New Old Bugs in the Linux Kernel

By Adam - March 12, 2021

Introduction

Dusting off a few new (old) vulns

Have you ever been casually perusing the source code of the Linux kernel and thought to yourself "Wait a minute, that can't be right"? That's the position we found ourselves in when we found three bugs in a forgotten corner of the mainline Linux kernel that turned out to be about 15 years old. Unlike most things that we find gathering dust, these bugs turned out to still be good, and one turned out to be useable as a Local Privilege Escalation (LPE) in multiple Linux environments.

Who you calling SCSI?

The particular subsystem in question is the SCSI (Small Computer System Interface) data transport, which is a standard for transferring data made for connecting computers with peripheral devices, originally via a physical cable, like hard drives. SCSI is a venerable standard originally published in 1986 and was the go-to for server setups, and iSCSI is basically SCSI over TCP. SCSI is still in use today, especially if you're dealing with certain storage situations, but how does this become an attack surface on a default Linux system?

Through the magic of extensive package dependencies, rdma-core is one of the packages that ends up being installed in any of the RHEL or CentOS base environments that include a GUI (Workstation, Server with GUI, and Virtualization Host), as well as Fedora Workstations. Additionally, rdma-core could be installed on other distributions, including Ubuntu and Debian, due to its use as a dependency for many other packages (see Figure 1). On Ubuntu Server 18.04 LTS and earlier, this was installed by default.

cockpit-machines	libguestfs-gobject-devel	librdmacm-utils	openmpi	virt-v2v
dpdk	libguestfs-inspect-icons	librpmem	openmpi-devel	
dpdk-devel	libguestfs-java	librpmem-devel	opensm	
dpdk-tools	libguestfs-java-devel	libvirt	opensm-libs	
fabtests	libguestfs-javadoc	libvirt-daemon-driver-storage	pcp-pmda-infiniband	
fence-virtd-libvirt	libguestfs-man-pages-ja	libvirt-daemon-driver-storage-iscsi-direct	perftest	
fence-virtd-serial	libguestfs-man-pages-uk	libvirt-daemon-kvm	perl-Sys-Guestfs	
fio	libguestfs-rescue	libvma	python3-libguestfs	
glusterfs-rdma	libguestfs-rsync	lorax-lmc-virt	qemu-kvm	
gnome-boxes	libguestfs-tools	lua-guestfs	qemu-kvm-block-iscsi	
ibacm	libguestfs-tools-c	mpitests-mvapich2	qemu-kvm-core	
infiniband-diags	libguestfs-winsupport	mpitests-mvapich2-psm2	qperf	
iwpmd	libguestfs-xfs	mpitests-openmpi	rdma-core-devel	
libfabric	libibumad	mvapich2	rpmemd	
libguestfs	libibverbs	mvapich2-psm2	ruby-libguestfs	
libguestfs-bash-completion	libibverbs-utils	opa-address-resolution	srp_daemon	
libguestfs-benchmarking	libiscsi	opa-basic-tools	systemtap-runtime-virthost	
libguestfs-devel	libiscsi-devel	opa-fastfabric	ucx	
libguestfs-gfs2	libiscsi-utils	opa-fm	virt-dib	
libguestfs-gobject	librdmacm	opa-libopamgt	virt-p2v-maker	

List of packages that depend on rdma-core on a default install of RHEL 8.3

RDMA (Remote Direct Memory Access) is a technology for high-throughput, low-latency networking whose application to data transfer and storage seems obvious. There are several implementations, but the only one that's relevant in the current discussion is Infiniband, found in the ib_iser kernel module.

If you're thinking "wait, is all of this just automatically up and running even if I don't use SCSI or iSCSI?", that's great because that line of questioning would lead to you to the concept of on-demand kernel module loading and an attack vector that's been around for a long time.

