Proteção de sessão pós-autenticação com *device fingerprinting* e *nonce* com *timestamp*: um estudo aplicado ao Sistema de Telemedicina e Telessaúde

Victória Pacheco Lobo[1], **Douglas D. J. de Macedo**[1]

[1] Departamento de Ciência da Informação — Centro de Ciências da Educação
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

victoriapachecolobo@gmail.com, douglas.macedo@ufsc.br

Abstract. *After successful authentication, web sessions remain exposed to session hijacking, in which an attacker reuses a captured access token, and replay attacks, in which an authenticated request is resent and accepted. This work implements and validates two post-authentication protection mechanisms in the Telemedicine and Telehealth System (STT): device fingerprinting, which binds the token to the originating device, and request control with nonce and timestamp, which ensures uniqueness and temporal validity. Attack simulations show that the system, which was previously vulnerable to both threats, now rejects requests from a different device and duplicated or expired requests.*

Resumo. *Após a autenticação, a sessão web permanece exposta ao sequestro de sessão, em que um atacante reutiliza um token de acesso capturado, e ao ataque de replay, em que uma requisição autenticada é reenviada e aceita. Este trabalho implementa e valida dois mecanismos de proteção pós-autenticação no Sistema de Telemedicina e Telessaúde (STT): o device fingerprinting, que vincula o token ao dispositivo de origem, e o controle de requisições com nonce e timestamp, que garante unicidade e validade temporal. Simulações de ataque mostram que o sistema, que antes estava vulnerável a ambas as ameaças, passa a rejeitar requisições de outro dispositivo e requisições duplicadas ou expiradas.*

1. Introdução

A digitalização dos serviços de saúde no Brasil, acelerada pela pandemia de COVID-19 e consolidada por marcos regulatórios como a Lei nº 13.989/2020 [BRASIL 2020] e a Lei nº 14.510/2022 [BRASIL 2022], reduziu o tempo de espera, diminuiu deslocamentos e ampliou o acesso a especialistas, sobretudo em áreas afastadas dos grandes centros [Caetano et al. 2020]. Essa digitalização, porém, também expôs o setor a um volume crescente de ciberataques, figurando entre os principais alvos e com o maior custo médio de violação de dados no mundo [Verizon Business 2025, IBM Security 2025]. No Brasil, a Lei Geral de Proteção de Dados (LGPD) classifica informações de saúde como dados sensíveis e prevê sanções significativas em caso de vazamento [BRASIL 2018]; episódios como a alegada exposição de dados de cerca de 205 milhões de usuários do sistema CADSUS reforçam a importância de práticas robustas de segurança nesse setor [Franco et al. 2025].

Mecanismos como a autenticação multifator tornam o login mais robusto, mas protegem apenas o momento de entrada no sistema. Após o login bem-sucedido, a segurança passa a depender dos mecanismos de proteção da sessão, e mesmo sistemas com MFA podem ter a sessão posteriormente comprometida [NIST 2025, Microsoft 2022]. É nesse cenário pós-autenticação que se concentram duas ameaças. O **sequestro de sessão**, em que um atacante que captura um *token* válido por *sniffing*, XSS ou adivinhação passa a agir no lugar do usuário legítimo [Hwang et al. 2022]; e o **ataque de *replay***, no qual uma requisição já autenticada é reenviada ao servidor, que a aceita sem perceber que já foi processada [NIST 2025].

Este trabalho volta-se ao Sistema de Telemedicina e Telessaúde de Santa Catarina (STT), do Laboratório de Telemedicina da UFSC, considerado a principal iniciativa pública de telemedicina da América Latina [Macedo et al. 2015], que atende todos os estados do Brasil e já registra mais de 400.000 laudos de teledermatologia e 436.000 teleconsultorias respondidas [Saúde Digital UFSC 2025]; o alcance nacional e o volume de dados clínicos sensíveis tornam o sistema um objeto de estudo relevante. A preocupação com a segurança o acompanha desde os primeiros anos: já em 2008, [Wallauer et al. 2008] apontavam o usuário como o ponto mais frágil da autenticação por senha, e pesquisas recentes do PGCIN/UFSC investigaram dados de proveniência para auditoria e conformidade com a LGPD [Sembay et al. 2024]. O objetivo geral é implementar mecanismos de proteção da sessão pós-autenticação no STT, combinando *device fingerprinting* e controle de requisições com *nonce* e *timestamp*. Os objetivos específicos são: (1) revisar os mecanismos de ataque e proteção de sessões web, com ênfase em sequestro e *replay*; (2) analisar o fluxo de sessão do STT e identificar fragilidades; e (3) desenvolver e validar os mecanismos, demonstrando sua viabilidade técnica para futura adoção. A contribuição deste trabalho não está em propor novos atributos ou técnicas de *fingerprinting*, e sim em medir o custo, em desempenho e em usabilidade, do vínculo entre *token* e dispositivo quando construído sob a restrição de minimização de dados da LGPD, em um sistema de saúde digital real e em uso. Para isso, a solução integra *device fingerprinting* e controle de *replay* com *nonce* e *timestamp* em um único fluxo REST/JWT, tratando conjuntamente dois ataques usualmente abordados em separado, e emprega APIs consolidadas como fontes de *fingerprint* (Canvas, WebGL e AudioContext), em vez de fontes cuja entropia se reduziu, como o CSS3 do SHPF, ou de identificadores inacessíveis via JavaScript, como os do ATEUI.

