Entrega 1 - Redes de Computadores e Cibersegurança

Introdução

O projeto consiste no desenvolvimento de uma solução de automação para salas de aula, voltada a instituições de ensino que desejam otimizar recursos, melhorar a gestão acadêmica e proporcionar maior organização no ambiente escolar.

A proposta integra diferentes funcionalidades de automação, como controle inteligente de iluminação e climatização, identificação de professores e alunos por meio de etiquetas RFID, gerenciamento de horários de aula e monitoramento de presença em avaliações. Essas funcionalidades têm como objetivo principal aumentar a eficiência operacional das instituições, reduzir desperdícios de energia elétrica e oferecer maior confiabilidade nos processos de controle acadêmico.

Além disso, a solução busca agregar valor para professores, alunos e gestores escolares ao proporcionar um ambiente mais confortável, organizado e transparente. Para a instituição, os dados coletados geram relatórios estratégicos sobre frequência, pontualidade e utilização dos recursos da sala, permitindo tomadas de decisão mais embasadas.

Relatório de Vulnerabilidade

Começamos olhando o sketch de exemplo do RC522 e o circuito de bancada, sem ferramentas avançadas, isso por si só já revela muita coisa. O primeiro problema é depender do UID do cartão em texto claro: se o firmware ou os logs expõem esse UID, a clonagem fica mais fácil. Outro ponto, em protótipos é aceitar leituras infinitas e muito rápidas do mesmo cartão, o que pode inflar presença ou travar o leitor. Também percebemos que manter a Serial em modo de “debug” em produção vaza estados internos. Do lado físico, sem um sensor de violação, qualquer pessoa pode abrir a caixa do dispositivo sem deixar rastro. E, olhando para os cartões em si, o uso de MIFARE Classic com chaves padrão é um risco bem conhecido, não é que quebramos criptografia, mas é uma configuração fraca comum em alguns exemplos. Por fim, existe a dimensão de privacidade: vincular UID diretamente a uma pessoa é

tratamento de dado pessoal, então precisamos de cuidado com LGPD, mesmo em protótipo.

As correções que implementamos seguem a lógica do que está ao nosso alcance agora. Em vez de guardar ou comparar UID em claro, o firmware calcula um hash FNV-1a do UID concatenado com um salt e compara apenas esse hash com uma lista de autorizados. Isso não “blinda” o cartão, mas impede que o microcontrolador carregue uma lista de UIDs legíveis. Para conter abuso e leituras repetidas, criamos uma janela anti-passback de quinze segundos para a mesma credencial e um rate-limit que bloqueia o sistema por sessenta segundos quando acontecem cinco falhas em trinta segundos. Reduzimos os logs no modo produção para não vazar informação desnecessária. Adicionamos um micro-switch de tamper: ao abrir a caixa, o dispositivo registra o evento e sinaliza localmente. E ajustamos a inicialização para manter as saídas em estado seguro ao ligar ou reiniciar, evitando que um relé ligue algo sem querer.

Testamos esses pontos de forma objetiva. Ao apresentar vários cartões inválidos em sequência, o bloqueio temporário entra em ação como esperado. Ao tentar passar o mesmo cartão antes do intervalo definido, a leitura é recusada. Ao acionar o tamper, temos registro e sinalização. E, ao reiniciar o Arduino de propósito, as saídas voltam ao estado seguro e o leitor retoma a operação normal. Esses testes são fáceis de reproduzir e geram evidências que podemos anexar ao trabalho.

Em relação ao que vem depois, a linha é clara: se a instituição quiser segurança forte no cartão, o caminho é migrar para tecnologias com mecanismos criptográficos melhores (como DESFire ou NTAG com recursos de segurança). Se houver rede, a comunicação deve usar TLS e autenticação de API com rotação de segredos. Do lado de privacidade, é importante formalizar base legal, política de retenção e controles de acesso, no microcontrolador, manter apenas o mínimo necessário, hash das credenciais já ajuda muito.

Partimos do exemplo do RC522, mapeamos as fragilidades mais evidentes de um protótipo de bancada e tratamos o que de fato pesa no uso real, tiramos o UID em claro comparando hash com salt, bloqueamos leituras repetidas e abuso (anti-passback de 15 s e rate-limit com bloqueio de 60 s), enxugamos os logs para produção, adicionamos detecção de violação física e garantimos estado seguro na inicialização. Essas mudanças foram guiadas por impacto e custo de implementação, e os testes em bancada confirmaram o comportamento esperado em cada caso.

Ficam claros os caminhos de evolução cartões com segurança mais forte (DESFire/NTAG), canal de comunicação protegido caso exista backend e formalização de LGPD, mas, dentro do escopo da disciplina, o sistema passou a refletir decisões técnicas justificadas por teste, não por suposição. O resultado é um

protótipo que a gente consegue explicar em poucos minutos, reproduzir com facilidade e que, na prática, ficou bem mais difícil de contornar no dia a dia de sala.]

