HashifyPass - Uma Ferramenta para Visualização de Hashes de Senhas*

Henrique Araújo de Carvalho, Jorge Miguel Ribeiro, Daniel Macêdo Batista, José Coelho de Pina

[1]Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo (USP)

henriquecarvalho@usp.br, jorgemiguelribeiro92@gmail.com

{batista, coelho}@ime.usp.br

Abstract. *For security reasons, it is recommended to create long, completely unstructured, random and different passwords for each of the personal accounts. Due to these requirements, some people may want to confirm that they have entered the correct password before submitting a login form. To assist in this confirmation in websites, many systems have the option to view the password, which makes authentication more vulnerable to shoulder surfing attacks. This paper introduces the* `HashifyPass` *software, which allows website passwords to be confirmed without displaying them. For this, an animation is used to visualize the password hash. The integration of the software into a login form attests to its effectiveness.*

Resumo. *Por motivos de segurança, é recomendável criar senhas longas, completamente desestruturadas, aleatórias e diferentes para cada uma das contas pessoais. Como consequência desses requisitos, algumas pessoas podem querer confirmar que digitaram a senha correta antes de submeterem um formulário de login. Para auxiliar nessa confirmação, muitos websites contam com a opção de visualizar a senha, o que torna a autenticação mais vulnerável a ataques de shoulder surfing. Este artigo apresenta a ferramenta* `HashifyPass`*, que permite que senhas de websites sejam confirmadas sem exibi-las. Para isso, é usada uma animação para a visualização do hash da senha. A integração do software em um formulário de login atesta sua eficácia.*

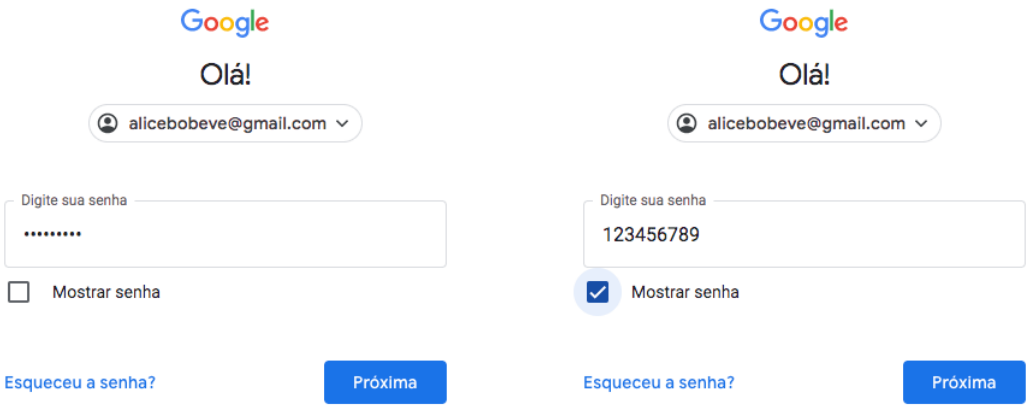
1. Introdução

Por motivos de segurança, as pessoas são forçadas a criar senhas diferentes, cada vez mais longas e completamente desestruturadas e aleatórias. Esses requisitos podem dificultar a memorização de senhas, e por isso algumas pessoas preferem digitar suas senhas apenas uma vez e mantê-las salvas pelo navegador web, principalmente quando estão usando

*Código-fonte: `https://github.com/decarv/hashify-pass` Documentação: `https://github.com/decarv/hashify-pass/blob/main/README.md` Vídeo: `https://drive.google.com/file/d/1v5xCD1OKjAvn2rAhFDI5_sk7iGHMYxDN/view` – Estas e todas as outras URLs referenciadas no artigo tiveram último acesso em 7 de Agosto de 2022

seus dispositivos pessoais. Apesar disso, algumas vezes precisa-se utilizar computadores públicos, e nesse caso necessita-se digitar a senha nesse novo dispositivo.

A digitação de senhas é um trabalho mecânico sujeito a erros, principalmente se a senha for digitada sem visibilidade, como ocorre em praticamente todos os sistemas computacionais que exigem digitação de senhas. Tais sistemas não ecoam a senha na tela, exibindo por exemplo * ou ● no lugar de cada caractere digitado ou, em alguns casos, não exibindo nenhum caractere. Dado que muitos sistemas possuem uma quantidade máxima de tentativas de entrada de senhas, é conveniente que digitemos as senhas corretamente antes de submetê-las para verificação do sistema de validação, evitando por exemplo o bloqueio de contas. Muitas vezes o sistema computacional a ser acessado é um *website* que possui a opção de visualizar a senha. Um exemplo dessa funcionalidade pode ser visto na Figura 1 que mostra a captura de tela do formulário de login do Gmail no momento em que a senha é requisitada. À esquerda a senha está oculta com ● para cada caractere digitado. À direita a senha está exibida pelo fato da opção "Mostrar senha" ter sido selecionada.