O artigo segue com o referencial e os trabalhos relacionados (Seção 2), o método (3), a análise do sistema atual (4), a implementação (5), os resultados e a discussão (6) e as conclusões (7).

2. Referencial e Trabalhos Relacionados

2.1. Sequestro de sessão e suas defesas

O sequestro de sessão consiste em assumir o controle não autorizado de uma sessão já autenticada [Kamal 2016]. Após o login, o servidor emite um *token* que o cliente apresenta nas requisições seguintes; o servidor reconhece o usuário pelo *token*, sem verificar quem o apresenta, de modo que qualquer entidade que o possua pode agir no lugar do usuário legítimo [Hwang et al. 2022, Jones and Hardt 2012]. Na camada de aplicação, o *token* pode ser capturado por *sniffing*, força bruta ou XSS [Baitha and Vinod 2018]. Defesas tradicionais (HTTPS, filtragem por IP, expiração e rotação de *token*) mostram-se

insuficientes, pois o atacante pode usar o *token* enquanto ele for válido ou reenviar o valor capturado [Hwang et al. 2022]. Essas defesas têm em comum o fato de apenas reduzirem a janela de exploração, sem atacar a causa do problema, que é a transferibilidade do *token*: nenhuma delas verifica *de onde* a requisição se origina. É essa verificação de origem que os mecanismos analisados a seguir buscam oferecer.

2.2. *Device fingerprinting*

A RFC 6973 define *fingerprinting* como o processo de identificar, com probabilidade suficientemente alta, um dispositivo ou instância de aplicação a partir de múltiplos elementos de informação [Cooper et al. 2013]. Um *browser fingerprint* reúne características de hardware, sistema operacional, navegador e configurações, coletadas por *scripts* que consultam APIs públicas e cabeçalhos HTTP [Laperdrix et al. 2020]. Embora estudado predominantemente como ameaça à privacidade desde [Mayer 2009], suas mesmas propriedades, a combinação de atributos diversos e a estabilidade entre sessões, viabilizam o uso construtivo na proteção de sessões [Laperdrix et al. 2020].

Três APIs destacam-se pela elevada entropia, com forças complementares. A API Canvas, proposta por [Mowery and Shacham 2012], desenha texto e formas em uma superfície gráfica e extrai os pixels resultantes, que variam conforme sistema, navegador, GPU e fontes; sua entropia é alta, mas depende do navegador e pode ser perturbada por proteções de privacidade que inserem ruído. A API WebGL identifica o hardware gráfico; [Cao et al. 2017] alcançaram identificação única de 99,24% intra-navegador e estabilidade de 91,44% entre navegadores na mesma máquina, a fonte mais robusta para distinguir dispositivos físicos. A Web Audio API, documentada em produção por [Englehardt and Narayanan 2016], gera um sinal processado por um compressor dinâmico que varia entre máquinas, com entropia independente das fontes gráficas, mas sujeita ao mesmo ruído do Canvas. Nenhuma fonte basta isoladamente: a robustez vem da combinação, com tratamento explícito da instabilidade das fontes ruidosas. O uso positivo para segurança já foi explorado no SHPF [Unger et al. 2013] e está na base de bibliotecas como a FingerprintJS [FingerprintJS 2025, Fingerprint 2025]. O *device fingerprinting* permite verificar se a requisição vem do ambiente que iniciou a sessão; não cobre, contudo, o reenvio de uma requisição inteira já validada, tratado na próxima subseção.

2.3. Ataques de *replay*, *nonce* e *timestamp*

O reenvio de uma requisição inteira já validada é o chamado ataque de *replay*. Ele difere do sequestro porque, em vez de montar novas requisições com o *token* roubado, o atacante captura uma requisição legítima e a reenvia sem modificação [NIST 2025]. Por ser uma cópia exata, todos os seus elementos são válidos, inclusive o *fingerprint* do dispositivo, de modo que apenas sua verificação aceitaria a requisição.

Para mitigar esse problema, o NIST observa que protocolos que usam *nonces* ou dados de atualidade resistem a esse tipo de ataque, pois o verificador percebe quando uma mensagem antiga é reenviada [NIST 2025]. Um *nonce* é um valor que nunca se repete com a mesma chave; o servidor o associa a cada requisição e, ao recebê-la, rejeita-a caso aquele valor já tenha sido processado. As soluções diferem no custo dessa memória: um *nonce* isolado obriga o servidor a guardar indefinidamente todos os valores já vistos, e *nonces* dependentes de interação do usuário, como em [Uma and Kannan 2013], restringem a proteção a fluxos interativos. A RFC 5849 [Hammer-Lahav 2010], que especifica

o OAuth 1.0, documenta a combinação de *nonce* com *timestamp* para contornar a primeira limitação, permitindo ao servidor descartar requisições antigas pelo *timestamp* e manter em memória apenas os *nonces* da janela de validade, ao custo de exigir tolerância a desvios de relógio entre cliente e servidor.

