# A01:2021-Broken Access Control

**Exceto para recursos públicos, negar por padrão.**

O middleware Authentication, responsável pelo sistema de login, já faz a maioria do trabalho. Os recursos que não precisam de autenticação, estão listados na constante src/Controller/UsersController.php::ACTIONS\_SEM\_AUTENTICACAO

**Implemente mecanismos de controle de acesso uma vez e reutilize-os em todo o aplicativo, incluindo a minimização do uso de Cross-Origin Resource Sharing (CORS).**

Somente para o CRUD de Users é necessário um controle de autorização.

Código que aplica o controle: src/Controller/UsersController.php->beforeFilter().

CORS não é utilizado.

**Os controles de acesso ao modelo devem impor a propriedade do registro em vez de aceitar que o usuário possa criar, ler, atualizar ou excluir qualquer registro.**

O KAW tem dois tipos de usuário, o comum e o root. A única diferença entre eles é que o root pode executar ações consideradas de configuração do sistema. O restante das permissões são iguais.

**Os requisitos de limite de negócios de aplicativos exclusivos devem ser impostos por modelos de domínio.**

As validações de autenticação e autorização são feitas no controller, sempre levando em consideração o usuário autenticado no servidor. Nunca confiando nas informações enviadas do frontend.

**Desative a lista de diretórios do servidor da web e certifique-se de que os metadados do arquivo (por exemplo, o .git) e os arquivos de backup não estejam presentes nas raízes da web (web roots).**

Em README > Configuração inicial da aplicação > Ambiente de produção > Item 2.1, ensina a configurar o Apache2 para considerar a pasta kaw/webroot/ como a raiz da aplicação.

O arquivo kaw/.htaccess faz o trabalho de redirecionar para a pasta webroot/

**Registrar falhas de controle de acesso e alertar os administradores quando apropriado (por exemplo, falhas repetidas).**

Criada a Atividades Suspeitas que com base nos logs registrados, mostra para o usuário atividades danosas na aplicação.

Arquivos:

* Sistema de logs: kaw/src/Log/\*
* Atividades Suspeitas: kaw/templates/Pages/home.php (Home do sistema)

**Limite de taxa o acesso da API e do controlador para minimizar os danos do conjunto de ferramentas de ataque automatizado.**

Foi criado o módulo de IPs Bloqueados. Quando um usuário erra consecutivamente as credenciais de acesso o IP é bloqueado.

**Os identificadores de sessão com estado devem ser invalidados no servidor após o logout. Os tokens JWT sem estado devem ter vida curta, para que a janela de oportunidade para um invasor seja minimizada. Para JWTs de longa duração, é altamente recomendável seguir os padrões OAuth para revogar o acesso.**

Melhorado a segurança da Session e Cookies. Ler o arquivo kaw/config/app.examplo.php > ‘Session’

# A02:2021-Cryptographic Failures

**Classifique os dados processados, armazenados ou transmitidos por um aplicativo. Identifique quais dados são confidenciais de acordo com as leis de privacidade, requisitos regulamentares ou necessidades de negócios.**[falta implementar]

**Não armazene dados confidenciais desnecessariamente. Descarte-o o mais rápido possível ou use tokenização compatível com PCI DSS ou mesmo truncamento. Os dados não retidos não podem ser roubados.**Por ser o chaveiro, a maioria dos dados são críticos. Mas uma das filosofias do projeto é sempre mantê-lo o mais enxuto possível.

**Certifique-se de criptografar todos os dados confidenciais armazenados.**Todas as informações críticas da entidade Entrada são criptografadas.

**Certifique-se de que algoritmos, protocolos e senhas de padrão forte e atualizados estejam em vigor; use o gerenciamento de senhas adequado.**

* Usando a Lib Sodium para criptografar e descriptografar
* Gerando a chave de criptografia assim: sodium\_bin2hex(sodium\_crypto\_secretbox\_keygen())
* Usando variáveis de ambiente do Apache2 para armazenar a chave de criptografia.