Automatic Module Loading, a.k.a. On-demand crufty kernel code

In an effort to be helpful and improve compatibility, the Linux kernel can load kernel modules on-demand if particular code notices some functionality is needed and can be loaded, like support for uncommon protocol families. This is helpful, but it also opens up the attack surface for local attackers because it allows unprivileged users to load obscure kernel modules which they can then exploit. Public knowledge of this has been around for over a decade, with grsecurity's GRKERNSEC_MODHARDEN being introduced as a defense against this sort of thing in 2009. Jon Oberheide's tongue-in-cheek named public exploit in 2010 made it plain that this was an issue, though Dan Rosenberg's suggested patch in 2010 implementing a sysctl option wasn't adopted. There have been other mitigations that have become available in the last few years, but support for them varies between Linux distributions.

So you're telling me there are bugs?

One of the nice parts about having source code is you could just be scanning visually and certain things might pop out and pique your interest, making you want to pull the thread on them. The first bug we found is like that. Seeing a bare sprintf means a buffer copy with no length, which means its game over if it's attacker-controlled data and no previous length validation (see Figure 2). So while there's a fair number of indirections and macros, and you have to jump through a number of files to follow the thread of execution, the main bug is a simple buffer overflow because sprintf was used. This is also an indication of a lack of security-conscious programming practices that was prevalent at the time this code was developed, as the other bugs show.

```
int iscsi_session_get_param(struct iscsi_cls_session *cls_session,
                enum iscsi_param param, char *buf)
{
    struct iscsi_session *session = cls_session->dd_data;
    int len;
                                        No length parameter for buf.
                                        That can't possibly be correct.
    switch(param) {
    ...
    case ISCSI_PARAM_USERNAME:
        len = sprintf(buf, "%s\n", session->username);
        break;
    case ISCSI_PARAM_USERNAME_IN:
        len = sprintf(buf, "%s\n", session->username_in);
        break;
    case ISCSI_PARAM_PASSWORD:
        len = sprintf(buf, "%s\n", session->password);
        break;
    case ISCSI_PARAM_PASSWORD_IN:
        len = sprintf(buf, "%s\n", session->password_in);
        break;
    case ISCSI_PARAM_IFACE_NAME:
        len = sprintf(buf, "%s\n", session->ifacename);
        break;
    case ISCSI_PARAM_INITIATOR_NAME:
        len = sprintf(buf, "%s\n", session->initiatorname);
        break;
    case ISCSI_PARAM_BOOT_ROOT:
        len = sprintf(buf, "%s\n", session->boot_root);
        break;
    case ISCSI_PARAM_BOOT_NIC:
        len = sprintf(buf, "%s\n", session->boot_nic);
        break;
    case ISCSI_PARAM_BOOT_TARGET:
        len = sprintf(buf, "%s\n", session->boot_target);
        break;
```

Overflows, overflows everywhere

The second bug is similar: using a kernel address as a handle. Clearly a leftover from when there wasn't any effort to keep the kernel from leaking pointers, but nowadays it's a handy Kernel Address Space Layout Randomization (KASLR) bypass, as it points to a structure full of pointers. And the final bug is simply failure to validate data from userland, which is a classic kernel programming issue.

Bug Identification

Linux Kernel Heap Buffer Overflow

- Vulnerability Type: Heap Buffer Overflow
- Location: iscsi_host_get_param() in `drivers/scsi/libiscsi.c`
- Affected Versions: Tested on RHEL 8.1, 8.2, and 8.3
- Impact: LPE, Information Leak, Denial of Service (DoS)
- CVE Number: CVE-2021-27365