2.4. Trabalhos Relacionados

Cinco trabalhos fundamentam esta proposta e diferem nos dois eixos que a definem. No vínculo com o dispositivo, o SHPF [Unger et al. 2013] é o mais próximo, combinando *fingerprinting* de navegador e proteção contra *replay*, mas com fontes que envelheceram (CSS3) e arquitetura presa a sessões HTTP; o ATEUI [Hwang et al. 2022] recorre a identificadores fixos de hardware (BIOS, placa-mãe, UUID), hoje inacessíveis via JavaScript e problemáticos perante a LGPD; e [Cao et al. 2017] demonstram a viabilidade das fontes atuais, mas visando identificação persistente entre navegadores, escopo oposto ao vínculo por sessão adotado aqui. No controle de *replay*, [Uma and Kannan 2013] combinam *nonce* dinâmico e *timestamp*, porém em SOAP/WS-Security e com *nonce* dependente de interação do usuário. [Saraiva et al. 2014] sistematizam os conceitos, sem implementar um mecanismo de proteção. O Quadro 1 sintetiza a comparação.

Quadro 1. Comparação dos trabalhos relacionados.

Trabalho	**Ataque**	**Mecanismo / tecnologia**	**Limitação / lacuna**
[Unger et al. 2013]	Sequestro + *replay*	SHPF: CSS3, PHP5, framework *server-side*	Atributos com baixa entropia hoje
[Hwang et al. 2022]	Sequestro	ATEUI: hardware fixo + cifra dupla	Identificadores inacessíveis em navegadores atuais
[Uma and Kannan 2013]	*Replay*	*Nonce* dinâmico + *timestamp*; SOAP/WS-Security	*Nonce* depende de interação do usuário
[Cao et al. 2017]	Identificação	*Fingerprinting* entre navegadores (SO/hardware)	Foco em identificação persistente
Este trabalho	**Sequestro + *replay***	**Canvas, WebGL, AudioContext, *nonce+timestamp*; REST/JWT, Redis**	**Comparação exata pode gerar falsos positivos em cenários não medidos**

Do conjunto analisado emerge a lacuna que este trabalho busca ocupar: nenhum dos trabalhos combina, simultaneamente, (i) fontes de *fingerprint* de entropia elevada acessíveis nos navegadores atuais; (ii) proteção conjunta contra sequestro e *replay* em arquitetura REST/JWT; e (iii) avaliação do custo dessa proteção, em desempenho e estabilidade, sob a minimização de dados da LGPD, em um sistema real em uso.

3. Método

A pesquisa caracteriza-se como aplicada e exploratória, pois parte da identificação de vulnerabilidades em um sistema real e propõe mecanismos concretos para mitigá-las, organizando-se em quatro etapas: revisão da literatura em segurança da informação e sistemas web; análise do mecanismo atual de sessões do STT, por inspeção do código-fonte e observação de requisições em ambiente de desenvolvimento; desenvolvimento dos dois mecanismos, em JavaScript (Node.js no servidor e React no cliente), com persistência no Redis; e validação por simulação de cenários de ataque, comparando o comportamento do sistema antes e depois da implementação.

A validação organiza-se em dois cenários. No primeiro, o *token* de uma sessão autenticada é copiado e reapresentado a partir de outro ambiente (troca de navegador na mesma máquina e troca de dispositivo físico), simulando o sequestro. No segundo, uma requisição completa, com *token* e *fingerprint*, é reenviada sem modificação, simulando o *replay*. Como o STT foi executado em ambiente local, o teste com dois dispositivos exigiu expor temporariamente os serviços por um túnel HTTP (Cloudflare Tunnel), restrito ao ambiente de teste. Todos os testes foram realizados em ambiente de desenvolvimento, com contas de teste e sem acesso a dados reais de pacientes. Complementarmente, avaliou-se a estabilidade do *fingerprint* em oito dispositivos físicos, regenerando o *hash* após recarga, redimensionamento da janela, reabertura do navegador e reinicialização da máquina.

Para fins de transparência, declara-se que assistentes de IA generativa (Claude, da Anthropic, e Gemini, do Google) apoiaram a revisão da redação e a depuração do código, e o Google Tradutor auxiliou a leitura de artigos em inglês. A concepção da pesquisa, a análise das vulnerabilidades, o desenvolvimento, a interpretação dos resultados e as conclusões são de responsabilidade dos autores, que revisaram todo o conteúdo produzido com apoio dessas ferramentas.

4. Análise do Sistema Atual

Por se tratar de um sistema real, o comportamento dos *endpoints* é descrito em nível conceitual, sem reproduzir rotas, esquemas de requisição ou valores de *token*, preservando o valor demonstrativo dos testes sem facilitar a exploração do sistema.

A comunicação segue o modelo REST sobre HTTPS, com mensagens em JSON. No login, o servidor retorna um `access_token` JWT que opera como *bearer token*: sua apresentação no cabeçalho `Authorization` basta para conceder acesso, e a validação limita-se à integridade e à validade temporal do *token*.

Dois testes em ambiente de desenvolvimento evidenciaram as implicações dessa limitação. No primeiro, o *token* de uma sessão autenticada no Chrome foi reapresentado em uma aba privada do Edge e em um segundo dispositivo físico: em ambos os casos a sessão do administrador carregou sem restrição, confirmando a **ausência de proteção contra sequestro de sessão**. No segundo, cinco requisições `GET` idênticas foram enviadas a um *endpoint* protegido com o mesmo *token*; o servidor respondeu HTTP 200 a todas, confirmando a **ausência de proteção contra *replay***. Os testes não executam, por si, um ataque real, mas demonstram que o sistema não dispõe dos mecanismos que impediriam tais ataques caso um *token* ou uma requisição fossem capturados.

