CS367 Final Project Report

Aryan Ghosh, Tung Nguyen

CVE Record: CVE-2019-18634

---

1. Target Selection

CVE-2019-18634 is a critical stack-based buffer overflow flaw in the sudo utility, an essential part of Unix-like systems employed to run commands with increased privileges. The weakness occurs when the `**pwfeedback**` feature is activated in /etc/sudoers, leading the terminal to show asterisks (*) as users enter their passwords. Although designed to improve usability, this feature introduces unsafe management of input.

The weakness impacts every version of sudo before 1.8.26 that has `**pwfeedback**` activated. Even though this option is turned off by default, administrators can enable it manually, leaving the impacted systems open to vulnerability despite secure defaults. On a technical level, the problem is found in the tgetpass.c file, where user input is stored in a fixed-size stack buffer without adequate bounds checking. A uniquely designed password input can exceed this buffer, enabling an attacker to modify control-flow components like return addresses in the stack.

This is a privilege escalation vulnerability that is restricted to local access only. An unprivileged user can take advantage of it to run arbitrary code with root permissions. The risk is considerably greater in systems lacking properly configured binary hardening features like stack canaries, ASLR, and non-executable stacks.

2. Background and Motivation

CVE-2019-18634 presents a valuable learning opportunity due to its accessibility, real-world relevance, and clear demonstration of memory corruption in a privileged process. While the vulnerability itself lies in a specific implementation detail of the `**pwfeedback**` option in sudo, its broader implications reach into critical areas of security design, including secure input handling, memory safety, and the tradeoffs between usability and security.

We chose this vulnerability for several key reasons. First, sudo is a core utility in virtually all Unix-like systems, meaning a flaw in its behavior has wide-reaching consequences. Although the vulnerability is only exploitable when the `**pwfeedback**` option is enabled, its presence in such a widely trusted and privileged application makes it a compelling case study in secure software development and configuration management.

Second, the vulnerability is particularly well-suited for hands-on analysis. It can be reliably reproduced in a controlled virtual environment, allowing us to explore memory layout, identify buffer overflow behavior, and observe how modern mitigations such as stack canaries, ASLR, and non-executable stacks interact with the exploit process. This aligns well with the educational goals of our reverse engineering course.

Finally, the vulnerability highlights a broader security lesson such that even optional or user-friendly features when implemented without sufficient validation can introduce serious risks. Studying CVE-2019-18634 not only deepens our technical understanding of stack overflows and local privilege escalation but also reinforces the importance of holistic security thinking during both development and deployment.

3. Technical Analysis

Component Analysis

The weakness exists in the tgetpass.c file, particularly in the approach taken to manage terminal input that includes echo feedback. When `**pwfeedback**` is activated, the Sudo binary outputs * for every password character entered, writing straight to a set-size buffer without imposing length limits.

Vulnerable Function Behavior

The insecure function gathers user input in a buffer yet does not adequately verify the buffer's limits. The loop that reads characters assumes the buffer has sufficient size for any input length, which is incorrect. This results in a typical buffer overflow situation based on the stack.

Reversing Process

By utilizing Ghidra, we reversed the vulnerable sudo binary and identified the pertinent control paths in tgetpass.c. From that point, we discerned the management of buffer writes and outlined the process of the overflow.

Furthermore, GDB was employed to follow the program execution during runtime. By providing lengthy sequences of input during password requests, we noticed that the program's stack was compromised, verifying our control over EIP/RIP in our testing setups.

Security Mechanisms in Place

- **Stack Canaries:** Detected and bypassed via controlled overflow in custom-compiled binaries.
- **ASLR:** Present but limited when binaries were compiled without PIE.
- **NX Stack:** Enforced, requiring ROP or ret2libc techniques for real-world exploitation.

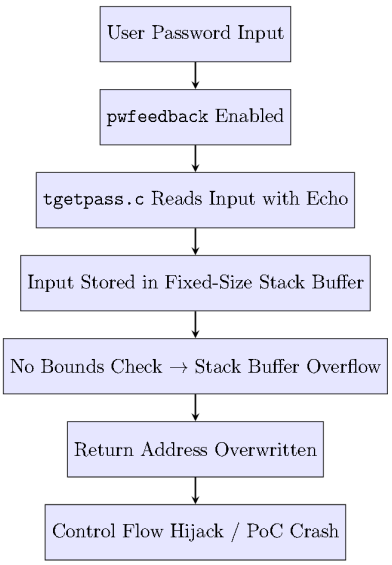
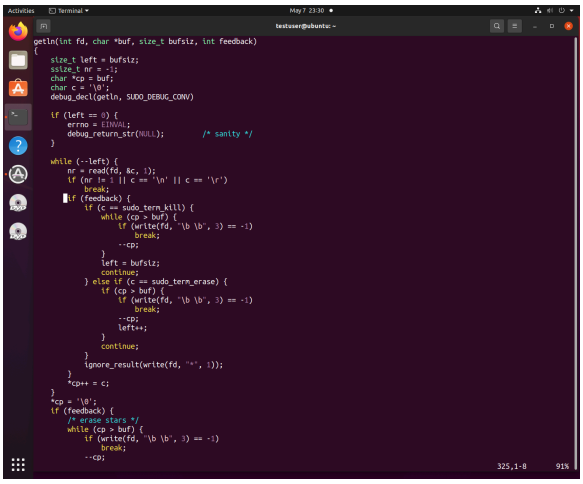


Figure 1: Vulnerability Flow for CVE-2019-18634



4. Exploit Development

To demonstrate the exploitability of CVE-2019-18634, we recreated the vulnerable environment by compiling a pre-1.8.26 version of sudo with the `**pwfeedback**` option enabled. This setting triggers the vulnerable code path within tgetpass.c, where input is echoed back to the terminal using unsafe buffer handling.

