Global Information Assurance Certification Paper

The FreeBSD Jail: When Chroot is not Enough

Name: Syahrul Sazli Shaharir
Assignment Version: 1.2f

Introduction

Offering a service to the Internet, such as the web service, has an associated risk: there exists the uncertainty of undiscovered security holes in its implementation. Even if they do get discovered and become publicly known, services often do not get fixed or patched in time, sometimes not until a few days, weeks, or not at all, depending on the constraints faced by the system developers and administrators. During that time, the vulnerable service is threatened by would-be attackers all over the Internet, who are equipped with freely downloadable tools used to look for such holes, exploit them, and cover their tracks or even go undetected. The impact can be serious. A successful exploit on a single vulnerable service usually leads to the compromise of confidentiality, integrity, and availability of the whole system, or other systems within reach. If these systems are secured behind perimeter firewalls, the impact could be devastating, since the attacker is now located within the trusted network, usually laden with sensitive proprietary data.

In order to mitigate such risks, "defense in depth" must be applied in setting up Internet services. In UNIX-based systems, the chroot[7] system call has been an indispensable tool used to limit, to some extent, what services can have access to in the Operating System. It acts as a "safety net": if a bug in the service is exploited and access to the Operating System is gained, the attacker's ability to make use of that access for breaching the rest of the system (or other systems) will be reduced.

To seek further improvements to chroot in confining Internet services, Poul-Henning Kamp[12] has developed the jail[8] kernel feature, which in addition to chroot, imposes more limitations to system services. Currently, it is only implemented in the FreeBSD Operating System.

The goal of this paper is to introduce the jail concept, comparing it with chroot and other alternatives, and walk through a simple HOWTO on setting up a jail partition. Finally, we look into the future of jail, and end with some concluding remarks.

What's Wrong with Chroot?

What chroot essentially does is it alters the current process's environment such that the root directory (/) points to any given directory beneath it. From then on, all filesystem access will be relative to this new root directory. A typical chroot usage is summarized as follows [2]:-

- Prepare a directory containing only the service to be run, and its dependencies.
- Change to that directory, and then chroot into it, making it the new root directory.
- Become non-root user.
- Run the service.

From here, it is evident that chroot is designed only to restrict filesystem access, nothing more. All other resources of the running kernel, such as networking resources, and processes outside of the chroot directory, are theoretically available for interaction by the chrooted process, since chroot is not designed to cover that much scope. [1][16]

The weakness which comes out of this is the following: as long as any possibility exists that a service can re-gain root access in its operation, then there is a possibility of a "chroot escape", i.e. the process breaks out of the chroot filesystem scope, and gains access to the rest of the host filesystem. [15]

This makes chroot protection inadequate for situations which require delegation of high privileges to untrusted services or users. A classic example is an Internet Service Provider, hosting multiple independent and mutually untrusted web sites. Each site owner has the privilege to install customized components such as Java, PHP, MySQL, etc. to their sites. Here, site owners can do nearly whatever they want with their sites, but strictly no access to others, as well as the base Operating System. [12] Another example is a more common situation: due to certain constraints, system administrators are forced to run services which are known to be vulnerable, needs the root privilege, or have dependencies all over the whole filesystem. The risks involved in these kinds of situations cannot be effectively mitigated using chroot, and will be referred to as "the System Partitioning Problem".

In order to solve this problem, many started to wrote features on top of, in addition to, or totally replacing chroot, all with a common goal: to further isolate, the service from the rest of its environment. Jail is one such effort, unique to the FreeBSD kernel.

How Does Jail Work?

At the time of writing, the current release version of FreeBSD is 4.4: the jail implementation may have changed since then. Some definitions are established here before proceeding:-

- "host": the main Operating System.
- "kernel": the main Operating System module which resides in memory, handling core tasks such as process and memory management, networking, I/O, and task scheduling.
- "userland": the outer layer of an Operating System which performs actual intended tasks, and usually interacts between the user and the kernel.