Código Fonte

```
#include <SPI.h>
#include <MFRC522.h>
#include <string.h> // strlen()

#define DEBUG  0   // 1 = logs de teste; 0 = produção
#define ENROLL 0   // 1 = imprime hash para cadastro no Serial

// Pinos (Arduino UNO): RC522 em SPI; periféricos opcionais para feedback
const uint8_t PIN_SS=10, PIN_RST=9, PIN_LED_OK=6, PIN_LED_NO=5,
PIN_BUZZER=4, PIN_TAMPER=7;
MFRC522 r(PIN_SS, PIN_RST);

const char* SALT = "sala2025@protecao";

// Adicione aqui os hashes autorizados (formato 0xXXXXXXXX)
uint32_t WL[] = {
  0xFE2AF2B7, // exemplo
  0x9C1123A5  // exemplo
};
const size_t WLN = sizeof(WL)/sizeof(WL[0]);

// Janelas/limites (ajuste conforme o uso)
const unsigned long AP_MS = 15000UL;  // anti-passback
const uint8_t       FAIL_LIM = 5;     // falhas por janela
const unsigned long WIN_MS  = 30000UL;// janela de contagem
const unsigned long BLK_MS  = 60000UL;// bloqueio

// Estado
unsigned long lastOKms = 0, winStart = 0, blockUntil = 0;
uint32_t lastHash = 0;
uint8_t  fails = 0;

inline uint32_t fnv1a_init(){ return 0x811C9DC5UL; }
inline uint32_t fnv1a_upd(uint32_t h, uint8_t b){ h^=b; h*=16777619UL; return h; }
uint32_t hash_uid_salt(const MFRC522::Uid &u){
  uint32_t h = fnv1a_init();
  for(byte i=0;i<u.size;++i) h = fnv1a_upd(h, u.uidByte[i]);
  for(size_t i=0;i<strlen(SALT);++i) h = fnv1a_upd(h, (uint8_t)SALT[i]);
  return h;
}
bool in_wl(uint32_t h){
  for(size_t i=0;i<WLN;++i) if(WL[i]==h) return true;
```

```
  return false;
}

// Feedback simples (LED + buzzer)
void tone_ok(){ digitalWrite(PIN_LED_OK, HIGH); tone(PIN_BUZZER,1800,100);
delay(120); digitalWrite(PIN_LED_OK,LOW); }
void tone_no(){ digitalWrite(PIN_LED_NO, HIGH);  tone(PIN_BUZZER, 600,300);
delay(320); digitalWrite(PIN_LED_NO,LOW); }

void setup(){
  pinMode(PIN_LED_OK,OUTPUT); pinMode(PIN_LED_NO,OUTPUT);
  pinMode(PIN_BUZZER,OUTPUT); pinMode(PIN_TAMPER,INPUT_PULLUP);
  digitalWrite(PIN_LED_OK,LOW); digitalWrite(PIN_LED_NO,LOW);

  Serial.begin(115200);
  SPI.begin(); r.PCD_Init();

  winStart = millis();
#if DEBUG
  Serial.println("RC522 pronto");
#endif
}

void loop(){
  const unsigned long now = millis();

  // Tamper: LOW = violação detectada
  if(digitalRead(PIN_TAMPER)==LOW){
#if DEBUG
    Serial.println("TAMPER");
#endif
    for(int i=0;i<2;i++){ tone_no(); delay(120); }
  }

  // Bloqueio ativo (rate-limit)
  if(now < blockUntil){
    digitalWrite(PIN_LED_NO, (now/250)%2); // pisca para indicar bloqueio
    return;
  } else {
    digitalWrite(PIN_LED_NO, LOW);
  }

  // Renova a janela de contagem de falhas
  if(now - winStart > WIN_MS){ winStart = now; fails = 0; }

  // Aguarda cartão
  if(!r.PICC_IsNewCardPresent() || !r.PICC_ReadCardSerial()) return;
```

```
  uint32_t h = hash_uid_salt(r.uid);

#if ENROLL
  Serial.print("HASH 0x"); Serial.println(h, HEX);
#endif

  // Anti-passback (mesma credencial em intervalo curto)
  if(h==lastHash && (now - lastOKms) < AP_MS){
#if DEBUG
    Serial.println("AP negado");
#endif
    tone_no();
    r.PICC_HaltA(); r.PCD_StopCrypto1();
    return;
  }

  if(in_wl(h)){
    lastHash = h; lastOKms = now;
#if DEBUG
    Serial.println("OK");
#endif
    tone_ok();
  } else {
    fails++;
#if DEBUG
    Serial.print("NEG "); Serial.println(fails);
#endif
    if(fails >= FAIL_LIM){
      blockUntil = now + BLK_MS;
#if DEBUG
      Serial.println("BLOQUEIO");
#endif
    }
    tone_no();
  }

  // Encerra sessão com o cartão
  r.PICC_HaltA();
  r.PCD_StopCrypto1();
}
```