Minisudo: Uma Implementação Simplificada do Sudo em Rust

Leonardo Castro[1], **Álefe Alves**[1]

[1]Universidade Federal de Roraima (UFRR)
Centro de Ciência, Tecnologia e Inovação – Boa Vista, RR – Brasil

student.leonardocastro@gmail.com, alefealvesdacosta@gmail.com

Abstract. *Este artigo apresenta o projeto minisudo, uma implementação simplificada do utilitário sudo, desenvolvido na linguagem Rust. O objetivo principal é reforçar conceitos de autenticação, controle de permissões e segurança em sistemas operacionais. O programa simula a execução de comandos privilegiados, mediante verificação de senha e autorização via arquivo de configuração. A aplicação foi testada em ambiente Linux virtualizado com QEMU.*

Resumo. *Este artigo apresenta o minisudo, uma implementação simplificada do utilitário sudo, desenvolvida em Rust. O projeto tem como propósito aplicar conceitos fundamentais de autenticação, permissões e segurança no contexto de sistemas operacionais. A aplicação simula a execução de comandos administrativos, exige autenticação por senha e realiza registros em log. Os testes foram realizados em ambiente virtualizado com QEMU.*

1. Introdução

A segurança em sistemas operacionais é um tema de relevância crescente, especialmente quando se trata de controle de acesso e execução de comandos privilegiados. O utilitário *sudo*, amplamente utilizado em sistemas Unix-like, permite que usuários executem comandos como superusuário de forma temporária e controlada. Inspirado por esse conceito, o presente trabalho propõe a criação de uma ferramenta simplificada denominada **minisudo**, implementada em *Rust*, com foco didático e de aprendizagem.

2. Objetivos do Projeto

O projeto *minisudo* tem por objetivo:

- Aplicar os conceitos de autenticação baseada em senha;
- Simular permissões administrativas por meio de regras personalizadas;
- Registrar as ações do usuário para fins de auditoria;
- Reforçar a prática segura de desenvolvimento com *Rust*.

3. Tecnologias Utilizadas

A linguagem escolhida para o desenvolvimento foi o **Rust**, devido à sua forte ênfase em segurança de memória e concorrência segura. As principais **crates** utilizadas incluem:

`bcrypt` Para verificação segura de senhas com hashes.
`rpassword` Para entrada de senha no terminal sem exibição.
`clap` Para parsing de argumentos da linha de comando.
`users` Para obter o nome do usuário atual.

`chrono` Para geração de timestamp nos logs.

O projeto foi desenvolvido e testado em um ambiente Linux virtualizado utilizando o **QEMU**, proporcionando um ambiente isolado e controlado para simulações seguras. Para os testes automatizados, foi utilizado o comando `cargo test`, aliado às crates `assert_cmd`, `tempfile` e `predicates`, permitindo verificar o comportamento da aplicação em diferentes cenários, como falha de autenticação, permissões negadas e geração correta de logs.

4. Arquitetura da Aplicação

A estrutura de diretórios do projeto é organizada conforme apresentado abaixo:

```
minisudo/
|- Cargo.toml
|- Cargo.lock
|- config/
|  |- minisudo_password
|  `- minisudoers
|- logs/
|  `- minisudo.log
|- src/
|  `- main.rs
`- tests/
   `- integrations.rs
```

Durante a instalação no ambiente Alpine Linux, os arquivos de configuração, logs e binários são posicionados em diretórios específicos do sistema:

```
/etc/
`- minisudoers
/var/log/
`- minisudo.log
/usr/local/bin/
`- minisudo
```

A aplicação segue um fluxo lógico dividido em cinco etapas principais:

1. Leitura do usuário atual do sistema;
2. Busca e verificação do hash da senha;
3. Autenticação com até três tentativas;
4. Validação de permissões no arquivo `minisudoers`;
5. Registro do comando autorizado no arquivo de log.

5. Implementação

A autenticação no *minisudo* é baseada na leitura de um arquivo localizado em `/etc/minisudo_password`, cujo formato segue a estrutura:

```
[nome_do_usuario]:[hash_da_senha]
```

Cada linha representa um par usuário-hash, sendo o hash gerado com a `crate bcrypt`, usando um custo padrão de complexidade 12, que oferece um bom equilíbrio entre segurança e desempenho.