(a) Senha oculta **(b) Senha visível**

Figura 1. Captura de tela do formulário de login do Gmail. À esquerda, tela sem a senha ser exibida. À direita, tela com a senha exibida por conta da opção "Mostrar senha" ter sido selecionada.

Caso uma pessoa opte pela visualização da senha, como no caso exibido na Figura 1b, a autenticação estará mais vulnerável a ataques de *shoulder surfing* [Bošnjak and Brumen 2020] [English and Poet 2011]. Nesse tipo de ataque, um atacante espiona o(a) usuário(a) de um dispositivo eletrônico a fim de obter informações pessoais, como a senha por exemplo [Lexico 2022].

Este artigo apresenta o `HashifyPass`, um software que possibilita a confirmação da senha digitada em *websites* sem exibi-la. No `HashifyPass`, o *hash* da senha é utilizado como entrada para um algoritmo que gera uma animação única para cada *hash*. A animação pode ser comparada com uma versão prévia armazenada em um dispositivo confiável, por exemplo um smartphone, antes de ser submetida. Dessa forma, mesmo que um atacante realize um ataque de *shoulder surfing*, ele não conseguirá obter a senha. A integração do software em um formulário de login atesta sua eficácia.

O restante deste artigo está organizado da seguinte forma: A Seção 2 apresenta

trabalhos relacionados. A Seção 3 descreve a arquitetura e as principais funcionalidades do `HashifyPass`. A Seção 4 apresenta um protótipo de formulário de login protegido pelo `HashifyPass`. A Seção 5 descreve uma demonstração do `HashifyPass` usando como base o protótipo de formulário de login que foi desenvolvido. Conclusões e trabalhos futuros são exibidos na Seção 6.

2. Trabalhos Relacionados

[Bošnjak and Brumen 2020] revisa a literatura de forma sistemática com o objetivo de avaliar a qualidade de estudos que atestam a vulnerabilidade ao ataque de *shoulder surfing*. Nessa revisão, os autores destacam e focam na utilização de senhas gráficas ao invés de senhas textuais. Embora senhas gráficas facilitem a memorização, assim como no caso de senhas textuais elas são vulneráveis a ataques de *shoulder surfing*. Setenta e sete artigos foram analisados e foi possível observar que não há um padronização ao avaliar o quão vulnerável a *shoulder surfing* é um dado esquema de autenticação baseado em senhas gráficas. O `HashifyPass` leva em consideração a importância da utilização de elementos gráficos para memorização de informações de autenticação mas, diferente dos setenta e sete trabalhos avaliados em [Bošnjak and Brumen 2020], esses elementos são aplicados para visualização do *hash* de uma senha textual, impedindo que um atacante seja capaz de descobrir a senha digitada.

[English and Poet 2011] discute potenciais ataques contra esquemas de autenticação que utilizam senhas gráficas. O modelo de ameaças utilizado pelos autores considera *shoulder surfing*, ataques de intersecção, escuta para posterior ataque de *replay*, *phishing* e táticas de engenharia social para adivinhar a senha. A motivação do `HashifyPass` foca na proteção contra ataques de *shoulder surfing* dada a facilidade de realização desse ataque, no sentido de que o atacante não necessita de muitos conhecimentos técnicos sobre sistemas computacionais e nem de equipamentos específicos. Além disso, diferente de [English and Poet 2011], considera-se esquemas de autenticação baseados em senhas textuais.

O `HashifyPass` é baseado no *software* `hashify` [Ribeiro et al. 2020], ferramenta que busca facilitar a comparação de *hashes* criptográficos por pessoas através da geração de animações de 2 segundos de duração. Entende-se ser mais agradável, rápido e seguro comparar duas figuras, com cores evidentes e movimentos específicos, a comparar grandes sequências de caracteres. Em [Carvalho et al. 2021], o desempenho do `hashify` foi avaliado para atestar a sua eficiência quando executado em dispositivos móveis. Foi possível observar que, em termos de geração de quadros, consumo de memória e consumo de rede, o `hashify` pode ser integrado em aplicativos para dispositivos móveis. O `HashifyPass` é a utilização da ideia do `hashify` para certificar que senhas digitadas em *websites*, cujos caracteres são geralmente escondidos, estão corretas. Isso é possível porque o `hashify` foi projetado para receber um *hash* como entrada independente da semântica relacionada a esse *hash*. O algoritmo desenvolvido funciona independente do *hash* ser o *digest* de um certificado digital ou ter sido gerado a partir de uma senha por exemplo.

