsr|SUNWcsu|SUNWhea|SUNWman'
system    SUNWcar    Core Architecture, (Root)
system    SUNWcsr    Core Solaris, (Root)
system    SUNWcsu    Core Solaris, (Usr)
system    SUNWhea    SunOS Header Files
system    SUNWman    On-Line Manual Pages
soltest#
```

It is advised to have a separate partition for the logs since one of the biggest problems with BSM auditing is it can consume a huge amount of space in an active system in a short amount of time.

Enabling BSM

If the earlier mentioned packages are installed, to enable BSM, we just need to enable the kernel support and ensuring auditd daemon is started at boot time. The auditd is started automatically by **S99audit** script in **/etc/rc2.d** which first check the presence of file **/etc/security/audit_startup** to decide to start the auditd daemon. Auditd daemon is the one that will tell the kernel which file to write to and which will do the basic management to ensure that there is space to write the audit trail records.

To ease the process of enabling or disabling BSM, SUN provides us with two scripts, **/etc/security/bsmconv** which sets up the system to run the BSM after the next reboot and **/etc/security/bsmunconv** which will do the reverse process.

As root or user that is being given the Audit Control under RBAC (Solaris 8), we need to run **bsmconv**.

Since running **bsmconv** will modify **/etc/system** to load the BSM kernel module, it is better to backup your **/etc/system** first.

The **bsmconv** script will also move your **/etc/rc2.d/S92volmgt** to **/etc/security/audit/spool**.

```
soltest#cp -p /etc/system /etc/system.bak.30102001
soltest# cd /etc/security
soltest# ./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.
soltest#
```

Verify the changes made by bsmconv to your /etc/system.

```
soltest# diff system system.bak.30102001    ** the system.bak.30102001 is backup file
< set c2audit:audit_load = 1                we made 83,84d82 in the previous step.
< set c2audit:audit_load = 1
< set abort_enable = 0
soltest#
```

The **set abort_enable = 0** will disables **Stop-A** and if you wish to not disable **Stop-A** or if you have already done this by updating **/etc/default/kbd**, you can remove this line from **/etc/system**. Since **bsmconv** also sets up device allocation, **vold** daemon is also disabled. . If you don't want to use device allocation and want vold continue to run then move **/etc/security/audit/spool/S92volmgt** back to **/etc/rc2.d/**

```
soltest#reboot
```

After rebooted, we should verify that BSM kernel module is loaded and that the auditd daemon is running.

```
soltest# modinfo | grep c2audit
26 f5934000 c644 186 1 c2audit (C2 system call)
soltest# ps -eafd | grep auditd | grep -v grep
root 249 1 0 10:04:48 ? 0:00 /usr/sbin/auditd
soltest#
```

Now we are ready for C2 auditing process.

Configuring BSM for Auditing

All configuration files used by BSM are stored inside the **/etc/security** directory. Knowing them and their functions will helps us to better understand how BSM works and where to look when planning for auditing policies and doing troubleshooting.

The files are listed below and their roles and functions are explained later as you go through this documentation.

```
/etc/security/audit_event
/etc/security/audit_class
/etc/security/audit_control
/etc/security/audit_user
/etc/security/audit_warn
/etc/security/audit_startup
```

BSM does the auditing process based on event and each event is defined inside the file **/etc/security/audit_event** in the form of :

event number:event name:event description:event classes (comma separated)

example :

```
10:AUE_CHMOD:chmod(2):fm
```

Common sets of events, such as system calls associated with file read access, are grouped into a single class.

Classes are defined inside the file **/etc/security/audit_class** in the form of :

mask:name:description

example :

0x00000002:fw:file write

the middle portion, **name**, is referred to as **audit_flag** which we will be using in the configuration file, **/etc/security/audit_control** and **/etc/security/audit_user** to determine our auditing policies. We also use this flag when doing audit trail analysis.

In BSM, policies can be configured for **system wide** and also for **user specific** auditing.

For **system wide** configuration, BSM uses the file **/etc/security/audit_control**. This is actually the main configuration file for BSM which determine how auditing will behave.

Let us take a look at the following **/etc/security/audit_control** file which is configured for the minimum recommended configuration, that is to log all login and logout activities and also all administrative activities.

