Physical Security Attack: Bypassing Linux Authentication via Boot Parameters

Summary

This technical write-up demonstrates a critical physical security measure that is often overlooked or neglected in many default Linux installations, including Kali Linux. By modifying boot parameters through the bootloader, an attacker with physical access can obtain root shell access without authentication, completely bypassing the operating system's login mechanisms.

Background Conext: The Initial Problem

While working with a Kali Linux virtual machine in VirtualBox, I encountered a storage saturation issue that caused the VM to crash (less than 1GB left on Virtual Disk). Upon restart, the kernel logged errors indicating insufficient resources to complete the boot process, preventing access to the standard login screen (`It was at this moment, he knew he...` moment lol).

The kernel output looked similar to:

```
[FAILED] Failed to start Load Kernel Modules
kernel: blk_update_request: I/O error, dev sda, sector XXXXX
kernel: EXT4-fs error (device sda1): ext4_find_entry: reading directory
Out of memory: Kill process 1234 (systemd)
```

After consulting my assistant (ChatGPT lol) in order to resolve this, I needed to:

1. Verify the disk's encryption status and file system integrity
2. Expand the virtual disk using VirtualBox's CLI tools (`VBoxManage modifyhd`)
3. Boot into a recovery environment (GParted) to resize the partition

However, during troubleshooting, I discovered a significant security implication worth documenting.

The Vulnerability: Boot Parameter Manipulation

What Are Boot Parameters?

Boot parameters are arguments passed to the Linux kernel during the boot process. They control various aspects of system initialization, including:

- Hardware detection and driver loading
- Root filesystem location and mounting options
- Init system behavior
- Debug and recovery modes

The bootloader (typically GRUB2 in most Linux distributions) allows users with physical access to temporarily modify these parameters before the kernel loads.

The Attack Vector

By changing a single boot parameter, specifically adding `init=/bin/bash`, an attacker can redirect the initialization process to spawn a root shell instead of following the normal boot sequence.

Normal boot process:

```
Bootloader → Kernel → Init System (systemd/init) → Login Manager → User Authentication
```

Modified boot process:

```
Bootloader → Kernel → /bin/bash (root shell) [Authentication bypassed]
```

Step-by-Step Exploitation

Prerequisites

- **Physical access** to the target machine (or console access for VMs/servers)
- Ability to interact with the bootloader (unencrypted boot partition)
- Target system without boot parameter restrictions

Attack Procedure

Step 1: Access the Bootloader Menu

1. Power on or reboot the target system
2. During the initial boot phase, access the bootloader menu:
    - **GRUB2 (most Linux distributions):** Press `Esc` or hold `Shift` during boot
    - **Kali Linux/Debian-based:** Often shows menu by default or use `Esc`

You should see a menu listing available kernel versions and boot options.

Step 2: Edit Boot Parameters

1. Highlight the default boot entry
2. Press `e` to edit the boot parameters
3. Navigate to the line starting with `linux` or `linux16` (contains kernel path and parameters)
4. This line typically looks similar to:

```
linux /boot/vmlinuz-5.x.x-kali root=UUID=xxxx ro quiet splash
```

Step 3: Modify the Init Parameter

1. Move to the end of the kernel command line

> `[Optional]` Remove parameters that might interfere:
>
> - Remove `quiet` and `splash` for verbose output
> - Change `ro` (read-only) to `rw` (read-write) for immediate write access

2. Append the critical parameter at the end:

```
init=/bin/bash
```

Step 4: Boot with Modified Parameters

1. Press `Ctrl + X` or `F10` to boot with the modified parameters
2. The system will bypass all normal initialization processes
3. After kernel loading completes, you'll be presented with a root shell prompt:

```
root@(none):/#
```

You now have root access without providing any credentials.

Why This Works: Technical Explanation

The Init Parameter

The `init` parameter tells the kernel which program to execute as the first user-space process (PID 1). Normally, this is `/sbin/init` (or `/lib/systemd/systemd` on systemd-based systems), which:

1. Mounts filesystems according to `/etc/fstab`
2. Starts system services
3. Launches the login manager
4. Enforces authentication

By specifying `init=/bin/bash`, we:

- Instruct the kernel to run Bash as PID 1
- Skip the entire init system and all its security checks
- Inherit root privileges (init always runs as root)
- Bypass authentication mechanisms entirely

The Initramfs Factor

Important clarification: When using `init=/bin/bash`, you're running bash from within the **initramfs** (initial RAM filesystem), not from the actual root partition. The initramfs is:

- Always unencrypted (it contains the tools to decrypt LUKS volumes)
- Loaded directly by the bootloader into memory
- Contains a minimal Linux environment (BusyBox utilities)

This means:

- Even with full-disk LUKS encryption, you still get a root shell
- The encrypted root filesystem hasn't been mounted yet
- You're in a limited environment but with root privileges
- You have access to critical tools: `cryptsetup`, `lvm`, `fsck`, etc.

