Global Information Assurance Certification Paper

Linux Kernel Hardening

Taylor Merry
November 18, 2003

Abstract

While not inherently insecure, the standard Linux kernel lacks advanced features to prevent or contain certain types of malicious attacks. This paper explores two approaches to hardening the standard Linux kernel: address space (memory) protection and advanced access control. Additions to the kernel which place restrictions on an application's address space make it possible to prevent many types of buffer overflows attacks. The addition of an access control system can remove many, if not all of the privileges assigned to the traditional superuser account.

After brief overviews of three methods of address space protection and three advanced access control systems, this paper outlines the installation and configuration of a Grsecurity-enhanced kernel. The Grsecurity kernel patch provides both address space protection and an advanced access control system.

Linux kernel hardening is an effective strategy for preventing many forms of attacks and providing enhanced host-level security, however the approach described in this paper should not be expected to prevent all attacks against Linux hosts.

Background

Like any other operating system, application level security flaws leave Linux vulnerable to a variety of malicious attacks. Over the years, many tools and techniques have been developed to “harden” Linux hosts in an attempt to mitigate the risk posed by buggy software. Removing or disabling unnecessary services and daemons, properly configuring services and changing vendor defaults, applying the appropriate security and bug patches, setting up backups, and configuring monitoring tools are all essential steps in building a secure system.

Once an operating system has been set up and placed in production, daily monitoring of vendor notices and security forums is necessary to ensure that software is kept current with the latest security issues. Unfortunately, this method of administration is entirely reactionary and leaves hosts susceptible to compromise before vulnerabilities are publicly announced and fixes have been distributed. While many systems are compromised due to lack of proper maintenance (ie. patches not applied when they are made available), there is a strong need for methods to further reduce the risks associated with flawed software. Keeping systems up to date with vendor patches will prevent the casual attacker from gaining access to a system, but will not always keep out an attacker that is targeting a system.

In today's computing environments, there is an ever increasing number of applications that require network connections. It is rare to find a machine that can simply be

disconnected from a network in an attempt to prevent it from being compromised.

The Carnegie Mellon University's Computer Emergency Response Team Coordination Center (CERT/CC) reports a dramatic increase in the number of security incidents reported in the past 10+ years:

1988-1989

Year	1988	1989
Incidents	6	132

1990-1999

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
Incidents	252	406	773	1,334	2,340	2,412	2,573	2,134	3,734	9,859

2000-2003

Year	2000	2001	2002	1Q-3Q 2003
Incidents	21,756	52,658	82,094	114,855

Total incidents reported (1988-3Q 2003): 297,318

Please note that an incident may involve one site or hundreds (or even thousands) of sites. Also, some incidents may involve ongoing activity for long periods of time(1).

While the number of computers connected to the Internet can be directly correlated with this trend, the number of attacks and the amount of damage incurred will only increase as more people become dependent on this connectivity. There is a need to reassess the trade-off made between host-level security and the convenience offered by various types of services and features if the intent is to keep these hosts connected to the Internet.

In general, the initial goal of an attacker attempting to gain access to a Linux host (or any other UNIX variant) is to gain control of the superuser (aka. root) account. The traditional superuser account has unrestricted access to all components of the system. In most cases, there is little to no monitoring or auditing of actions taken by the superuser. Even when configured, an attacker with superuser privileges has the ability to disable these services, or cover their tracks by modifying log files. Once an attacker has control of the superuser account, even the most novice attacker will likely install rootkits and kernel modules that will allow them to continue to use the system for nefarious purposes without being easily noticed.

Aside from instances where an attacker is able to bypass physical access restrictions, the ability to gain control of a system is often due to “trusted” programs. A “trusted” program is one that an administrator allows to run with superuser privileges. This is not always the fault of the administrator – certain applications do not function correctly without these privileges. The exploitation of bugs in flawed software can allow an attacker to gain the privileges of a compromised program. Cowan, et al., (2) outlines buffer overflows, race conditions and special character processing as the most widely known application vulnerabilities that allow an attacker to acquire superuser privileges.

- Buffer Overflows
  The result of a program failing to properly check the boundaries of user input. If more data is entered into a memory buffer than it was intended to store, the data will overwrite data in adjacent buffers potentially giving an attacker control of the vulnerable application and access to the system privileges granted to that application.
- Race Conditions
  A vulnerability in a program that does not properly check for the previous existence of a file before writing to it. In some cases, the delay between a check and the creation of the file may afford an attacker enough time to create a link to an important system file so that when the application tries to create the new file, it ends up overwriting or corrupting the other file.
- Special Character Processing
  A vulnerability that allows an attacker to insert special characters into URLs pointing to scripts that process web forms. These character sequences could allow attackers to execute arbitrary code on the web server. Although most web servers are configured to run as unprivileged users today, these type of attacks can allow for unauthorized entry into a system which can be used to leverage attacks against other flawed applications.

Methods to prevent or mitigate the risk posed by a successful attack are generally applied on either the application level or the operating system level. A standard way to mitigate risk on the application level is by granting it the least amount of privilege required to perform its duties. While most applications do not need to run as a privileged user, there are some system level applications which require superuser privileges to perform certain tasks. In the past, many attacks against applications such as the Apache web server, the Berkeley Internet Name Domain (BIND) server, and the Sendmail mail server led to full system compromise as they required superuser privileges to function. In the 2.4 kernel series, enhancements to the kernel capabilities system makes it possible for applications to execute initially as root and then drop superuser privileges to run as a regular user. These enhancements are quite beneficial to overall system security, and today, most Linux distributions configure applications such as Apache, BIND, and Sendmail to run as unprivileged users by default.