The first vulnerability is a heap buffer overflow in the iSCSI subsystem. The vulnerability is triggered by setting an iSCSI string attribute to a value larger than one page, and then trying to read it. Internally, a sprintf call (line 3397 in `drivers/scsi/libiscsi.c` in the kernel-4.18.0-240.el8 source code) is used on the user-supplied value with a buffer of a single page that is used for the seq file that backs the iscsi attribute. More specifically, an unprivileged user can send netlink messages to the iSCSI subsystem (in `drivers/scsi/scsi_transport_iscsi.c`) which sets attributes related to the iSCSI connection, such as hostname, username, etc, via the helper functions in `drivers/scsi/libiscsi.c`. These attributes are only limited in size by the maximum length of a netlink message (either 2**32 or 2**16 depending on the specific code processing the message). The sysfs and seqfs subsystem can then be used to read these attributes, however it will only allocate a buffer of PAGE_SIZE (single_open in `fs/seq_file.c`, called when the sysfs file is opened). This bug was first introduced in 2006 (see `drivers/scsi/libiscsi.c`, commits a54a52caad and fd7255f51a) when the iSCSI subsystem was being developed. However, the kstrdup/sprintf pattern used in the bug has been expanded to cover a larger number of fields since the initial commit.

Linux Kernel Pointer Leak to Userspace

- Vulnerability Type: Kernel Pointer Leak
- Location: show_transport_handle() in drivers/scsi/scsi_transport_iscsi.c
- Affected Versions: Tested on RHEL 8.1, 8.2, and 8.3
- Impact: Information Leak
- CVE Number: CVE-2021-27363

In addition to the heap overflow vulnerability, GRIMM discovered a kernel pointer leak that can be used to determine the address of the iscsi_transport structure. When an iSCSI transport is registered with the iSCSI subsystem, the transport's "handle" is available to unprivileged users via the sysfs file system, at `/sys/class/iscsi_transport/$TRANSPORT_NAME/handle`. When read, the show_transport_handle function (in `drivers/scsi/scsi_transport_iscsi.c`) is called, which leaks the handle. This handle is actually the pointer to an iscsi_transport struct in the kernel module's global variables.

Linux Kernel Out-of-Bounds Read

- Vulnerability Type: Out-of-Bounds Read
- Location: iscsi_if_recv_msg() in drivers/scsi/scsi_transport_iscsi.c
- Affected Versions: Tested on RHEL 8.1, 8.2, and 8.3
- Impact: Information Leak, DoS
- CVE Number: CVE-2021-27364

The final vulnerability is an out-of-bounds kernel read in the libiscsi module (`drivers/scsi/libiscsi.c`). This bug is triggered via a call to send_pdu (lines 3747-3750 in `drivers/scsi/scsi_transport_iscsi.c` in the kernel-4.18.0-240.el8 source code). Similar to the first vulnerability, an unprivileged user can craft netlink messages that specify buffer sizes that the driver fails to validate, causing a controllable out-of-bounds read. There are multiple user-controlled values that are not validated, including the calculation of the size of the preceding header, allowing for a read of up to 8192 bytes at a controllable 32-bit offset from the original heap buffer.

Technical Analysis

Exploit

GRIMM developed a Proof of Concept (PoC) exploit that demonstrates the use of the first two vulnerabilities.

In its current state, the PoC has support for the `4.18.0-147.8.1.el8_1.x86_64`, `4.18.0-193.14.3.el8_2.x86_64`, and `4.18.0-240.el8.x86_64` releases of the Linux kernel. Other versions of the Linux kernel are also vulnerable, but symbol addresses and structure offsets will need to be gathered before they can be exploited. See the `symbols.c`, `symbols.h`, and `utilities/get_symbols.sh` script for a semi-automated symbol gathering approach. Testing was performed against RHEL 8.1 through 8.3, but other Linux distributions using the same underlying kernel images are vulnerable as well.

KASLR Leak

Before we can start modifying kernel structures and changing function pointers, we've got to bypass KASLR. Linux randomizes the base address of the kernel to hinder the exploitation process. However, due to numerous sources of local information leak, KASLR can often be bypassed by a local user. This exploit is no exception, as it includes two separate information leaks which enable it to bypass KASLR.