What an attacker can do from initramfs:

1. **Dump the LUKS header** for offline cracking:

```
dd if=/dev/sda3 of=/external_usb/luks_header.bin bs=512 count=10000
```

2. **Modify the initramfs** itself to add backdoors

3. **If they know the passphrase** (or crack it), manually unlock and access everything:

```
cryptsetup luksOpen /dev/sda3 cryptroot
vgchange -ay
mount /dev/mapper/vg-root /mnt
# Now they have full access to your encrypted data
```

Real-World Impact – Why This Matters

Vulnerability Scope

This technique works on:

- Most Linux distributions (Debian, Ubuntu, Fedora, CentOS, Arch, Kali, etc.)
- Systems with default bootloader configurations (most personal Laptops are)
- Both encrypted and unencrypted installations (with different impacts)
- Virtual machines with console access
- Physical servers with KVM/iLO/iDRAC access

Time Required

An experienced attacker can complete this exploit in under **30 seconds** (literally using AI lol):

Mitigations and Hardening Measures

This attack vector exists in default configurations, but several security measures can prevent or detect it:

1. **GRUB Bootloader Password Protection** (Quick Win)

Set a password for editing boot parameters:

```
# Generate password hash
grub-mkpasswd-pbkdf2
# Enter password twice
# Copy the generated hash starting with "grub.pbkdf2.sha512..."

# Edit /etc/grub.d/40_custom
sudo nano /etc/grub.d/40_custom

# Add these lines:
set superusers="admin"
password_pbkdf2 admin grub.pbkdf2.sha512.10000.HASH_HERE

# Update GRUB configuration
sudo update-grub    # Debian/Ubuntu
sudo grub2-mkconfig -o /boot/grub2/grub.cfg    # RHEL/Fedora/CentOS
```

Result: Users can still select and boot OS entries normally, but editing boot parameters (`e` key) or accessing GRUB command line (`c` key) requires the password.

Note: This only protects GRUB editing, not booting itself. To require password for booting, add `--users ""` to menuentry lines.

2. **Full Disk Encryption (FDE)**

Encrypt the entire disk including root partition using LUKS:

```
# During installation, enable full disk encryption
# This protects data at rest but NOT from:
# - LUKS header extraction
# - Weak passphrase attacks
# - Evil maid attacks without Secure Boot
```

Important: FDE alone is insufficient. An attacker can still:

- Extract the LUKS header via `init=/bin/bash`
- Perform offline brute-force attacks on weak passphrases
- Install a malicious bootloader (evil maid attack)

3. **Secure Boot + TPM (Modern Best Practice)**

Enable UEFI Secure Boot with TPM integration:

- Verifies bootloader and kernel signatures
- Prevents loading of unsigned/modified kernels

- TPM can store encryption keys, auto-unsealing only if boot chain is unmodified

Setup:

1. Enable Secure Boot in UEFI firmware
2. Configure LUKS with TPM auto-unlock (systemd-cryptenroll)
3. If boot chain is tampered with, TPM won't release the key

4. **BIOS/UEFI Firmware Password**

Set a password in BIOS/UEFI firmware to:

- Prevent booting from external media (USB, CD, network)
- Restrict boot device order changes
- Lock firmware settings
- Require password to access BIOS setup

Warning: BIOS passwords can sometimes be bypassed via:

- Removing CMOS battery
- Using manufacturer backdoor passwords
- Hardware attacks on the SPI flash chip

5. **Physical Security Controls**

- Lock server rooms and data centers with access control systems
- Use cable locks for laptops (Kensington locks)
- Implement physical access logs and video monitoring
- Use tamper-evident seals on server chassis
- Deploy chassis intrusion detection switches
- Implement clean desk policies for sensitive areas