Another tactic adopted to secure certain system level applications is through the use of the *chroot* program. *chroot* makes an application execute inside a special shell with a new root directory, effectively isolating it from the rest of the system. If an attacker were able to exploit a vulnerability in an application running in a chroot jail, they would only have access to this limited root file system containing the essential programs, configuration files and libraries necessary for the application to operate. Unfortunately, chroot jails are not exempt from security flaws – in some situations when an application inside a chroot jail is running as a superuser, there are methods which allow attackers to break out of the jails.(3)

The millions of lines of source code that comprise the applications used on today's Linux systems make it nearly impossible to fully audit each application and remove all potential vulnerabilities. While still a critical component of a defense in-depth strategy,

hardening techniques at the application level are typically inadequate as a means in protecting the rest of the system from flaws in the applications. Regardless of the time spent hardening applications, a vulnerability exploited in one application will often lead to a full system compromise. As the dependency upon network connectivity increases, the need to look at lower level methods for preventing successful attacks becomes apparent.

The Linux Kernel

The lowest software level components of "[e]ach computer system includes a basic set of programs called the operating system. The most important program in the set is called the kernel. It is loaded into RAM when the system boots and contains many critical procedures that are needed for the system to operate."(4) While the system is running, the kernel acts as a mediator between the hardware components and the processes running on the system – none of the processes directly access any hardware components on their own. At an even lower level, features built into hardware components help to enforce the separation between those that can ("kernel mode") and cannot ("user mode") directly interact with the hardware components.

The Linux kernel provides for a multiuser operating system. This means that any user can run any program at any time without worrying about the other users. Due to the multiuser nature of the system, the kernel is also responsible for providing mechanisms for user authentication and access control to prevent users or applications from interfering with the activity of other users or programs. The access control system built into the standard Linux kernel is based on the traditional Discretionary Access Control (DAC) model. DAC is discussed in further detail later in this document.

Linux Kernel Modules

At a time when memory (RAM) was an expensive commodity, the idea of a modular kernel was introduced in a few commercial UNIX distributions which allowed the loading and unloading of pieces of kernel code to free up memory without having to rebuild the entire kernel. Although memory is much less expensive today, the flexibility that loadable kernel modules offer is one of the main reasons that they are still used heavily today. The Linux kernel is one of the few modern monolithic kernels with the ability to load and unload modules on the fly. This can prove to be a big time-saver as it prevents an administrator from having to recompile the whole kernel and then reboot the system for every minor change.

"Linux was first developed for 32-bit i386-based PCs. These days it also runs on (at least) the Compaq Alpha AXP, Sun SPARC and UltraSPARC, Motorola 68000, PowerPC, PowerPC64, ARM, Hitachi SuperH, IBM S/390, MIPS, HP PA-RISC, Intel IA-64, DEC VAX, AMD x86-64 and CRIS architectures."(5) Due to the variance in hardware components between the platforms, many Linux distributions provide stock kernels that rely heavily on the use of kernel modules. Not only does this approach greatly assist in the initial setup of a Linux system, but it also allows Linux distributions to maintain a small number of pre-built kernel images optimized for the various CPU configurations. While this approach eliminates the need for a custom built kernel on

each system, it also provides a facility that allows an attacker to load harmful bits of code into a running kernel on a compromised machine.

A statically complied kernel with the loadable module option disabled is one suggested method for hardening a standard Linux kernel. While it is true that this approach will likely prevent an inexperienced attacker from installing rootkits that use kernel modules on a compromised host, it is unlikely that it will deter an experienced and determined attacker. Superuser privileges are required for the loading and unloading of kernel modules. If an attacker is able to load kernel modules, then they already have full access to the entire system. Furthermore, it has been shown that it is possible to patch Linux kernel modules into a statically built kernel.(6) As one can deduce, disabling kernel modules in the standard Linux kernel does not offer any significant increase in the overall security of the system.

The Linux kernel developers have merged a security framework into the 2.5 development kernel series to make it easier for administrators to implement security frameworks as Linux kernel modules. These new features are discussed later in this document.

Address Space Protection

Buffer overflows are one of the leading vulnerabilities exploited to gain control of a system. While processing data, applications create finite buffers in the computer's memory for temporary storage of data such as user input. The problem arises when a user is able to insert more data into this buffer than the programmer originally allocated. This overflow of data into adjacent buffers could crash the application or even the entire system. When intentionally crafted, an attacker could use the application's address space to introduce and execute arbitrary code in an attempt to gain control of the system. For an in depth look at buffer overflows, see the “Buffer Overflows for Dummies” paper located in the SANS Reading Room.

Three projects that have researched and introduced implementations to provide some protection against buffer overflows include the Openwall Project, Exec Shield and PaX.

Openwall Linux kernel patch

The Openwall Project created one of the earlier security patches for the Linux kernel that addressed the problems created by an executable user stack area. This feature prevents attacks where a buffer overflow overwrites return pointers allowing an attacker to execute arbitrary code inserted into the stack. “This patch also changes the default address that shared libraries are mmap()'ed at to make it always contain a zero byte. This makes it impossible to specify any more data (parameters to the function, or more copies of the return address when filling with a pattern), -- in many exploits that have to do with ASCIIZ strings.”(7) While an excellent strategy in buffer overflow prevention, this system only provides protection against these specific types of buffer overflows. These features are also only available for Linux kernels running on i386 architectures.

Other (architecture independent) security enhancements provided by the Openwall

kernel patch include:

- restricted links and FIFOs in /tmp;
- restricted access to /proc;
- enhanced enforcement of the number of user processes;
- destruction of shared memory segments not in use;
- other enhancements for pre-2.4 kernels.

Exec Shield

The exec-shield kernel patch takes the approach used by the Openwall Project's implementation of a non-executable stack and extends it in an attempt to protect all virtual memory segments. “The exec-shield feature provides protection against stack, buffer or function pointer overflows, and against other types of exploits that rely on overwriting data structures and/or putting code into those structures. The patch also makes it harder to pass in and execute the so-called 'shell-code' of exploits.”(8) At this time, the exec-shield kernel patch is limited to prevention of address space attacks, and does not provide any other features. As with the Openwall kernel patch, this patch is designed for i386 based architectures.