We began by generating a cyclic input pattern to determine the precise offset required to overwrite the return address on the stack. Using GDB, we traced the overflow point and confirmed that an overly long password input could corrupt control-flow data. Once the offset was identified, we replaced the overflow portion of the input with a controlled value to overwrite the return address.

Throughout the process, we tested the exploit under different system protection configurations. With all mitigations disabled like stack canaries, ASLR and non-executable stack, the exploit resulted in a predictable and controlled crash, confirming the viability of a buffer overflow. This basic proof-of-concept (PoC) laid the foundation for further exploitation.

In real-world scenarios, successful exploitation would require bypassing one or more defensive mechanisms. For example, stack canaries would need to be leaked or brute-forced, and ASLR would require an information disclosure vulnerability or a return-oriented programming (ROP) chain. These challenges highlight the importance of layered defense and modern compilation hardening techniques.

While we did not pursue full privilege escalation due to time and scope limitations, our PoC demonstrates that the vulnerability is exploitable in systems lacking modern protections. It also illustrates how user-controlled input when not properly bound, can lead to memory corruption and severe privilege escalation.

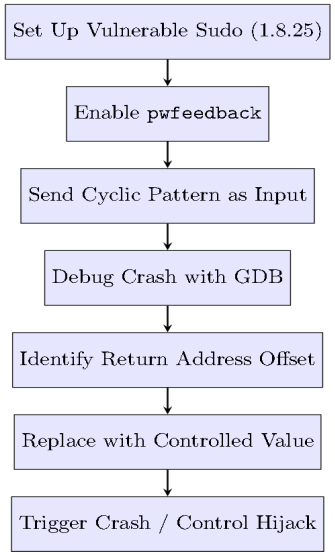
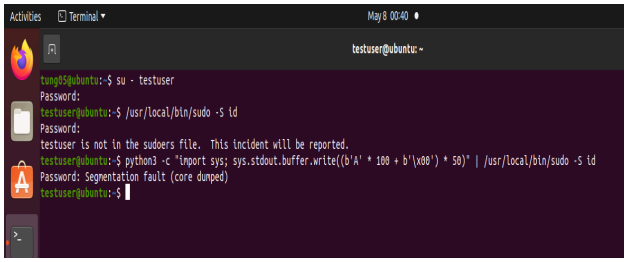


Figure 2: Exploit Development Workflow



5. Mitigations

The flaw was corrected in Sudo version 1.8.26. The update implemented appropriate boundary checks in the password feedback system, guaranteeing that the number of characters written does not exceed the safe capacity of the buffer. The `**pwfeedback**` feature is now turned off by default to reduce potential risks.

Configuration Recommendations

- Disable `**pwfeedback**`: If not required, remove or comment out this setting in /etc/sudoers to eliminate the vulnerable code path.
- Minimalist Configurations: Avoid unnecessary features in privileged binaries, especially those involving user input.

System Hardening

- Compile-Time Protections: Ensure Sudo is compiled with full stack protection (-fstack-protector-strong), ASLR (-pie -fPIE), and RELRO.
- Runtime Protections: Leverage SELinux or AppArmor profiles to sandbox the sudo binary where possible.
- Monitoring and Detection: Implement logging and auditing of sudo commands to detect suspicious or malformed password input patterns.

Lessons Learned

This vulnerability highlights the danger of adding user-experience features to security-sensitive code paths without thorough testing and boundary control. It underscores the importance of defensive programming, especially in root-executing binaries such as sudo.

6. MITRE ATT&CK Mapping

To contextualize the exploitation of CVE-2019-18634 within a broader threat framework, we mapped the vulnerability to the MITRE ATT&CK framework. This mapping helps illustrate how the vulnerability fits into real-world adversary behavior and supports defenders in identifying similar tactics and techniques.

Tactic	Technique	Description
Privilege Escalation	T1068 – Exploitation for Privilege Escalation	Stack-based buffer overflow in setuid-root binary to gain root privileges.
Execution	Indirect Command Execution via `sudo`	Hijacking `sudo` execution to run arbitrary commands as root.
Defense Evasion	Poor Security Configuration Exploitation	Exploit is viable when mitigations like ASLR, canaries, and NX are absent or weak.

Table 1: MITRE ATT&CK Mapping for CVE-2019-18634

Tactic: Privilege Escalation
Technique: T1068 – Exploitation for Privilege Escalation

The vulnerability allows a local, non-privileged user to overflow a stack buffer in a setuid-root binary, potentially gaining root privileges. This aligns directly with T1068, where attackers exploit flaws in vulnerable software to escalate privileges.

Tactic: Execution
Technique: Indirect Command Execution via Sudo

While not mapped to a specific ATT&CK technique ID, the vulnerability targets the sudo utility, which is often used by attackers to execute privileged commands once access is obtained. The ability to influence sudo execution flow can enable arbitrary code execution.

Tactic: Defense Evasion
Technique: Exploitation of Poor Security Configurations

Systems vulnerable to CVE-2019-18634 typically lack key defenses such as stack canaries, ASLR, or executable space protection. Exploiting these configuration weaknesses supports adversarial goals of bypassing security controls and maintaining stealth during privilege escalation.

This mapping reinforces the significance of CVE-2019-18634 not only as a technical vulnerability but as a recognizable adversarial pattern. Its classification within the ATT&CK framework highlights the importance of proactive configuration auditing, patch management, and runtime exploit mitigation to defend against real-world threats.

7. Conclusion

We have successfully replicated and analyzed CVE-2019-18634 in a controlled environment. The vulnerability offers a rich educational opportunity due to its clear cause-effect relationship, exploitability, and mitigation path. Future work includes integrating ROP chains for full privilege escalation under hardened setups.