The first information leak comes from a non-null terminated heap buffer. When an iSCSI string attribute is set via the `iscsi_switch_str_param` function (shown below), the `kstrdup` function is called on the user provided input (in `new_val_buf`). However, the buffer containing the user input is not initialized upon allocation, and the kernel does not enforce that the user's input is NULL terminated. As a result, the `kstrdup` function will copy any non-NULL bytes after the client input, which can later be retrieved by reading back the attribute. The exploit abuses this information leak by specifying a string of 656 bytes, which results in the address of the `netlink_sock_destruct` being included in the `kstrdup`'ed string. Later, the exploit reads back the attribute set here, and obtains the address of this function. By then subtracting off the base address of the `netlink_sock_destruct`, the exploit can calculate the kernel slide. The allocation which sets the `netlink_sock_destruct` function pointer is performed in the `__netlink_create` function (`net/netlink/af_netlink.c`) as a natural side effect of sending a netlink message.

```
int iscsi_switch_str_param(char **param, char *new_val_buf)
{
  char *new_val;

  if (*param) {
    if (!strcmp(*param, new_val_buf))
      return 0;
  }

  new_val = kstrdup(new_val_buf, GFP_NOIO);
  if (!new_val)
    return -ENOMEM;

  kfree(*param);
  *param = new_val;
  return 0;
}
```

The second information leak obtains the address of the target module's `iscsi_transport` structure via the second vulnerability. This structure defines the transport's operations, i.e. how it handles each of the various iSCSI requests. As this structure is within the target kernel module's global region, we can use this information leak to obtain the address of its kernel module (and thus any other variables within it).

Obtaining a Kernel Write Primitive

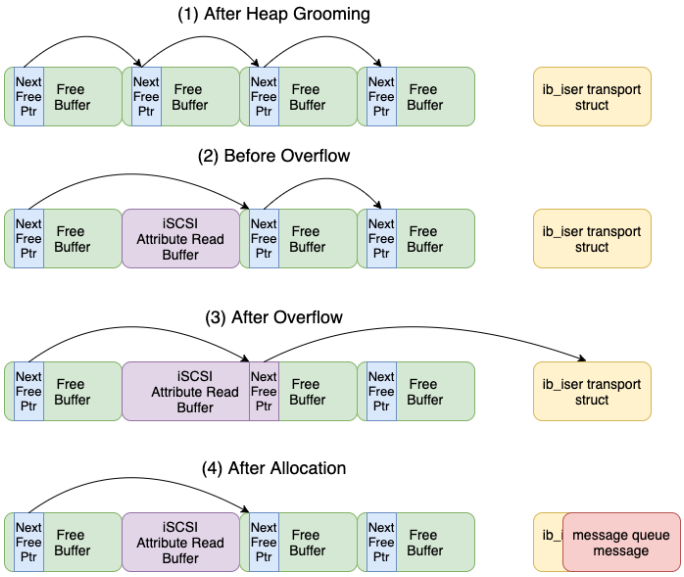
The Linux kernel heap's SLUB allocator maintains a cache of objects of the same size (in powers of 2). Each free object in this cache contains a pointer to the next free item in the cache (at offset 0 in the free item). The exploit uses the heap overflow to modify the freelist pointer at the beginning of the adjacent slab. By redirecting this

pointer, we can choose where the kernel will make an allocation. By carefully choosing the allocation location and controlling the allocations in this cache, the exploit obtains a limited kernel write primitive. As explained in the next section, the exploit uses this controlled write to modify the iscsi_transport struct.

In order to obtain the desired heap layout, the exploit utilizes heap grooming via POSIX message queue messages. The exploit operates on the 4096 kmalloc cache, which is a comparatively low-traffic cache. As such, abusing the freelist to redirect it to an arbitrary location is unlikely to cause issues. Message queue messages were selected as the ideal heap grooming allocation as they can be easily allocated/freed from userland, and their contents and size can be mostly controlled from userland.

