Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Humberto Borgas Bulhões

Melhoria da segurança da informação em navegadores: uma proposta baseada em APIs Javascript e HTML

São Paulo

2017

Humberto Borgas Bulhões

Melhoria da segurança da informação em navegadores: uma proposta baseada em APIs Javascript e HTML

Exame de Qualificação apresentado ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para a obtenção do título de mestre em Engenharia de Computação.

Data da aprovação ____/____/_____

_____
Marcelo Novaes de Rezende (Orientador)

Membros da Banca Examinadora:

Marcelo Novaes de Rezende (Orientador)

Humberto Borgas Bulhões

Melhoria da segurança da informação em navegadores: uma proposta baseada em APIs Javascript e HTML

Exame de Qualificação apresentado ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para a obtenção do título de mestre em Engenharia de Computação.

Área de Concentração: Engenharia de Software

Orientador: Marcelo Novaes de Rezende

São Paulo

Dezembro/2017

Humberto Borgas Bulhões
Melhoria da segurança da informação em navegadores: uma proposta baseada em APIs Javascript e HTML/ Humberto Borgas Bulhões. – São Paulo, 2017-
27 p. : il. (algumas color.) ; 30 cm.

Orientador Marcelo Novaes de Rezende

Dissertação de Mestrado – Instituto de Pesquisas Tecnológicas do Estado de São Paulo, 2017.

CDU 02:141:005.7

RESUMO

O desenvolvimento de aplicações destinadas ao ambiente de execução dos navegadores da web requer atenção aos riscos de vazamento da informação de usuário contida no navegador, sejam esses riscos explorados de forma intencional e maliciosa, sejam eles acionados por consequência acidental do próprio funcionamento de uma aplicação web. Embora ao longo do tempo os navegadores venham sendo melhorados para que consigam detectar e mitigar alguns desses riscos, esse esforço é marcado por concessões às funcionalidades esperadas pelas aplicações web, fazendo com que a segurança da informação no navegador seja um campo de conhecimento com práticas consideradas incoerentes entre si. Um cipoal de iniciativas e padrões de segurança se impõe aos desenvolvedores de aplicações, que nem sempre podem prever o grau de exposição dos dados de seus usuários no navegador uma vez que esta propriedade é produto de uma combinação entre a configuração dos servidores de aplicação, o nível de confiança entre as partes componentes das páginas web, o conjunto de extensões ativadas pelo navegador, e o teor dos *scripts* envolvidos. Recursos de programação poderiam ser empregados para que informações sensíveis ficassem fora do alcance de participantes não confiáveis, particularmente *scripts*. No âmbito da segurança da informação, este trabalho propõe uma abordagem que proporcione ao desenvolvedor uma barreira de proteção incorporada às aplicações, complementar aos recursos de segurança disponíveis no *backend* e ao navegador. A validação dessa proposta ocorrerá por meio de um protótipo de sistema submetido a situações de risco de vazamento da informação em páginas web, em correspondência com ocorrências documentadas pela literatura. O protótipo será preparado para que a abordagem proposta, sob a forma de um script baseado em APIs padronizadas de HTML e Javascript, seja colocada à prova no seu propósito de neutralizar tais situações de risco.

Palavras-chave: Segurança da informação, vazamento de dados, HTML, Javascript, DOM.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

*

LISTA DE ABREVIATURAS E SIGLAS

CORS	Cross-origin Resource Sharing
DAC	Discretionary Access Control
DOM	Document Object Model
IFC	Information Flow Control
JSON	JavaScript Object Notation
MAC	Mandatory Access Control
SOP	Same Origin Policy
XSS	Cross-Site Scripting

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

A diversidade das aplicações disponíveis na web reflete a demanda de seus usuários por funcionalidade útil e interessante. A popularidade dessa plataforma denota o grau de confiança depositada pelos usuários em sua infraestrutura, ou não haveria procura pela utilização de serviços como comércio eletrônico, transações bancárias, redes sociais e troca de mensagens, para mencionar algumas das categorias de aplicações que fazem uso intensivo de dados pessoais, incluindo dados sigilosos ou de uso restrito.

A confiança com que os usuários interagem com a web, no entanto, é constantemente desafiada por falhas sistêmicas e ataques deliberados que culminam com o roubo e a adulteração das informações roteadas pelos diversos componentes que dão suporte às funcionalidades das aplicações web. Muitas vezes, o risco de vazamento de informação se manifesta justamente na ponta mais próxima do usuário: seu software navegador de internet. Ataques como o redirecionamento de *scripts* em sites do *bureau* de crédito norte-americano Equifax (SEGURA, 2017), a invasão e adulteração de scripts da rede de distribuição de conteúdo BootstrapCDN (DORFMAN, 2013), a ousada exploração de *malware* atingindo usuários do site E-Bay (VANUNU, 2016), e as divulgações recentes do comprometimento de extensões do aplicativo Chrome (FORREST, 2017) apontam para vulnerabilidades intrínsecas nos recursos do navegador que dão suporte ao “conteúdo ativo”, mecanismo essencial para as aplicações web interativas modernas (HEDIN et al., 2016).

Assim, o desenvolvimento de uma aplicação segura para a web demanda esforços para que seja evitada a exposição e a manipulação indevidas das informações do usuário. Para esse propósito, o desenvolvedor conta com um conjunto de práticas e recomendações estabelecidas, efetivamente protegendo a aplicação e seus usuários de uma série de vulnerabilidades. Conjuntos de regras, como a SOP (*same-origin policy*), e protocolos como o CORS (*cross-origin resource sharing*) elevam a capacidade do navegador em manter um ambiente de execução seguro. Ambos se baseiam na

noção do “domínio” como identificador da origem e, por consequência, da confiabilidade de um recurso: o domínio da página, denotado pela combinação do *protocolo*, *nome do host* e *porta TCP* de onde o navegador requisitou o conteúdo carregado, é tido como o mais confiável, enquanto recursos requisitados de domínios diferentes são considerados menos confiáveis.

Contudo, tal ambiente é protegido dentro da condição de que todo conteúdo ativo carregado por uma aplicação está sob o conhecimento e confiança de seu desenvolvedor, o que nem sempre é o caso (HEULE et al., 2015). Dentro da estrutura de documento da página web as informações dos usuários permanecem fundamentalmente expostas a *scripts* mal-intencionados ou mal-escritos, executados em contexto da página ou como extensões do navegador. Criar um *script* destinado a ler o conteúdo potencialmente sigiloso de uma página da web e revelá-lo a terceiros não autorizados é uma tarefa que exige pouca habilidade e que pode passar despercebida pelo aparato de segurança disponível, incluindo as restrições de SOP e CORS.

Código *inline* ou *scripts* baixados pelas páginas da web são executados com os mesmos privilégios e mesmo nível de acesso à estrutura de documento do navegador, o chamado DOM (*document object model*) (DeRyck et al., 2012, p. 2-3), não importando o domínio de origem dos scripts. Uma demonstração do problema pode ser exemplificada na figura 1 e listagem de código 1.1. Nesse exemplo, um script tido como benigno é incorporado a uma página web a partir de um domínio de CDN (*content delivery network*), diferente daquele da aplicação que efetivamente publica a página. O servidor da página, pelo protocolo CORS, sinaliza ao navegador que o domínio da CDN é confiável. O script externo pode, então, iniciar requisições ao seu domínio de origem – uma consequência desejada pelos autores da página, pois o script depende desse acesso para efetuar suas funções.

Em momento posterior, o script servido pelos servidores da CDN é substituído por código malicioso que, além de efetuar as funções do script benigno, captura o conteúdo da página armazenado no DOM 2. O script pode buscar informações específicas e potencialmente sensíveis como identificação do usuário, senhas e endereços. Por causa da autorização concedida pelo protocolo CORS, o código mal intencionado tem a chance de transmitir o conteúdo capturado para um serviço anômalo.

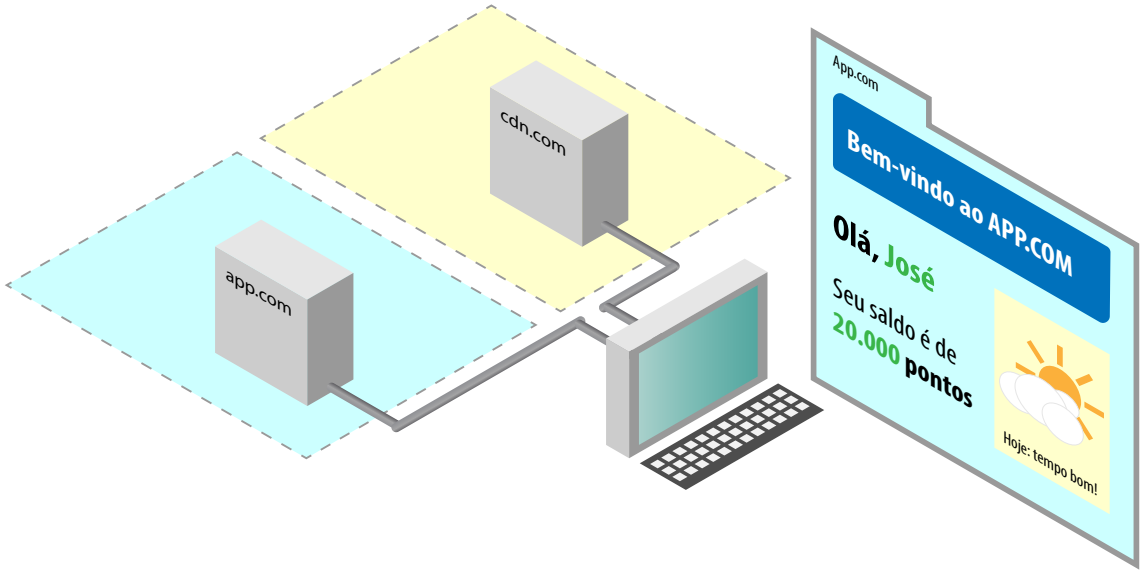

Figura 1 – Aplicação web composta por conteúdo proveniente de duas origens.

```
1  <html>
2    <head>
3      <title>App.com</title>
4      <script src="//cdn.com/previsao-do-tempo.js"></script>
5    </head>
6
7    <body>
8      <h1>Bem-vindo ao APP.COM</h1>
9
10     <div>
11       <div>
12         <p>Olá, <span class="userInfo">José</span></p>
13
14         <p>Seu saldo é de <span class="userInfo">20.000</span> pontos</p>
15       </div>
16
17       <div id="appletPrevisaoTempo"></div>
18     </div>
19   </body>
20 </html>
```

Código 1.1 – Incorporação de script de outra origem (linha 4)

Acessar, capturar e modificar informações contidas no DOM também são efeitos de extensões do navegador. Mas, diferentemente dos scripts incorporados em páginas, extensões são executadas em modo privilegiado e podem afetar todas as páginas carregadas pelo navegador, não sendo confinadas a domínios específicos. Extensões como as do Google Chrome são publicadas exclusivamente em site específico e protegido, mas não é impossível que o código fonte de extensões seja descaracterizado e publicado pela ação de *hackers* (SPRING, 2017), afetando a todos os usuários que atualizarem a extensão – um processo automático por padrão (GOOGLE, 2017).

Na raiz das vulnerabilidades está a forma inconsistente e parcial com que a lingua-

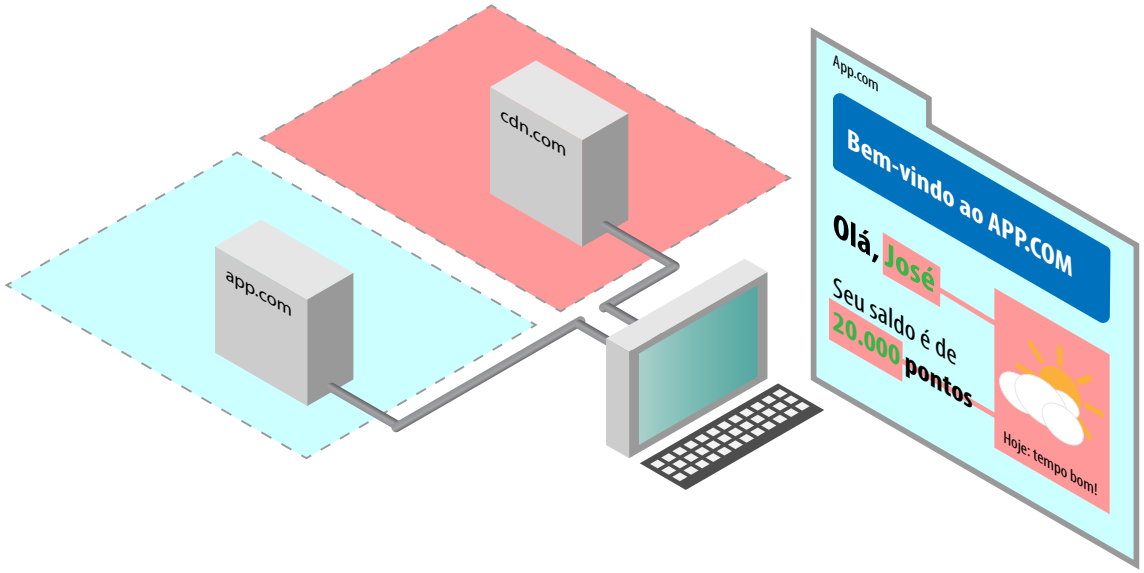

Figura 2 – Domínio de CDN comprometido, capturando informações do usuário.

gem Javascript e as APIs do navegador tratam o relacionamento entre os *scripts*, o acesso à rede, e à estrutura do DOM. Uma das formas propostas para a solução dessas inconsistências seria a introdução de uma política de segurança que monitorasse o fluxo da informação na linguagem Javascript (**??**, p.3). Dessa forma, toda cópia, referência ou manipulação de dados na linguagem dependeria de uma validação dos contextos de segurança atribuídos ao dado e aos recipientes envolvidos. Isso impediria, por exemplo, que informações sensíveis presentes em um nó do DOM fossem lidas e transmitidas para endereços da web desautorizados – um impedimento que não seria obedecido de forma discricionária, imperativa, mas de modo mandatório, declarativo e integrado ao *runtime* da linguagem. Essa abordagem, conhecida pela sigla IFC (*information flow control*), não é compatível com os navegadores existentes. Mecanismos de suporte ao IFC são implementados como navegadores experimentais (**??**BICHHAWAT et al., 2014), o que impede sua adoção maciça.

Assim, se os meios existentes para proteção contra o vazamento de informação contida no DOM são insuficientes, e se um mecanismo robusto como o IFC ainda não foi incorporado aos navegadores tradicionais, pode ser importante propor uma abordagem que desse ao desenvolvedor um mecanismo que, embora discricionário, implementasse uma barreira que impedisse o acesso não prescrito aos nós, ou regiões, do DOM, a critério do desenvolvedor. Isso tornou-se uma possibilidade com a introdução de APIs como a de suporte ao *Shadow DOM* (**??**), que fornecem um grau

de invisibilidade a determinados conteúdos de HTML. Propor uma melhoria da segurança da informação baseada em APIs padronizadas, imediatamente disponíveis aos desenvolvedores e usuários, é a proposta deste trabalho.

1.2 Objetivo

O objetivo deste trabalho é propor uma abordagem para a ocultação da informação contida no DOM usando recursos padronizados de HTML e Javascript. A proposta deverá permitir ao desenvolvedor a delimitação de regiões do DOM cujo conteúdo seja opaco para scripts incorporados e extensões do navegador.

A efetividade da proposta deve ser avaliada segundo requisitos não funcionais enumerados por (DeRyck et al., 2012):

a) Efetividade da separação do DOM: a estrutura de documento mantida em regiões ocultas pela abordagem proposta é separada do restante da página;
b) Efetividade do isolamento de scripts: scripts carregados em regiões ocultas não poderão sofrer influência de scripts externos a essas regiões;
c) Confidencialidade: a informação mantida em regiões ocultas só poderá ser lida por scripts especialmente criados para esse fim pelo desenvolvedor da aplicação;
d) Integridade: a informação mantida em regiões ocultas só poderá ser modificada por scripts especialmente criados para esse fim pelo desenvolvedor da aplicação;
e) Autenticidade: o protocolo de comunicação com regiões ocultas deve suportar apenas participantes que confiem uns aos outros.

Requisitos funcionais da proposta devem satisfazer sua compatibilidade com os navegadores modernos, não-experimentais:

a) Permitir que qualquer combinação de elementos HTML seja encapsulada em regiões ocultas;

b) Ser compatível com bibliotecas e *frameworks* de desenvolvimento em Javascript, HTML e CSS;

c) Expor uma interface de programação para a leitura e modificação das informações contidas em regiões ocultas.

Como objetivo secundário, será implementada uma prova de conceito que valide os requisitos estabelecidos.

1.3 Contribuições

(STEFAN et al., 2014) propõe um navegador cujo ambiente de execução de Javascript é refatorado para suportar controle de acesso ao estilo MAC, em detrimento da abordagem alinhada ao estilo DAC implementado pelos navegadores comuns. MAC, como proposto por (STEFAN et al., 2014), é alcançado pela implementação do controle do fluxo da informação (IFC) no *runtime* de Javascript como premissa para a segurança da informação. Outros trabalhos (HEDIN et al., 2016; BICHHAWAT et al., 2014) propõem abordagens similares, vinculadas ao emprego do IFC.

(MAGAZINIUS et al., 2014) e (DeRyck et al., 2012) comparam estratégias voltadas para a segurança da informação e seus casos de uso. Aplicar IFC, no momento, é uma opção que exclui todos os navegadores comuns, exigindo a utilização de software experimental. Esta não é uma alternativa para o desenvolvedor de aplicações web, já que estas são disponibilizadas para uso em qualquer navegador, em múltiplas plataformas. Em oposição a essas propostas, este trabalho contribui com uma abordagem DAC baseada em APIs disponíveis em navegadores de ampla utilização.

1.4 Método de trabalho

Os objetivos deste trabalho serão o produto de uma sequência de atividades dedicadas à exploração do problema e elaboração da solução. O método de trabalho, então, é composto das atividades enumeradas a seguir.

a) **Levantamento bibliográfico** Esta atividade realiza-se pela pesquisa de trabalhos relacionados à segurança da informação no software navegador, incluindo

contribuições relevantes que deram embasamento a esses trabalhos.

b) **Coleta de evidências** Nesta atividade, os problemas-alvo motivadores deste trabalho serão materializados em simulações e casos de teste. As evidências deverão provar que é possível efetuar as seguintes ações sem o conhecimento ou consentimento do usuário:

   – scripts provenientes de domínios diferentes podem observar o conteúdo de páginas da web, incluindo identificações, senhas, códigos de cartão de crédito e números de telefone, desde que essas informações estejam presentes no DOM;
   – scripts agindo em extensões do navegador podem observar o conteúdo de páginas da web e de seus *iframes*;
   – scripts de qualquer natureza podem registrar o comportamento do usuário ao interagir com a página, capturando eventos de teclado e de mouse;
   – scripts de qualquer natureza conseguem interceptar APIs e com isso extrair informações que transitem pelas interfaces de programação do DOM e da linguagem Javascript.

c) **Proposição** O objetivo desta atividade é elaborar um método para que os objetivos do trabalho sejam alcançados, em aderência aos requisitos funcionais e não-funcionais estabelecidos.

d) **Implementação** Esta atividade tem a finalidade de produzir um componente de HTML e Javascript compatível com os requisitos estabelecidos pela proposta.

e) **Avaliação do método** Nesta tarefa, o componente implementado será submetido à avaliação de sua eficácia frente às vulnerabilidades, e de sua compatibilidade em relação aos requisitos do método.

f) **Síntese dos resultados** A partir das observações produzidas na atividade de avaliação, será elaborada uma síntese dos resultados alcançados em contraste com os objetivos desta proposta.

1.5 Organização do trabalho

A seção 2, Estado da Arte, enquadra o tema sob três pontos de vista: (1) das vulnerabilidades derivadas da tecnologia atual, (2) dos recursos implementados pelos navegadores para a detenção de determinados ataques à segurança da informação, e (3) das propostas experimentais para a mitigação de vulnerabilidades. O panorama formado por esses três pontos de vista corresponde ao contexto em que as contribuições deste trabalho estão inseridas.

A seção 3, Encapsulamento da Informação via *shadow DOM*, descreve um método para o desenvolvimento de componentes de HTML que mantenham invisíveis, para o restante da página, as informações mantidas ou geradas por esses componentes, ao mesmo tempo em que expõe uma interface de programação baseada em controle do acesso à informação encapsulada. São apresentadas nesta seção a disponibilidade dos recursos necessários para a implementação do método, bem como suas limitações de uso.

Na seção 4, Avaliação, são propostos critérios para a verificação da eficácia do método proposto: disponibilidade nas plataformas de navegação, limites de proteção versus vulnerabilidades mitigadas, e requisitos de funcionamento. A seção se completa com a aplicação desses critérios sobre o método proposto, em comparação com trabalhos embasados pela abordagem de IFC – o controle de fluxo de informação define, no âmbito do problema, maior granularidade na segurança da informação em Javascript, ao custo da compatibilidade com a base instalada de navegadores.

O conteúdo da seção 5, Conclusões, deriva da reflexão crítica sobre a implementação do método proposto em contraponto aos resultados observados na avaliação qualitativa. Recomendações sobre a aplicação do método, além de oportunidades a serem exploradas por trabalhos futuros, fecham a conclusão dos esforços deste trabalho.

2 PRINCIPAIS CONCEITOS E ESTADO DA ARTE

2.1 Principais conceitos

Nesta seção são apresentados os conceitos que embasam este e outros trabalhos relacionados ao tema da segurança da informação em aplicações da web.

2.1.1 Segurança da informação

Segundo (ISO, 2016), segurança da informação é um processo com os objetivos de "preservação da confidencialidade, integridade e disponibilidade da informação". (FOSTER et al., 1998) elabora esses objetivos, descrevendo a confidencialidade como a condição na qual a informação só pode ser acessada pelos agentes autorizados, integridade como a capacidade de proteger a informação contra modificações não autorizadas, e a disponibilidade como a capacidade de garantir acesso à informação quando necessário; (FOSTER et al., 1998) ainda atribui mais duas características a um sistema de segurança da informação: *accountability* como a possibilidade de se atribuir um agente para cada ação ocorrida dentro do sistema, e *assurance* como o grau de confiabilidade na segurança do sistema em relação aos seus objetivos declarados.

Neste trabalho, qualquer definição de segurança da informação será restrita aos sistemas de informação relacionados com a navegação de usuários através da web: provedores de serviço (*sites*, servidores da web), protocolos de comunicação em rede (HTTP, HTTPS, *web sockets*), navegadores (*browsers*) e os ambientes de execução de Javascript embutidos nos navegadores. Isto delimita a área de conhecimento relevante para este trabalho.

2.1.2 Modelos de controle de acesso

Enquanto a definição dos requisitos de segurança da informação estabelece seus objetivos, os modelos definem os sistemas derivados desses objetivos (GOGUEN; MESEGUER, 1982). (FOSTER et al., 1998) menciona diferentes modelos de controle de acesso, categorizados de modo amplo como modelos discricionários (DAC – *discretio-*

nary access control) e mandatórios (MAC – *mandatory access control*). Modelos discricionários se baseiam na definição dos relacionamentos de segurança entre agentes e objetos em um sistema, como, por exemplo, a política de que um script – a parte *agente* – não pode iniciar conexões com domínios diferentes do seu próprio – a parte *objeto*. Modelos discricionários são os mais comumente utilizados para estabelecer mecanismos de segurança nos navegadores. O campo de atuação desses modelos é limitado aos relacionamentos de segurança estabelecidos, e portanto não podem garantir a segurança da informação quando esta ultrapassa o domínio desses relacionamentos. Isto significa, por exemplo, que dados legitimamente obtidos dentro de regras discricionárias pode ser replicado para um contexto não-seguro sem qualquer impedimento derivado do modelo de segurança.

Modelos mandatórios não atribuem explicitamente as regras de controle de acesso aos objetos e agentes de um sistema. Ao invés disso, estabelecem níveis de confidencialidade utilizados para classificar os participantes do sistema de informação, viabilizando o controle dinâmico do trânsito da informação entre os agentes. Num modelo mandatório, o nível de segurança de um dado impede que ele seja obtido ou modificado por agentes com níveis de segurança mais baixos. O controle do fluxo da informação faz dos MACs modelos mais robustos do que os DACs (FOSTER et al., 1998).

2.1.3 Controle do fluxo de informações

O controle do fluxo de informações (IFC – *information flow control*) é um mecanismo que atua, em tempo de execução, nos meios de propagação dos valores entre os espaços de armazenamento de um sistema computacional de modo a impedir fluxos não autorizados dos dados (DENNING, 1976). IFC é um modelo discricionário e baseia-se em *classes de segurança* “altas” e “baixas”, simbolizadas pelas letras `<h>` e `<l>`, respectivamente, para indicar graus de confidencialidade das informações e dos seus espaços de armazenamento (*heap*, pilha, redes, dispositivos etc). Operações entre entidades com classes de segurança diferentes, como a cópia do valor de uma variável `<h>` (confidencial) para a variável `<l>` (pública), são automaticamente impedidas de prosseguir.

IFC distingue entre fluxos de informação explícitos e implícitos. Um fluxo explícito

ocorre quando uma informação classificada como “alta” é diretamente copiada para um contexto de classificação “baixa”, como na listagem de código 2.1. Em um fluxo implícito, não é a informação em si que transita entre contextos de classificação diferente, mas sim alguma informação derivada dela através da qual seja possível fazer qualquer inferência sobre seu conteúdo. Um exemplo de fluxo implícito encontra-se la listagem 2.2. Um mecanismo que suporte IFC deve ser capaz de interromper vazamento de informação em ambos os tipos de fluxo.

```
1  var revelaH = function(h) {
2    // O valor do parâmetro <h>, tido como confidencial, é explicitamente
3    // propagado para o domínio www.evil.com:
4    makeAjaxCall('www.evil.com/tell/secret/' + h);
5  }
6
7  var h = document.getElementById('password').value;
8  revelaH(h);
```

Código 2.1 – Vazamento de dados em fluxo explícito de informação

```
1  var revelaH = function(h) {
2    // Uma pista sobre o valor do parâmetro <h>, tido como confidencial,
3    // é propagada para o domínio www.evil.com:
4    var pista = h.length;
5
6    // Embora o valor <h> não tenha sido propagado, uma característica
7    // implícita do seu conteúdo foi revelada.
8
9    makeAjaxCall('www.evil.com/tell/hint?passwordLength=' + pista);
10   // Agora, www.evil.com sabe o tamanho da senha utilizada, e pode usar essa
11   // informação para fazer uma inferência sobre a senha utilizada.
12 }
13
14 var h = document.getElementById('password').value;
15 revelaH(h);
```

Código 2.2 – Vazamento de dados em fluxo implícito de informação

2.1.4 SOP – Same Origin Policy

A política de segurança SOP foi estabelecida para que os navegadores conseguissem dar suporte a páginas com conteúdo proveniente de domínios mistos com um mínimo de segurança contra o vazamento de informação entre esses domínios (HILL, 2016). Através desta política, os navegadores podem impedir um conjunto de ataques

conhecido como *cross-site resource forgery*, em que um domínio tenta instruir o navegador a fazer requisições para outro domínio em nome do usuário.

O termo *origem* é intercambiável com a expressão *domínio* e ambos representam, para fins desta política, componentes do endereço de URL associado com cada recurso da web – a saber, o *protocolo*, o *nome do host* e a *porta TCP* de onde o recurso foi transferido (BARTH, 2011). Os exemplos a seguir representam recursos de mesma origem:

Endereço	Protocolo	Nome do *host*	Porta
`http://exemplo.com/`	http	exemplo.com	80
`http://exemplo.com:80/`	http	exemplo.com	80
`http://exemplo.com/path/file`	http	exemplo.com	80

Os endereços a seguir representam recursos de origens diferentes:

Endereço	Protocolo	Nome do *host*	Porta
`http://exemplo.com/`	http	exemplo.com	80
`http://exemplo.com:8080/`	http	exemplo.com	8080
`http://www.exemplo.com/`	http	www.exemplo.com	80
`https://exemplo.com:80/`	https	exemplo.com	80
`https://exemplo.com/`	https	exemplo.com	443
`http://exemplo.org/`	http	exemplo.org	80

Segundo a SOP, as atividades derivadas da inclusão de recursos de origens mistas são categorizadas em três ações (RUDERMAN, 2017):

a) **Escrita:** atividades deste tipo instruem o navegador para que ocorra alguma forma de navegação entre páginas, o que inclui a interação com *links*, redirecionamento e submissão de formulários. Em geral SOP não restringe este tipo de ação;

b) **Incorporação:** SOP permite que recursos incorporados à página tenham origens mistas. Isto significa que é possível a inclusão de imagens, vídeos, scripts e do elemento `<iframe>`, entre outros, provenientes de origens mistas e dentro de uma mesma página.

c) **Leitura:** atividades de leitura permitiriam que o conteúdo dos recursos carregados pudesse ser consultado entre origens. SOP permite que um subconjunto de funcionalidades de leitura possam ocorrer entre domínios diferentes.

Um aspecto importante da SOP é o tratamento dado a scripts incorporados. Quando uma página inclui um script proveniente de outras origens, por exemplo pelo uso de uma CDNs (*content distribution networks*), esses scripts são executados em contexto da origem do documento em que eles foram incorporados. Isto permite, por exemplo, que *frameworks* populares como jQuery e Angular.js possam ser disponibilizados em CDNs sem perder funcionalidades importantes, como a capacidade de iniciar chamadas assíncronas pela técnica AJAX. Esta concessão da SOP, porém, abre a possibilidade de que esses scripts, se adulterados, executem atividades maliciosas sem impedimentos.

2.1.5 CSP – Content Security Policy

CSP foi criada como um complemento à SOP, elevando a capacidade do navegador de servir como plataforma razoavelmente segura para composição de aplicações *mashup* ao estabelecer um protocolo para o compartilhamento de dados entre os componentes da página que residam em domínios diferentes. CSP define um conjunto de diretivas (codificadas como cabeçalhos HTTP) para a definição de *whitelists* – o conjunto de origens confiáveis em um dado momento – pelas quais navegador e provedores de conteúdo estabelecem o controle de acesso e o uso permitido de recursos embutidos como scripts, folhas de estilos, imagens e vídeos, entre outros. Através desse protocolo, ataques de XSS que podem ser neutralizados desde que todos os componentes na página sejam aderentes à mesma política de CSP.

2.1.6 CORS – Cross-Origin Resource Sharing

Assim como a CSP, o mecanismo CORS (W3C, 2014) complementa a SOP estabelecendo um conjunto de diretivas (cabeçalhos HTTP) para a negociação de acesso via Ajax/XHR a recursos hospedados em domínios diferentes. CORS determina que exista um vínculo de confiança entre navegadores e provedores de conteúdo, dificul-

tando vazamento de informação ao mesmo tempo em que flexibiliza as funcionalidades das APIs. O uso de CORS permite que os autores de componentes e desenvolvedores de aplicações *mashup* determinem o grau de exposição que cada conteúdo pode ter em relação aos outros conteúdos incorporados.

CSP e CORS são recomendações do comitê W3C (BARTH et al., 2016) (W3C, 2014), sendo incorporados por todos os navegadores relevantes desde 2016 (DEVERIA, 2016a) (DEVERIA, 2016b).

2.1.7 Vulnerabilidades

Violações de privacidade são possíveis nos navegadores por causa da natureza dinâmica da linguagem Javascript e de sua ausência de restrições de segurança em tempo de execução (JANG et al., 2010). Seus usuários estão expostos a ataques sutis com objetivos diversos como roubar *cookies* e *tokens* de autorização, redirecionar o navegador para sites falsos (*phishing*), observar o histórico de navegação e rastrear o comportamento do usuário através dos movimentos do ponteiro do mouse e eventos de teclado. Para que scripts mal-intencionados sejam incorporados a páginas benignas, *hackers* fazem uso de vulnerabilidades como *cross-site scripting (XSS)* (OWASP, 2016) e comprometimento de extensões (HEULE et al., 2015) do navegador.

2.1.7.1 Cross-Site Scripting (XSS)

Em Javascript, todos os recursos de código carregados dentro de uma mesma página possuem os mesmos privilégios de execução. Ataques do tipo *cross-site scripting* tiram proveito dessa característica para injetar código malicioso em contextos onde seja possível observar e retransmitir informação sigilosa como *cookies* do usuário, endereço do navegador, conteúdo de formulários, ou qualquer outra informação mantida pelo DOM.

O emprego de medidas para prevenção de ataques XSS (WILLIAMS et al., 2016) não elimina riscos inerentes à tecnologia do navegador. Uma vez que componentes incorporados, como anúncios e *players* de mídia, conseguem carregar scripts tidos como confiáveis dinamicamente, um único trecho de código comprometido pode colo-

car informações em risco sem qualquer interferência dos dispositivos de segurança.

2.1.7.2 Comprometimento de extensões

Os mecanismos de extensibilidade oferecidos pelos navegadores[1] melhoram a funcionalidade da web para os usuários, e o código de que são feitos é executado com privilégios mais elevados do que o dos scripts incorporados pelos *sites*. Por isso, os usuários precisam confirmar ao navegador que aceitam que uma extensão seja instalada, sendo informados a respeito dos privilégios que a extensão pretende utilizar. O fato de que esse processo precisa se repetir a cada vez que uma extensão necessita de um conjunto de privilégios diferente faz com que os desenvolvedores optem por solicitar, de antemão, uma gama de privilégios maior que a estritamente necessária (HEULE et al., 2015).

Uma extensão que tiver sido comprometida (por exemplo, ao usar scripts de terceiros que, por sua vez, tenham sido redirecionados ou adulterados) terá assim poder para ler e transmitir todo o conteúdo carregado e exibido pelo navegador, com o potencial de causar os mesmos efeitos observados em um ataque XSS, mas em escopo e poder aumentados, já que poderiam afetar todas as páginas abertas e todas as APIs publicadas pelo navegador.

[1] "Extensões" e *"apps"* do Chrome; e "complementos" do Firefox e do Internet Explorer

3 PROPOSTA (EM DESENVOLVIMENTO)

3.1 Roteiro para a elaboração da proposta

a) **Mapeamento de cenários.** Partindo dos requisitos não funcionais estabelecidos, serão definidos cenários de testes capazes de evidenciar as condições necessárias para a não-conformidade com esses requisitos. Os cenários assim definidos delimitarão o escopo de ação da abordagem proposta por este trabalho.

b) **Especificação da proposta.** A abordagem derivada dos cenários de testes será formalmente especificada, em termos das interfaces e funcionalidades esperadas, visando orientar o desenvolvimento de uma implementação de teste e provas de conceito.

c) **Implementação do mecanismo de ocultação.** A materialização da abordagem proposta será efetuada pelo desenvolvimento dos scripts necessários para a implementação das interfaces esperadas, em conformidade com os requisitos funcionais esperados.

d) **Validação.** Os cenários de testes serão implementados na forma de provas de conceito para validação da proposta e do mecanismo implementado. O objetivo é conhecer a aderência do mecanismo em relação aos requisitos.

REFERÊNCIAS

BARTH, A. **The Web Origin Concept**. 2011. Disponível em: <https://tools.ietf.org/html/rfc6454>.

BARTH, A. et al. **Content Security Policy Level 2**. [S.l.], 2016. Https://www.w3.org/TR/2016/REC-CSP2-20161215/.

BICHHAWAT, A. et al. Information Flow Control in WebKit's JavaScript Bytecode. In: **Principles of Security and Trust**. [S.l.: s.n.], 2014. p. 159–178. ISBN 978-3-642-54792-8.

DENNING, D. E. A lattice model of secure information flow. **Commun. ACM**, v. 19, n. 5, p. 236–243, 1976. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/360051.360056>.

DeRyck, P. et al. Security of web mashups: A survey. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 7127 LNCS, p. 223–238, 2012. ISSN 03029743.

DEVERIA, A. **Can I Use: Content Security Policy 1.0**. 2016. Disponível em: <http://caniuse.com/\#search=Contentsecuritypolicy>.

____. **Can I Use: Cross-Origin Resource Sharing**. 2016. Disponível em: <http://caniuse.com/\#feat=cors>.

DORFMAN, J. **BootstrapCDN Security Post-Mortem**. 2013. Disponível em: <https://www.maxcdn.com/blog/bootstrapcdn-security-post-mortem/>.

FORREST, C. **Warning: These 8 Google Chrome extensions have been hijacked by a hacker**. 2017. Disponível em: <https://www.techrepublic.com/article/warning-these-8-google-chrome-extensions-have-been-hijacked-by-a-hacker/>.

FOSTER, I. et al. A security architecture for computational grids. In: **Proceedings of the 5th ACM conference on Computer and communications security - CCS '98**. New York, New York, USA: ACM Press, 1998. v. 44, n. 2, p. 83–92. ISBN 1581130074. Disponível em: <http://portal.acm.org/citation.cfm?doid=288090.288111>.

GOGUEN, J. A.; MESEGUER, J. Security Policies and Security Models. In: **1982 IEEE Symposium on Security and Privacy**. IEEE, 1982. p. 11–11. ISBN 0-8186-0410-7. Disponível em: <http://ieeexplore.ieee.org/document/6234468/>.

GOOGLE. **Autoupdating**. 2017. Disponível em: <https://developer.chrome.com/extensions/autoupdate>.

HEDIN, D. et al. Information-flow security for JavaScript and its APIs. **Journal of Computer Security**, v. 24, n. 2, p. 181–234, 2016. ISSN 0926227X.

HEULE, S. et al. The Most Dangerous Code in the Browser. **Usenix**, 2015.

HILL, B. **CORS for Developers**. 2016. Disponível em: <https://w3c.github.io/webappsec-cors-for-developers/>.

ISO. ISO/IEC 27000: 2016 Glossary. **ISO.org [Online]**, v. 2016, p. 42, 2016. Disponível em: <http://standards.iso.org/ittf/PubliclyAvailableStandards/>.

JANG, D. et al. An empirical study of privacy-violating information flows in JavaScript web applications. **Proceedings of the 17th ACM conference on Computer and communications security - CCS '10**, p. 270, 2010. ISSN 15437221.

MAGAZINIUS, J. et al. Architectures for inlining security monitors in web applications. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 8364 LNCS, p. 141–160, 2014. ISSN 16113349.

OWASP. **Cross-site Scripting (XSS)**. OWASP, 2016. Disponível em: <https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)>.

RUDERMAN, J. **Same-origin Policy**. 2017. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin{_}pol>.

SEGURA, J. **Malvertising on Equifax, TransUnion tied to third party script**. 2017. Disponível em: <https://blog.malwarebytes.com/threat-analysis/2017/10/equifax-transunion-websites-push-fake-flash-player/>.

SPRING, T. **Copyfish Browser Extension Hijacked to Spew Spam**. 2017. Disponível em: <https://threatpost.com/copyfish-browser-extension-hijacked-to-spew-spam/127125/>.

STEFAN, D. **Principled and Practical Web Application Security**. 30–31 p. Tese (Doutorado) — Stanford University, December 2015.

STEFAN, D. et al. Protecting Users by Confining JavaScript with COWL. **OSDI**, 2014.

VANUNU, O. eBay Platform Exposed to Severe Vulnerability. **Check Point Threat Research**, 2016. Disponível em: <http://blog.checkpoint.com/2016/02/02/ebay-platform-exposed-to-severe-vulnerability/>.

W3C. **Same Origin Policy**. 2010. Disponível em: <https://www.w3.org/Security/wiki/Same_Origin_Policy>.

____. **Cross-Origin Resource Sharing**. 2014. Disponível em: <https://www.w3.org/TR/cors/>.

WILLIAMS, J. et al. **XSS (Cross Site Scripting) Prevention Cheat Sheet**. OWASP, 2016. Disponível em: <https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet>.