5. Implementação

5.1. Modelo de ameaça

O que se protege é a sessão autenticada do usuário do STT, ou seja, o *token* de acesso gerado no login e os dados clínicos que ele libera. O atacante considerado é alguém que obteve um *token* válido de forma isolada (por vazamento em *logs*, cópia do armazenamento do navegador ou captura em trânsito) e tenta reutilizá-lo. Como o *fingerprint* não é um valor armazenado, mas recalculado a cada requisição a partir do ambiente do dispositivo, um *token* reapresentado em outra máquina gera um *fingerprint* distinto do registrado

no login, e a sessão é recusada. O *nonce* cobre o caso em que a requisição inteira é reenviada, hipótese em que o *fingerprint* capturado também seria válido. Ficam fora desse alcance o *phishing*, em que o próprio usuário fornece as credenciais; XSS executado no dispositivo legítimo, de onde *token*, *fingerprint* e *nonce* podem ser produzidos; e cenários em que o atacante reproduz por inteiro o ambiente de origem após romper o TLS. Mecanismos como mTLS [Campbell et al. 2020], *token binding* [Popov et al. 2018] e DPoP [Fett et al. 2023] dão garantia mais forte, ao exigir também uma chave privada, mas demandam mudança de protocolo e gestão de chaves no cliente, incompatível com a adoção incremental sobre o REST/JWT existente.

5.2. Arquitetura da solução

Os mecanismos distribuem-se por três camadas: o *frontend* de autenticação, uma biblioteca que centraliza a geração de *fingerprint* e *nonce* e o serviço de autenticação no *backend*. Ambos são armazenados no Redis, cache em memória que o STT já utiliza: o *fingerprint* fica vinculado à sessão e é descartado ao fim dela; cada *nonce* é gravado com TTL igual à janela de frescor, em linha com a RFC 5849 [Hammer-Lahav 2010]. A verificação atua, por padrão, no fluxo central da sessão (revalidação e renovação do *token*); para estendê-la às rotas dos módulos, uma lista configurável é consultada pelo ForwardAuth do Traefik, que desvia cada requisição a um *endpoint* de autorização. A decisão concentra a verificação no ponto de autorização centralizada, evitando sobrecarga nas rotas de alto volume, ao custo de configuração explícita por rota adicional.

5.3. Geração do *fingerprint* no cliente

A função `gerarFingerprint()` coleta oito atributos selecionados por três critérios: estabilidade intra-sessão, contribuição não-redundante e minimização de dados (princípio da necessidade do art. 6º, III, da LGPD), concatena seus valores e produz um *hash* SHA-256 via Web Crypto API (Código 1). Os atributos derivados da renderização de hardware (Canvas, WebGL e AudioContext) seguem o padrão da FingerprintJS [FingerprintJS 2025]. As funções de Canvas e áudio geram o valor duas vezes e o comparam: quando navegadores com proteções de privacidade inserem ruído aleatório e os valores divergem, retornam o marcador `'unstable'`, em vez de um *hash* que mudaria a cada requisição; quando a API não existe, retornam `'unsupported'`. Apenas o *hash* de 64 caracteres chega ao servidor, atendendo ao princípio da minimização. O atributo de proporção de tela proposto por [Cao et al. 2017], embora inicialmente implementado, foi removido porque o redimensionamento da janela alterava o *fingerprint* e deslogava o usuário legítimo. Pelo mesmo motivo, o *userAgent* é normalizado com a remoção dos números de versão: como os navegadores se atualizam automaticamente em ciclos de poucas semanas, o valor bruto invalidaria sessões legítimas de toda a base a cada atualização. A normalização preserva a família do navegador e o sistema operacional, mantendo a detecção de troca de navegador ou de dispositivo.

```
const data = {
    userAgent: _normalizarUA(navigator.userAgent),
    platform: navigator.platform || 'unknown',
    timezone: Intl.DateTimeFormat().resolvedOptions().timeZone,
    colorDepth: screen.colorDepth,
    hardwareConcurrency: navigator.hardwareConcurrency || 'unknown',
    canvasFingerprint: await _canvasFingerprint(),
```

```
    webGLFingerprint: await _webGLFingerprint(),
    audioFingerprint: await _audioFingerprint(),
};
const concatenado = Object.values(data).join('||');
// hash SHA-256 do valor concatenado via Web Crypto API
const hashBuffer = await crypto.subtle.digest('SHA-256',
    new TextEncoder().encode(concatenado));
return Array.from(new Uint8Array(hashBuffer))
    .map(b => b.toString(16).padStart(2, '0')).join('');
```

Código 1. Núcleo da função `gerarFingerprint` (cliente).

No login, o *frontend* envia o *fingerprint* em um cabeçalho `X-Device-Fingerprint`; o servidor o grava na sessão, no Redis, como `id_dispositivo`. A cada requisição protegida, o servidor compara, por igualdade exata, o *fingerprint* recebido com o gravado e rejeita a sessão (HTTP 401) em caso de divergência. O mesmo identificador é propagado ao fluxo de renovação do *token*.

