Análise do algoritmo de hash argon2

Nelson Vieira
Faculdade Ciências Exatas e da Engenharia
Universidade da Madeira
Funchal, Portugal
2080511@student.uma.pt

***Abstract*—Este artigo serve como uma análise literária sobre o algoritmo de hash argon2, será realizada uma análise do algoritmo, tentando perceber a necessidade da sua criação, os pontos fortes e os pontos fracos, e o que este algoritmo oferece em relação a outros algoritmos de hash já existentes. Sobretudo este artigo procura responder as seguintes questões: porquê que este algoritmo está a ser recomendado e será que devemos substituir os algoritmos que estão em uso por este?**

***Index Terms*—argon2, criptografia, algoritmo de hash**

I. Introdução

Senhas, apesar de todas as suas desvantagens, permanecem a forma principal de autenticação em vários serviços da Web. As senhas são geralmente armazenadas num formato hash na base de dados de um servidor. Essas bases de dados são bastante frequentemente capturados pelos adversários, que aplicam ataques de dicionário, já que as senhas tendem a ter entropia baixa. Os designers de protocolo usam um número de truques para mitigar esses problemas. A partir do final dos anos 70, as senhas passaram a ser hashed com um valor de salt aleatório para evitar a detecção de senhas idênticas em diferentes utilizadores e serviços. Os cálculos de função de hash, que se tornaram mais rápidos e mais rápidos devido à lei de Moore, foram chamados várias vezes para aumentar o custo de tentativas de senha para o adversário. [1]

O Argon2 é um algoritmo de hash criptográfico que tem como objetivo resolver os problemas atuais do processo de hashing, criado por Alex Biryukov, Daniel Dinu e Dmitry Khovratovich da Universidade de Luxemburgo. O Argon2 possui 6 parâmetros de entrada: password, salt, custo de memória (o uso de memória do algoritmo), custo de tempo (o tempo de execução do algoritmo e o número de iterações), fator paralelismo (o número de fios paralelos), comprimento da hash.

Este algoritmo de hash criptográfico é uma função de derivação chave que foi selecionada como o vencedor da Password Hashing Competition [2] em 2015, ganhou esta competição de entre 24 algoritmos. Para além de ganhar esta competição também foi aceite como um standard RFC 9106 [3] em 2021.

II. Razão da Criação do Algoritmo

A. Os esquemas existentes de hashing e os seus problemas

A solução trivial para hash de senha é uma função de hash digitada como o HMAC, mas esta solução requer o uso de chaves secretas. Se o designer de protocolo preferir hashing sem chaves secretas para evitar todos os problemas com geração, armazenamento e atualização de chaves, ele tem poucas alternativas: o modo genérico PBKDF2, o bcrypt baseado no Blowfish, e scrypt. Entre esses, apenas scrypt visa a alta memória, mas a instabilidade de uma troca de tempo trivial permite implementações compactas com o mesmo custo de energia. O design de uma função memory-hard provou ser um problema difícil. Desde o início dos anos 80, sabe-se que muitos problemas criptográficos que aparentemente exigem memória grande, na verdade, permitem uma troca de tempo de memória, onde o adversário pode negociar memória por tempo e fazer o seu trabalho no hardware rápido com memória baixa. Em aplicação aos esquemas de hash de senha, isso significa que os crackers de senha ainda podem ser implementados num hardware dedicado, mesmo que por algum custo adicional. Outro problema com os esquemas existentes é a sua complexidade. O mesmo scrypt chama uma pilha de subprocessos, cuja razão de design não foi totalmente motivada. É difícil analisar e, além disso, difícil conseguir confiança. Finalmente, não é flexível em separar o tempo e os custos de memória. Ao mesmo tempo, a história das competições criptográficas demonstrou que os projetos mais seguros vêm com simplicidade, onde cada elemento é bem motivado e um criptoanalista tem apenas poucos pontos de entrada.