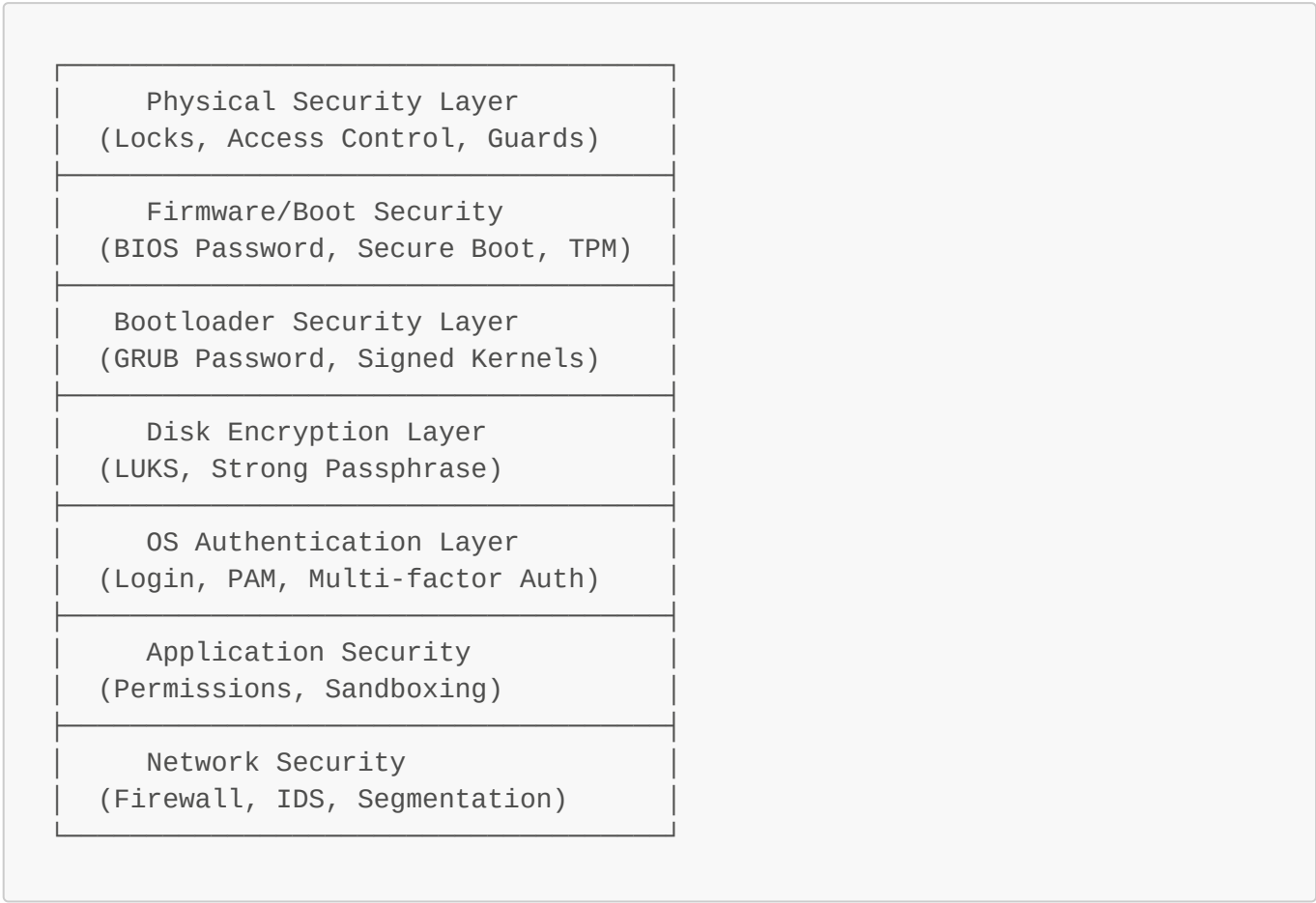
6. **Boot Integrity Monitoring**

Configure systems to detect boot tampering:

- Use TPM for measured boot (records PCR values)
- Implement remote attestation to verify boot integrity
- Log boot parameter changes
- Alert on unexpected system reboots

Defense in Depth Approach

Physical security should be part of a comprehensive security strategy:



No single layer is sufficient; each layer compensates for weaknesses in others.

Log Analysis Challenges

The attack may leave minimal -digital- traces:

- No authentication failures (authentication was bypassed)
- Possible boot-time irregularities in system logs
- Modified file timestamps if attacker changed configuration files
- No network connection established (unless attacker configured it)

Important: An experienced attacker will:

- Clean logs after making changes
- Modify file timestamps to hide modifications
- Use memory-only backdoors that don't touch disk

Conclusion

This demonstration reveals a fundamental principle in information security: **physical access often equals root access**. While organizations invest heavily in firewalls, intrusion detection systems, and network security, the physical security layer is sometimes neglected.

Disclaimer

This information is provided for **educational purposes only**. Unauthorized access to computer systems is illegal under laws such as:

- Computer Fraud and Abuse Act (CFAA) in the United States
- Computer Misuse Act in the United Kingdom
- Similar legislation worldwide

Always obtain explicit written permission before testing security measures on systems you do not own or have authorization to test. Use this knowledge responsibly to improve security, not to harm others.

References and Further Reading

- GRUB2 Manual - Security
- Linux Kernel Boot Parameters
- Debian: Encrypted Boot
- NIST Physical Security Guidelines (SP 800-116 Rev. 1)
- Red Hat: Protecting GRUB with a Password
- Ubuntu Community: GRUB2 Passwords
- Arch Wiki: dm-crypt/System Configuration

Published: November 2025