3. Arquitetura e Principais Funcionalidades do HashifyPass

A Figura 2 apresenta a interação de um(a) usuário(a) e dos componentes de software em um sistema de autenticação web protegido pelo `HashifyPass`. A interação inicia

com a digitação do nome de usuário e da senha (1). Em qualquer momento durante a digitação, a habilitação do `HashifyPass` pode ser realizada (2). Uma vez habilitado, o `HashifyPass` passará a receber a senha e um *salt* (3) que serão utilizados como entrada para a geração da animação que é exibida para o(a) usuário(a) (4). De posse da animação, o(a) usuário(a) deve verificar se ela está correta (5). Caso a pessoa não se lembre da animação correta, ela pode comparar com uma versão previamente armazenada, por exemplo, no seu smartphone. Após confirmar que a animação, e consequentemente a senha, estão corretas, o(a) usuário(a) envia as suas credenciais ao componente de autenticação do *website* (6) que por sua vez libera o acesso ao sistema (7). No caso de uma senha incorreta, o acesso não será liberado e os passos terão que ser repetidos. O restante desta seção explica com mais detalhes o funcionamento do sistema resumido na Figura 2.

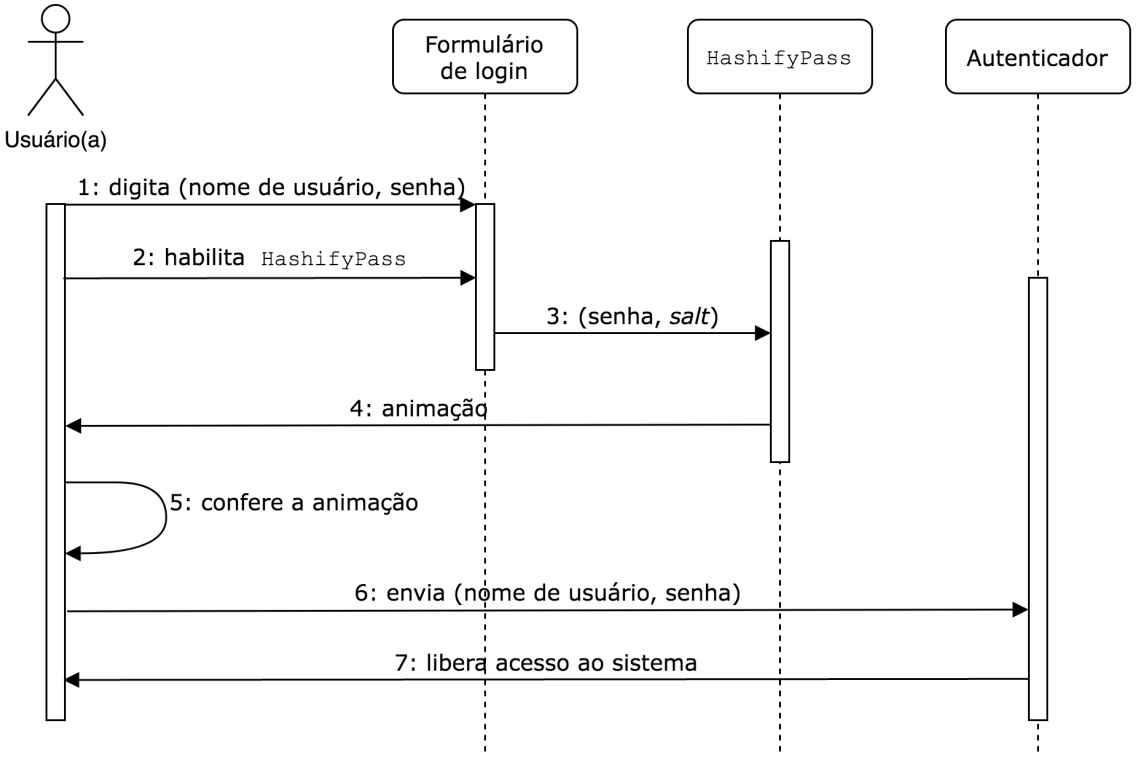


Figura 2. Interação de um(a) usuário(a) e dos componentes de software em um sistema de autenticação web protegido pelo `HashifyPass`

Enquanto o `hashify` permitia a criação de animações por meio de um *hash* no formato hexadecimal passado como parâmetro, o `HashifyPass` é capaz de exibir dinamicamente uma animação criada pelo `hashify` durante a digitação da senha pelo(a) usuário(a) em um *website*. Ou seja, à medida que um texto é digitado em um campo, o `HashifyPass` exibe uma sequência de animações criadas pelo `hashify`.