A Password Hashing Competition, que começou em 2014, destacou os seguintes problemas:

- O que acontece se o endereçamento de memória for independente do input ou dependente do input, ou híbrido?
- O primeiro tipo de esquemas, onde o local de leitura da memória é conhecida com antecedência, é imediatamente vulnerável aos ataques de time-space tradeoff, já que um adversário pode pré-computar o bloco ausente no momento em que é necessário. Por sua vez, os esquemas dependentes do input são vulneráveis a ataques side-channel, como as informações de tempo permitem uma pesquisa de senha muito mais rápida.
- É melhor encher a memória mas sofrer de compensações de espaço-tempo, ou fazer mais passes pela memória para ser mais robusto? Esta questão foi bastante difícil de responder devido à ausência de ferramentas de tradeoff genéricas, que analisaria a segurança contra ataques de troca e a ausência de métrica unificada para medir os custos do adverário. Como os endereços independentes do input devem ser computados? Várias opções aparente-

mente seguras foram atacadas.

- Quão grande bloco de memória único deve ser? Ler blocos menores colocados aleatóriamente é mais lento (em ciclos por byte) devido ao princípio da localidade espacial do cache do CPU. Por sua vez, blocos maiores são dificeis de processar devido ao número limitado de registradores longos. Se o bloco for grande, como escolher a função de compactação interna? Deve ser criptograficamente seguro ou mais leve, fornecendo apenas uma mistura básica das entradas? Muitos candidatos simplesmente propuseram uma construção iterativa e argumentavam contra transformações criptograficamente fortes.
- Como explorar várias cores das CPUs modernas, quando estiverem disponíveis? Paralelizar chamadas para a função de hashing com qualquer interação está sujeita a ataques tradicionais simples.

[1]

B. Solução proposta pelo Argon2 para os problemas de esquemas existentes

O Argon2 é uma função memory-hard. É um design simplificado. Visa a maior taxa de enchimento de memória e uso efetivo de várias unidades de computação, enquanto ainda fornece defesa contra ataques de trade-off. O Argon2 é otimizado para a arquitetura X86 e explora a organização de cache e memória dos recentes processadores Intel e AMD. O Argon2 tem uma variante principal, Argon2id e duas variantes suplementares, Argon2d e Argon2i. O Argon2d usa acesso à memória dependente de dados, o que o torna adequado para criptomoedas e aplicações sem ameaças de ataques de temporização de canal lateral. O Argon2i usa acesso à memória independente de dados, que é preferido para hash de senha e derivação de chave baseada em senha. O Argon2id funciona como Argon2i para a primeira metade da primeira passagem sobre a memória e como Argon2d para o resto, fornecendo assim a proteção de ataque de canal lateral e as economias de custo de força bruta devido a trade-offs de memória. O Argon2i faz mais passes pela memória para proteger de ataques de trade-off. [3]

Argon2 é um tipo de computação memory-hard em que, como paradigma genérico, consiste no seguinte: cada tarefa é amalgamada com um determinado procedimento que requer acesso intensivo à RAM tanto em termos de tamanho quanto (muito importante) largura de banda, de modo que a transferência de computação para GPU, FPGA e até mesmo ASIC (application-specific integrated circuit) traz pouca ou nenhuma redução de custos. Esquemas criptográficos que são executados neste contexto tornam-se igualitários no sentido de que os utilizadores e os invasores são iguais nas condições de relação de desempenho de preços. [4]

III. Performance do Algoritmo

O tempo necessário para executar as operações de hash na versão do console são semelhantes às da versão PHP, com o aumento no tempo dependendo da memória adquirida para uso pelo algoritmo Argon2. Também é importante notar que a eficiência do algoritmo é amplamente influenciada pelo número de threads usados. Com 1 GB de memória usada, seis iterações e um fio, o tempo gasto para realizar esta operação foi de quase 12 segundos. Com quatro threads, caiu quase para metade e com oito threads atingiu pouco mais de 4,5 segundos. Claro, o exemplo mencionado acima é apenas um exemplo limitante do algoritmo Argon2 no ambiente de produção, pois é cobrado com um pesado uso do poder do processador e da memória. O uso de memória de 16MB ou 32MB em três iterações e dois threads é a solução ideal em termos de tempo e uso de recursos de hardware. [5]

IV. Análise Criptográfica

Existem alguns artigos publicados que têm foco em ataques ao algoritmo Argon2.