5.4. Proteção contra *replay* com *nonce* e *timestamp*

Cada requisição carrega um *nonce* gerado no cliente: um `timestamp` concatenado a um UUID v4, em Base64 (Código 2). O `crypto.randomUUID()` garante unicidade mesmo no mesmo milissegundo, e o `timestamp` embutido permite descartar requisições antigas sem armazenar *nonces* indefinidamente.

```
const timestamp = Date.now().toString();
const nonce = btoa(`${timestamp}:${crypto.randomUUID()}`);
```

Código 2. Geração do *nonce* no cliente.

A validação no servidor executa quatro verificações em sequência: presença do cabeçalho; decodificação do Base64 e extração de um `timestamp` numérico; janela temporal de cinco minutos (via `Math.abs`, com tolerância a desvios de relógio [Hammer-Lahav 2010]); e unicidade no Redis, por uma única operação atômica `SET nonce:<valor> '1' EX 300 NX`, que cria a chave apenas se ela não existir [Redis 2024], eliminando condições de corrida. O TTL de 300 s é igual à janela temporal, pois *nonces* mais antigos já seriam rejeitados pelo *timestamp*. A Figura 1 sintetiza a validação combinada na primeira requisição protegida.

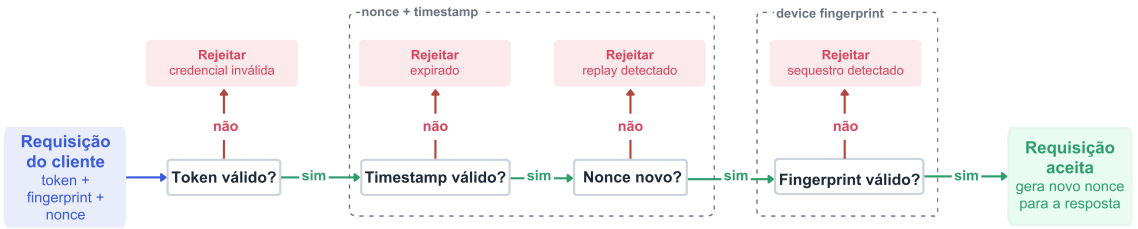

Figura 1. Fluxo de validação combinada de *fingerprint* e *nonce* na primeira requisição protegida após o login.

6. Resultados e Discussão

6.1. Validação contra sequestro de sessão

Reproduzindo o cenário de [Baitha and Vinod 2018], em que um atacante de posse de um *token* válido tenta operar a sessão de outro ambiente, a validação foi feita em dois momentos. Na troca de navegador na mesma máquina, ao reapresentar o *token* do Chrome no Edge, a primeira requisição ao *endpoint* de revalidação comparou o *fingerprint* do Edge com o `id_dispositivo` gravado no login, detectou a divergência e respondeu HTTP 401 ("Sessão inválida. Faça login novamente."), redirecionando para o login. Antes da implementação, o mesmo *token* funcionava sem restrição.

Como Chrome e Edge compartilham motor e hardware, a divergência vem sobretudo do User-Agent. Para exercitar os atributos de hardware, o cenário foi repetido em quatro dispositivos físicos distintos: o *fingerprint* divergiu do gravado no login, como esperado dos atributos de hardware (entre eles o identificador da GPU via WebGL), e a sessão foi rejeitada, sem falsos negativos nas repetições. A Figura 2 ilustra um desses casos.

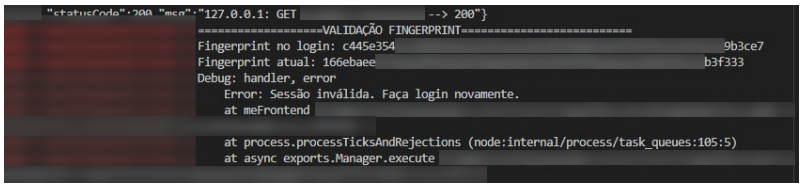

Figura 2. *Log* do servidor em um dos testes com dispositivos distintos: o *fingerprint* do login, o *fingerprint* atual divergente e a rejeição da sessão.

6.2. Validação contra ataque de *replay*

Uma requisição real ao *endpoint* de revalidação foi capturada no DevTools, e seus cabeçalhos (`Authorization`, `X-Device-Fingerprint` e `X-Request-Nonce`) foram reenviados cinco vezes por um *script* de console, em dois momentos. Dentro da janela de cinco minutos, as cinco requisições foram bloqueadas com "Requisição já processada": o *nonce* capturado já havia sido consumido pela requisição legítima e registrado no Redis, falhando a unicidade. Mais de cinco minutos após a captura, foram bloqueadas com "Requisição expirada. Faça login novamente.", pela verificação temporal, antes mesmo de a unicidade ser avaliada, confirmando que as duas verificações estão ativas. Antes da implementação, a mesma requisição podia ser reenviada repetidamente.

A extensão da proteção foi verificada em uma rota de módulo real submetida ao ForwardAuth: uma requisição com *fingerprint* divergente foi rejeitada com HTTP 401, e o reenvio com o mesmo *nonce* também, confirmando que a verificação opera no ponto de autorização centralizada, e não apenas no *endpoint* de revalidação.