Embora o algoritmo central do `hashify` para geração de imagens não tenha sido alterado, o `HashifyPass` inova ao adicionar a composição dinâmica do `hashify` durante a digitação da senha. O software aguarda um evento de digitação de *input* e transforma esse *input* em *hash* dinamicamente. Para isso, utiliza-se uma função que re-

cebe texto e retorna um SHA-256, implementada na biblioteca `jsSHA`[1], desenvolvida por Brian Turek e distribuída sob a licença BSD. Com isso, a baixa probabilidade de colisão com dados de entrada, independentemente do comprimento desta entrada, é mantida, uma vez que as propriedades do SHA-256 passado não serão modificadas. Sabe-se que a probabilidade de colisão do *hash* visual do `hashify` é de aproximadamente 2^{-48} [Ribeiro et al. 2020], o que garante, com alto grau de certeza, que se a animação gerada pela *string* digitada corresponde à animação conhecida pelo(a) usuário(a) quando este digita sua senha, então a *string* digitada é idêntica à senha.

Considera-se possível que um agente mau intencionado seja capaz de descobrir a senha por meio de força bruta, após visualizar a animação gerada pelo `HashifyPass` para outro(a) usuário(a). Ainda assim, a probabilidade disso acontecer é muito pequena, uma vez que o `HashifyPass` usa 48 bits na composição da animação e, portanto, pode gerar 2^{48} animações, o que exigiria do agente tentar, na pior das hipóteses, 2^{48} senhas, sem considerar que este agente precisaria tentar senhas de tamanhos diferentes, o que cria uma nova camada de dificuldade. Portanto, é mais razoável supor que dois usuários possuem a mesma senha e que o primeiro deles tenha visualizado o *hash* animado do segundo, tomando conhecimento, portanto, da senha do segundo, que é idêntica à sua.

Diante disso, o `HashifyPass` usa o nome de usuário como um *salt* para compor a animação do `hashify`. Assim, para dois usuários diferentes com senhas idênticas, a animação resultante é diferente. A Figura 3 ilustra a integração do `HashifyPass` com o `hashify`.

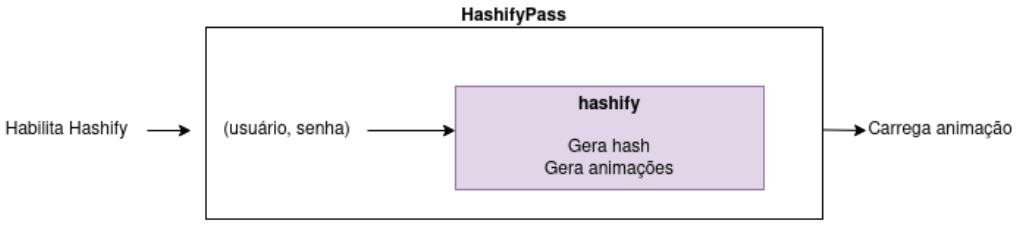


Figura 3. Esquema HashifyPass

4. Implementação e Resultados

O `HashifyPass` é escrito em JavaScript e, portanto, pode ser facilmente portado para *websites* para ser utilizado. Para tanto, basta que os diretórios `js` e `libs` do repositório sejam copiados para o diretório do *website* e sejam carregados na página html.

Para atestar a eficácia do `HashifyPass`, um protótipo de website foi criado e está disponível em `https://decarv.github.io`. Trata-se de um formulário de *login*, com a aparência exibida na Figura 4.

O(a) usuário(a) habilita o `HashifyPass` ao clicar no botão "Hashify". O resultado pode ser visto na Figura 5 (A imagem na figura corresponde a um quadro específico da animação. Para observar a animação por completo deve-se aguardar 2 segundos. A animação permanecerá sendo exibida em *loop*). Vale observar que a habilitação do `HashifyPass` pode ser realizada mesmo antes da senha ter sido completamente

[1]`https://github.com/Caligatio/jsSHA/wiki`. Último acesso em 23/06/2022

Figura 4. Protótipo de formulário – Página inicial

digitada. Caso a habilitação seja feita antes da senha ter sido digitada por completo, a animação segue sendo automaticamente atualizada a cada novo caractere digitado, removido ou modificado.