PaX

The PaX Team has created a kernel patch to prevent various classes of software vulnerabilities “that give an attacker arbitrary read/write access to the attacked task's address space.”(9) By separating the writable and executable properties on memory pages, the NOEXEC feature is comprised of the three following options:

- Paging based non-executable pages (PAGEEXEC)
  This option utilizes the CPU's paging features to implement non-executable pages – geared towards non-i386 architectures.
- Segmentation based non-executable pages (SEGMEXEC)
  This option utilizes the segmentation features unique to i386 CPUs for the implementation of non-executable pages on i386 architectures.
- mmap() and mprotect() restrictions (MPROTECT)
  These features are to help “prevent the introduction of new executable code into the task's address space”(10) by adding restrictions to the mmap() and mprotect() functions. These features are mostly architecture independent.

A side-effect of the PAGEEXEC and SEGMEXEC features is that certain bits of necessary runtime code generated by some applications will fail to work. Two features, EMUTRAMP and EMUSIGRT, have been created by the PaX Team as a workaround for two specific cases. When enabled, these features allow gcc trampolines and kernel signal return stubs to function correctly.

In other cases where applications do not function as a result of the NOEXEC features, the PaX team has also created a tool called *chpax* which allows an administrator to enable or disable PaX features on individual binaries. The source code for the *chpax* program is available on the PaX site.

The Address space layout randomization (ASLR) feature introduces randomness into the locations of pieces of data that are stored in memory. This feature is implemented in four different areas:

- Stack addresses used by processes in user mode (RANDUSTACK);
- Stack addresses used by processes in kernel mode (RANDKSTACK);
- Other “memory regions handled by the do_mmap() kernel interface”(11) (RANDMMAP);
- “[M]ain executable file mapping addresses”(12) (RANDEXEC).

The ASLR feature makes it extremely difficult to execute attacks that depend on advance knowledge of the location of data stored in memory. Although workarounds such as convincing the application to divulge information regarding the ASLR layout, or by a brute-forced attack against the randomization are possible, it is unlikely this event would go unnoticed as an application would crash in the process of such an attack.

Each of the patches outlined above make it more difficult for an attacker to execute attacks against an application's address space. While there are varying levels of protection provided by each, the use of any of the patches can greatly increase the overall security of a Linux system.

Access Control

Modern computing platforms provide access control systems to ensure that only authorized users are able to access objects that they have permission to access. On standard Linux distributions, there are two access levels by default – regular users and the superuser.

Discretionary access control (DAC) allows regular users to modify any objects that they own. DAC is based solely on the identity of a given user, the ownership of an object, and the permissions that have been granted to that user. The superuser, as the owner of all objects on the system, is not subject to any of these restrictions and can override any restrictions set up by regular users.

With the implementation of a strong security policy and some application layer restrictions, access to the root account can be restricted by:

- restricting root login to system console only;
- restricting use of the *su* command to a group of administrators;
- requiring administrators to use tools such as *sudo* to run commands as root.

These techniques along with a strong password policy are excellent methods of protecting against accidental password disclosure, mistakes made by administrators logged in as root, and brute-force password attacks against the root account. However, these techniques do little to protect the system from applications that run as root.

Various different approaches have been proposed regarding access control

mechanisms, but there has been a general disagreement between Linux developers as to which approach is the best – this is partially the reason that none of these systems have been implemented in the mainstream Linux kernel. The complexity and large time commitments required to learn and maintain these systems often makes their use unattractive to administrators. In some cases, these systems interfere with applications and functions of the kernel. In attempting to increase the overall security of Linux hosts, the trade-off between convenience and security is something that needs reexamining.

Some alternatives to DAC include:

- Mandatory Access Control (MAC)
  A system which requires a security administrator to define the access control attributes assigned to objects, users and processes. These policies cannot be modified by regular users. By comparing the access control attributes of an object with the access control attributes of the user or process that is trying to access that object, the system will determine whether to allow or prevent that access.
- Role Based Access Control (RBAC)
  A system which assigns users to groups and then assigns the groups the ability to perform specified tasks on specified pieces of data.
- Rule Set Based Access Control (RSBAC)
  A modular combination of security models which divides access control "into enforcement, decision and data structures, and [groups] all access modes...into abstract request types...the controlled object types include interprocess communication as well as devices..."(13)
- Domain and Type Enforcement (DTE)
  A system which separates processes into domains and objects into types. The system can then be configured to allow or prevent specific domains from accessing specified types. Access controls can also be set up to allow or prevent domains from interacting with each other.

While seemingly most useful on systems which require separate and distinct classification levels for data and users, advanced access control methods can also be highly effective in preventing an attacker from gaining full control of a system compromised by a buggy piece of software. Systems such as those outlined above enable an administrator to take away many, if not all, of the privileges assigned to the superuser on a standard Linux system. This is possible by using the least privilege principle when defining access controls – only allowing a user or process the minimum amount of privilege necessary to complete a task. In its purest form, strict mandatory access control is an extremely time consuming system to implement which mostly explains its lack of integration into operating systems. Some of the other alternatives allow for a more granular access control.

As with most modern UNIX derived operating systems, DAC is the only option available in standard Linux kernels up to, and including, the 2.4 series. Various projects have released their solutions for alternate access control systems in the form of patches to be applied to the kernel source tree before building it. While it would make sense to use the flexibility of Linux kernel modules to leverage these implementations, the

functionality of these additions requires the modification of core portions of the standard kernel.

Outlined below are a few examples of systems that add alternate access control functionality into the standard Linux kernel:

Linux Intrusion Detection System

The Linux Intrusion Detection System (LIDS) Project has released a kernel patch and administrative tool that adds mandatory access control to the kernel. Fine grained access control via ACLs serve to protect both objects and network connections. While this system could be considered more of an intrusion prevention system, an integrated port scan detector also helps to detect attempt at unauthorized access.

When a LIDS system is booted, file restrictions are enforced immediately. Once the system has come up, the *lidsadm -I* command will seal off the kernel, preventing any additional kernel modules from affecting it. All capability restrictions are also enforced once the kernel has been sealed. An administrator logs in to a shell free of the LIDS restrictions when they need to further configure the system. For a more detailed overview of LIDS, see the Securityfocus Infocus series “An Overview of LIDS”.