```
soltest# pwd
/etc/security
soltest# cat audit_control
#
# Copyright (c) 1988 by Sun Microsystems, Inc.
#
#ident @(#)audit_control.txt 1.3 97/06/20 SMI
#
dir:/var/audit/log1
dir:/var/audit/log2
flags:lo,ad,na
minfree:20
naflags:lo
soltest#
```

dir keyword tells where to write the binary audit trails to. The default directory is **/var/audit** but we can also use other directory. The best practice is to have these directories in a separate file system to avoid from disturbing other operations since the binary audit trails can grow very fast and fills up the space depending on our policies and activities in the system. The binary audit trail file name are stored in the format of :

yyyymmddhhmm.yyyyymmddhhmm.hostname

or

yyyymmddhhmm.not_terminated.<hostname>

for the currently active one.

The yyyymmddhhmm is actually the %Y%m%d%H%M%S format as defined in manual page of **date** command.

Example :

```
soltest# ls /var/audit
20011106082030.20011108035548.soltest
20011108035551.not_terminated.soltest
soltest#
```

In a very critical system, since it is a normal Unix File System (UFS), we can mirror the audit trail to a remote machine to avoid from being removed etc.

flags specifies what classes to audit. The list of classes, as mentioned earlier, can be found in the file `/etc/security/audit_class`. Each class is separated with comma. The above example, “**lo**” logs login and logout events, “**ad**” logs admin events such as filesystem mounts or creation of users and “**na**” tells the system to log all non-attributable events. These include Stop-A, which we cannot be sure was done by any particular user. We can tell the system to only log the success event by prefixing the flag with a “+” and “-” for a failed event. To log all events, we can simply put “all” flag but we need to be careful since logging all events will generate tons of records in our binary audit trails.

minfree specifies the minimum total percentage of free space available in the audit directories. If the threshold falls below the number, in our example is 20 percent, then the script `/etc/security/audit_warn` will send an alert to a mail alias called **audit_warn** in aliases file.

naflags specifies which audit classes are to be logged when the system events can not be attributed to a specific user.

For **user specific** configuration, BSM uses the file `/etc/security/audit_user`. The configuration for specific user in this file will overwrite or fine tune the previously defined system wide configuration in the file `/etc/security/audit_control`.

The syntax of the file is as follows :

username:always:never

Let us take a look at below example.

fared: all,^-fr

This means that we are auditing all classes for user fared by the “**all**” class, overwriting the “**only audit lo, ad and na**” defined in system wide configuration file. The ^ prefix tells the system to turn off the logging of failed attempt of class “**fr**” (**file read**) for user fared which may have been defined in the system wide configuration file.

Finally, we should check the configuration of kernel audit event to class mappings by issuing an **auditconfig** command with **-chkconf** as argument.

```
soltest# auditconfig -chkconf
soltest#
```

If the runtime class mask of a kernel audit event does not match the configured class mask, a mismatch will be reported.

Audit Trail Analysis

To convert the binary format audit trail into human readable ASCII format, Solaris provides us with two tools, the `/usr/sbin/auditreduce` and `/usr/sbin/praudit`.

The **auditreduce** command performs two functions :

- **Audit trail selection**
- **Audit trail management**

Since **auditreduce** output is still in binary format, we need to pipe the output of **auditreduce** to **praudit**. The only job of **praudit** is to convert this binary output of **auditreduce** into the human readable ASCII format. Experience tells us that understanding how to use options available in the **auditreduce** command help us a lot when doing audit trail analysis. Thus, I am suggesting that we go through the manual page of **auditreduce** and also to have a look at the content of the file **/etc/security/audit_event** and **/etc/security/audit_class**.

Now, let us try to search for a login type of record by user fared

```
soltest# auditreduce -c lo -u fared | praudit
header,81,2,login - telnet,,Sun Nov 04 12:44:34 2001, + 282713857 msec
subject,fared,fared,staff,fared,staff,2937,2937,24 6 serv1.jubaihah.com.my
text,successful login
return,success,0
header,81,2,login - telnet,,Mon Nov 05 15:10:22 2001, + 229431317 msec
subject,fared,fared,staff,fared,staff,3216,3216,24 6 serv1.jubaihah.com.my
text,invalid password
return,failure: Interrupted system call,-1
file,Mon Nov 05 15:10:22 2001, + 0 msec,
soltest#
```

As you can see from the output above, the records produced by BSM auditing is stored in its own format. Each audit trail record's event will normally have at least the following :

Header
Subject
Return

Each event starts with **header** as follows :

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

In the example above, we understand that there are two records where each of them indicates a login session via telnet.

Then follows with **subject** which explain the user's related information.

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

In the example above, the audit user id was fared. Both the real user id and effective user id are fared. Real user id, as the name spells, means the "real user id" and the effective user id is the user that we run a program as. When we run a setuid application, our effective id is the uid of the owner of the application file.

Each record will be ended with **return** which reports the status of the event.

return, success or failure with exit status and error description.

In the above example, we saw the record of both success and failed login event of telnet session.

Example : Keeping Track of All Users Activity

Previously, for some internal reason, I was told to keep track of all our local users activities including the system administrators activities inside few in-house production machines which they

are using for updating files, answering e-mail and many other activities including their personal files.

The suggestions that I got from the newsgroup and from some readings include to keep the users **.history** files to a secure place, restricting user's activities via **sudo** and use it's log for tracking their activities and also to use the **process accounting**.

Since users can easily change the **HISTFILE** value, the first suggestion is not practical. We then decide to utilize both **sudo** and the **process accounting**.

The **process accounting** logs all users commands and operate at kernel level as opposed to other **syslog** dependent application based logging mechanism like **wtmp** etc. But then, the problem with the **process accounting** is it does not log the arguments supplied by the users when invoking a command.

An example of an administrator, doing **su** to root and then use a **vi** editor to edit the file **/root/.rhosts**, the accounting records at one of our machine running on **BSDi OS** which does not have a C2 auditing capability as what we have with **Solaris's BSM**, will just show something like the following :