The jail[8] system call is typically called by a userland binary called jail[9], which takes 4 arguments:-

```
jail path hostname ip-number command ...
```

1."path": An arbitrary directory in the host containing the jail filesystem environment. Similar to chroot, this directory can contain minimal service binaries, dependent libraries, devices, and configurations with minimum permissions set up for the service to run. However, in jail environments, it is common for the whole FreeBSD distribution to be installed into this directory, with only small exceptions, such as most device nodes and the kernel. [12]

2."hostname": The hostname assumed by the jail environment. This entry is currently used for rudimentary jail process accounting.

3."ip-number": A unique IP address, an alias of the host interface. The jail will not be able to bind and utilize network resources on the host, except the ones associated with this IP address.

4."command": The program that is to be run in the jail, typically a shell such as /bin/sh.

The first three arguments is then passed to the jail[8] system call, which does the following:-

1.Calls chroot to change the root filesystem environment to the supplied "path". Here, the usual rules of chroot applies, along with some FreeBSD-specific protections against known chroot escapes. [5]

2.Assigns a "prison" structure to the current process. The structure holds the "hostname" and "ip-number" pair associated with the jailed process, which gets inherited to its children. This prison structure gets associated with the process until the process is eliminated. In other words, the process and all its children are imprisoned, in the same jail, for life. [6][13]

The "command" is then executed via the execve system call, thus becoming a new process, inside the chroot environment, bound by the prison structure.

How does the kernel recognize a jailed process, and how to restrict its movements?

In various parts of the FreeBSD kernel, additional checks are made to see if the current process is associated with a prison structure, so called a jail "inmate". If the process is an inmate, restrictions will be imposed to the process, which can roughly be divided as follows [6][13]:-

1.Restrictions on network resources.

An inmate will only be able to bind and utilize the single IP address "ip-number" associated to it in its prison structure. Requests to use other available IP addresses (such as the host system's and other jails', including loopback address 127.0.0.1) will be silently diverted to this IP address.

2.Restrictions on processes.

An inmate can only interact with its fellow inmates, i.e. other processes in the same jail. Processes in other jails and the host system will not be accessible, or even visible, at all. Even with prior knowledge of processes running in other hosts, most inter-process communications and memory sharing mechanisms between them are prohibited.

3.Restrictions on the root privilege set.

An inmate with root privileges will be prohibited from:-
- Modifying the running kernel and its parameters (e.g. sysctl, securelevel, kldload)
- Modifying any network resource (e.g. ifconfig, route)
- Mounting and unmounting file systems (e.g. mount, umount)
- Creating device nodes (e.g. mknod)
- Changing file flags (e.g. chflags)
- Accessing raw, divert or routing sockets (e.g. ping, tcpdump, packet generators)

If a process has no prison structure, it is deemed as a host process, and is exempted from all the above restrictions.

The end result: root's powers within jail are severely limited, and what's left of these powers are only effective inside the jail. On the other hand, the host system root has full authority over all jails.

A rough visualization of the difference between chroot and jail is shown in the diagram below.