**Criptografe todos os dados em trânsito com protocolos seguros, como TLS com cifras de sigilo de encaminhamento (FS), priorização de cifras pelo servidor e parâmetros seguros. Aplique a criptografia usando diretivas como HTTP Strict Transport Security (HSTS).**Forçado a conexão HTTPS e aplicando o HSTS: kaw/src/Application.php

**Desative o armazenamento em cache para respostas que contenham dados confidenciais.**Feito no commit f88e4e5979556c05b5b1b6d1e37eaecb8fad056a

**Aplique os controles de segurança necessários de acordo com a classificação de dados.**[falta implementar]

**Não use protocolos legados, como FTP e SMTP, para transportar dados confidenciais.**A ferramenta ainda não utiliza SMTP para transportar dados.

**Armazene senhas usando fortes funções de hash adaptáveis e saltadas com um fator de trabalho (fator de atraso), como Argon2, scrypt, bcrypt ou PBKDF2.**Implementando a criptografia utilizando a Lib Sodium: kaw/src/Model/Entity/Entrada.php

**Os vetores de inicialização devem ser escolhidos de acordo com o modo de operação. Para muitos modos, isso significa usar um CSPRNG (gerador de números pseudo-aleatórios criptograficamente seguro). Para modos que requerem um nonce, o vetor de inicialização (IV) não precisa de um CSPRNG. Em todos os casos, o IV nunca deve ser usado duas vezes para uma chave fixa.**Gerado um nonce único para cada criptografia, com o tamanho da constante SODIUM\_CRYPTO\_SECRETBOX\_NONCEBYTES, através da função random\_bytes(). Código: kaw/src/Model/Entity/Entrada.php:123

**Sempre use criptografia autenticada em vez de apenas criptografia.**Utilizando a função sodium\_crypto\_secretbox() já atende esse requisito.

**As chaves devem ser geradas de forma criptograficamente aleatória e armazenadas na memória como um *array* de *bytes*. Se uma senha for usada, ela deve ser convertida em uma chave por meio de uma função de derivação de chave de base de senha apropriada.**No READM, na seção AMBIENTE DE PRODUÇÃO, no tópico ‘Configure variáveis de ambiente.’, é ensinado como criar variáveis de ambiente no Apache 2 para fazer a criptografia das informações.No README, na seção GERANDO CHAVES DE SEGURANÇA, é recomendado usar a função sodium\_bin2hex(sodium\_crypto\_secretbox\_keygen()) para gerar a senha de criptografia.

**Certifique-se de que a aleatoriedade criptográfica seja usada quando apropriado e que não tenha sido usada uma semente de uma forma previsível ou com baixa entropia. A maioria das APIs modernas não exige que o desenvolvedor propague o CSPRNG para obter segurança.**Para gerar o NONCE da função sodium\_crypto\_secretbox(), é utilizado o random\_bytes(SODIUM\_CRYPTO\_SECRETBOX\_NONCEBYTES); Conforme orientação da documentação https://www.php.net/manual/en/function.sodium-crypto-secretbox

**Evite funções criptográficas e esquemas de preenchimento obsoletos, como MD5, SHA1, PKCS número 1 v1.5.**A função sodium\_crypto\_secretbox() implementa o algoritmo XSalsa20, um dos mais modernos. https://libsodium.gitbook.io/doc/advanced/stream\_ciphers/xsalsa20

**Verifique de forma independente a eficácia das configurações.**[falta implementar]

[**A03:2021-Injection**](https://owasp.org/Top10/A03_2021-Injection/)

[falta implementar]

[**A04:2021-Insecure Design**](https://owasp.org/Top10/A04_2021-Insecure_Design/)

[falta implementar]

[**A05:2021-Security Misconfiguration**](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)

[falta implementar]

[**A06:2021-Vulnerable and Outdated Components**](https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/)

[falta implementar]

[**A07:2021-Identification and Authentication Failures**](https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/)

[falta implementar]

[**A08:2021-Software and Data Integrity Failures**](https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/)

[falta implementar]

[**A09:2021-Security Logging and Monitoring Failures**](https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/)

[falta implementar]

[**A10:2021-Server-Side Request Forgery**](https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/)

[falta implementar]