Humberto Borgas Bulhões

Método para encapsulamento da informação em aplicações de HTML e Javascript

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
Método para encapsulamento da informação em aplicações de HTML e Javascript/ 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

Impulsionados por recursos avançados de HTML e Javascript, os programas navegadores da web tornaram-se plataformas de desenvolvimento e publicação de aplicativos. O aumento em escopo de funcionalidade dos navegadores, porém, vem acompanhado de vulnerabilidades na segurança da informação que podem expor dados do usuário ao uso não autorizado. Muitas dessas vulnerabilidades se manifestam de formas sutis, fora do alcance das políticas de segurança padronizadas e adotadas pela indústria, e afetando a confidencialidade da informação que trafega pelo navegador. Para neutralizar esse problema diversas iniciativas experimentais têm sido propostas, sem que tenham encontrado, até o momento, adoção em massa. Ao mesmo tempo, novos recursos de programação têm sido incorporados aos navegadores, e alguns deles, quando utilizados em conjunto, levantam a possibilidade de implementação do encapsulamento da informação, tornando-a invisível a agentes não autorizados. O objetivo deste trabalho é propor um método que, fundamentado em capacidades de programação padronizadas e difundidas, implemente o encapsulamento da informação em HTML e Javascript, o que poderia tornar realidade alguns dos requisitos de segurança que apenas abordagens experimentais têm conseguido assegurar. Para tanto, este trabalho apresentará uma especificação desse método, uma implementação modelo e uma avaliação da eficácia do método em relação a determinados casos de teste que evidenciam problemas de confidencialidade da informação.

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

Sumário

1 INTRODUÇÃO 7
1.1 Motivação 7
1.2 Objetivo 9
1.3 Resultados esperados e contribuições 9
1.4 Método de trabalho 10
1.5 Organização do trabalho 12

2 ESTADO DA ARTE 15
2.1 Estado da arte 15
2.1.1 Vulnerabilidades 15
2.1.1.1 Cross-Site Scripting (XSS) 16
2.1.1.2 Comprometimento de extensões 16
2.1.2 Abordagens tradicionais para contenção de ataques 17
2.1.2.1 SOP – Same Origin Policy 17
2.1.2.2 CSP – Content Security Policy 17
2.1.2.3 CORS – Cross-Origin Resource Sharing 18
2.1.3 Abordagens experimentais para contenção de ataques 18
2.1.3.1 Controle do fluxo de informações 19
2.1.3.2 An empirical study of privacy-violating information flows in JavaScript web applications (JANG et al., 2010) 20
2.1.3.3 Security of web mashups: A survey (DeRyck et al., 2012) 20
2.1.3.4 Information-Flow Security for a Core of JavaScript (HEDIN; SABELFELD, 2012) 20
2.1.3.5 Toward Principled Browser Security (YANG et al., 2013) 21
2.1.3.6 JSFlow: Tracking information flow in JavaScript and its APIs (HEDIN et al., 2014) 21
2.1.3.7 Information Flow Control in WebKit's JavaScript Bytecode (BICHHAWAT et al., 2014) 22
2.1.3.8 Protecting Users by Confining JavaScript with COWL (STEFAN et al., 2014) 23
2.1.3.9 Architectures for inlining security monitors in web applications (MAGAZINIUS et al., 2014) 24
2.1.3.10 Information Flow Control for Event Handling and the DOM in Web Browsers (RAJANI et al., 2015) 24

REFERÊNCIAS 25

1 Introdução

1.1 Motivação

É difícil desenvolver aplicações seguras para a web. Mesmo sob a defesa de *firewalls*, protocolos de segurança, software antivírus e atualizações periódicas dos navegadores, as informações dos usuários permanecem fundamentalmente expostas ao acesso indevido por *scripts* mal-intencionados ou mal-escritos. Essa condição se expressa de forma crítica por meio do chamado conteúdo ativo, recurso do navegador que viabiliza a existência de aplicações interativas nas linguagens HTML e Javascript. Criar um *script* com a intenção de 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.