The exploit takes the following steps to obtain a kernel write primitive from the heap overflow:

1. Send a large number of message queue messages to ourselves, but do not receive them. This will cause the kernel to create the associated message queue structures in the kernel, flooding the 4096 kmalloc cache.
2. Receive a number of the message queue messages. This will cause the kernel to free the kernel message queue structures. If successful, the resulting 4096 kmalloc cache will contain a number of free items in a row, such as shown in the image (1) below.
3. The exploit triggers the overflow. The overflow allocation will take one of the cache's free entries and overflow the next freelist pointer for the adjacent free item, as shown in (2) and (3) in the image below. The exploit uses the overflow to redirect the next freelist pointer to point at the `ib_iser` module's `iscsi_transport` struct.
4. After the overflow, the exploit sends more message queue messages in order to reallocate the modified free item. The kernel will traverse the freelist and return an allocation at our modified freelist pointer, as shown in (4) in the image below.



Heap Grooming and Overflow Cache Layout

While this approach does allow the exploit to write to kernel memory at a controlled location, it does have a few caveats. The selected location must start with a NULL pointer. Otherwise, the item's allocation will attempt to link the pointer at this value into the freelist. Subsequent allocations will then use this pointer causing memory corruption and likely a kernel panic. Additionally, the kernel message queue structures include a header of 0x30 bytes before the user controlled message body. As such, the exploit cannot control first 0x30 bytes that it writes.

Target Selection and Exploitation

With KASLR bypassed and the ability to write to arbitrary content to kernel memory, the next task is to use these primitives to obtain stronger kernel read/write primitives and escalate privileges. In order to do this, the exploit aims the arbitrary write at the `ib_iser` module's `iscsi_transport` structure. As shown below, this structure contains a number of iSCSI netlink message handling function pointers, which the iSCSI subsystem calls with partially user controlled parameters. By modifying these function pointers, the exploit can call arbitrary functions with a few user controlled parameters.

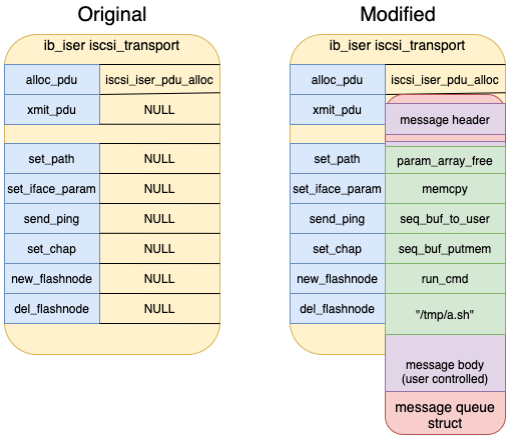
```
struct iscsi_transport {
  ...
  int (*alloc_pdu) (struct iscsi_task *task, uint8_t opcode);
  int (*xmit_pdu) (struct iscsi_task *task);
  ...
  int (*set_path) (struct Scsi_Host *shost, struct iscsi_path *params);
  int (*set_iface_param) (struct Scsi_Host *shost, void *data,
    uint32_t len);
  int (*send_ping) (struct Scsi_Host *shost, uint32_t iface_num,
    uint32_t iface_type, uint32_t payload_size,
    uint32_t pid, struct sockaddr *dst_addr);
  int (*set_chap) (struct Scsi_Host *shost, void *data, int len);
  int (*new_flashnode) (struct Scsi_Host *shost, const char *buf,
    int len);
  int (*del_flashnode) (struct iscsi_bus_flash_session *fnode_sess);
```

```
  ...
};
```

