Linux Incident Response - Sticky Bits, SUID and SGID.

Taz Wake | Halkyn Consulting Ltd | Originally published 2024-07-01

When responding to an intrusion, responders need to be able to identify elements that might help the attackers or traces of attacker behaviour. In this article, we will look at three ways files (in Linux) can be modified and how it can change the file's behaviour.

This is important because files with SUID/SGID bits set are often left by well-intentioned administrators, but can end up giving an adversary the ability to elevate privileges or compromise a device entirely.

During the "Scoping" phase of IR (or threat hunting) it is a good idea to search for these files and then prioritise reviewing any activity relating to them. Incident Responders need to do this, because our adversaries are certainly doing so.

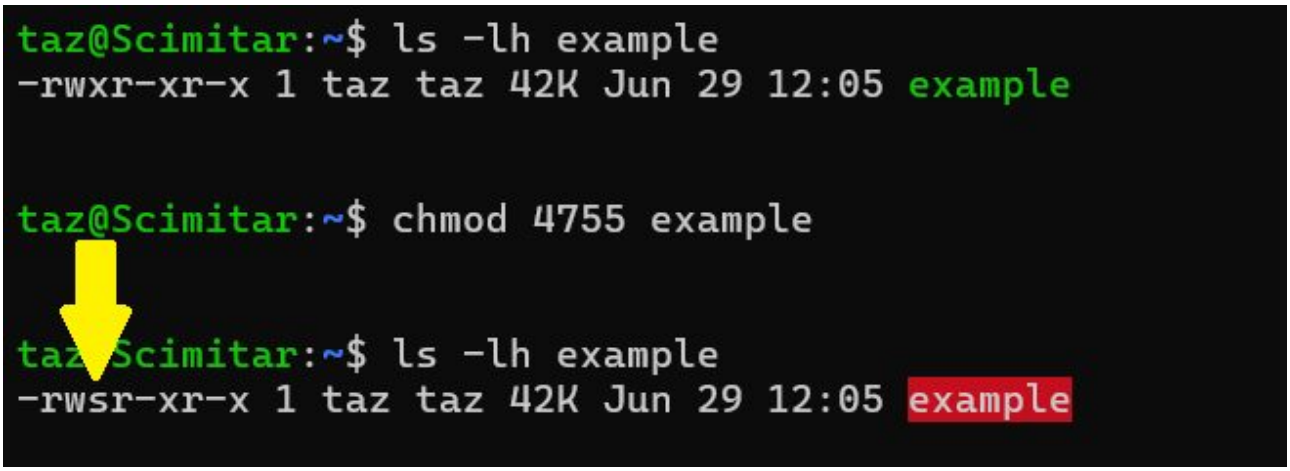
What do the terms mean

SUID

The SUID (Set User ID) bit is a special permission flag in Linux (and other Unix and Unix-like) operating systems that allows a file to be executed with the privileges of the file's owner. This powerful feature enhances functionality in specific scenarios but requires careful management to prevent potential security risks.

When set on an executable file, the SUID bit modifies the standard behaviour of file execution. Instead of running with the privileges of the user who initiated the execution, the program temporarily assumes the privileges of the file owner. This mechanism is commonly used for programs that require elevated permissions to perform specific tasks, such as changing passwords or accessing protected system resources.

In octal notation, the SUID bit is represented by the value 4 in the first digit as we read it (e.g., 4755), while in symbolic notation, it's denoted by an 's' or 'S' in the execute permission field for the owner (e.g., -rwsr-xr-x). Setting the SUID bit can be done with the chmod command:

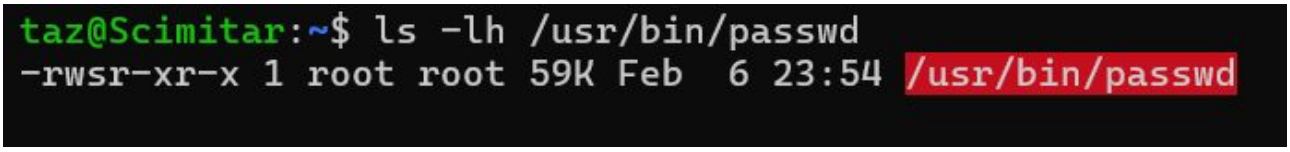


Setting the SUID bit, so whenever this runs, it runs as the user taz - whoever executes it.