Figura 5. Protótipo de formulário – Geração da animação para usuário "usuário" e senha "minhasenha"

Pela Figura 6 e pela Figura 7 nota-se que duas animações diferentes são geradas se os nomes de usuário forem diferentes, mas tiverem a mesma senha. Por fim, se a senha estiver correta, o(a) usuário(a) terá um login bem sucedido. No caso da página criada, como se trata apenas de um protótipo, qualquer nome de usuário e senha leva a um login considerado bem sucedido.

Diferente do `hashify`, o `HashifyPass` permite que uma animação seja formada a partir de uma senha qualquer em formato de sequência de caracteres, ao invés da animação só poder ser gerada a partir de um *hash*. A complexidade adicional do `HashifyPass` em relação ao `hashify` é decorrente da transformação da senha em *hash*, o que é feito pela biblioteca `jsSHA`. Se a complexidade da geração de um *hash* a partir de uma *string* é linear em relação ao tamanho da string e se a string tem um tamanho limitado, como ocorre em uma senha, a complexidade do cálculo do *hash* é constante. Conclui-se, portanto, que em termos de desempenho, o `HashifyPass` possui um desempenho similar ao do `hashify`, podendo ser executado até mesmo em dispositivos móveis.

Figura 6. Protótipo de formulário – Animação para usuário “usuário” e senha “nãoéminhasenha”

5. Demonstração

O protótipo de *website* descrito na seção anterior é a melhor demonstração do `HashifyPass` em funcionamento. Ele pode ser acessado facilmente em um navegador web, tanto em um computador quanto em um smartphone. Qualquer nome de usuário e senha podem ser usados nos campos do formulário do protótipo.

O `HashifyPass` pode ser integrado em formulários de login por meio da cópia dos diretórios `js` e `libs` disponíveis no repositório da ferramenta. O vídeo disponibilizado em `https://drive.google.com/file/d/1v5xCD1OKjAvn2rAhFDI5_sk7iGHMYxDN/view` mostra um passo a passo de como realizar essa integração.

6. Conclusões e Trabalhos Futuros

A motivação para o desenvolvimento do `HashifyPass` é a criação de um sistema de *software* que seja capaz de garantir que usuários(as) digitem corretamente suas senhas em um *website*, sem comprometerem sua segurança pela exibição da senha, o que tornaria a autenticação mais vulnerável a ataques de *shoulder surfing*. Esse problema é resolvido pelo `HashifyPass` por meio da exibição de animações únicas para o *hash* de cada senha digitada.

O `HashifyPass` pode ser utilizado e modificado livremente por qualquer pessoa. Seu código-fonte está disponível em `https://github.com/decarv/hashify-pass` sob licença MIT. Neste repositório também é disponibilizado um protótipo de formulário de login protegido pelo `HashifyPass` e que atesta a eficácia do mesmo.

Como trabalhos futuros, pretende-se desenvolver um aplicativo para smartphone que armazene, de forma fácil e intuitiva, as animações para todas as senhas de uma pessoa. Dessa forma, caso uma pessoa não lembre da animação da sua senha correta, ela poderá tirar a dúvida pelo aplicativo. Também pretende-se realizar pesquisas com diver-

Figura 7. Protótipo de formulário – Animação para usuário "outrousuário" e senha "nãoéminhasenha"

sas pessoas para se avaliar a utilidade, a facilidade de uso e a resistência a ataques do `HashifyPass`.

Agradecimentos

Esta pesquisa é parte do INCT da Internet do Futuro para Cidades Inteligentes, financiado por CNPq (proc. 465446/2014-0), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001 e FAPESP (procs. 14/50937-1 e 15/24485-9). Também é parte dos projetos FAPESP proc. 21/04266-1 e 18/23098-0.

Referências

Bošnjak, L. and Brumen, B. (2020). Shoulder Surfing Experiments: A Systematic Literature Review. *Computers & Security*, 99:102023.

Carvalho, H., Ribeiro, J., Batista, D., and Pina, J. (2021). Análise de Desempenho de uma Ferramenta para Visualização de Hashes em Dispositivos Móveis. In *Anais Estendidos do XXI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pages 248–255.

English, R. and Poet, R. (2011). Towards a Metric for Recognition-based Graphical Password Security. In *Anais da 5th International Conference on Network and System Security*, pages 239–243.

Lexico (2022). SHOULDER SURFING — Meaning & Definition for UK English — Lexico.com. `https://www.lexico.com/definition/shoulder_surfing`.

Ribeiro, J. M., Batista, D. M., and de Pina, J. C. (2020). hashify: Uma Ferramenta para Visualização de Hashes com Animações. In *Anais Estendidos do XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pages 109–116.