Rule Set Based Access Control for Linux

Rule Set Based Access Control for Linux is a kernel patch and set of administrative tools. The current patch includes implementations of various modules for access control among other advanced security options:

- an advanced access control system based on MAC;
- RBAC modules which allow security and systems administrators to access information required to perform their duties;
- a privacy module for storing personal data;
- a module which scans files for malware;
- a module which adds new file flags to the traditional read, write and execute flags used on objects (ie. append-only, secure-delete and no_delete_or_rename);
- access control lists for objects;
- authorization enforcement to further control the ability of a user to switch to another user;
- a capabilities module to restrict the privileges traditionally granted to the root account and grant those privileges to other users;
- advanced resource limits for users and applications.

The RSBAC implementation takes a modular approach which allows an administrator to choose the methods of access control appropriate for a given system. An administrator is even able to create custom access control modules with the RSBAC runtime registration facility. This kernel patch also includes advanced logging system to alert an administrator of unauthorized actions taken by any user or process.

Security-enhanced Linux

Security-enhanced Linux (SELinux) is a project that has integrated Role Based Access Control and Type Enforcement into the standard Linux kernel. Some of the distinctive characteristics of the Security-enhanced Linux system include:

- Clean Separation of Policy from Enforcement
- Well-Defined Policy Interfaces
- Independent of Specific Policies and Policy Languages
- Independent of Specific Security Label Formats and Contents
- Individual Labels and Controls for Kernel Objects and Services
- Caching of Access Decisions for Efficiency
- Support for Policy Changes
- Controls over Process Initialization and Inheritance and Program Execution
- Controls over File Systems, Directories, Files, and Open File Descriptions
- Controls over Sockets, Messages, and Network Interfaces
- Controls over Use of 'Capabilities'(14)

Originally designed as a standalone kernel patch based on the Flask (Flux Advanced Security Kernel) micro-kernel, SELinux has now been fully integrated into the Linux Security Modules framework.

Linux Security Modules

Differing opinions on the best OS level security system, enhanced security features and access control mechanisms have prevented many of the proposed solutions from being integrated into the mainstream Linux kernel. While Linux has long supported loadable kernel modules, “creating effective security modules is problematic since the kernel does not provide any infrastructure to allow kernel modules to mediate access to kernel objects.”(15)

In response to a presentation on SELinux, Linus Torvalds, the lead developer and inventor of the Linux kernel, outlined a general framework “that would provide a set of security hooks to control operations on kernel objects and a set of opaque security fields in kernel data structures for maintaining security attributes” that he would consider integrating into the Linux kernel.(16)

In the 2.5 development kernel series, the privileges of the traditional superuser account has been separated out into a capabilities system. With the integration of the Linux Security Modules (LSM) framework, it is possible to replace the traditional superuser account with an alternate system of access control in the mainstream Linux kernel. Some of the current systems that have been ported for implementation through the LSM framework include: POSIX.1e Capabilities, SELinux, Domain and Type Enforcement, LIDS, and a port of some features of the Openwall kernel patch.

As of November 2003, development on the 2.5 kernel series has ended and the stable 2.6 Linux kernel is in a pre-release testing phase. The new kernel will likely be released

in the first quarter of 2004.

Distributions featuring hardened kernels

In addition to projects focusing on the kernel, other projects were formalized to package various security enhancements into entire Linux distributions. While many of the mainstream distributions package some of these components, they generally are not implemented by default during system installation. Three projects that focus on building a highly secure operating system include Adamantix, Owl GNU/*/Linux, and the Hardened Gentoo Project.

Adamantix

The Adamantix project has created a full Linux distribution based on the current stable version of Debian GNU/Linux. By integrating and enabling security features not currently used in mainstream distributions, the Adamantix project has created a distribution that is much more secure out of the box.

The Adamantix kernel is patched with the PaX and RSBAC kernel patches to provide extra protection against address space attacks and a fine-grained access control system. There are three different types of kernel configurations available for use on these systems. The base kernel has RSBAC disabled; the image with the "-sec" suffix has RSBAC enabled and enforces all of the security policies; and the image with the "-soft" suffix has RSBAC enabled, logs all security policy violations but does not enforce them. As all of the kernels have PaX enabled, installing the *chpax* utility will allow an administrator to modify PaX flags should issues with these features arise.

All of the binary packages, including the kernels, available with this distribution were compiled with a version of the *gcc* compiler that was patched with IBM's Stack Smashing Protector. This helps to take buffer overflow prevention a step further.

Some of the additional features that are provided in the Adamantix distribution include:

- The Zorp application level proxy firewall;
- Integrated IPSec support;
- Integrated encrypted loopback device (ie. for encrypting home partitions);
- and WLAN Host-AP mode drivers that allow Linux to be used as a WLAN base station.

To an administrator familiar with the Debian installation process, switching to Adamantix is a fairly easy undertaking. Near the end of the Debian 3.0 (woody) installation process, both the tasksel/dselect steps are skipped, and the installer is exited. After logging into the system as root, the /etc/apt/sources.list file is modified so that it only contains one of the sources from the Adamantix mirrors page. By running "apt-get update", "apt-get install libncurses5" and finally "apt-get dist-upgrade" the system will be upgraded to the latest version of Adamantix.

While it is possible to use Adamantix binaries in Debian 3.0, or Debian 3.0 binaries in

Adamantix, incompatibilities may prevent certain packages from functioning correctly.

Owl GNU//Linux*

In addition to providing kernel patches for the Linux 2.0, 2.2, and 2.4 kernels, the Openwall Project offers a full security enhanced Linux distribution for servers. Owl uses a combination of methods to provide a secure operating system. Source code that runs as a privileged user, or is used to process data over the network was audited for specific types of software vulnerabilities before being included in the distribution. “This covers relevant code paths in many of the system libraries, all SUID/SGID programs, all daemons and network services.”(17) Any issues found during the audit are corrected, or the components are left out of the distribution. During the packaging process, applications are configured with safe default settings and set to run with the least privilege. Security enhanced applications developed by the Openwall project, such as pam_passwdqc, and a blowfish-enabled crypt are also included in the distribution.

The Owl distribution also features a build environment that allows an administrator to rebuild the entire system (except for the kernel) from source code by running the command “*make buildworld*”.