```
bsditest# ls -ld /root/.rhosts          ** /root/.rhosts was edited
-rw-r--r-- 1 root wheel 371 Nov 6 14:27 /root/.rhosts
bsditest#lastcomm root | grep vi       ** When did root used vi command
vi      -   root      ttyr2   0.00 secs Tue Nov 6 14:21
vi      -   root      ttyr5   0.00 secs Tue Nov 6 14:25
bsditest#lastcomm ttyr2 | grep login    ** who was root at that time ?
login_krb- -S   fared      ttyr2   0.00 secs Tue Nov 6 14:21
login_krb- -S   joe        ttyr5   0.00 secs Tue Nov 6 11:10
bsditest#grep 'su:' /var/log/messages  **who was root at that time ?
messages:Nov 6 14:21:22 bsditest su: fared to root on /dev/ttyr2
messages:Nov 6 11:10:49 bsditest su: joe to root on /dev/ttyr5
bsditest# last fared                   ** how long fared was inside bsditest ?
fared   ttyr2 fared.jubaihah.com.my Sun Nov 6 12:09 - 14:43 (02:34)bsditest
bsditest# last joe                     ** how long joe was inside bsditest ?
joe     ttyr5 joe.jubaihah.com.my   Sun Nov 6 14:01 - 20:12 (06:11)bsditest
bsditest#
```

In the above example, both users, fared and joe were connecting to the same machine, **bsditest**, and then did a **su** to root and use a **vi** editor. We are now, puzzled to who edited the **/root/.rhosts** file and add some ip address inside it. For this kind of reason, I found that logging the arguments supplied to each command is very important when tracking down who did what.

Sudo, a short name of **super user do**, did a very good job of not only as an access control mechanism for us but also because it logs whatever commands executed by users via **sudo** along with it's argument and other useful information such as the terminal the user is connected to.

An example of a user fared using **sudo** to edit the file **/root/.rhosts**, the log shows as follow :

```
bsditest#tail /var/log/sudo.log
Nov 6 14:48:44 bsditest.jubaihah.com.my sudo: fared : TTY=ttyt7 ;
PWD=/home2/fared ; USER=root ; COMMAND=/usr/bin/vi /root/.rhosts
bsditest#
```

What are the problem with **sudo** ? The problem that I faced is, if a user is also granted to use **sudo** to execute **su** command, we will then have to face the same problem described for **process accounting** above, as the user is already escaped from **sudo**'s log.

How BSM solved this issue for us ? First, in BSM we can turn on the **ex** flag inside the file **/etc/security/audit_control** by specifying :

flags:ex

or

flags:all

which will do the logging by intercepting each `execve(2)` call being made and this means that there is no chance for user to bypass from being logged by modifying user configurable variable for example the value of `LD_LIBRARY_PATH`.

In BSM, we can tell auditing to log the arguments together with the command by running :

auditconfig -setpolicy +argv

The `+` prefix adds the policies specified to the current audit policies.

To have this being set up at each reboot, we have to define the default policy in the file **/etc/security/audit_startup**. The **audit_startup script** is used to initialize the audit subsystem before the audit daemon is started.

Please note that the default is

auditconfig -setpolicy none

which does not log arguments.

Below is an example of an audit trail analysis with **auditreduce** and **praudit** that shows the activity of a user **fared** doing a `vi` on the file **/root/.rhosts** as user **root**. Please note that even the user **fared** has already `su` to root, the **audit user id** column at the second line of the row that starts with **subject** is still maintained as **fared**. This makes us easier to keep track of any activity involving each user inside our system because it never change.

The **effective user id** and **real user id** tell us that the `vi` command was executed by **root**. The arguments are clearly mentioned at the fourth and fifth line including the number of arguments involved.