Exemplos de manipulação maliciosa de scripts incluem sequestro de redes de distribuição de conteúdo (CDN, *content distribution network*) (DORFMAN, 2013), injeção de scripts por extensões, por vírus ou por redirecionamento (KINLAN, 2015), e ainda a inclusão de código não verificado pelo publicador (VANUNU, 2016). Tais tipos de ataques não tiram proveito de defeitos dos navegadores; ao invés disso, funcionam de acordo com os mecanismos de segurança da informação padronizados pela indústria. Isto ocorre porque parte da tecnologia que dá suporte ao conteúdo ativo é fundamentalmente limitada no nível de segurança da informação que pode oferecer, especialmente quando é considerado o rico ambiente de execução proporcionado pelo software navegador. Nele, a confidencialidade da informação é vulnerável a agentes maliciosos que operam em dois âmbitos:

a) Componentes de *script* incorporados às páginas da web são executados com os mesmos privilégios e mesmo nível de acesso à página como um todo (DeRyck et al., 2012, p. 2-3), sejam eles benignos ou não. O navegador oferece diretivas de segurança limitadas para conter o vazamento de informação entre componentes, uma vez que não há um meio de especificar relações de confiança entre eles (JANG et al., 2010).

b) *Plugins* e extensões do navegador têm nível de acesso mais elevado aos recursos de sistema do que aquele definido para componentes de páginas da web, permitindo que *plugins* tomem controle de funcionalidades como gerenciamento

de conexões de rede, eventos de navegação e o DOM. Ainda que dependam da permissão explícita do usuário no momento de sua instalação, os *plugins* e extensões podem se comportar como agentes de vazamento de informação de forma sutil e não necessariamente proposital (HEULE et al., 2015a).

Na raiz das vulnerabilidades está a forma inconsistente e parcial com que a linguagem Javascript e as APIs do navegador tratam o isolamento entre *scripts* e dados. A caracterização e a solução desse problema fazem parte de um campo de pesquisas ativo (STEFAN et al., 2014), (HEDIN et al., 2014), (BICHHAWAT et al., 2014), (MAGAZINIUS et al., 2014) e em busca por padronização (W3C, 2017b), mas que ainda não resultou em práticas, ferramentas e protocolos de amplo alcance pois dependem da adoção de políticas de segurança experimentais (HEDIN et al., 2014), (BICHHAWAT et al., 2014) e potencialmente degradantes de desempenho (STEFAN et al., 2014, p. 14) por parte dos desenvolvedores de navegadores.

No campo das tecnologias experimentais estão as estratégias baseadas no controle do fluxo da informação (IFC, *information flow control*). IFC, inicialmente descrito por (DENNING, 1976), estabelece que cada espaço de armazenamento em um programa – arquivos, segmentos de memória, conexões de rede ou variáveis, por exemplo – seja rotulado por uma classe de segurança. Em função disso, o trânsito de informação entre espaços de armazenamento deve ser monitorado e, eventualmente, interrompido quando os rótulos da origem e do destino da informação não forem compatíveis. IFC é um mecanismo inexistente nas implementações da linguagem Javascript dos navegadores da web; implementá-lo em uma linguagem dinâmica como Javascript significa introduzir uma checagem de rótulos a cada leitura e escrita em objetos mutáveis (HEULE et al., 2015b, p.3), acarretando perdas significativas na velocidade da execução de *scripts*.

Os desenvolvedores de aplicações em conteúdo ativo, por sua vez, têm limitadas opções para a publicação de *sites* e páginas web seguras. Ainda que sigam as práticas recomendadas para a neutralização dos ataques mais comuns à segurança da informação (W3C, 2014), (W3C, 2010), (BARTH et al., 2016), o conteúdo das páginas da web permanece acessível aos *scripts* incorporados e às extensões do navegador.