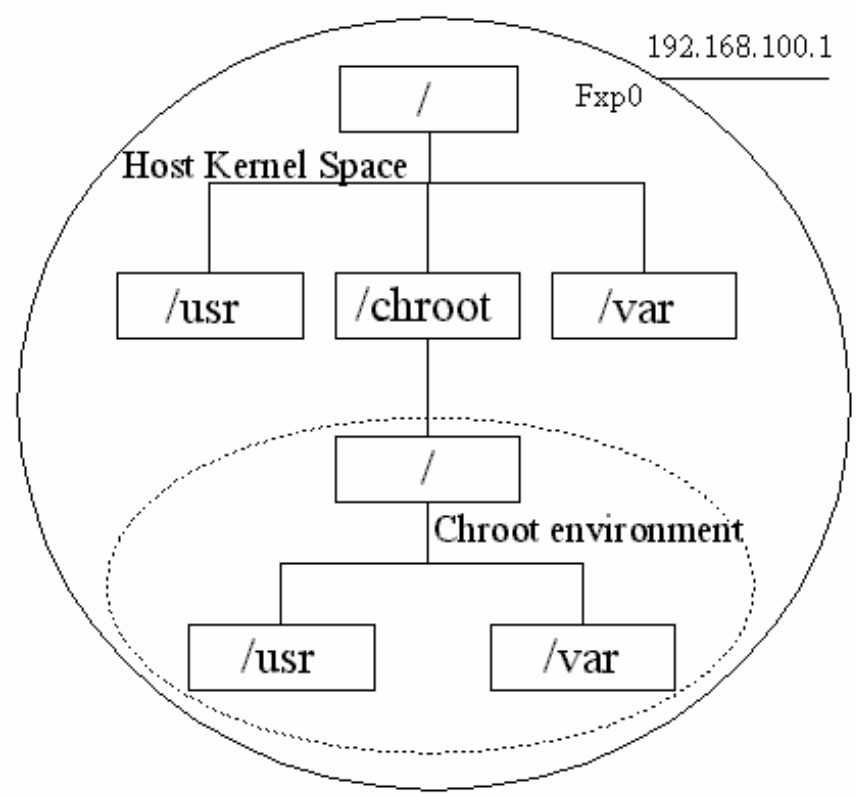

Figure 1: Chroot environment still has full access to most kernel resources

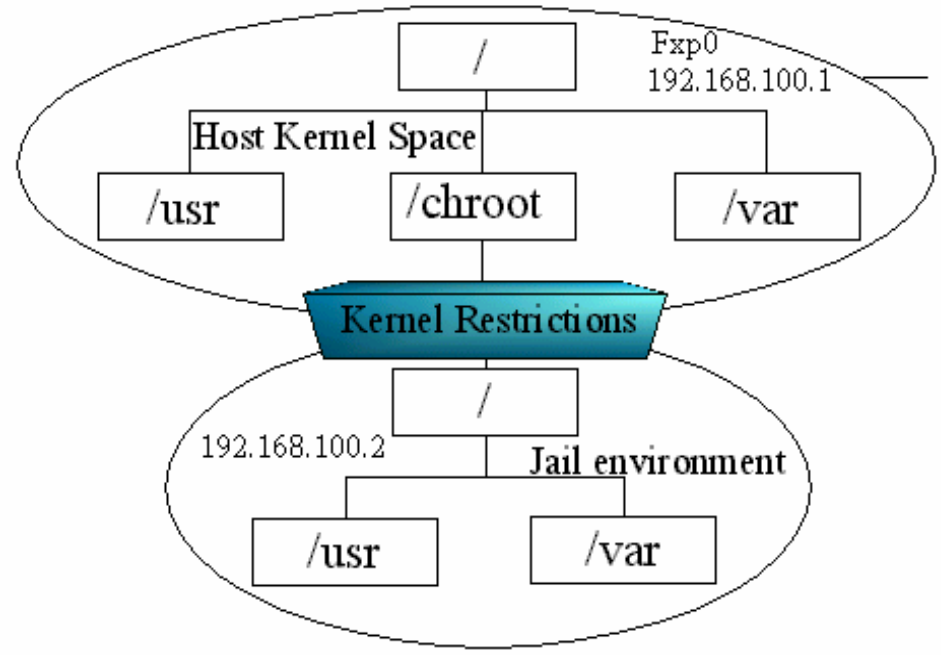

Figure 2: Jail environment goes through mandatory checking and restrictions at the kernel. All network activity in jail is redirected to an aliased IP address (192.168.100.2).

A Short Generic HOWTO

Setting up jail partitions is fairly straightforward (This assumes familiarity with FreeBSD or any UNIX-based Operating System):-

PART 1: Preparing the Host System

The following preparations are done in the host system, as root. The steps can easily be put in scripts for faster maintenance:-

First, we need to create IP address aliases for our jails. If the host is equipped with a single network interface denoted as 'fxp0', and located inside a hypothetical subnet 192.168.100.0/24, put the following entries in /etc/rc.conf, defining alias for each jail:-

```
ifconfig_fxp0="inet 192.168.100.1 netmask 255.255.255.0"
ifconfig_fxp0_alias0="inet 192.168.100.2 netmask 255.255.255.255"
ifconfig_fxp0_alias1="inet 192.168.100.3 netmask 255.255.255.255"
....
```

To activate an alias manually, type:-

```
# ifconfig fxp0 inet 192.168.100.1 netmask 255.255.255.255 alias
```

1.Build the FreeBSD world distribution, as normal:-

```
# cd /usr/src; make buildworld
```

2.Create the jail partition, e.g. /jail. For fast performance, /jail should be mounted to a separate, fast disk subsystem.