Obtaining a Stable Kernel Read/Write Primitive

The exploit modifies the `iscsi_transport` struct function pointers as shown in the below figure. After the modifications, the `send_ping` and `set_chap` function pointers point to the `seq_buf_to_user` and `seq_buf_putmem` functions. The exploit uses these functions to obtain a stable kernel read and write primitive. These functions take a `seq_buf` structure, buffer, and length as parameters, and then read or write to kernel memory respectively. The passed in `seq_buf` defines where these functions will read from memory or write to memory. However, as can be seen in the struct definition above, the overwritten function pointers will pass a `Scsi_Host` struct as the first parameter, rather than the `seq_buf` that our exploit needs to provide. A simple workaround for this issue is to modify the `Scsi_Host` struct associated with our connection to resemble a `seq_buf` struct. As the beginning of the `Scsi_Host` struct does not contain any necessary values for the exploited functionality, the exploit uses the modified `set_iface_param` function pointer to call `memcpy` and overwrite the `Scsi_Host` struct. Afterwards, the `seq_buf_to_user` and `seq_buf_putmem` functions can be called to read or write kernel memory.

The exploit then uses the kernel read and write primitives to remove the overlapping freelist region from the 4096 kmalloc cache and fix some memory corruption that occurred as result of the overlapping allocations.



The `ib_iser` module's `iscsi_transport` before and after the exploit's modifications

Privilege Escalation

With the ability to call arbitrary functions and read/write kernel memory, root privileges can be obtained in several different ways. The exploit obtains privileged code execution by calling the kernel `run_cmd` function via a chained call to `param_array_free`. This kernel function takes a command to run and executes it as root in the `kernel_t` SELinux context. The exploit calls this function with a pointer to the `/tmp/a.sh` string in the `iscsi_transport` struct, causing it to run the post-escalation payload. As neither the privileged escalation or kernel read/write primitives directly dereference or execute memory in userland from the kernel, the exploit is able to bypass Supervisor Mode Execution Prevention (SMEP), Supervisor Mode Access Prevention (SMAP), and Kernel Page Table Isolation (KPTI).

While this exploit works on some Linux distributions, its technique will not work on others. Other Linux distributions protect the free list pointers in the heap to prevent exploitation (`CONFIG_SLAB_FREELIST_HARDENED`). This exploit takes advantage of the fact that some distributions do not include this config option. The exploit will need to be reworked using an alternative technique for this bug to be used to exploit a Linux distribution with this option enabled.

Testing

To test the provided PoC exploit, ensure you have a compatible RHEL and kernel version. You can check your release version with cat `/etc/redhat-release` and kernel version with uname -r, but the exploit will also detect and warn if you are on an unsupported version. Install the required packages and build the exploit, then copy the post-escalation script and run the exploit:

```
$ sudo yum install -y make gcc
$ make
$ cp a.sh /tmp/
$ chmod +x /tmp/a.sh
$ ./exploit
```

The exploit concludes by running a script at a hardcoded location (`/tmp/a.sh`) as root. The repository includes a demonstration script which creates a file owned by root in `/tmp`. The exploit is not 100% reliable, so may require multiple runs. Possible error conditions include warnings such as "Failed to detect kernel slide" and "Failed to overwrite iscsi_transport struct (read 0x0)", and kernel panics. Output from a successful run is shown below:

```
$ ./exploit
Got iscsi iser transport handle 0xffffffffc0e1a040
KERNEL_BASE=0xffffffff98600000
Setting up target heap buffer
Triggering overflow
Allocating controlled objects
Cleaning up
```

```
Running escalation payload
Success
```

Due to the way the PoC is written, additional attempts to run either the same PoC or the send_pdu_oob PoC will fail gracefully (until the system is restarted).

```
$ ls -l /tmp/proof
-rwsrwxrwx. 1 root root 14 Jan 14 10:09 /tmp/proof
```