If it has a capital 'S', it indicates that the SUID bit is set but the execute permission is not granted to the owner. *This configuration is unusual and typically indicates a misconfiguration, as SUID is generally only useful on executable files*.

When a user executes a file with the SUID bit set, the kernel performs additional checks and temporarily elevates the user's privileges to match those of the file owner. This elevation only applies for the duration of the program's execution. Once the program terminates, the user's original privileges are restored. This mechanism allows for controlled privilege escalation, enabling users to perform specific tasks that would otherwise require root or elevated access.

Common examples of SUID-enabled programs include `passwd` for changing user passwords, and `sudo` for executing commands with superuser privileges.



Example of the passwd command showing the SUID bit set.

However, improper use of SUID can pose significant security risks. Attackers may exploit vulnerabilities in SUID programs to gain unauthorised elevated privileges. Therefore, system administrators must carefully manage and regularly audit SUID-enabled files to maintain system security and integrity.

SGID

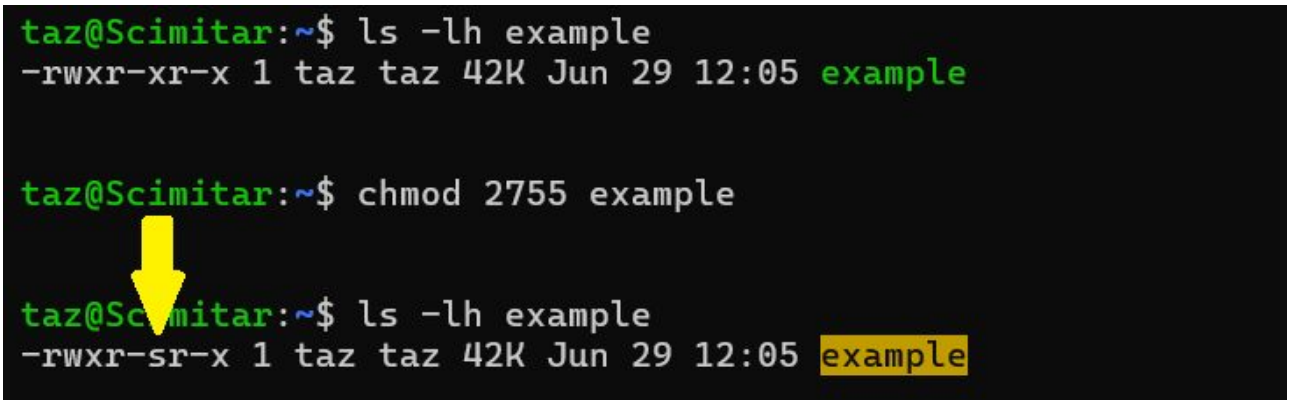
The SGID (Set Group ID) bit is a special permission flag in Linux (and other Unix or Unix-like) operating systems that modifies the behaviour of files and directories with respect to group permissions. This feature enhances collaboration and access control in shared environments but requires careful management to maintain security.

When set on an executable file, the SGID bit alters the standard execution behaviour. Instead of running with the privileges of the user's primary group, the program

temporarily assumes the group privileges of the file's group owner. This mechanism is useful for programs that need to access resources owned by a specific group.

For directories, the SGID bit has a different effect. When set on a directory, it causes new files and subdirectories created within it to inherit the group ownership of the parent directory, rather than the primary group of the user creating the file. This feature is particularly valuable for maintaining consistent group ownership in shared project directories.