3.For extra protection, edit /etc/sysctl.conf, and for extra put:-

```
jail.set_hostname_allowed=0
```

This is to protect root accounts in jail from being able to change the jail's hostname, thus possibly disrupting jail process accounting.

4.Make sure the host system is not running any service, but if some is needed, make sure that it listens only to the host IP address. This is done by doing:-

```
# netstat -an | grep LISTEN
```

and make sure no service is listening to all addresses (INADDR.ANY, or denoted as '*' in the netstat IP address output).

PART 2: Preparing each Jail Subsystem

1.For each of the jail directories e.g. /jail/dir, install copies of the compiled FreeBSD distribution:-

```
# mkdir /jail/dir
# cd /usr/src; make installworld DESTDIR=/jail/dir
# cd /usr/src/etc; make distribution DESTDIR=/jail/dir
NO_MAKEDEV=yes
```

NOTE: If this is an upgrade, do cd /usr/src; mergemaster -D /jail/dir to merge the configuration changes.

Note also that this step can be replaced with chroot-style directory preparations, such as limiting directory contents with only needed binaries, libraries and devices. [2]

2.For each jail, create the jail-relevant devices:-

```
# cd /jail/dir/dev; sh MAKEDEV jail
```

3.For each jail, create bogus kernel and fstab:-

```
# cd /jail/dir; ln -sf dev/null kernel
# touch /jail/dir/etc/fstab
```

4.Mount procfs for jail process accounting:-

```
# mount -t procfs proc /jail/dir/proc
```

5.Next, the sysinstall binary is created for each jail:-

```
# mkdir /jail/dir/stand; cp /stand/sysinstall /jail/dir/stand
```

6.Enter the jail by executing:-

```
# jail /jail/dir jail-hostname 192.168.100.2 /bin/sh
```

This will open up a shell in the new jail. From this point on, all operations below are run within the jail subsystem.

7.Run /stand/sysinstall. Choose "Do Post-install configuration..".

8.Go to time zone, and define the time zone. Exit sysinstall, and the jail shell.

Now we have created a fully independent FreeBSD system inside the jail. To start or "boot" the jail environment, do the following:-

```
# jail /jail/dir jail-hostname 192.168.100.2 /bin/sh /etc/rc
```

Once the jail is up, the administrator can log on into it, install and configure required packages, as with any normal FreeBSD host. (e.g. pkg_add or 'make install' in /usr/ports tree).

Further Enforcement on Jails

What if a malicious attacker, who managed to gain root in a jail environment, uses the jail environment as a launch pad for attacking other hosts reachable in the network?

Recall that a jail is always spawned from the single host kernel. All low level operations in the jail, including networking, will go through the host kernel. Therefore, any security policies enforced on the host system kernel will be inherited by the jails. Also recall that root cannot in any way modify the running kernel and its parameters. This would include the enforced security policies, thus protecting it from being changed by the jailed root.

For example, a host administrator might want to limit and monitor all networking activity in the jails. To achieve this, the administrator can configure the host kernel to perform packet filtering, redirect certain traffic to a proxy, or sniff all traffic and analyze them using a Network Intrusion Detection System. Next, the administrator might also want to enforce disk quota on the partitions where the jail directories reside. All jail subsystems will be subjected to these mechanisms without any way of getting around them.

A demonstration of such a controlled jailed environment can be found in OpenRoot.org, a site which gives out root accounts to any interested visitors. The root accounts are safely tucked inside jailed partitions, having full power within the jail, but totally powerless to the outside world. For example, the only network connection allowed is FTP access to an anonymous FTP site. [14]

A Comparison with Other Partitioning Efforts

Other approaches to solving the partitioning problem follows:-

1.*Trusted Operating Systems, Derivatives and Subsets*

Trusted Operating Systems, first introduced in the early 1980s, offer much more fine-grained compartmentalization of system resources. For example, the UNIX file permissions model is replaced with stronger access control list, covering every resource in every possible access methods. The powerful set of privileges previously owned by root are broken apart into "capabilities", each capability given to processes and users in a strict "need to" basis, consistent with the "least privilege" principle. Users are classified in military-style multi-level security. Examples of such systems include Trusted Solaris, Trusted IRIX, Trusted AIX, and TrustedBSD. [17] Efforts which apply a subset of or alternative to these features also exist, notably Immunix and its Subdomain component. [3]