1.2 Objetivo

Buscando alternativas de desenvolvimento defensivo que possam ser utilizadas nos navegadores comuns, e não somente em software experimental a exemplo do mecanismo IFC, este trabalho investiga um método para o confinamento de informação usando recursos padronizados de HTML e Javascript. Interfaces de programação (APIs – *application programming interface*) suportadas (ou em vias de adoção) pelos navegadores modernos, como o *shadow DOM* (W3C, 2017a), dispõem de recursos que possibilitariam o encapsulamento de informações em componentes cuja estrutura seja opaca tanto para *scripts* incorporados na página HTML como para *plugins* do navegador. Hipoteticamente, é possível propor um método que, combinando tais recursos de APIs, ofereça ao desenvolvedor os meios para a composição de páginas imunes a ataques de vazamento de informação. Implementando o método e colocando-o à prova por meio de testes que simulam esse tipo de ataque, o objetivo deste trabalho é avaliar a efetividade do método como meio de ofuscação (1) das informações incorporadas e exibidas na página e (2) do comportamento do usuário ao interagir com componentes HTML e Javascript desenvolvidos conforme o método proposto.

1.3 Resultados esperados e contribuições

Abordagens não-experimentais contra vazamentos de dados são preocupação escassa na literatura sobre segurança da informação em Javascript. Este trabalho, ao descrever uma estratégia em que a inviolabilidade da informação contida no software navegador seja assegurada pelo emprego de APIs padronizadas, não-experimentais, contribui com o estado da arte em segurança nas aplicações da web.

Derivado da estratégia avaliada, será proposto um método para encapsulamento de informação que torne determinadas regiões do DOM invisíveis a *scripts* maliciosos. A eficácia do método será validada por testes que simulem, sob diferentes condições, casos específicos de ataques e tentativas de acesso indevido ao conteúdo das regiões protegidas do DOM. É esperado que o método impeça que scripts não autorizados observem e manipulem o conteúdo dessas regiões, bem como torne indetectáveis os eventos de interação do usuário com os elementos das regiões protegidas.

Além do método para encapsulamento, outra contribuição esperada é um algoritmo

para a detecção de sobrecargas inesperadas em métodos e propriedades do DOM. Sem tal algoritmo, o método proposto não terá como prevenir uma forma de intrusão onde um *script* intercepta determinadas APIs do navegador para monitorar e, eventualmente, capturar informação que transite através de suas chamadas. Tal intrusão permitiria tomar controle das próprias APIs utilizadas pelo método, abrindo a possibilidade de vazamento de informação e inutilizando a estratégia.

1.4 Método de trabalho

O método para encapsulamento de informação por *shadow DOM* resultante deste trabalho será o produto de uma sequência de tarefas para a exploração do problema e elaboração da solução. O método de trabalho, então, é composto das atividades enumeradas a seguir.

a) **Coleta de evidências** Nesta primeira atividade, os problemas abordados pelo método proposto serão materializados em simulações e casos de teste que evidenciem acessos não autorizados a informações contidas em componentes de páginas da web. Especificamente, as evidências devem 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 na estrutura da página;
   – 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 funcionalidades do navegador para extrair informações que transitem pelas interfaces de programação do DOM e de Javascript.

   Artefatos derivados dos casos de testes, codificados como páginas da web, serão insumo das atividades de avaliação do método proposto. Parte desta tarefa será dedicada à automação dos casos de teste pelo *framework* de programação

de testes Selenium[1]. Os scripts de automação serão agregados ao conjunto de artefatos de código derivados deste trabalho.

b) **Proposição do método** O objetivo desta atividade é projetar um método de encapsulamento com o qual desenvolvedores de páginas da web possam definir componentes de HTML imunes ao vazamento de informação por meio de Javascript. O método precisará atender aos seguintes requisitos:

   – Permitir que qualquer combinação de elementos HTML seja encapsulável em um componente inviolável por scripts externos a si;
   – Ser compatível com bibliotecas e *frameworks* de desenvolvimento em Javascript, HTML e CSS;
   – Estabelecer um protocolo de confiança entre si e agentes (scripts) externos;
   – Sob esse mesmo protocolo, expor uma interface de programação para a leitura e modificação das informações encapsuladas;
   – Ser compatível com recursos padronizados e não-experimentais de HTML e Javascript.

