II Encontro de Iniciação Científica e Tecnológica
II EnICT
ISSN: 2526-6772
IFSP – Câmpus Araraquara
26 e 27 de Outubro de 2017

TROCA DE CAPTURAS DE TELA ENTRE USUÁRIOS UTILIZANDO CRIPTOGRAFIA PONTA-A-PONTA

Malcom F. B. Silva[1], Lourenço Alves Pereira Júnior[2]

[1] Graduando em Análise e Desenvolvimento de Sistemas, IFSP Campus Araraquara.
[2] Docente, IFSP, Campus Araraquara, ljr@ifsp.edu.br

Área de conhecimento (Tabela CNPq): Arquitetura de Sistemas de Computação – 1.03.03.04-9

RESUMO: O compartilhamento de capturas de tela entre usuários está se tornando cada vez mais comum, porém as ferramentas atuais armazenam e/ou trafegam as imagens compartilhadas de forma insegura. Este trabalho visa desenvolver uma especificação que permita a transferência segura de capturas de tela entre usuários e obter uma maior privacidade. A especificação proposta utiliza criptografia ponta-a-ponta com o objetivo de dificultar a visualização das imagens compartilhadas e sendo acessível apenas ao remetente e o destinatário.

PALAVRAS-CHAVE: Captura de tela; Criptografia ponta-a-ponta; Segurança; Privacidade

INTRODUÇÃO

Uma das formas de facilitar a comunicação entre pessoas na sociedade atual é por meio de envio de capturas de telas, ou seja, registrar a saída de vídeo de um dispositivo em um determinado momento no formato de imagem. O envio dessas capturas geralmente se dá por meio online, em que um usuário envia a captura de tela para um serviço de hospedagem e compartilha o *link* para outros usuários acessarem. Dependendo do serviço, o usuário que envia a captura pode alterar as configurações de visibilidade para ser acessível apenas a um determinado grupo. No entanto, um administrador do serviço pode acessar todos os arquivos enviados por usuários e encontrar informações que deveriam ser privadas.

Os serviços de hospedagem de imagem podem não possuir proteção contra acesso não autorizado pelos funcionários e/ou administradores do serviço, podendo expor informações confidenciais de usuários, como dados bancários, números de cartões de crédito, informações pessoais, entre outros. Há também a possibilidade de o serviço ser invadido e ter os arquivos roubados, expondo as informações dos usuários e comprometendo a privacidade.

Dado o exposto, este artigo apresenta uma especificação que permite o compartilhamento de capturas de telas entre usuários de tal forma que dificulte o acesso não autorizado aos dados. A criptografia pode ser utilizada para garantir a confidencialidade e a integridade dos dados. A confidencialidade é alcançada ao impedir que um administrador do serviço consiga acessar os dados enviados pelos usuários por estarem criptografados. Ao utilizar um Código de Autenticação de Mensagem (MAC) junto ao arquivo criptografado, é possível garantir a integridade dos dados, pois alterações maliciosas ou acidentais poderão ser detectadas.

FUNDAMENTAÇÃO TEÓRICA

Com o aumento crescente de atores maliciosos em redes de computadores, se faz necessário aumentar a resistência a alterações maliciosas e visualização do tráfego na rede por terceiros, como provedores de Internet. Uma forma de garantir a privacidade na rede se dá pelo uso de criptografia. Sua finalidade é permitir que a informação seja mantida, mas que somente o destinatário possa decifrá-la e compreender o seu conteúdo (FERREIRA, 2003).

A criptografia está sendo cada vez mais usada em sistemas online, em que está sendo adotado um protocolo para transferência segura de dados pela rede, Transport Layer Security (TLS), definido no RFC 5246. O protocolo TLS pode ser utilizado em conjunto com o protocolo HTTP (Hyper Text Transfer Protocol), definido no RFC 7540, para assegurar a integridade e confidencialidade durante a transmissão de dados e, portanto, prover uma navegação segura na Web. Tal combinação de protocolos é denominada HTTPS (Hyper Text Transfer Protocol Secure), de acordo com RFC 2818.

O protocolo HTTPS protege os dados em trânsito, ou seja, quando estão sendo trafegados pela rede. Ao chegar no servidor, os dados enviados podem ser acessados em texto puro. Mesmo sendo desejável na maior parte das aplicações, em certos casos, é possível ter uma maior segurança ao privar o servidor de ter acesso aos dados originais, deixando-os disponíveis somente para o remetente e o destinatário da mensagem, caso o propósito do servidor for de repassar mensagens entre usuários.

Um exemplo de aplicação são serviços de mensagem instantânea, em que há um remetente (autor da mensagem), um destinatário (receptor da mensagem) e intermediários (provedores do serviço). É desejável utilizar uma abordagem que somente o remetente e o destinatário possam visualizar a mensagem, privando qualquer intermediário de conseguir interceptar ou modificar a mensagem original.

Tal abordagem é chamada de criptografia ponta-a-ponta. Uma das pontas (remetente) converte a mensagem original em texto cifrado e envia a chave de criptografia para o destinatário utilizando outro meio de comunicação. O texto cifrado é enviado ao servidor sem fornecer a chave de criptografia, tornando os dados ilegíveis no servidor e, consequentemente, ao operador do serviço, e em seguida o texto cifrado é enviado ao destinatário. Como o destinatário obteve uma cópia da chave de criptografia, é possível acessar os dados originais.

Há várias ferramentas de captura de tela disponíveis para uso, algumas delas possuem opção de *upload* e compartilhamento, as quais é o foco deste trabalho. Foram analisadas 2 ferramentas de captura de tela: Lightshot e Greenshot. Todas as ferramentas permitem selecionar uma área da tela, adicionar texto e/ou desenho à imagem e salvar o resultado localmente em formatos de imagem como PNG (*Portable Network Graphics*), JPEG (*Joint Photographic Experts Group*), dentre outros.

Lightshot é um software gratuito de captura de tela para Windows e Mac desenvolvido pela empresa Skillbrains. A ferramenta permite ao usuário enviar a captura de tela gerada para os servidores do serviço e receber uma URL (*Uniform Resource Locator*) curta para compartilhamento. A funcionalidade de encurtamento de URL gera códigos sequenciais, ou seja, o próximo código será uma unidade maior que o anterior. Por exemplo, se o código for “akbgh”, o próximo código seria “akbgi”, pois segue a ordem alfabética. Isso permite que um usuário malicioso acesse capturas de telas de outros usuários, pois o endereço é facilmente encontrável, além de não haver controle de acesso, portanto qualquer usuário possui acesso a qualquer captura de tela. As capturas de tela são transferidas para o serviço em texto puro, sem utilizar HTTPS ou criptografia ponta-a-ponta, permitindo que tanto agentes maliciosos na rede, quanto os administradores do site possam ver o que foi enviado pelo usuário.

Greenshot é um software *open-source* e gratuito para Windows. A ferramenta não possui servidor de *upload* nativo, porém o recurso pode ser adicionado por meio de *plug-ins*. Uma vantagem desta abordagem é a possibilidade de implementar ou utilizar outro serviço de *upload*, não ficando atrelado a um serviço específico. No entanto, o usuário deve ter confiança em duas entidades: nos desenvolvedores do Greenshot, por não adicionar código malicioso e no serviço de hospedagem escolhido pelo usuário, no qual pode haver diferentes mitigações de segurança entre as opções disponíveis. Nenhum dos *plug-ins* testados oferecem criptografia ponta-a-ponta, permitindo que o administrador do serviço visualize as capturas enviadas.

A partir do levantamento realizado, pode-se notar que as ferramentas analisadas (Lightshot, Greenshot) oferecem um nível de privacidade ao usuário que não contempla criptografia ponta-a-ponta, havendo a possibilidade de dados privados ficarem acessíveis aos operadores do serviço. Dado a facilidade de uso das ferramentas, há a possibilidade de um usuário inexperiente enviar uma captura de tela com dados sensíveis, como número de cartão de crédito, saldo bancário, dados pessoais, entre outros. Há também a possibilidade de haver um vazamento de informações no serviço de hospedagem e expor milhares de imagens que deveriam ser privadas.

Neste contexto, é interessante projetar uma abordagem de envio de capturas de tela utilizando criptografia ponta-a-ponta, em que somente o usuário que fez o envio e os contatos que receberem o *link* poderão ver a captura de tela, o operador do serviço e da rede não serão capazes de visualizar as imagens. Caso houver um ataque ao serviço, todas as imagens estarão criptografadas e de difícil acesso.

No caso do Greenshot, o sistema sugerido poderia ser mais facilmente implementado por possuir suporte a *plug-ins*, não necessitando realizar alterações no código-fonte principal do programa.

METODOLOGIA

Conforme mencionado anteriormente, o projeto consiste na definição de uma especificação para permitir uma transferência segura de capturas de tela entre usuários.

Como parte do projeto, foram feitos diagramas utilizando UML (Unified Modeling Language) para representar o protótipo de testes. Os diagramas modelados são do tipo de sequência, retratando duas etapas no uso do protótipo: envio da captura de tela (Figura 1) e recuperação da captura de tela (Figura 2).

A Figura 1 detalha a implementação da especificação descrita anteriormente para o envio de capturas de tela. O software irá gerar uma chave de criptografia e criptografar a captura de tela com a chave gerada. A imagem criptografada é então enviada para um servidor de armazenamento, no qual retorna um código identificador para poder realizar o *download* da imagem posteriormente. Por fim, o software devolve ao usuário uma representação amigável contendo o código identificador e a chave de criptografia.

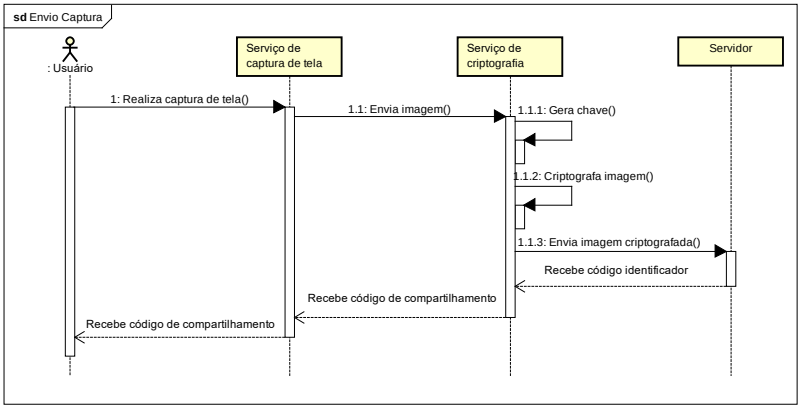

FIGURA 1. Diagrama de sequência para envio de capturas de tela
Fonte: Elaborado pelo autor

A Figura 2 especifica a recuperação de capturas de tela por um usuário após receber o código identificador e a chave de criptografia de outro usuário. O usuário fornece o código identificador e a chave de criptografia gerada durante o *upload* da captura de tela em uma representação amigável. Em seguida, o software irá extrair o código identificador e a chave de criptografia da representação amigável, realizar o *download* da imagem criptografada do servidor de armazenamento e efetuar a descriptografia da imagem. Por fim, a imagem descriptografada é exibida ao usuário.

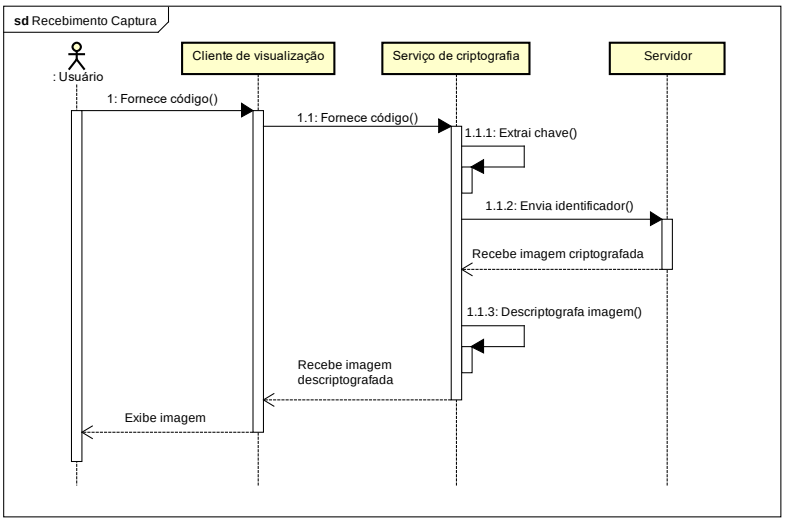

FIGURA 2. Diagrama de sequência para recuperação de capturas de tela
Fonte: Elaborado pelo autor

IMPLEMENTAÇÃO

Foram desenvolvidos dois protótipos para avaliar a especificação proposta. O primeiro protótipo utiliza uma abordagem de servidor *proxy* local para o Lightshot. O *proxy* local foi desenvolvido utilizando a linguagem Javascript e a plataforma Node.JS (Figura 3). O servidor de *upload* foi escrito em PHP. Ao enviar uma captura de tela, o pedido HTTP é interceptado pelo protótipo e as operações citadas anteriormente são executadas. Isso só é possível por conta dos servidores do Lightshot utilizarem o protocolo HTTP em vez de HTTPS, permitindo a execução de um ataque de homem no meio.

Para realizar os processos de criptografia do protótipo, foi utilizado o algoritmo AES (*Advanced Encryption Standard*) em modo GCM (*Galois/Counter Mode*). O algoritmo AES é uma cifra de bloco de 128 bits e um tamanho de chave de 128, 192 ou 256 bits (STALLINGS, 2008). O protótipo gera uma chave aleatória de 256 bits a cada *upload* realizado, não havendo qualquer ligação entre chaves e/ou *uploads* anteriores ou posteriores, provendo confidencialidade aos dados. No modo de operação GCM, uma *tag* de autenticação (MAC) de 128 bits para garantir a integridade da mensagem cifrada com AES.

Após o servidor retornar o código identificador, o *proxy* gera uma URL no formato `https://exemplo.com/<código identificador>#<chave>`, em que <chave> corresponde à chave de criptografia, e envia para a aplicação. Os caracteres após a cerquilha formam o fragmento da URL, como descrito na seção 3.5 do RFC 3986. Em browsers, o fragmento é geralmente utilizado para identificar uma porção do documento e não deve ser enviado para o servidor durante um pedido. Portanto, o browser envia ao servidor apenas o código identificador e mantendo a chave localmente, podendo ser acessada posteriormente utilizando Javascript. Para a descriptografia da imagem no browser, foi utilizada a API Web Crypto.

```
function onPart(part, res) {
    const key = crypto.randomBytes(32);
    const iv = crypto.randomBytes(16);
    const cipher = crypto.createCipheriv('aes-256-gcm', key, iv);

    let encrypted = Buffer.alloc(0);
    let partSize = 0;

    part.on('data', (data) => {
        partSize += data.length;
        encrypted = Buffer.concat([encrypted, cipher.update(data)]);
    });
    part.on('end', () => {
        encrypted = Buffer.concat([encrypted, cipher.final()]);
        const tag = cipher.getAuthTag();

        uploadToServer(iv.toString('base64'), tag.toString('base64'),
            encrypted, key, res);
    });
}
```

FIGURA 3. Trecho de código de upload do *proxy* local
Fonte: Elaborado pelo autor

O segundo protótipo utiliza uma abordagem de *hooking* no Greenshot. Foi desenvolvido um *plug-in* que recebe a captura de tela utilizando a API do software e realiza as operações descritas acima. Uma vantagem dessa abordagem é a utilização de componentes oficiais do software, diferente do primeiro protótipo em que é utilizado um detalhe de implementação que pode ser alterado a qualquer momento.

Foi utilizada a biblioteca Sodium para realizar as operações de criptografia. O algoritmo oferecido pela biblioteca é XSalsa20, de 256 bits, em conjunto do algoritmo Poly1305 para MAC, de 128 bits, para autenticação. A implementação do *plug-in* foi feita utilizando a linguagem de programação C#, o código do servidor foi escrito em Node.JS.

Diferente da primeira implementação, no segundo protótipo o usuário deve utilizar uma aplicação específica para visualizar as capturas de tela. A aplicação de *upload* entrega ao usuário um código compartilhável no formato `<código identificador>!<chave>`. Na aplicação de visualização, a chave é separada do identificador e é feita uma requisição ao servidor para obter os dados criptografados, mantendo a chave localmente. O caractere "!" serve somente como delimitador, não possuindo um significado específico.

Como pode ser visto na Figura 4, também foi implementada checagens de assinatura digital utilizando Curve25519. As respostas do servidor são assinadas e enviadas no cabeçalho HTTP. O cliente verifica a assinatura utilizando a chave pública do servidor para garantir que não houve modificação dos dados durante a transmissão.

```
public async Task<APIUploadModel> Upload(string json)
{
    try
    {
        var response = await DoRequest(json);

        string content = await response.Content.ReadAsStringAsync();

        return ValidateSignature(response, content) ??
            JsonConvert.DeserializeObject<APIUploadModel>(content);
    }
    catch (Exception)
    {
        return new APIUploadModel("Unknown error");
    }
}
```

FIGURA 4. Trecho de código de upload do *plug-in*
Fonte: Elaborado pelo autor

RESULTADOS E DISCUSSÕES

As duas implementações de teste mostraram ser funcionais de acordo com a especificação, permitindo uma captura de tela ser enviada e visualizada sem que o servidor de armazenamento tivesse acesso imediato. Também foi possível observar que a especificação não está ligada com uma linguagem de programação específica, podendo ser implementada da forma que for necessária para satisfazer um determinado caso de uso.

A especificação também não limita os algoritmos de criptografia ou funcionalidades de segurança da implementação. O uso de assinatura digital pode mitigar ataques de homem no meio caso não for possível utilizar HTTPS para transporte dos dados. Os protótipos utilizam diferentes algoritmos de criptografia (AES e XSalsa20) para demonstrar diferentes implementações da mesma especificação.

Vale ressaltar que o objetivo do trabalho é impedir o acesso aos dados enquanto são trafegados pela rede e armazenados nos servidores. Não cabe ao escopo do trabalho garantir a segurança no lado do cliente, ou seja, quando um usuário envia um código compartilhável a outro usuário.

CONCLUSÕES

O compartilhamento de capturas de tela é cada vez mais utilizado por usuários, porém as ferramentas atuais não garantem a privacidade do usuário perante o administrador do serviço e/ou provedores de Internet (Lightshot; Greenshot).

Este trabalho propõe uma especificação que permite a transferência segura de captura de tela entre usuários, garantindo confidencialidade, integridade e disponibilidade por meio da utilização de criptografia, MAC e criptografia ponta-a-ponta, respectivamente.

Dado o exposto, espera-se que a especificação apresentada melhore a privacidade e segurança dos usuários ao compartilhar capturas de tela. Foram obtidos resultados positivos após o desenvolvimento e avaliação de protótipos utilizando abordagens distintas, porém implementando a mesma especificação.

REFERÊNCIAS

FERREIRA, Fernando, 2003, **Segurança da Informação** – Ciência Moderna.

Greenshot. Disponível em: <http://getgreenshot.org/>. Acesso em: 05 Maio. 2017.

Introduction · libsodium. Disponível em: <https://download.libsodium.org/doc/>. Acesso em: 02 Setembro. 2017.

Lightshot - ferramenta para captura de tela para Mac & Win. Disponível em: <https://prnt.sc/>. Acesso em: 05 Maio. 2017.

RFC 2818 - HTTP Over TLS. Disponível em: <https://tools.ietf.org/html/rfc2818>. Acesso em: 30 Agosto. 2017.

RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2). Disponível em: <https://tools.ietf.org/html/rfc7540>. Acesso em: 30 Agosto. 2017.

RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2. Disponível em: <https://tools.ietf.org/html/rfc5246>. Acesso em: 30 Agosto. 2017.

RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax. Disponível em: <https://tools.ietf.org/html/rfc3986>. Acesso em: 30 Agosto. 2017.

STALLINGS, William. **Criptografia e segurança de redes: 4. ed**. São Paulo: Pearson Prentice Hall, 2008.

Web Cryptography API. Disponível em: <https://www.w3.org/TR/WebCryptoAPI/>. Acesso em: 02 Setembro. 2017.