6.3. Estabilidade do *fingerprint* em condições benignas

Para caracterizar a taxa de estabilidade relatada apenas empiricamente durante o desenvolvimento, o *fingerprint* foi regenerado em oito máquinas físicas distintas sob quatro condições benignas: recarga da página, redimensionamento da janela e reabertura do navegador, com cinco repetições por condição, e reinicialização do sistema operacional,

com três, totalizando 144 medições. Em todas elas, o *hash* foi idêntico ao registrado originalmente (100% de estabilidade), sem qualquer atributo divergente, confirmando que os atributos selecionados são insensíveis à geometria da janela e persistem entre execuções do navegador e reinicializações.

A única condição que produziu divergência foi o modo responsivo das ferramentas de desenvolvedor, reproduzida em todas as máquinas: o navegador troca o *userAgent* real pelo do dispositivo escolhido, o *hash* regenerado deixa de coincidir com o do *login* e a sessão é encerrada. O comportamento é esperado, pois o servidor não distingue essa mudança de um *token* usado em outro dispositivo, e restringe-se a um cenário de desenvolvimento, improvável no uso clínico. Complementarmente, observou-se que a troca do monitor em configuração multi-tela na mesma máquina não alterou o fingerprint, mantendo a sessão ativa. Atualizações de *driver* e GPUs híbridas não foram testadas.

6.4. Avaliação de impacto sobre o desempenho

Para dimensionar o custo dos mecanismos, mediu-se o tempo de geração do *fingerprint* no cliente, o de validação do *nonce* no servidor e, como referência, o tempo total de uma requisição de leitura de sessão (Tabela 1). A geração do *fingerprint* levou em média 23,71 ms ($n = 30$ execuções independentes, descartadas três iniciais de aquecimento) e ocorre uma única vez por sessão; nas requisições seguintes, o valor vem do cache em menos de 0,01 ms. A validação do *nonce* levou em média 1,74 ms por requisição ($n = 100$; mediana de 0,97 ms), mantida baixa pelo Redis em memória, ante 7,22 ms (mediana de 5 ms) da requisição de referência; os desvios-padrão refletem picos isolados do ambiente de execução. Em ambos os pontos o custo é pequeno diante do ganho de segurança: no cliente, amortizado ao longo da sessão; no servidor, mediana abaixo de 1 ms, inferior à variação de latência da rede.

Tabela 1. Tempo medido nos pontos críticos da solução (em milissegundos).

Métrica	n	**Média**	**Mín.**	**Máx.**	**Desv. pad.**
Geração do *fingerprint* (cliente)	30	23,71	14,90	38,70	5,80
Validação do *nonce* (servidor)	100	1,74	0,62	38,76	4,22
Requisição de leitura de sessão (total)	100	7,22	4,00	46,00	5,93

6.5. Discussão

Os testes confirmam o comportamento esperado: requisições de outro navegador ou dispositivo são rejeitadas pelo *fingerprint*, e requisições reenviadas, pelo controle de *replay*; antes, o sistema aceitava ambos os cenários, com custo agregado pequeno. Os resultados, contudo, foram obtidos em ambiente de desenvolvimento, com amostras de até $n = 100$ que caracterizam a tendência central e a dispersão do custo, não seu comportamento sob concorrência ou carga em produção, e devem ser entendidos como demonstração de viabilidade técnica.

Há uma tensão de projeto entre minimização de dados e robustez: ao reduzir os oito atributos a um único *hash* SHA-256, atende-se ao princípio da necessidade (art. 6º, III, da LGPD), pois o servidor jamais recebe os atributos brutos; em contrapartida, a alteração de um único atributo produz um *hash* inteiramente distinto, sem noção de proximidade. Embora as condições medidas na Seção 6.3 não tenham produzido divergências,

cenários não exercitados, como GPUs híbridas e atualizações de *driver*, podem alterar o *fingerprint* entre sessões e gerar falsos positivos. Uma comparação atributo a atributo, com tolerância parcial, mitigaria esses casos, mas exigiria transmitir e armazenar os atributos individualmente, reequilibrando a relação com a minimização. Medir a frequência real desses falsos positivos na base do STT é o passo prévio a qualquer adoção em produção.

Cabe explicitar por que o *fingerprint* atua como sinal de risco, e não como identificador único. Este trabalho não mediu entropia própria; estudos de larga escala reportam ao menos 18,1 bits de entropia por navegador [Eckersley 2010], 89,4% de *fingerprints* únicos entre 118.934 coletados [Laperdrix et al. 2020] e, em [Cao et al. 2017], 99,24% de unicidade intra-navegador. A garantia aqui exigida é mais fraca que a desses cenários de identificação global: a comparação ocorre apenas dentro de uma mesma sessão, entre o *hash* do login e o recalculado a cada requisição, descartado ao fim dela; os *hashes* das oito máquinas da Seção 6.3 foram mutuamente distintos. O cabeçalho expõe apenas o *hash*, não reversível aos atributos; por trafegar a cada requisição, porém, ele é reapresentável, o que o controle de *replay* cobre na janela temporal e reforça, conforme [Mello et al. 2022], que o identificador apoia a análise de risco sem ser, por si, um fator de autenticação.

A adoção em produção depende ainda de atualizar a política de privacidade do STT para informar a coleta de atributos do navegador e de documentar o teste de balanceamento de legítimo interesse previsto pela ANPD [ANPD 2024]. Há também uma limitação de escopo: a geração do *fingerprint* está acoplada ao login local, de modo que sessões iniciadas por provedores externos, como o gov.br, ficam fora do vínculo de dispositivo, o que demanda estender o mecanismo a esses fluxos. A proposta dialoga com o ATEUI [Hwang et al. 2022] e o SHPF [Unger et al. 2013], dos quais herda o princípio de verificar o dispositivo de origem, diferenciando-se nas fontes de entropia atuais e na arquitetura REST/JWT.