   O resultado desta atividade é uma especificação técnica da solução proposta, incluindo requisitos e limitações de uso.

c) **Implementação do método** Esta atividade tem a finalidade de produzir um componente de HTML e Javascript compatível com os requisitos estabelecidos pela proposta, seguido pela incorporação desse artefato às páginas web derivadas dos casos de teste.

d) **Avaliação do método** Nesta tarefa, o componente implementado será submetido à avaliação de sua eficácia frente às vulnerabilidades e sua compatibilidade em relação aos requisitos do método. Esse resultado será verificado pelo acionamento dos testes automatizados, que demonstrarão se os problemas evidenciados são neutralizados pela implementação.

e) **Síntese dos resultados** O produto do acionamento dos testes indicará não apenas o sucesso ou falha em cada caso de teste, mas também fornecerá informações como a diferença no desempenho (medido em função do tempo de execução e memória do navegador) e nos erros ou exceções capturadas em

[1] http://www.seleniumhq.org/

tempo de execução, antes e depois da aplicação do método. Eventualmente, falhas de segurança que não forem mitigadas serão diagnosticadas com base nas especificações das APIs utilizadas nos testes e na implementação – isto é, será avaliado se a falha de segurança é uma manifestação do comportamento esperado daquela API. A partir dessas observações será elaborada uma síntese dos resultados alcançados, incluindo uma reflexão sobre a abrangência e utilidade dos artefatos produzidos. As realizações e limitações deste trabalho serão comparadas com aquelas encontradas em trabalhos relacionados experimentais como (HEDIN et al., 2014) e (STEFAN et al., 2014), posicionando definitivamente a contribuição deste trabalho dentro do panorama de segurança da informação.

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 Estado da Arte

2.1 Estado da arte

O estado da arte em segurança da informação nos navegadores é resultado de uma progressiva adaptação deste tipo de software aos recursos de composição e de publicação das aplicações web. Vulnerabilidades têm se manifestado desde o momento em que os navegadores e servidores passaram a dar suporte a *cookies*, à linguagem Javascript, ao DOM e aos recursos de incorporação de mídia, porém as formas de mitigação dessas vulnerabilidades competem entre si e com as funcionalidades de que os desenvolvedores dependem, resultando em regras e práticas inconsistentes e incompletas (HILL, 2016).

Segurança da informação é o processo de “preservação da confidencialidade, integridade e disponibilidade da informação” (ISO, 2016). Neste trabalho, tal definição 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.

Os objetos de estudo são (a) as vulnerabilidades derivadas dos fluxos de informação (GOGUEN; MESEGUER, 1982) (DENNING, 1976) entre os sistemas envolvidos e (b) os meios para a mitigação das vulnerabilidades. Neste capítulo será avaliado o estado da arte dos objeto de estudo fazendo, no caso de (b), uma distinção entre as abordagens “tradicionais” e “experimentais”: as primeiras representam padrões já adotados na indústria e implementados nas plataformas de navegação mais comuns, enquanto as segundas incluem ferramentas e abordagens desenvolvidas experimentalmente para a solução de vulnerabilidades que, embora fundamentais, ainda não foram remediadas nativamente pelos navegadores.

2.1.1 Vulnerabilidades

Violações de privacidade são possíveis nos navegadores por causa da natureza extremamente 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., 2015a), problemas para os quais os navegadores não oferecem ainda proteção total.

2.1.1.1 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 colocar informações em risco sem qualquer interferência dos dispositivos de segurança.

2.1.1.2 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

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

solicitar, de antemão, uma gama de privilégios maior que a estritamente necessária (HEULE et al., 2015a).

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.

2.1.2 Abordagens tradicionais para contenção de ataques

2.1.2.1 SOP – Same Origin Policy

Implementada desde o primeiro navegador com suporte a Javascript, SOP (W3C, 2010) é uma política que impõe limites aos meios pelos quais uma página ou script efetuam requisições a recursos que se encontram em *domínios diferentes*[2]. SOP promove isolamento de informações ao impedir que o conteúdo em um domínio acesse conteúdo que tenha sido carregado em domínios diferentes.