Hardened Gentoo Project

The Gentoo Linux distribution strives to provide a system that is highly configurable and optimized for each specific application. One of Gentoo's biggest features is its Portage software distribution and packaging system. Portage uses a set of scripts provided by Gentoo developers to build custom packages from source code for each system. As it is possible to build all system binaries from source, the integration of advanced security features becomes an easier task in some cases.

The Hardened Gentoo project is a subproject of the Gentoo Linux distribution which researches, tests and integrates advanced security applications into the main Gentoo distribution. Currently, teams of developers are focusing on integrating the features of projects such as SELinux, PaX, Grsecurity, and Openwall into the main Gentoo distribution. The project homepage contains a number of guides on implementing these features into a Gentoo system.

The standard kernel package for Gentoo, “gentoo-sources”, currently contains the Grsecurity kernel patch which will be discussed below. The “hardened-sources” kernel package patches a kernel with various of the patches provided by the Hardened Gentoo Project. While it is true that many other mainstream distributions offer packaged versions of kernel security patches, the custom building and configuration of kernel images is generally not as standard a procedure in other distributions as it is with Gentoo.

Hardening the Linux kernel with Grsecurity

In the sections above, various options were examined that address concerns regarding the prevention of certain types of attacks against Linux hosts through the use of

memory restrictions and alternate systems of access control. An administrator who does not wish to migrate their Linux hosts to a new distribution in order to take advantage of their hardened kernels will need to look at building a custom kernel.

While the section below goes into some depth regarding the building and installation of a Linux kernel, the Linux Documentation Project also has a comprehensive guide called the "Linux Kernel HOWTO" which outlines the procedures more thoroughly.

As each of the systems that have been outlined have strengths in various areas, an administrator wishing to secure the address space on a Linux box in addition to replacing discretionary access control might not have the time necessary to try out all the combinations of patches to find what works best. The Grsecurity kernel patch is a more comprehensive patch that addresses the need for security enhancements in both areas.

Process based MAC, chroot restrictions, address space modification protection, enhanced auditing, larger entropy pools, restrictions on the /proc file system, symlink and hard link restrictions in /tmp, FIFO and Dmesg restrictions, and enhanced Trusted Path Execution are a few of the features provided by this patch. The upcoming release of Grsecurity 2.0 will also include a Role Based Access Control system to enhance the current access control features.

On the network side, Grsecurity adds an additional module to the Netfilter options. This "stealth" module allows an administrator to create an iptables rule that will drop all TCP and UDP packets that have been sent to unused ports helping to prevent remote OS fingerprinting and slow down port scanners. This document will not go into depth on the use of this feature – in order to use it, is necessary to:

- enable "stealth match support" in the Netfilter options of a Grsecurity patched kernel;
- build and install a Grsecurity patched version of iptables;
- add a rule to the iptables rule-set.

Building a Grsecurity enhanced kernel

While this example jumps right in to the creation of a Grsecurity-enhanced kernel, it is always good idea to first build, install and reboot into a plain kernel to ensure that everything functions correctly before you begin applying third party patches. This will greatly assist in the debugging process should there be problems. In this example, it is assumed that you are currently running a typical vendor supplied kernel and have not already built your own custom kernel.

One can obtain the latest version of the kernel source code from the Linux Kernel Archives. Although most Linux distributions offer pre-packaged kernel sources, these versions of the kernel source usually include patches applied by the vendor which could prevent the Grsecurity patch from applying cleanly.

At the time of this writing, the current stable Linux source is 2.4.22. Using the *wget*

utility, the kernel source code and the OpenPGP signature are downloaded. The signature is used to verify the integrity of the source code tarball.

```
$ wget http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.22.tar.bz2
  <output deleted>
$ wget http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.22.tar.bz2.sign
  <output deleted>
```

If you don't already have a copy of the OpenPGP key for the Linux Kernel Archives (LKA), it can first be imported into GnuPG as shown below. When verifying the source tarball, the warning message “This key is not certified with a trusted signature!” only means that you or someone you trust (in the sense of OpenPGP trust) hasn't signed the LKA key. All of the information regarding the signatures is available at the Linux Kernel Archives. Once the kernel source tarball has been verified, it can be decompressed using the *tar* utility.

```
$ gpg --keyserver wwwkeys.pgp.net --recv-keys 0x517D0F0E
gpg: key 517D0F0E: public key "Linux Kernel Archives Verification Key
<ftpadmin@kernel.org>" imported
gpg: Total number processed: 1
gpg:               imported: 1
$ gpg --verify linux-2.4.22.tar.bz2.sign linux-2.4.22.tar.bz2
gpg: Signature made Mon Aug 25 07:54:52 2003 EDT using DSA key ID 517D0F0E
gpg: Good signature from "Linux Kernel Archives Verification Key
<ftpadmin@kernel.org>"
Could not find a valid trust path to the key.  Let's see whether we
can assign some missing owner trust values.

No path leading to one of our keys found.

gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
gpg: Fingerprint: C75D C40A 11D7 AF88 9981  ED5B C86B A06A 517D 0F0E
$ tar xjvf linux-2.4.22.tar.bz2
  <output deleted>
```

The latest Grsecurity kernel patches and the gradm source code are obtained from the Grsecurity site. The patches on the site each refer to a specific kernel version number – you should take care to match up the version numbers, or the patch might not apply cleanly. Again, the OpenPGP signatures can be obtained to verify the integrity of the patch as done above.

The next step is to apply the patch to the kernel:

```
$ patch -p0 < grsecurity-1.9.12-2.4.22.patch
  <output deleted>
```

After successfully patching the kernel tree, you can change into the base of the unpacked source (*cd linux-2.4.22*), and prepare to configure the kernel.