7. Conclusões e Trabalhos Futuros

Este trabalho implementou e validou dois mecanismos de proteção da sessão pós-autenticação no STT: o *device fingerprinting* e o controle de requisições com *nonce* e *timestamp*. O sistema, antes vulnerável a sequestro de sessão e a *replay*, passa a rejeitar requisições antes aceitas, com custo de poucos milissegundos e *fingerprint* estável nas condições medidas. A contribuição central está na combinação dos dois mecanismos, com fontes acessíveis nos navegadores atuais, em uma solução REST/JWT validada em um sistema de saúde digital real em uso.

As limitações são: (1) validação restrita ao ambiente de desenvolvimento, sem homologação ou produção e sem cenários de GPU híbrida ou atualização de *driver*; (2) cobertura restrita a sequestro e *replay*; e (3) rigidez da comparação exata, que pode recusar sessões legítimas.

Como trabalhos futuros, destacam-se medir a frequência de falsos positivos; explorar hashing tolerante a similaridade, que permite a comparação de proximidade preservando a minimização de dados; estendê-lo aos logins externos; migrar para vínculo criptográfico (DPoP [Fett et al. 2023]); e adotar autenticação adaptativa, em que a divergência aciona um fator adicional em vez de encerrar a sessão.

Referências

ANPD (2024). *Guia Orientativo das Hipóteses Legais de Tratamento de Dados — Legítimo Interesse*. Autoridade Nacional de Proteção de Dados, Brasília. Acesso em: 17 abr. 2026.

Baitha, A. K. and Vinod, S. (2018). Session hijacking and prevention technique. *International Journal of Engineering and Technology*, 7(2.6):193–198. DOI: 10.14419/ijet.v7i2.6.10566. Acesso em: 23 mar. 2026.

BRASIL (2018). Lei nº 13.709, de 14 de agosto de 2018. Lei Geral de Proteção de Dados Pessoais (LGPD). Diário Oficial da União, Brasília, DF, 15 ago. 2018. Acesso em: 14 mar. 2026.

BRASIL (2020). Lei nº 13.989, de 15 de abril de 2020. Dispõe sobre o uso da telemedicina durante a crise causada pelo coronavírus (SARS-CoV-2). Diário Oficial da União, Brasília, DF, 16 abr. 2020. Acesso em: 14 mar. 2026.

BRASIL (2022). Lei nº 14.510, de 27 de dezembro de 2022. Altera a Lei nº 8.080, de 19 de setembro de 1990, para autorizar e disciplinar a prática da telessaúde em todo o território nacional, e a Lei nº 13.146, de 6 de julho de 2015; e revoga a Lei nº 13.989, de 15 de abril de 2020. Diário Oficial da União, Brasília, DF, 28 dez. 2022. Acesso em: 14 mar. 2026.

Caetano, R., Silva, A. B., Guedes, A. C. C. M., Paiva, C. C. N. d., Ribeiro, G. d. R., Santos, D. L., and Silva, R. M. d. (2020). Desafios e oportunidades para telessaúde em tempos da pandemia pela COVID-19: uma reflexão sobre os espaços e iniciativas no contexto brasileiro. *Cadernos de Saúde Pública*, 36(5). DOI: 10.1590/0102-311X00088920. Acesso em: 14 mar. 2026.

Campbell, B., Bradley, J., Sakimura, N., and Lodderstedt, T. (2020). OAuth 2.0 mutual-TLS client authentication and certificate-bound access tokens. RFC 8705, Internet Engineering Task Force (IETF). Acesso em: 20 mar. 2026.

Cao, Y., Li, S., and Wijmans, E. (2017). (cross-)browser fingerprinting via OS and hardware level features. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. DOI: 10.14722/ndss.2017.23152. Acesso em: 05 maio 2026.

Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and Smith, R. (2013). Privacy considerations for internet protocols. RFC 6973, Internet Architecture Board (IAB). Acesso em: 23 mar. 2026.

Eckersley, P. (2010). How unique is your web browser? In *Proceedings of the 10th International Symposium on Privacy Enhancing Technologies (PETS)*, pages 1–18. Springer. DOI: 10.1007/978-3-642-14527-8_1.

Englehardt, S. and Narayanan, A. (2016). Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM. DOI: 10.1145/2976749.2978313. Acesso em: 05 maio 2026.

Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and Waite, D. (2023). OAuth 2.0 demonstrating proof of possession (DPoP). RFC 9449, Internet Engineering Task Force (IETF). Acesso em: 29 maio 2026.

Fingerprint (2025). Introduction. Fingerprint Documentation. Acesso em: 20 mar. 2026.

FingerprintJS (2025). FingerprintJS: Browser fingerprinting library. GitHub. Acesso em: 20 mar. 2026.

Franco, M. F., Soares, L. R., and Nobre, J. C. (2025). Saúde sob ataque: Da avaliação de riscos ao desenvolvimento de estratégias de investimentos em cibersegurança na área da saúde. In *Minicursos do XXV Simpósio Brasileiro de Computação Aplicada à Saúde (SBCAS 2025)*. Sociedade Brasileira de Computação. DOI: 10.5753/sbc.16731.5.1. Acesso em: 18 abr. 2026.