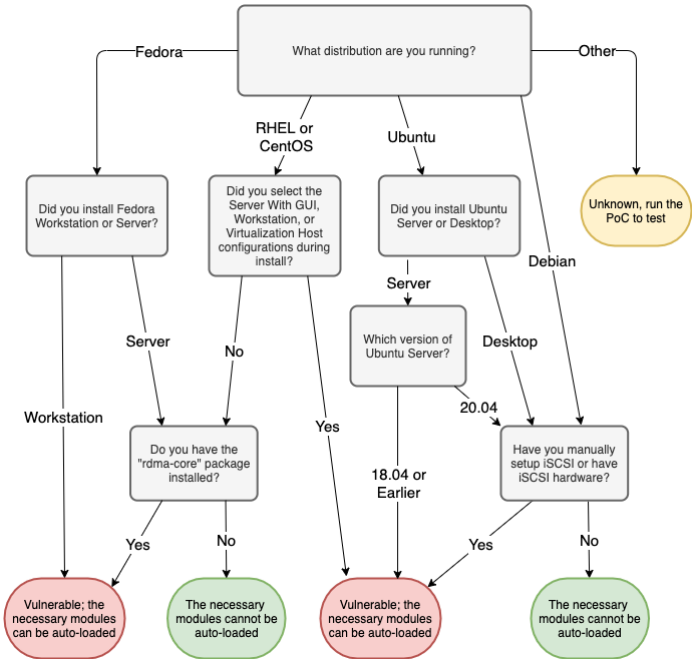
A PoC for the third vulnerability was developed as well. The send_pdu_oob PoC supplies a static large offset which causes a wild read into (likely) unmapped kernel memory, which causes the kernel to panic. The compiled PoC should be run with root privileges. Since the size and offset of the data read are both controlled, this PoC could be adapted to create an information leak by only leaking a small amount of information beyond the original heap buffer. The primary restriction on the usefulness of this bug is that the leaked data is not returned directly to the user, but is stored in another buffer to be sent as part of a later operation. It's likely an attacker could design a dummy iSCSI target to receive the leaked information, but that is beyond the scope of this PoC.

Impact

Due to the non-deterministic nature of heap overflows, the first vulnerability could be used as an unreliable, local DoS. However, when combined with an information leak, this vulnerability can be further exploited as a LPE that allows an attacker to escalate from an unprivileged user account to root. A separate information leak is not necessary, though, since this vulnerability can be used to leak kernel memory as well. The second vulnerability (kernel pointer leak) is less impactful and could only serve as a potential information leak. Similarly, the third vulnerability (out-of-bounds read) is also limited to functioning as a potential information leak or even an unreliable local DoS.

Affected Systems

In order for these bugs to be exposed to userland, the scsi_transport_iscsi kernel module must be loaded. This module is automatically loaded when a socket call that creates a `NETLINK_ISCSI` socket is performed. Additionally, at least one iSCSI transport must be registered with the iSCSI subsystem. The ib_iser transport module will be loaded automatically in some configurations when an unprivileged user creates a `NETLINK_RDMA` socket. Figure 5 shows a flowchart which helps to determine if you are impacted by this vulnerability.



Impact Flowchart

On CentOS 8, RHEL 8, and Fedora systems, unprivileged users can automatically load the required modules if the rdma-core package is installed. This package is a dependency of several popular packages, and as such is included in a large number of systems. The following CentOS 8 and RHEL 8 Base Environments include this package in their initial installation:

- Server with Graphical User Interface (GUI)
- Workstation
- Virtualized Host

The following CentOS 8 and RHEL 8 Base Environments do NOT include this package in their initial installation, but it can be installed afterwards via yum:

- Server
- Minimal Install
- Custom OS

The package is installed in the base install of Fedora 31 Workstation, but not Fedora 31 Server.

On Debian and Ubuntu systems, the rdma-core package will only automatically load the two required kernel modules if the RDMA hardware is available. As such, the vulnerability is much more limited in scope.

Conclusions

The vulnerabilities discussed above are from a very old driver in the Linux kernel. This driver became more visible due to a fairly new technology (RDMA) and default behavior based on compatibility instead of risk. The Linux kernel loads modules either because new hardware is detected or because a kernel function detects that a module is missing. The latter implicit autoload case is more likely to be abused and is easily triggered by an attacker, enabling them to increase the attack surface of the kernel.