In octal notation, the SGID bit is represented by the value 2 in the first digit as we read it (e.g., 2755), while in symbolic notation, it's denoted by an 's' or 'S' in the execute permission field for the group (e.g., -rwxr-sr-x). Setting the SGID bit can be done with the chmod command:



Setting the SGID bit on a file

If it has a capital 'S', it indicates that the SGID bit is set but the execute permission is not granted to the group. *This configuration is unusual for **files** and typically indicates a misconfiguration, as SGID on files is generally only useful on executable files.* ***However, it is a valid and useful configuration for directories***.

When a user executes a file with the SGID bit set, the kernel performs additional checks and temporarily modifies the process's group ID to match that of the file's group owner. This change only applies for the duration of the program's execution. Once the program terminates, the user's original group privileges are restored. This mechanism allows for controlled group-based privilege modification, enabling users to perform specific tasks that require access to group-owned resources.

Common use cases for SGID include shared directories for collaborative projects, where consistent group ownership of files is crucial, and programs that need to access group-specific resources.

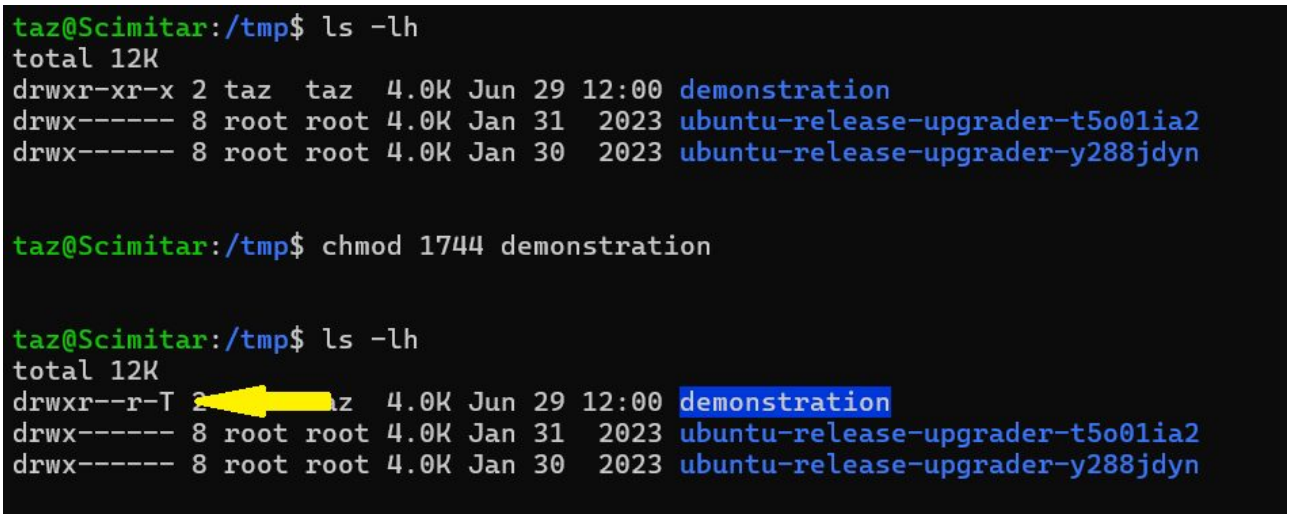
However, as with SUID, improper use of SGID can pose security risks. Attackers may exploit vulnerabilities in SGID programs to gain unauthorised group privileges. Therefore, system administrators must carefully manage and regularly audit SGID-enabled files and directories to maintain system security and integrity.

Sticky Bit

The sticky bit is a permission flag in Linux (and other Unix/Unix-like) operating systems which enhances security and access control for files. It is sometimes called the "restricted deletion flag" or "saved text attribute."

When set on a directory, it modifies the standard behaviour of file deletion and renaming within that directory. The primary function of the sticky bit is to restrict the deletion or renaming of files to only the file owner, the directory owner, or the root user, regardless of the write permissions on the directory.