In the event that you do not have your own custom configuration file, then a good starting point would be to look in the /boot directory for a file with a name like “config-2.4.20”. If there are multiple configuration files, you can compare their version numbers with the output of the “*uname -r”* command to determine which is the configuration for

the kernel currently in use. This file can be copied to a file called ".config" so that it will be read when we begin the configuration with *make menuconfig*:

```
$ cp /boot/config-2.4.20 .config
$ make menuconfig
  <output deleted>
```

As you navigate through the menus of configuration options, you will see that a generic vendor kernel has a lot of options enabled that you won't need on your system. Some of these are built in as modules (flagged with an "M"), and others are built in statically (flagged with a "*"). In many cases you will know that you do not have or do not need support for certain types of hardware – ie. extra SCSI controllers, Firewire support, or Telephony. In other instances where you aren't sure what is needed you may need to reference the current system to determine which modules have been loaded, or which components were detected upon startup.

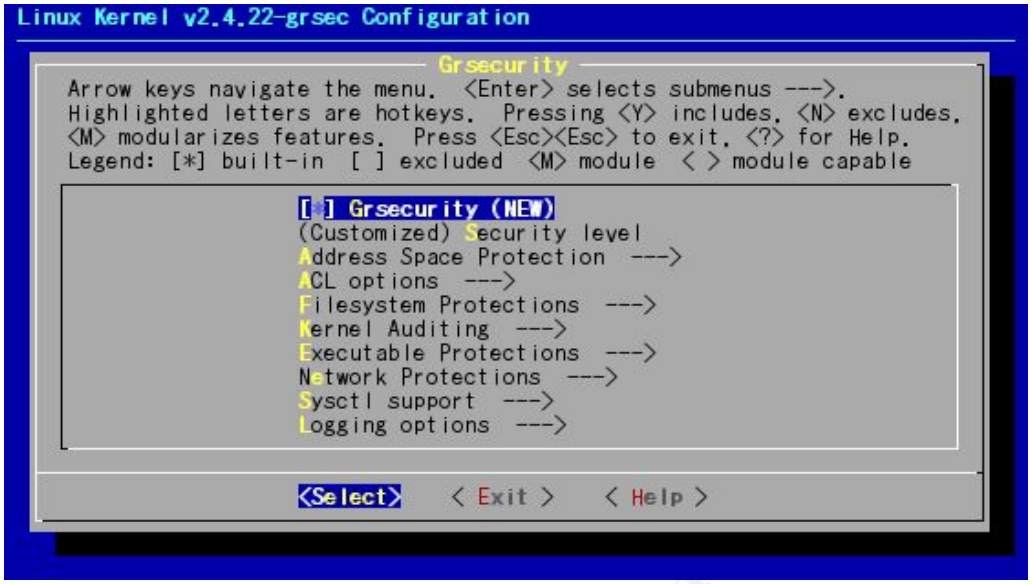
There are three ways to determine whether a particular option needs to be enabled or not. The first is by examining the help pages for the options in question. After highlighting a particular option, the left and right arrow keys can be used to highlight the word "Help". Pressing enter will then display a page describing the option. In many cases, the help pages provide information regarding the necessity of a particular option.

Secondly, the *lsmod* command will allow you to view all the modules that are currently loaded. When it is not clearly apparent which option corresponds to a particular module, searching through the Configure.help file located in the Documentation directory of the kernel source tree can be quite helpful. The Configure.help file is the one that provides the content for the help menus mentioned above.

```
$ sudo lsmod
Module                  Size  Used by    Not tainted
keybdev                 1664   0  (unused)
usbkbd                  2848   0  (unused)
input                   3072   0  [keybdev usbkbd]
usb-uhci               20708   0  (unused)
usbcore                48032   0  [usbkbd usb-uhci]
```

Another way to determine which options are in use is by reading through the system startup messages. These can be viewed by looking at the file "/var/log/dmesg", or by running the command *dmesg*. Disabling all of the unnecessary options can help to reduce the overall size of the kernel, but it may take a little bit of trial and error to assemble the smallest possible kernel that is still fully operational.

At the bottom of the main list of configuration menus, there is a Grsecurity menu which contains the Grsecurity options. After enabling Grsecurity, a detailed set of options is displayed:



At this point, there are two ways to proceed: the first is to go through all of the options one-by-one to enable the features that are needed, and the second is to select a pre-set “Security Level” from the second menu item as shown above. In a test environment, it may be desirable to build kernels at each of these levels to see if any necessary applications are adversely affected. The features enabled by each of the security levels are outlined below:

Low Security:

- linking restrictions
- fifo restrictions
- random pids
- enforcing nproc on execve()
- restricted dmesg
- random ip ids
- enforced chdir("/") on chroot

Medium Security (includes all of the Low Security options):

- random tcp source ports
- altered ping ids
- failed fork logging
- time change logging
- signal logging
- deny mounts in chroot
- deny double chrooting
- deny sysctl writes in chroot
- deny mknod in chroot
- deny access to abstract AF_UNIX sockets out of chroot
- deny pivot_root in chroot
- denied writes of /dev/kmem, /dev/mem, and /dev/port

- /proc restrictions with special gid set to 10 (usually wheel)
- address space layout randomization

High Security (includes all of the Low and Medium Security options):

- additional /proc restrictions
- chmod restrictions in chroot
- no signals, ptrace, or viewing processes outside of chroot
- capability restrictions in chroot
- deny fchdir out of chroot
- priority restrictions in chroot
- segmentation-based implementation of PaX
- mprotect restrictions
- removal of /proc/<pid>/[maps|mem]
- kernel stack randomization
- mount/unmount/remount logging
- kernel symbol hiding(18)

Outside of the options in these categories are a few additional options that can be enabled manually:

- PaX: PAGEEXEC
- PaX: EMUTRAMP
- PaX: EMUSIGRT
- PaX: Disallow ELF text relocations (DANGEROUS)
- Disable privileged I/O (should not use with XFree86)
- Hide kernel processes
- Allow a user group access to /proc
- Auditing options
    - Set up a single group that is the only one audited
    - Exec logging
    - Log execs within chroot
    - Chdir logging
    - IPC logging
- Trusted path execution
- Socket restrictions
- Sysctl support
- Netfilter Configuration: stealth match support(18)

It is important to carefully read through the documentation for each option. Having a basic understanding of the options will help to prevent incompatibilities between the Grsecurity features and system applications. The “Sysctl support” option is also useful as it allows an administrator to enable and disable some of the Grsecurity options on the fly. This feature is discussed in more depth below.

When the kernel configuration has been completed, the kernel dependencies are built with the command “make dep”, and then the kernel image with “make bzImage”. If any kernel options are enabled as modules, these are built with the command “make

modules".

```
$ make dep
  <output deleted>
$ make bzImage
  <output deleted>
$ make modules
  <output deleted>
```

The next steps are to install the kernel modules (if any), copy the kernel image, System.map and .config files to the /boot directory, and ensure that the boot loader (ie. LILO or GRUB) knows where to find the new image next time the system is booted.

```
$ sudo make modules_install
  <output deleted>
$ sudo cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.22-grsec
$ sudo System.map /boot/System.map-2.4.22-grsec
$ sudo cp .config /boot/config-2.4.22-grsec
```

Make sure that the old kernel image is set as a backup in the boot loader configuration in the event that the new kernel does not boot. At this point, rebooting the system will run the new Grsecurity-enhanced kernel.

Grsecurity Sysctl support

If the Grsecurity "Sysctl support" option was enabled, the Grsecurity options that make use of this feature will be disabled when the system is started. Through the use of the *sysctl* tool, or by echoing "1" to the various files under /proc/sys/kernel/grsecurity, it is possible to enable these options. The options are also disabled with the sysctl tool or by echoing "0" to the appropriate files.

One important sysctl option to note is "grsec_lock". When set to "1", no changes are allowed to any of the Grsecurity sysctl settings – only a system reboot will undo this setting. Once fully configured, modifications to the /etc/sysctl.conf file make it so that the appropriate options are enabled when the system is rebooted in the future.

Since the Grsecurity kernel patch integrates PaX features, the *chpax* utility is also obtained from the PaX site and installed for modifying PaX flags if necessary. The use of sysctl support, and the installation of the *chpax* tool could be considered security risks when used on a production system without the use of ACLs to strictly control access to these facilities.

Access Control Lists

In order to begin configuring Grsecurity access control lists, it is necessary to install the *gradm* tool. The source code for this software is installed as shown below. When running "make install", you will also be prompted to enter a gradm password. If lost, the only way to recover this password is by rebooting into a non-Grsecurity kernel, deleting the file /etc/grsec/pw, and running *gradm -P* to create a new one.

```
$ tar xzf gradm-1.9.12.tar.gz
$ cd gradm
$ make
  <output deleted>
$ sudo make install
  <output deleted>
Setting up grsecurity ACL password:
Password:
Re-enter Password:
Password written to /etc/grsec/pw.
```

After the installation of gradm, a sample ACL file is installed in /etc/grsec/acl. The "debian_secure_acls" directory under the gradm source tree contains some sample ACLs for various system daemons – minor adjustments to the pathnames in these files may be necessary for use on non-Debian systems. These, or other additional ACL files can be included by adding a line:

```
include </path/to/acl>
```

to the main /etc/grsec/acl file. It is also possible to include a whole directory of ACLs by naming the appropriate path.

The Grsecurity ACL Documentation defines the syntax of an ACL as follows:

```
<path of subject process> <optional subject modes> {
      <file object> <optional object modes>
      [+|-]<capability>
      <resource name> <soft limit> <hard limit>
      connect {
            <ip>/<netmask>:<low port>-<high port> <type> <proto>
      }
      bind {
            <ip>/<netmask>:<low port>-<high port> <type> <proto>
      }
}(19)
```

A default ACL must be defined for the root of the file system ("/") – all other ACLs will inherit the settings outlined in the default ACL if they have not been explicitly overridden. When creating ACLs, it is necessary to define absolute paths for each of the subject processes or file objects. Access to a file is granted only when both the file's DAC settings, and Grsecurity ACL system allow it. For a more in-depth look at all the attributes of an ACL entry, see the Grsecurity ACL Documentation.

The goal in configuring the ACL system is to make the default ACL as restrictive as possible. The more restrictive the default ACL, the more the root account functions as a regular user account. Obviously, this is a great undertaking as it requires writing ACLs for all applications, but the long-term benefits of reaching this state far outweigh the time invested in configuring it.

To enable the ACL system, the command *gradm -E* is used. An administrative mode which exempts the current user from ACL restrictions is available by using the command *gradm -a* (requires gradm password). When in administrative mode, it is possible to view all hidden files and processes as well as make modifications to the

ACLs. After making changes to the configuration files, the ACL system is reloaded with the command *gradm -R*. To completely disable the ACL system, *gradm -D* is used.

gradm Learning mode

The *gradm* tool has a learning mode feature that creates ACLs outlining appropriate files and sockets that an application needs to access as well as any other required capabilities or resources. To use the learning mode, a plain ACL is set up for the process with an "l" flag in the subject mode (see the ACL syntax as outlined above). After enabling the ACL system in learning mode, the specified process must be subjected to normal usage so that the learning system can determine the appropriate levels of access to grant. Once this process is complete, the ACL system can be disabled, and the command:

```
gradm -L -O /etc/grsec/newacl
```

will create (or append to if it already exists) the file /etc/grsec/newacl. After reviewing and making any necessary modifications to the new rule-set, it is then used to replace the learning ACL that had been previously set up.

The configuration of any access control system is a time consuming task. The sets of ACLs provided with the source code for the gradm tool help to provide a solid starting point.

Before installing security enhanced kernels on production Linux systems, it is important to read all the documentation and fully test the new features in a development environment. Poor planning and configuration of enhanced access control systems could be detrimental to the overall security of a host.

Conclusions

Hardening a Linux kernel through the use of one or more of these security patches greatly improves the overall security of a system. Patches that reduce the effectiveness of many types of address space attacks can prevent many buffer overflow issues from being exploited to gain access to a system. In the event that an attacker is able to exploit a buggy piece of software, the implementation and proper configuration of a non-discretionary access control system will prevent an attacker from gaining full control of that system. An excellent approach is to implement a combination of patches that use both methods to provide for a highly secure system.

The systems outlined in this paper provide varying levels of security enhancements, but even a combination of the most restrictive kernel patches is not a replacement for proper system maintenance. Software still must be kept up to date with vendor security fixes. Standard application-level hardening techniques must be applied. The physical security of the host must be examined as well as the people who have the authority and ability to access the host. All of these steps are crucial to a defense-in-depth strategy on a Linux host.

Resources

Adamantix. http://www.adamantix.org/ (9 Nov 2003).

Exec Shield. http://people.redhat.com/mingo/exec-shield/ (9 Nov 2003).

Linux Intrusion Detection System. http://www.lids.org/ (9 Nov 2003).

Linux Kernel Archives. http://www.kernel.org/ (9 Nov 2003).

Linux Security Modules. http://lsm.immunix.org/ (9 Nov 2003).

Openwall Project. http://www.openwall.com/ (9 Nov 2003).

PaX Project. http://pageexec.virtualave.net/ (9 Nov 2003).

Rule Set Based Access Control for Linux. http://www.rsbac.org/ (9 Nov 2003).

Security-enhanced Linux. http://www.nsa.gov/selinux/ (9 Nov 2003).

References

Boissiere, G. “Linux Kernel 2.6 Status.” 17 Aug 2003. http://www.kernelnewbies.org/status/ (9 Nov 2003).

Dragovic, B. “LinSec - Linux Security Protection System” Apr 2002. http://www.linsec.org/doc/final/final.html (9 Nov 2003).

Hallyn, S., Kearns., P. “Domain and Type Enforcement for Linux.” 8 Sep 2000 http://www.usenix.org/publications/library/proceedings/als00/2000papers/papers/full_papers/hallyn/hallyn_html/ (9 Nov 2003).

Hatch, B. “An Overview of LIDS.” 17 Oct 2001. http://www.securityfocus.com/infocus/1496 (9 Nov 2003).

Neliben, J. “Buffer Overflows for Dummies.” http://www.sans.org/rr/papers/60/481.pdf (9 Nov 2003).

Pranevich, J. “The Wonderful World of Linux 2.6.” 26 Oct 2003. http://kniggit.net/wwol26.html (9 Nov 2003).

Vasudevan, A. “The Linux Kernel HOWTO.” 27 Oct 2003. http://www.tldp.org/HOWTO/Kernel-HOWTO (9 Nov 2003).

Wreski, D. “Linux 2.4: Next Generation Kernel Security.” 1 Mar 2001. http://www.linuxsecurity.com/feature_stories/kernel-24-security.html (9 Nov 2003).

Quoted References

1. CERT/CC Statistics 1998-2003. 17 Oct 2003. http://www.cert.org/stats/cert_stats.html (9 Nov 2003).

2. Cowan, C., Beattie, S., Kroah-Hartman, G., Pu, C., Wagle, P., and Gligor, V. SubDomain: Parsimonious Server Security. 16 Jan 2002. http://www.usenix.org/events/lisa2000/full_papers/cowan/cowan_html/index.html (9 Nov 2003).

3. Simes. "How to break out of a chroot() jail." 12 May 2002. http://www.bpfh.net/simes/computing/chroot-break.html (9 Nov 2003).

4. Bovet, D., Cesati, M. Understanding the Linux Kernel, 2nd Edition. O'Reilly & Associates, Inc., 2003. [1.4 Basic Operating System Concepts]

5. The Linux Kernel Archives. http://www.kernel.org/ (9 Nov 2003).

6. jbtzhm. "Static Kernel Patching." Phrack 60. 28 Dec 2002. http://www.phrack.org/show.php?p=60&a=8 (9 Nov 2003).

7. "Linux kernel patch from the Openwall Project." http://www.openwall.com/linux/README.shtml (9 Nov 2003).

8. Molnar, Ingo. "Exec Shield." 2 May 2003. http://people.redhat.com/mingo/exec-shield/ANNOUNCE-exec-shield (9 Nov 2003).

9. Documentation for the PaX project: The main document, overall description. http://pageexec.virtualave.net/docs/pax.txt (9 Nov 2003).

10. Documentation for the PaX project: mmap() and mprotect() restrictions. http://pageexec.virtualave.net/docs/mprotect.txt (9 Nov 2003).

11. Documentation for the PaX project: mmap() randomization. http://pageexec.virtualave.net/docs/randmmap.txt (9 Nov 2003).

12. Documentation for the PaX project: non-relocatable executable file randomization. http://pageexec.virtualave.net/docs/randexec.txt (9 Nov 2003).

13. Ott, A. "The Rule Set Based Access Control (RSBAC) Linux Kernel Security Extension." 3 Dec 2001. http://www.rsbac.org/Linux-kongress/index.html (9 Nov 2003).

14. "Security-Enhanced Linux Frequently Asked Questions (FAQ)." http://www.nsa.gov/selinux/faq.html (9 Nov 2003).

15. Wright, C., Cowan, C., Morris, J., Smalley, S., Kroah-Hartman, G. "Linux Security Modules: General Security Support for the Linux Kernel." 3 June 2002. http://lsm.immunix.org/docs/lsm-usenix-2002/html/ (9 Nov 2003).

Quoted References (continued)

16. Smalley, S., Fraser, T., Vance., C. “Linux Security Modules: General Security Hooks for Linux.” http://lsm.immunix.org/docs/overview/linuxsecuritymodule.html (9 Nov 2003).

17. “'Owl', -- a security-enhanced server platform.“ 26 Jan 2003. http://www.openwall.com/Owl/CONCEPTS.shtml (9 Nov 2003).

18. Grsecurity source code. http://www.grsecurity.org/grsecurity-1.9.12-2.4.22.patch (9 Nov 2003).

19. Spengler, B. “Grsecurity ACL Documentation”. 1 Apr 2003. http://www.grsecurity.net/papers.php (9 Nov 2003).

Last Updated: July 25th, 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
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
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 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
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
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
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
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
SANS Copenhagen 2017	Copenhagen, Denmark	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
Community SANS Indianapolis SEC401	Indianapolis, IN	Oct 09, 2017 - Oct 14, 2017	Community SANS