Na prática, essa política restringe funcionalidades importantes sem solucionar completamente o problema do vazamento de informações. SOP impede, por exemplo, que um script inicie requisições assíncronas[3] para outros domínios, ao mesmo tempo em que permite que um script incorporado através de XSS efetue vazamento da identidade do usuário. Em geral, os navegadores diminuem certas restrições da SOP para permitir APIs mais funcionais, e implementam meios mais flexíveis para proteção contra ataques de XSS.

2.1.2.2 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 com-

[2] "Domínios" identificam origens de recursos pela combinação do protocolo, do nome do host e da porta utilizados para o acesso ao recurso.

[3] Através das APIs Ajax e XHR

ponentes 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.

2.1.2.3 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, dificultando 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).

2.1.3 Abordagens experimentais para contenção de ataques

Os mecanismos tradicionais são discricionários, pois as aplicações devem pré-estabelecer explicitamente seus parâmetros de segurança da informação (seja através de CSP ou CORS) para que o navegador configure um contexto de segurança correspondente (STEFAN, 2015, p. 31). Trata-se, ademais, de um controle estático, imutável durante todo o ciclo de vida da página, o que é inadequado para aplicações ao estilo *mashup* em que os componentes da página podem ser desconhecidos no momento em que as diretivas de segurança são aplicadas pelo navegador.

Uma característica fundamental das abordagens tradicionais é sua ênfase na comunicação de rede entre domínios distintos. Esse é um enfoque pertinente: a confiabi-

lidade entre domínios é essencial para garantir um mínimo de segurança. No entanto, o navegador como um todo pode abrir vulnerabilidades que independem de conexões de rede para se concretizarem. *Plugins* e extensões são executados com privilégios elevados e têm acesso a todas as partes do navegador com as quais os usuários interagem, podendo extender dinamicamente a linguagem Javascript para modificar seu funcionamento e rastrear de informações.

2.1.3.1 Controle do fluxo de informações

O

```
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);
```

Listing 2.1 – Fluxo explícito de informação

```
9  var revelaH = function(h) {
10    // Uma pista sobre o valor do parâmetro <h>, tido como confidencial,
11    // é propagada para o domínio www.evil.com:
12    var pista = h.charAt(0) === 's';
13    makeAjaxCall('www.evil.com/tell/hint?startsWith=s&hint=' + pista);
14    // Agora, www.evil.com sabe se o valor confidencial é
15    // um texto iniciado com a letra "s"
16 }
17
18 var h = document.getElementById('password').value;
19 revelaH(h);
```

Listing 2.2 – Fluxo implícito de informação

Abordando a segurança da informação no navegador de forma holística, mecanismos experimentais têm sido propostos para implementar *controle do fluxo de informação* (IFC – *information flow control*) como estratégia de segurança não-discricionária e dinâmica.

2.1.3.2 An empirical study of privacy-violating information flows in JavaScript web applications (JANG et al., 2010)

Este artigo é um dos primeiros trabalhos relevantes sobre as vulnerabilidades expostas pela linguagem JavaScript e seu tratamento através de IFC. Os autores apresentam situações em que scripts maliciosos podem subverter o comportamento normal das aplicações e causar falhas de segurança da informação. É proposto um mecanismo de detecção e neutralização desse tipo de ataque.

Contribuição. A formalização das vulnerabilidades na linguagem JavaScript, a metodologia dos testes efetuados e a natureza da ferramenta descrita no artigo serviram como referenciais para a proposição de novas abordagens, algumas das quais são referenciadas nesta pesquisa. De forma presciente, os autores apontam o IFC como um caminho a ser seguido. Diversas iniciativas posteriores, algumas revisadas neste documento, seguem nessa direção.