Durante a execução, o sistema identifica o usuário ativo por meio da crate `users`, busca sua entrada no arquivo de senhas, e solicita a senha via terminal (sem eco), utilizando a crate `rpassword`. O trecho a seguir ilustra a verificação da senha com bcrypt:

```
let senha = rpassword::prompt_password(
    format!("[minisudo] senha para {}: ", username)
).unwrap();

if verify(&senha, &hash).unwrap_or(false) {
    autenticado = true;
}
```

Para mitigar ataques de força bruta, a autenticação é limitada a no máximo 3 tentativas. Caso todas falhem, a execução é encerrada imediatamente com uma mensagem de erro.

Se a autenticação for bem-sucedida, o sistema verifica se o comando requisitado é autorizado ao usuário, por meio do arquivo `/etc/minisudoers`, no formato:

```
usuario comando1 comando2 ...
```

Comandos podem ser autorizados individualmente ou de forma genérica, utilizando a palavra-chave `ALL`.

Todas as execuções válidas são registradas no arquivo `/var/log/minisudo.log`, seguindo o formato:

```
[2025-08-06 14:35:12] usuario: joao, comando: apt update
```

Esse log é gerado com auxílio da crate `chrono` e possui permissões restritas para preservar a integridade das informações registradas.

6. Resultados e Testes

Para garantir o correto funcionamento do *minisudo*, foram criados testes automatizados utilizando as crates `assert_cmd`, `tempfile` e `predicates`. O objetivo foi simular diferentes cenários de uso, incluindo permissões negadas, senhas incorretas e comandos autorizados.

Um exemplo de teste automatizado consiste em verificar a falha de autenticação quando um usuário fornece uma senha inválida por três vezes consecutivas:

```
#[test]
fn erro_de_senha() {
    let hash_errada = bcrypt::hash("senha", bcrypt::DEFAULT_COST
        ).unwrap();
    let (_dir, minisudoers_path, password_path) =
        criar_ambiente("usuario", &hash_errada, "ls");

```

```
    let mut cmd = Command::cargo_bin("minisudo").unwrap();
    cmd.env("MINISUDO_TEST_USER", "usuario")
        .env("MINISUDOERS_PATH", minisudoers_path)
        .env("MINISUDO_PASSWORD_PATH", password_path)
        .arg("ls")
        .write_stdin("senha_errada\nsenha_errada\nsenha_errada\n
            ");

    cmd.assert()
        .failure()
        .stderr(contains("3 tentativas de senha incorreta"));
}
```

Outros testes incluem:

- Execução negada a usuários não autorizados mesmo com senha válida;
- Permissão negada a comandos não listados no `minisudoers`;
- Verificação de logs corretamente gerados após execução autorizada.

Através desses testes, foi possível validar a robustez da autenticação e a precisão dos mecanismos de autorização e logging do sistema.

7. Considerações Finais

A construção do *minisudo* proporcionou a consolidação de conceitos importantes de segurança em sistemas operacionais, autenticação e controle de permissões. A linguagem *Rust* mostrou-se eficaz para o desenvolvimento de ferramentas seguras e robustas. Projetos como este contribuem significativamente para o aprendizado prático em cursos de Sistemas Operacionais e Segurança da Informação.

8. Disponibilidade do Código

O código-fonte completo está disponível publicamente em:

- Leonardo Castro: https://github.com/thetwelvedev/FinalProject_OS_UFRR_Desc_9_2025

Referências

[1] TANENBAUM, Andrew S.; WOODHULL, Albert S. **Sistemas operacionais: projeto e implementação**. 3. ed. Porto Alegre: Bookman, 2008.

[2] *sudo (8) — manual page*. Disponível em https://linux.die.net/man/8/sudo. Acessado em 22 de julho de 2025.

[3] Docs.rs. *Docs.rs*, disponível em https://docs.rs/. Acessado em 22 de julho de 2025.

[4] QEMU. *QEMU Documentation*, disponível em https://www.qemu.org/documentation/. Acessado em 24 de julho de 2025.