```
soltest# auditreduce -u fared -c ex -d 20011106 | praudit
.
.
header,141,2,execve(2),,Tue Nov 06 16:21:47 2001, + 423023213 msec
path,/usr/bin/vi
attribute,100555,bin,bin,118,54125,0
exec_args,2,
vi,/root/.rhosts
subject,fared,root,other,root,other,3473,3464,24 3 serv1.jubaihah.com.my
return,success,0
.
.
soltest#
```

Common Problems and Solutions

Enabling C2 auditing, as expected, will incur some performance penalty which SUN suggest at around 5% to 10% as at each system call being made, an additional auditing code will be executed even for those system calls that are not being audited. This would be the main reason that many Solaris system administrator would rather chose not to use BSM. In the system where performance is the only thing expected and security auditing is not a really important aspect, I chose not to enable BSM.

Being able to retrieve as much information from BSM audit trail is the best thing about it but as previously mentioned, the tons of audit trail data will keep growing until someone stops the **auditd** daemon or issue the **audit -n** or if the file system is filled up. BSM does not provides any utility to handle this. As a result, we have to properly plan, how to rotate the audit trail binary. We can use a command called **audit** and give a **-n** flags which will signal the audit daemon to close the current audit file and open a new audit file in the current audit directory. The **audit -n** command is actually runned by BSM at each starts. We can then move the old audit trail binary to other archive medium such as a CDR (writable CD) or tape device. A script should be runned out of cron to automate this process.

The difficulties in selecting the record that we really want is also a challenge for especially the new users of BSM. I would really suggest that we go through the manual page of auditreduce and try to play around with the options to narrow down the result rather than using other system utility such as grep which will just create a burden.

To ease the process of searching the audit trail for analysis, a GUI interface called BSM Event Viewer, can be used to ease the searching process. It is written in **perl** by **Kevin Wenchel** and **Stephen Michaels** from **John Hopkins University Applied Physics Laboratory** and can be downloaded for free from Sys Admin Magazine website, <http://www.sysadminmag.com/code/>. Those who normally use GUI or perhaps the NT administrator may like this tool very much. Alternatively, for those who prefer java, they may download a java based GUI interface from **Jay Daniels** homepage at <http://home.twmi.rr.com/jayd/bsm.html>.

Another important issue to be addressed is the problem with the Secure Shell (SSH) which was not yet tailored to add BSM auditing capability inside. In my environment, this was a big problem since we have totally stopped using telnet or rlogin to tighten up security. A workaround to this is to configure our Open SSH daemon to use `/bin/login` which will then cause an audit record to get written, but it will appear as a telnet login not ssh. In my opinion, this is better than opening a hole for someone to evesdropping password or other important informations in the network. In order to enable this, we need to change the **UseLogin** parameter which default to **no** to **yes** inside the Open SSH (SSH2) configuration file, `/usr/local/etc/sshd_config` as follows.

UseLogin yes

The default configuration in ssh is :

UseLogin no

Conclusion

We have discussed about the process of implementating Basic Security Module (BSM) for doing C2 auditing in Solaris environment, the basic of the BSM audit trail analysis and also few problems arises when we start deploying BSM.

Besides C2 auditing, there are many other useful purposes that BSM auditing capability is being utilized due to the rich audit trail which was not discussed in this document, such as **reconstruction of event** and **host based intruder detection system (IDS)**. An example commercial product using BSM for IDS purpose is **eXpert-BSM**.

The usage of BSM auditing today is still very low for reason that I would say the complexity and lack of documentation on the subject as well as application utilizing it.