O artigo de Boneh et al [6] mostra que é possível calcular uma função de passe de passagem única usando entre um quarto e um quinto do espaço desejado, sem penalidade de tempo, e calcular um passe de múltiplos Argon2i usando apenas N / E ¡N / 2.71 espaço com sem penalidade de tempo. Este tipo de ataque foi corrigido na versão mais recente do Argon2, de acordo com os designers [1], pelas experiências que fizeram, os resultados mostram que em 1-passe Argon2i, em média, 1/7 da memória é usado. Como no aplicativo simples, em média, 1/2 da memória é usado, a vantagem no produto da área de tempo é de cerca de 3,5. Se for usado o valor da memória de pico nos cálculos da área de tempo, a vantagem seria de 5 e 2.7, respetivamente. A versão 1.3 do Argon2 substitui a operação de sobrescrever com o XOR. Isso fornece sobrecarga mínima sobre o desempenho: para os requisitos de memória de 8 MB e maior a diferença de desempenho é entre 5% 3-Pass Argon2d v.1.2.1 em 1,8 GHz CPU com o Ubuntu é de 1,61 ciclos por byte, enquanto para V.1.3 é 1,7 CPB (ambos medidos para 2 GB de RAM, 4 threads).

O segundo artigo [7] consistiu em provar a efetividade do ataque de Alwen-Blocki, que na altura da sua criação afetava a revisão Argon2i-A (Versão 1.0), na revisão o Argon2i-B (Versão 1.3), com o objetivo de analisar a segurança dessa revisão em particular, tendo em consideração que o ataque seja realizado sob constrições de hardware realísticas. Conseguiram demonstrar que tal ataque, mesmo com tais constrições, é ainda possível de ser realizado na nova versão. Mesmo para definições de parâmetros pessimistas, o ataque conseguiu reduzir os custos por um fator de 2 usando apenas 1 GB de memória e que quando a largura de banda da memória onchip limita o paralelismo, só era preciso ajustar os parâmetros de ataque em conformidade, sem afetar a qualidade de ataque. No entanto, o Argon2i-B demonstrou oferecer melhor resistência ao ataque do que Argon2i-A. O artigo também salienta que a redução dos custos do iMHF (p. ex., por um fator de 10) podem aumentar a percentagem de palavras-passe comprometidas num ataque offline (por exemplo, por até 60%).

O terceiro artigo [8] teve como objetivo avaliar a qualidade do ataque de Alwen-Blocki. Conclui que o ataque apresenta uma ameaça teoricamente considerável para o Argon2, sendo

uma ameaça preocupante a qualquer MHF com gráficos computacionais de indegredo fixo, praticamente falando ainda é uma melhoria em relação à disparidadeentre os ASIC e os computadores, em perfeita harmonia com a exigência não-memória de palavras-passe. No entanto, indicaram que o ataque optimizado por hardware não é generalizável a diferentes membros da família Argon2, sendo pouco provável que a maioria dos alvos utilize o mesmo membro da família Argon2. Também salientaram aência da compra de equipamentos generalizados sub-pares ou conjuntos diferentes de ASIC's para diferentes alvos, sendo uma limitação pragmática significante.

V. Comparação com outros algoritmos existentes

Existem vários algoritmos de hash disponíveis para além do algoritmo em análise, Argon2, como por exemplo o MD5, SHA1, SHA256, PBKDF2, Bcrypt, Scrypt, ou simplesmente usar plaintext. Torna-se complicado escolher o algoritmo a usar de entre várias escolhas, portanto nesta secção será realizada uma comparação dos algoritmos acima descritos com o Argon2 e as suas derivações, Argon2i, Argon2d e Argon2id.

