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.

4. Arquitetura da Aplicação

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

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

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 seguir, um trecho simplificado da lógica de autenticação:

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

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

Após a autenticação, o sistema verifica se o comando é permitido para o usuário consultando o arquivo `minisudoers`, e caso seja, registra a ação:

```
let entrada_log = format!(
    "[{}] usuario: {}, comando: {}\n",
    agora, username, comando_completo
);
```

A execução do comando é apenas simulada, sem alteração real de privilégios, reforçando o caráter educacional da ferramenta.

6. Resultados e Testes

O *minisudo* foi testado em um ambiente Linux leve, virtualizado com QEMU. As simulações envolveram diferentes usuários, senhas válidas e inválidas, bem como comandos permitidos e não permitidos. Todos os registros foram corretamente armazenados no arquivo `logs/minisudo.log`.

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

9. Referências