Hammer-Lahav, E. (2010). The OAuth 1.0 protocol. RFC 5849, Internet Engineering Task Force (IETF). Acesso em: 12 maio 2026.

Hwang, W. S., Shon, J. G., and Park, J. S. (2022). Web session hijacking defense technique using user information. *Human-centric Computing and Information Sciences*, 12(16). DOI: 10.22967/HCIS.2022.12.016. Acesso em: 20 mar. 2026.

IBM Security (2025). Cost of a data breach report 2025. Technical report, IBM Corporation, Armonk. Acesso em: 23 mar. 2026.

Jones, M. and Hardt, D. (2012). The OAuth 2.0 authorization framework: Bearer token usage. RFC 6750, Internet Engineering Task Force (IETF). Acesso em: 20 mar. 2026.

Kamal, P. (2016). State of the art survey on session hijacking. *Global Journal of Computer Science and Technology*, 16(1). Acesso em: 23 mar. 2026.

Laperdrix, P., Bielova, N., Baudry, B., and Avoine, G. (2020). Browser fingerprinting: A survey. *ACM Transactions on the Web*, 14(2). DOI: 10.1145/3386040. Acesso em: 23 mar. 2026.

Macedo, D. D. J. d., von Wangenheim, A., and Dantas, M. A. R. (2015). A data storage approach for large-scale distributed medical systems. In *Proceedings of the Ninth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, pages 486–490. IEEE. DOI: 10.1109/CISIS.2015.88.

Mayer, J. R. (2009). "any person... a pamphleteer": Internet anonymity in the age of web 2.0. Senior thesis (bachelor of arts), Woodrow Wilson School of Public and International Affairs, Princeton University, Princeton. Acesso em: 23 mar. 2026.

Mello, E. R. d., Chaves, S. A. d., Silva, C. E. d., Wangham, M. S., Brito, A., and Henriques, M. A. A. (2022). Autenticação e autorização: antigas demandas, novos desafios e tecnologias emergentes. In *Minicursos do XXII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2022)*. Sociedade Brasileira de Computação. DOI: 10.5753/sbc.10710.3.1. Acesso em: 24 mar. 2026.

Microsoft (2022). From cookie theft to BEC: Attackers use AiTM phishing sites as entry point to further financial fraud. Microsoft Security Blog. Acesso em: 24 mar. 2026.

Mowery, K. and Shacham, H. (2012). Pixel perfect: Fingerprinting canvas in HTML5. In *Proceedings of Web 2.0 Security and Privacy (W2SP)*. IEEE Computer Society. Acesso em: 05 maio 2026.

NIST (2025). Digital identity guidelines. NIST Special Publication 800-63B-4, National Institute of Standards and Technology. DOI: 10.6028/NIST.SP.800-63b-4. Acesso em: 23 mar. 2026.

Popov, A., Nystroem, M., Balfanz, D., and Hodges, J. (2018). The token binding protocol version 1.0. RFC 8471, Internet Engineering Task Force (IETF). Acesso em: 21 maio 2026.

Redis (2024). SET – Redis commands. Redis Documentation. Acesso em: 23 maio 2026.

Saraiva, A. R., Elleres, P. A. d. P., Carneiro, G. d. B., and Feitosa, E. L. (2014). Device fingerprinting: Conceitos e técnicas, exemplos e contramedidas. In *Minicursos do XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2014)*. Sociedade Brasileira de Computação. Acesso em: 20 mar. 2026.

Saúde Digital UFSC (2025). Quem somos. Núcleo Saúde Digital UFSC, Florianópolis. Acesso em: 20 mar. 2026.

Sembay, M. J., Macedo, D. D. J. d., and Marquez Filho, A. A. G. (2024). Gerenciamento de dados de proveniência em telemedicina e telessaúde: possíveis abordagens e perspectivas à luz da ciência da informação em saúde no brasil. *Asklepion: Informação em Saúde*, 3(1). DOI: 10.21728/asklepion.2024v3n1e-89. Acesso em: 18 abr. 2026.

Uma, E. and Kannan, A. (2013). The dynamic nonce based authentication scheme to defend intermediary attack for web services. *Journal of Computer Science*, 9(8). DOI: 10.3844/jcssp.2013.998.1007. Acesso em: 16 maio 2026.

Unger, T., Mulazzani, M., Frühwirt, D., Huber, M., Schrittwieser, S., and Weippl, E. (2013). SHPF: Enhancing HTTP(S) session security with browser fingerprinting. In *Proceedings of the 2013 Eighth International Conference on Availability, Reliability and Security (ARES)*. IEEE. DOI: 10.1109/ARES.2013.33. Acesso em: 28 mar. 2026.

Verizon Business (2025). 2025 data breach investigations report. Technical report, Verizon. Acesso em: 24 mar. 2026.

Wallauer, J., von Wangenheim, A., Andrade, R., and Macedo, D. D. J. d. (2008). A telemedicine network using secure techniques and intelligent user access control. In *Proceedings of the 21st IEEE International Symposium on Computer-Based Medical Systems (CBMS)*, Jyväskylä. IEEE. Acesso em: 18 abr. 2026.