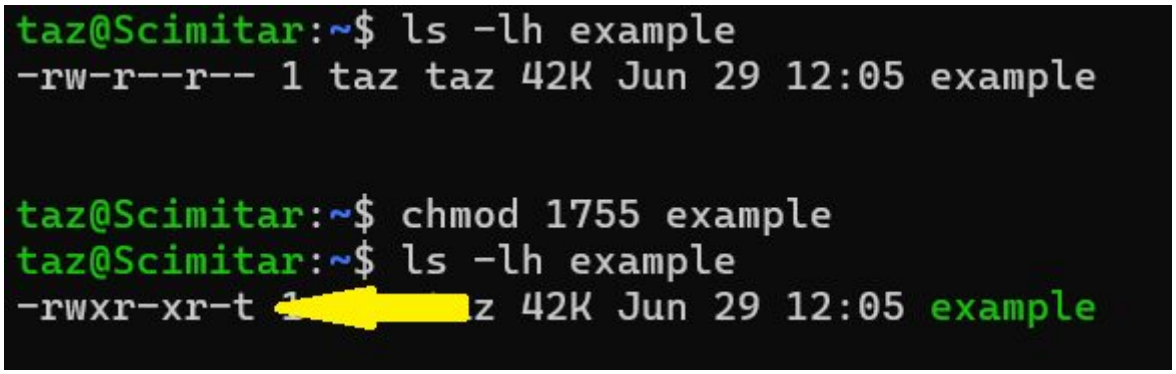
In octal notation, the sticky bit is represented by the value 1 in the fourth digit (e.g., 1755), while in symbolic notation, it's denoted by a 't' or 'T' in the execute permission field for "others" (e.g., drwxrwxrwt). Setting the sticky bit can be done with the chmod command



Setting the sticky bit for a folder.

The capital 'T' indicates that the sticky bit is set but the execute permission is not granted to others. This permission mode is commonly applied to shared directories like /tmp, where multiple users have write access but shouldn't be able to interfere with each other's files.

Although uncommon, individual files can also have the sticky bit set as follows:



Setting the sticky bit and allowing all users to execute a file.

When set on a file, the sticky bit historically had a different meaning. On some Unix systems, it would keep the program's text image in swap space after it finished executing, speeding up subsequent executions. However, in modern Linux systems, the sticky bit on files is largely ignored and doesn't affect file behaviour. Despite this, you may still encounter it in legacy systems or documentation.

When a user attempts to **delete** or **rename** a file in a directory with the sticky bit set, the kernel performs additional checks beyond the standard file permissions. It verifies whether the user is the owner of the file, the owner of the directory, or has root privileges. If none of these conditions are met, the operation is denied, even if the user has write permissions on both the file and the directory. This mechanism effectively prevents users from maliciously or accidentally removing or altering files owned by other users in shared spaces, enhancing system security and maintaining data integrity in multi-user environments.

Security Issues and DFIR

The primary security risks associated with SUID, SGID, and sticky bits stem from their potential for privilege escalation and unauthorised access. SUID and SGID executables, if compromised, can allow attackers to execute commands with elevated privileges, potentially leading to system-wide compromise. Vulnerabilities in these binaries, such as buffer overflows or race conditions, can be exploited to gain higher-level access. The sticky bit, whilst primarily a protective measure, can be misused if set incorrectly, potentially allowing unauthorised file modifications in shared directories.

Incident responders should prioritise the identification and analysis of files with these special permissions during investigations. Unusual or unexpected SUID/SGID binaries, especially in non-standard locations, warrant immediate scrutiny. Responders should verify the integrity of these files, cross-reference them with known-good versions, and analyse their recent execution history. For directories with the sticky bit set, attention should be paid to any anomalies in file ownership or unexpected modifications, particularly in shared areas like /tmp.