The disadvantages of trusted systems however are:-

(i) Trusted systems are complex, resulting in high set up and administration costs. Complexity also invites more bugs and security holes to the trusted system implementation itself. Furthermore, it also slows down development: trusted systems often lag behind their mainstream counterparts in terms of features and capabilities. [11]

(ii) Not fully compatible with applications: most applications are not designed to run on trusted systems. As a result, applications need to be fully studied and tested to determine what resources they need, and what privileges they need to have on them. [11][3]

In contrast, FreeBSD jail preserves the simple UNIX Security model in each jail, thus being more friendly to system owners, administrators, and applications alike.

2.*Userland Virtual machines*

These are a newer breed of solutions which usually involves booting a whole Operating System on top of a userland emulation layer. Examples include VMware and User-mode Linux [4]. Although their design and purpose may not be in the same vein as chroot and jail, they can be used to solve the partitioning problem. In each virtual machine, every system resource, including disks and network interfaces, are emulated, all as a non-root user. As a result, the emulated system is securely isolated from the base system and other virtual machines. As each virtual machine is run as non-root, resource limiting can easily be enforced on them, to avoid host availability attacks from within each virtual machine.

However, this also becomes a disadvantage: each virtual machine needs to run its own copy of the kernel in the host's memory space. Thus each virtual machine has a much bigger memory footprint. Furthermore, performance of virtual machines are slower, since every low level operation has to go through the emulation layer. In contrast, FreeBSD jails interact with a single kernel, controlled by the underlying host.

Disadvantages of FreeBSD Jail

FreeBSD jail does have some downsides:-

1.Jail is not a portable solution: it is only available on FreeBSD.

2.Jail is a relatively recent, immature implementation. It was first released along with FreeBSD 4.0, in March 2000. As with any new feature, the security and robustness of the implementation is not yet proven. Jail management utilities, for administration and monitoring from the host system is also lacking. For example, there is no distinction between processes in different jails via 'ps'. For host availability protection, per-jail resource limiting mechanisms are not possible: they share a common limit imposed at the host kernel. As a result, a jailed root can easily perform denial of service attacks on the host system resources, and other jails. Although these problems can be solved using third party tools, it is beyond the current scope of jail.

3.Jail is also arguable as a kernel feature. Some view FreeBSD jail as a "kernel bloat" and a case of "creeping featurism": the kernel goes out of its way to act as a safety net for applications, and in doing so, adds complexity to the otherwise simple kernel. This opens up more opportunities for bugs and security holes in itself. This argument probably explains why the feature has not been, or will never be ported to other BSD-based kernel implementations. [10]

The Future of FreeBSD Jail

The jail feature is under heavy development, and in some areas, being re-implemented, led by Robert Watson. [18] The new version is known as jailNG. Notable improvements in jailNG are improved jail management facilities, per-jail system policies, and better virtual networking support. The jail concept is also used in Watson's TrustedBSD project, an effort

to incorporate Trusted Operating System features into the FreeBSD kernel. [17]

Conclusion

Overall, the FreeBSD jail concept is an interesting addition to solving the system partitioning problem: it fills the gap between the traditional UNIX security model, and the more fine-grained security models such as the Trusted Operating System. Although it is still immature, and has problems of its own, the jail concept is still attractive as its design is simple and friendly to current applications and administrator skillsets. However, like chroot, in most cases jails act as "damage control" mechanism. The fact remains that for each service, there is always some amount of data associated with it: such that a successful exploit of a service, even trapped within a jail, can be an unacceptable risk. In this case, even jail is not enough: in addition, a secure implementation of the service is used, and always kept up-to-date with security fixes.