Besides Solaris, other operating system that provide a BSM liked C2 auditing capability are **HPUX** and **AIX**. There is also an ongoing project to provide such capability in **Linux** hosted by **University of California at Davis** which could stimulate the development of open source applications utilizing BSM audit trail. The web site is available at <http://linuxbsm.sourceforge.net>

References

Sun Microsystems Online Documentation

<http://docs.sun.com/ab2/coll.47.11/SHIELD/@Ab2TocView?Ab2Lang=C&Ab2Enc=iso-8859-1>

National Computer Security Center, Trusted Computer Evaluation Criteria.

<http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.pdf>

Intrusion Detection using Solaris' Basic Security Module

<http://www.securityfocus.com/cgi-bin/infocus.pl?id=1211>

Solaris BSM Auditing

<http://www.securityfocus.com/cgi-bin/infocus.pl?id=1362>

Practical UNIX & Internet Security, 2nd Edition from O'Reilly & Associates, Inc.

By Simson Garfinkel & Gene Spafford

Implementing C2 Auditing in the Solaris Environment

<http://www.samag.com/documents/s=1157/sam0013d/>

Auditing : The Ugly Duckling of Computers

<http://www.usenix.org/publications/login/1998-2/auditing.html>

An Approach to Unix Security Logging

<http://www.ce.chalmers.se/staff/sax/unix-sec-log.pdf>



Upcoming SANS Training

[Click Here for a full list of all Upcoming SANS Events by Location](#)

SANS Paris June 2018	Paris, FR	Jun 25, 2018 - Jun 30, 2018	Live Event
SANS Minneapolis 2018	Minneapolis, MNUS	Jun 25, 2018 - Jun 30, 2018	Live Event
SANS Vancouver 2018	Vancouver, BCCA	Jun 25, 2018 - Jun 30, 2018	Live Event
SANS London July 2018	London, GB	Jul 02, 2018 - Jul 07, 2018	Live Event
SANS Cyber Defence Singapore 2018	Singapore, SG	Jul 09, 2018 - Jul 14, 2018	Live Event
SANS Charlotte 2018	Charlotte, NCUS	Jul 09, 2018 - Jul 14, 2018	Live Event
SANSFIRE 2018	Washington, DCUS	Jul 14, 2018 - Jul 21, 2018	Live Event
SANS Cyber Defence Bangalore 2018	Bangalore, IN	Jul 16, 2018 - Jul 28, 2018	Live Event
SANS Pen Test Berlin 2018	Berlin, DE	Jul 23, 2018 - Jul 28, 2018	Live Event
SANS Riyadh July 2018	Riyadh, SA	Jul 28, 2018 - Aug 02, 2018	Live Event
Security Operations Summit & Training 2018	New Orleans, LAUS	Jul 30, 2018 - Aug 06, 2018	Live Event
SANS Pittsburgh 2018	Pittsburgh, PAUS	Jul 30, 2018 - Aug 04, 2018	Live Event
SANS San Antonio 2018	San Antonio, TXUS	Aug 06, 2018 - Aug 11, 2018	Live Event
SANS August Sydney 2018	Sydney, AU	Aug 06, 2018 - Aug 25, 2018	Live Event
SANS Boston Summer 2018	Boston, MAUS	Aug 06, 2018 - Aug 11, 2018	Live Event
Security Awareness Summit & Training 2018	Charleston, SCUS	Aug 06, 2018 - Aug 15, 2018	Live Event
SANS Hyderabad 2018	Hyderabad, IN	Aug 06, 2018 - Aug 11, 2018	Live Event
SANS New York City Summer 2018	New York City, NYUS	Aug 13, 2018 - Aug 18, 2018	Live Event
SANS Northern Virginia- Alexandria 2018	Alexandria, VAUS	Aug 13, 2018 - Aug 18, 2018	Live Event
SANS Krakow 2018	Krakow, PL	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Chicago 2018	Chicago, ILUS	Aug 20, 2018 - Aug 25, 2018	Live Event
Data Breach Summit & Training 2018	New York City, NYUS	Aug 20, 2018 - Aug 27, 2018	Live Event
SANS Prague 2018	Prague, CZ	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Virginia Beach 2018	Virginia Beach, VAUS	Aug 20, 2018 - Aug 31, 2018	Live Event
SANS San Francisco Summer 2018	San Francisco, CAUS	Aug 26, 2018 - Aug 31, 2018	Live Event
SANS Copenhagen August 2018	Copenhagen, DK	Aug 27, 2018 - Sep 01, 2018	Live Event
SANS SEC504 @ Bangalore 2018	Bangalore, IN	Aug 27, 2018 - Sep 01, 2018	Live Event
SANS Wellington 2018	Wellington, NZ	Sep 03, 2018 - Sep 08, 2018	Live Event
SANS Amsterdam September 2018	Amsterdam, NL	Sep 03, 2018 - Sep 08, 2018	Live Event
SANS Tokyo Autumn 2018	Tokyo, JP	Sep 03, 2018 - Sep 15, 2018	Live Event
SANS Tampa-Clearwater 2018	Tampa, FLUS	Sep 04, 2018 - Sep 09, 2018	Live Event
SANS MGT516 Beta One 2018	Arlington, VAUS	Sep 04, 2018 - Sep 08, 2018	Live Event
SANS Cyber Defence Canberra 2018	OnlineAU	Jun 25, 2018 - Jul 07, 2018	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced