# Prompt para Desenvolvimento de Plugin Better Auth: Conformidade PCI DSS

## 1. Objetivo

Desenvolver um plugin de comunidade para o Better Auth que implemente e enforceça políticas de segurança de senha em conformidade com os requisitos do PCI DSS (Payment Card Industry Data Security Standard), versão 4.0 ou superior. O plugin deve ser modular, extensível e seguir as melhores práticas de desenvolvimento para plugins Better Auth.

## 2. Contexto

O Better Auth é uma biblioteca de autenticação e autorização agnóstica a frameworks para TypeScript, conhecida por seu ecossistema de plugins que estende suas funcionalidades. Este plugin visa preencher uma lacuna na conformidade com o PCI DSS, fornecendo um conjunto robusto de regras de senha que podem ser facilmente integradas e configuradas por usuários do Better Auth que lidam com dados de cartão de crédito.

## 3. Requisitos de Segurança de Senha (PCI DSS)

Com base na análise do documento fornecido e pesquisa complementar, o plugin deve implementar as seguintes políticas de senha:

### 3.1. Comprimento Mínimo da Senha

- **Requisito:** A senha deve ter um comprimento mínimo de 12 caracteres. Se o sistema não suportar 12 caracteres, um mínimo de 8 caracteres deve ser usado. O plugin deve priorizar 12 caracteres e permitir a configuração de um mínimo menor apenas como fallback.

### 3.2. Complexidade da Senha

- **Requisito:** A senha deve incluir uma combinação de letras e números. Para maior robustez e alinhamento com as melhores práticas de segurança, o plugin deve enforceçar a inclusão de:
    - Pelo menos uma letra maiúscula.
    - Pelo menos uma letra minúscula.
    - Pelo menos um número.
    - Pelo menos um caractere especial (ex: !@#$%^&*()).

### 3.3. Frequência de Troca de Senha

- **Requisito:** As senhas devem ser redefinidas a cada 90 dias, a menos que a autenticação contínua baseada em risco seja implementada. O plugin deve permitir a configuração deste período.

### 3.4. Histórico de Senhas

- **Requisito:** Não permitir a reutilização das últimas 4 senhas. O plugin deve manter um histórico das senhas anteriores do usuário para enforceçar esta regra, permitindo a configuração do número de senhas a serem lembradas.

### 3.5. Bloqueio de Conta por Tentativas Inválidas

- **Requisito:** Limitar o número de tentativas de login inválidas. O plugin deve:
    - Bloquear a conta após um número configurável de tentativas falhas (ex: 3 a 6 tentativas).
    - Permitir a configuração da duração do bloqueio (ex: 30 minutos).
    - O documento menciona 3 tentativas para `emailOTP` e 10 tentativas em 30 minutos para `rate-limiter.middleware.ts`. O plugin deve unificar e permitir a configuração dessas políticas.

### 3.6. Tratamento de Contas Inativas

- **Requisito:** Contas inativas devem ser desativadas ou removidas após 90 dias. O plugin deve fornecer mecanismos (hooks ou endpoints) para identificar e

desativar automaticamente contas que não tiveram atividade de login por um período configurável.

### 3.7. Requisitos para a Primeira Senha de Novos Usuários

- **Requisito:** A senha inicial de novos usuários deve ser única e deve ser alterada obrigatoriamente no primeiro acesso. O plugin deve suportar a geração de senhas temporárias e enforceçar a troca na primeira autenticação.

### 3.8. Outros Requisitos de Autenticação Relevantes

- **MFA (Multi-Factor Authentication):** Embora o plugin seja focado em senhas, ele deve ser compatível e não interferir com a implementação de MFA existente no Better Auth para acessos críticos.
- **Expiração de Sessão:** As sessões devem expirar após 15 minutos de inatividade. O plugin deve ser ciente e compatível com as configurações de expiração de sessão do Better Auth.
- **Armazenamento Seguro:** As senhas devem ser armazenadas usando um hash seguro (ex: scrypt, conforme já utilizado pelo Better Auth) com salt único por usuário. O plugin não deve manipular diretamente o armazenamento de senhas, mas deve garantir que as políticas de senha sejam aplicadas antes do hashing.
- **Verificação de Identidade:** A identidade do usuário deve ser verificada antes de permitir modificações em fatores de autenticação (ex: troca de senha). O plugin deve integrar-se com os mecanismos de verificação de identidade do Better Auth.

## 4. Arquitetura e Implementação do Plugin

O plugin deve ser desenvolvido em TypeScript e seguir a estrutura de plugins do Better Auth, idealmente como um *server plugin*.

### 4.1. Estrutura do Plugin

- O plugin deve ser um módulo TypeScript exportando um objeto que satisfaça a interface `BetterAuthPlugin`.
- Deve ter um `id` único (ex: `pci-dss-password-policy`).

- Pode incluir `endpoints` para funcionalidades específicas (ex: verificação de conformidade de senha, forçar troca de senha).
- Deve utilizar `hooks` do Better Auth para interceptar eventos de autenticação (ex: `onPreAuth`, `onPostAuth`, `onPasswordChange`) e aplicar as políticas de senha.
- Pode estender o `schema` do banco de dados para armazenar informações adicionais necessárias para as políticas (ex: `passwordHistory`, `lastPasswordChangeDate`, `failedLoginAttempts`, `accountLockedUntil`).

## 4.2. Configuração

- Todas as políticas (comprimento mínimo, complexidade, frequência de troca, histórico, tentativas de bloqueio, inatividade) devem ser configuráveis através de opções passadas ao plugin na inicialização do Better Auth.
- Exemplo de configuração esperada:

  ```typescript import { auth } from './lib/auth'; import { pciDssPasswordPolicy } from 'better-auth-pci-dss-plugin';

  export const auth = betterAuth({ // ... outras configurações plugins: [ // ... outros plugins pciDssPasswordPolicy({ minLength: 12, minUppercase: 1, minLowercase: 1, minNumbers: 1, minSpecialChars: 1, passwordHistoryCount: 4, passwordChangeIntervalDays: 90, maxFailedLoginAttempts: 5, accountLockoutDurationMinutes: 30, inactiveAccountDeactivationDays: 90, forcePasswordChangeOnFirstLogin: true, }), ] }); ```

## 4.3. Validação e Mensagens de Erro

- O plugin deve fornecer mensagens de erro claras e informativas quando uma senha não atender aos requisitos, indicando qual regra foi violada.
- As validações devem ocorrer antes que a senha seja armazenada ou utilizada para autenticação.

## 4.4. Testes

- O plugin deve incluir um conjunto abrangente de testes unitários e de integração para garantir que todas as políticas de senha sejam aplicadas corretamente e que o plugin se integre bem com o Better Auth.
  ```

## 5. Considerações Adicionais

- **Documentação:** O plugin deve ser bem documentado, explicando como instalá-lo, configurá-lo e quais são as opções disponíveis. Exemplos de uso devem ser fornecidos.
- **Compatibilidade:** O plugin deve ser compatível com as versões mais recentes do Better Auth e do TypeScript.
- **Código Aberto:** O código deve ser limpo, legível e seguir as convenções de código aberto para facilitar a contribuição da comunidade.
- **Segurança:** O desenvolvimento deve seguir as melhores práticas de segurança de software para evitar vulnerabilidades.

Este prompt serve como um guia detalhado para o desenvolvimento do plugin. O desenvolvedor deve usar sua expertise para refinar a implementação e garantir a máxima conformidade e usabilidade.