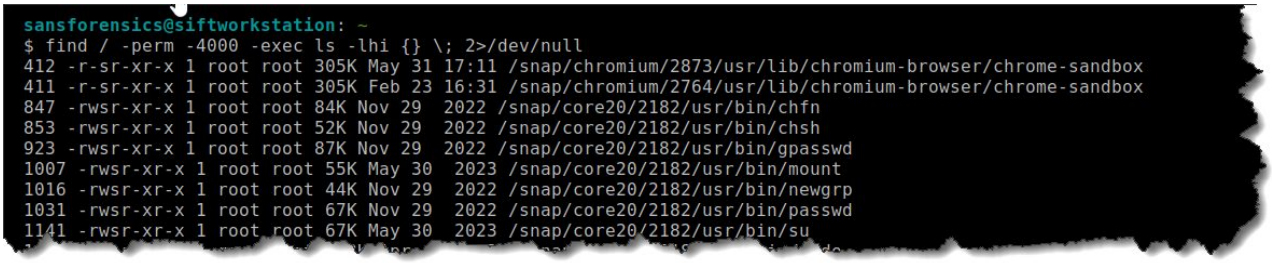
It is really important that responders understand the normal baseline of SUID/SGID files and sticky bit usage on the systems they're investigating. This knowledge allows for rapid identification of potential threats. Additionally, responders should be aware that attackers

often attempt to create or modify SUID/SGID binaries as a persistence mechanism. Therefore, any changes to these special permissions, especially on critical system files or in user home directories, should be treated as highly suspicious and investigated thoroughly.

Finding files with SUID/SGID/Sticky Bit set

One of the easiest methods is to use the find command. Normally we would use the -perm argument to specify the values we are interested in (for example, -4000 to look for SUID bits being set). It is the same for SGID and Sticky bit files so options are:

The find command allows us to add some additional steps so that when a file with the bit is set, we can get some additional information on it. For example, this is a way to search for files with the SUID bit set, and if found, run ls -lhi on the file to get the long format of ls, with human-readable sizes and the inode number.



Running a find command to look for SUID binaries, then run ls, showing long-format, human-readable timestamps and inode numbers for any items found. This command also pipes errors (FD2) to /dev/null to suppress any noisy error messages.

We could even expand this to look for it all at once with a command similar to:

This can be a quick way to search but remember there will be an increase in false positives (folder structures, mostly), so we can tidy it up a little bit by specifying we only want files.

An example showing the extended find command looking for all three combinations. Note - on most recent Ubuntu distros a lot of these files will be related to the Snap package manager.

What about the inode?

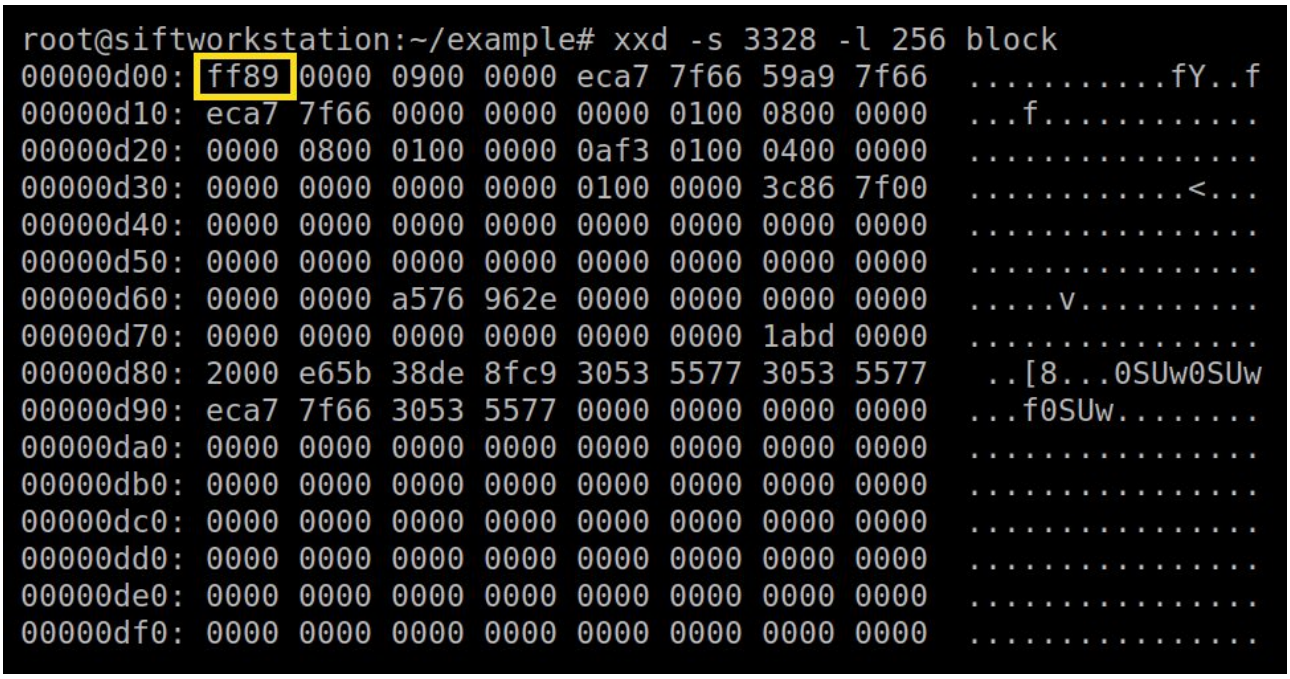
Here things get a little bit more complex but we can still identify SUID/SGID/Sticky bits. In an EXT4 inode, the permissions are the 16 bits (Little Endian) at offset 0x0 into the inode. Bits 9-11 here show any special permissions as follows:

- 001 - Sticky bit (0x2)
- 010 - Set Group ID (SGID) (0x4)
- 100 - Set User ID (SUID) (0x8)

The permissions are normally shown in hex as four bytes, which means some calculations are required. For example, a regular file with the sticky bit set and RWXRWXRWX (777) permissions will have FF89 as the first four bytes.

This translates as follows - first because it is little-endian, we actually read it 0x89 0xFF.

The leading 0x8 represents regular files. The second value - 0x9 is the result of the values for User Read (0x1) and SUID (0x8). The 0xFF represents all the other bits being set to give read/write/execute for all users.



Example of an EXT4 inode for a file with SUID bit set and RWXRWXRWX permissions.

With XFS it is a similar structure however it is located at byte offset 2-3 and stored in big-endian format. That means the example above would appear 0x89FF in the evidence.

What are you looking for?

As an incident responder or investigator, examples of things to check include

- Any user-created binaries or scripts with the SUID/SGID bit set and writable permissions. If the user has created these there is a risk that they might be exploitable. Check the file and validate this. A common example is users setting up scripts to back up their data. If an attacker can write to this file, they can modify the locations and use this to create a session running as the user. An example of a search to do this would be:

- Any binaries owned by root with the SUID/SGID bit set. You can (generally) allow-list the binaries provided by the distro but any others might introduce the risk of exploitation and privilege escalation. An example of a search could be:

- Combinations of the above - such as root-owned files, which are executable, with the SUID/SGID bits set and can be written to by other users (world writeable). A search for this could be:

Conclusion

Understanding SUID, SGID, and sticky bit permissions is crucial for Linux system administrators and incident responders. These special permissions, while powerful tools

for enhancing functionality and sometimes security, can pose significant risks if misused or exploited. Regular auditing of files with these permissions, particularly those owned by root or with world-writable access, is essential to maintain system integrity and prevent privilege escalation attacks.

Incident responders and security professionals should incorporate searches for these special permissions into their standard investigation procedures. Using tools like 'find' and understanding the underlying inode structures, responders can quickly identify security risks and unauthorised changes to system files. Ultimately, a proactive approach to managing these special permissions will significantly enhance the overall security posture of Linux systems.

About Halkyn Consulting

Halkyn Consulting Ltd is a UK-based security and risk management consultancy providing 24/7 incident response, digital forensics, threat hunting, virtual security leadership (vCISO) and security compliance support to organisations worldwide. We deliver clear, practical, vendor-independent advice based on real-world threats — no hype, no upsell.

More free security resources are available at www.halkynconsulting.co.uk/security-resources.

Web: www.halkynconsulting.co.uk | **Email:** info@halkynconsulting.co.uk | **Tel:** +44 1244 940858