SOURCES

[1] Al-Herbish, Thamer. "Secure UNIX Programming FAQ", May 1999. http://www.whitefang.com/sup/secure-faq.txt

[2] Borland, Matt. "Locking Down Your Daemons: An Overview of 'chroot jailing' Services in Linux", SANS Institute: Information Security Reading Room, May 2001. http://www.sans.org/infosecFAQ/linux/daemons.htm

[3] Cowan, Crispin; Beattie, Steve; Kroah-Hartman, Greg; Pu, Calton; Wagle, Perry; Gligor, Virgil. "SubDomain: Parsimonious Server Security", WireX Communications, Inc., 2000. http://www.immunix.org/subdomain.pdf

[4] Dike, Jeff. "The User-mode Linux Kernel Home Page: The Kernel", 2001. http://user-mode-linux.sourceforge.net/kernel.html

[5] FreeBSD Project, The. "CVS log for src/sys/kern/vfs_syscalls.c", November 2001. http://www.freebsd.org/cgi/cvsweb.cgi/src/sys/kern/vfs_syscalls.c

[6] FreeBSD Project, The. "CVS log for src/sys/kern/kern_jail.c", December 2001. http://www.freebsd.org/cgi/cvsweb.cgi/src/sys/kern/kern_jail.c

[7] FreeBSD Project, The. "FreeBSD Hypertext Man Pages: chroot(2)", June 1993. http://www.FreeBSD.org/cgi/man.cgi?query=chroot&sektion=2&apropos=0&manpath=FreeBSD+4.4-RELEASE

[8] FreeBSD Project, The. "FreeBSD Hypertext Man Pages: jail(2)", April 1999. http://www.FreeBSD.org/cgi/man.cgi?query=jail&sektion=2&apropos=0&manpath=FreeBSD+4.4-RELEASE

[9] FreeBSD Project, The. "FreeBSD Hypertext Man Pages: jail(8)", April 1999.
http://www.FreeBSD.org/cgi/man.cgi?query=jail&sektion=8&apropos=0&manpath=FreeBSD+4.4-RELEASE

[10] Google Groups. "Re: chroot() break", Online Discussion Thread in openbsd.misc, on the Merits of Jail as a Kernel Feature, May 2001.
http://groups.google.com/groupshl=en&th=95c4fab32332ec11&rnum=15

[11] Jacobs, Charles. "Trusted Operating Systems", SANS Institute: Information Security Reading Room, May 2001.
http://www.sans.org/infosecFAQ/securitybasics/trusted_OS.htm

[12] Kamp, Poul-Henning; Watson, Robert N.M. "Jails: Confining the omnipotent root", The FreeBSD Project, May 2000.
http://docs.FreeBSD.org/44doc/papers/jail/jail.html

[13] Sarmiento, Evan. "Inside Jail", Daemonnews, September 2001.
http://www.daemonnews.org/200109/jailint.html

[14] Sarmiento, Evan. "Openroot.org", 2000.
http://sektor7.ath.cx:8080/openroot/index.php

[15] Simes. "How to break out of a chroot() jail", January 2001.
http://www.bpfh.net/simes/computing/chroot-break.html

[16] suplist@whitefang.com. "Newbie Q : What about chroot?", Secure UNIX Programming Mailing List Thread, June 1999.
http://www.whitefang.com/sup/archive/msg00000.html

[17] Watson, Robert N.M. "Adding Trusted Operating System Features to FreeBSD", June 2001.
http://www.trustedbsd.org/documentation/implementation/trustedbsd-usenix-2001.ps.gz

[18] Watson, Robert N.M. "jailNG", April 2001.
http://www.watson.org/~robert/jailng/

Last Updated: July 24th, 2017

Upcoming Training

SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
Community SANS Omaha SEC401*	Omaha, NE	Aug 14, 2017 - Aug 19, 2017	Community SANS
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Trenton SEC401	Trenton, NJ	Aug 21, 2017 - Aug 26, 2017	Community SANS
Community SANS Pasadena SEC401 @ NASA	Pasadena, CA	Aug 23, 2017 - Aug 30, 2017	Community SANS
Mentor Session - SEC401	Minneapolis, MN	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Sep 06, 2017 - Oct 18, 2017	Mentor
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
SANS Copenhagen 2017	Copenhagen, Denmark	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Boise SEC401	Boise, ID	Sep 25, 2017 - Sep 30, 2017	Community SANS
Baltimore Fall 2017 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
Community SANS New York SEC401	New York, NY	Sep 25, 2017 - Sep 30, 2017	Community SANS
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Oct 02, 2017 - Oct 07, 2017	Community SANS
Mentor Session - SEC401	Arlington, VA	Oct 04, 2017 - Nov 15, 2017	Mentor