In the context of these specific vulnerabilities, the presence of loaded kernel modules relating to the iSCSI subsystem on machines that don't have attached iSCSI devices is a potential indicator of compromise. An even greater indicator is the presence of the following log message in a host's system logs:

```
localhost kernel: fill_read_buffer: dev_attr_show+0x0/0x40 returned bad count
```

While this message does not guarantee that the vulnerabilities described in this report have been exploited, it does indicate that some kind of buffer overflow has occurred.

This risk vector has been known for years, and there are several defenses against module autoloading including grsecurity's MODHARDEN and the eventual mainstream implementation of modules_autoload_mode, the modules_disabled sysctl variable, distributions blacklisting particular protocol families, and the use of machine-specific module blacklists. These are separate options that can be used in a defense-in-depth strategy, where server administrators and developers can take both reactive and proactive measures.

The bottom line is that this is still a real problem area for the Linux kernel because of the tension between compatibility and security. Administrators and operators need to understand the risks, their defensive options, and how to apply those options in order to effectively protect their systems.

Timeline

- 02/17/2021 - Notified Linux Security Team
- 02/17/2021 - Applied for and received CVE numbers
- 03/07/2021 - Patches became available in mainline Linux kernel
- 03/12/2021 - Public disclosure (NotQuite0DayFriday)

GRIMM's Private Vulnerability Disclosure (PVD) Program

GRIMM's Private Vulnerability Disclosure (PVD) program is a subscription-based vulnerability intelligence feed. This high-impact feed serves as a direct pipeline from GRIMM's vulnerability researchers to its subscribers, facilitating the delivery of actionable intelligence on 0-day threats as they are discovered by GRIMM. We created the PVD program to allow defenders to get ahead of the curve, rather than always having to react to events outside of their control.

The goal of this program is to provide value to subscribers in the following forms:

- Advanced notice of 0-days prior to public disclosure. This affords subscribers time to get mitigations in place before the information is publicly available.
- In-depth, technical documentation of each vulnerability.
- PoC vulnerability exploitation code for:
    - Verifying specific configurations are vulnerable
    - Testing defenses to determine their effectiveness in practice
    - Training
        - Blue teams on writing robust mitigations and detections
        - Red teams on the art of exploitation
- A list of any indicators of compromise
- A list of actionable mitigations that can be put in place to reduce the risk associated with each vulnerability.

The research is done entirely by GRIMM, and the software and hardware selected by us is based on extensive threat modeling and our team's deep background in reverse engineering and vulnerability research. Requests to look into specific software or hardware are welcome, however we can not guarantee the priority of such requests. In addition to publishing our research to subscribers, GRIMM also privately discloses each vulnerability to its corresponding vendor(s) in an effort to help patch the underlying issues.

If interested in getting more information about the PVD program, reach out to pvd@grimm-co.com.

Working with GRIMM

Want to join us and perform more analyses like this? We're hiring. Need help finding or analyzing your bugs? Feel free to contact us.

APPSEC EXPLOIT LINUX VULNERABILITY VULNERABILITY RESEARCH

Popular posts from this blog

No Hardware, No Problem: Emulation and Exploitation

By Unknown - April 22, 2022

Vulnerability Hunting for Sport If you've been following our blog, you might notice some favoritism when it comes to embedded targets... We've been exploring the NETGEAR R7000 for several blog posts . This pattern stems from a number of product characteristics, ...

READ MORE

SOHO Device Exploitation

By Adam - June 15, 2020

Netgear R7000 SOHO Device Exploitation After a long day of hard research, it's fun to relax, kick back, and do something easy. While modern software development processes have vastly improved the quality of commercial software as compared to 10-15 years ago, consumer netwoı ...

READ MORE

Archive

Labels

GRIMM

Careers with GRIMM
GRIMM GitHub
GRIMM listing at Carahsoft
GRIMM Website
GRIMM's Cyber Maturity Builder