A. Utilização de plaintext

Começando com o mais óbvio, usar plaintext (texto legível) para guardar informação delicada e portanto que deve ser escondida é o que deve ser evitado a todo o custo, pois qualquer pesquisa numa base de dados irá revelar todo o conteúdo de forma aberta a qualquer pessoa, daí a necessidade de usarmos algum algoritmo que torne esta informação encriptada. Ainda assim apesar da riqueza de métodos de criptografia existentes, algumas empresas ainda armazenam as senhas em plaintext, isso significa que qualquer pessoa com acesso pode ler todas as informações altamente confidenciais, como as senhas, datas de nascimento e números de cartão de crédito. Se uma senha estiver armazenada em plaintext, esta também pode ser rabiscada num bloco de notas e deixada na sala de espera para qualquer indivíduo mal itencionado ver. Bases de dados com senhas em plaintext tornam a contenção muito mais difícil porque o atacante compromete instantaneamente a segurança de todos os utilizadores do serviço web. Poderia ser considerado que um ataque directo num servidor é um evento extremamente improvável, mas incidentes recentes e repetidos em grande escala (afetando milhões de utilizadores) têm vindo a aumentar nos últimos anos [9] e principalmente com a guerra na Ucrânia temos visto isso a acontecer.

B. Comparação com MD5

Um dos algoritmos mais usados em bases de dados é o MD5, e é também um dos algoritmos mais velhos, normalmente por ser um algoritmo muito usado e pela sua idade, levaria a pensar que este seria um dos algoritmos mais seguros que existem, mas isto não se verifica Vamos comparar o argon2 com sha-256, md5, etc.

VI. Conclusão

O Argon2 é um dos algoritmos de hash mais seguros da atualidade, é inspirado nalguns algoritmos também bastante seguros como o bcrypt e por isso acaba por ser uma evolução lógica dos algoritmos de hash. Apenas uma versão do algoritmo tem vulnerabilidades, mas estas vulnerabilidades podem ser negadas como explicado na secção "Análise criptográfica". Este algoritmo tem muitas vantagens, mas uma desvantagem que pode causar problemas nalgumas situações é o fato do algoritmo poder ser muito lento caso sejam modificados alguns parâmetros, é necessário criar um equilíbrio nestas situações entre segurança e tempo de execução.

Acknowledgment

Gostaria de agradecer a todas as pessoas que me apoiaram até agora, incluindo família, amigos e professores.

References

[1] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: the Memory-Hard Function for Password Hashing and Other Applications," Mar. 2017. [Online]. Available: https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf

[2] (2019) Password Hashing Competition. Accessed on 2022-04-21. [Online]. Available: https://www.password-hashing.net/

[3] A. Biryukov, D. Dinu, D. Khovratovich, and S. Josefsson, "Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications," RFC 9106, Sep. 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9106

[4] A. Biryukov and D. Khovratovich, "Egalitarian Computing," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 315–326. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/biryukov

[5] M. Duka, "Elliptic-Curve Cryptography (ECC) and Argon2 Algorithm in PHP Using OpenSSL and Sodium Libraries," *Informatyka Automatyka Pomiary w Gospodarce i Ochronie Środowiska*, vol. 10, pp. 91–94, 09 2020.

[6] D. Boneh, H. Corrigan-Gibbs, and S. Schechter, "Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2016, pp. 220–248, https://ia.cr/2016/027.

[7] J. Alwen and J. Blocki, "Towards Practical Attacks on Argon2i and Balloon Hashing," in *2017 IEEE European Symposium on Security and Privacy (EuroS P)*. IEEE, 2017, pp. 142–157.

[8] M. Gu, R. Berg, and C. Chen, "Investigation of Practical Attacks on the Argon2i Memory-hard Hash Function," 2017.

[9] A. Lukehart. (2022) 2022 Cyber Attack Statistics, Data, and Trends. Accessed on 2022-05-22. [Online]. Available: https://parachute.cloud/2022-cyber-attack-statistics-data-and-trends/