2.1.3.3 Security of web mashups: A survey (DeRyck et al., 2012)

O artigo é motivado pelos requisitos de segurança de aplicações *mashup*. Os autores definem um conjunto de categorias de requisitos não funcionais de segurança e avaliam a conformidade desses requisitos versus funcionalidades do navegador. O critério de classificação estabelecido posiciona as diversas abordagens em quatro graduações que vão desde a separação total de componentes até sua integração completa.

Contribuição. O artigo contribui com a enumeração de requisitos que uma solução voltada à segurança da informação deve atender. Algumas das tecnologias mencionadas podem ter se tornado obsoletas ou de alcance limitado desde que o artigo foi escrito, o que não invalida o resultado pretendido pelos autores, que é considerado "estado da arte" (HEDIN et al., 2014) em pesquisa sobre segurança de aplicações de composição baseadas em Javascript.

2.1.3.4 Information-Flow Security for a Core of JavaScript (HEDIN; SABELFELD, 2012)

Este trabalho contém uma proposta conceitual para mitigação dos problemas de segurança da informação inerentes à implementação e execução da linguagem Ja-

vascript nos navegadores modernos. Apresentando casos de uso comuns, os autores empregam o conceito de *não-interferência* para introduzir um monitor de execução como sentinela de uso indevido de informação no sistema dinâmico de tipos em Javascript. O trabalho, puramente conceitual, alcança esse objetivo através da extensão de um subconjunto fundamental *(core)* da linguagem que, partindo da definição formal de Javascript[4], introduz anotações de código fonte que permitem a composição de programas à prova de vazamento de informação.

Contribuição. Este trabalho fornece uma prova da eficácia das abordagens baseadas no controle do fluxo de informações (IFC). Por se tratar de um exercício teórico, e propositalmente limitado a um subconjunto da linguagem, o conteúdo serve como introdução aos desafios e conceitos associados ao IFC. Por fim, a simplicidade e o rigor da solução apresentada também funcionam como exemplos a serem seguidos.

2.1.3.5 Toward Principled Browser Security (YANG et al., 2013)

Os autores analisam criticamente os mecanismos tradicionais SOP, CORS e CSP para expor suas heurísticas e políticas *ad-hoc* que, em troca de flexibilidade, abrem diversas vulnerabilidades de segurança da informação. Partindo dessa condição, e motivados pela robustez das abordagens de controle do fluxo de informações, os autores propõem um modelo baseado em IFC que, mesmo suportando todas as heurísticas existentes, é resistente aos algoritmos de ataque.

Contribuição. Este artigo enriquece o repertório a respeito de IFC aplicando essa abordagem ao escopo das funcionalidades além da execução de Javascript.

2.1.3.6 JSFlow: Tracking information flow in JavaScript and its APIs (HEDIN et al., 2014)

O trabalho, uma continuação de outro de mesma autoria(HEDIN; SABELFELD, 2012), é composto de duas partes: primeiro, os autores descrevem o panorama geral das pesquisas em segurança da informação em Javascript, detalhando as vulnerabilidades mais comuns e propondo como solução o controle do fluxo de informações; e em segundo, apresentam o projeto JSFlow[5], uma implementação da linguagem Javascript

[4] ECMA-262 – <http://www.ecma-international.org/publications/standards/Ecma-262.htm>
[5] JSFlow – <http://www.jsflow.net/>

com IFC puramente dinâmico integrado. Disponibilizado tanto através de extensão de navegador como ainda módulo *back-end* para o ambiente Node.js, e escrito na própria linguagem Javascript, JSFlow oferece segurança da informação de forma transparente e ubíqua, abrangendo a totalidade dos scripts executados no navegador – ainda que de modo experimental. O trabalho é concluído com um teste da eficácia do software.

Contribuição. O escopo do projeto JSFlow demonstra até que ponto é possível adotar uma abordagem puramente dinâmica para IFC. Fica evidente que tal abordagem abre oportunidade para a existência de "falsos positivos" durante a avaliação dos níveis de segurança associados a cada contexto de execução, um problema que os autores propõem mitigar através de uma abordagem estática complementar (ao que denominam "análise híbrida"). Outros trabalhos, ainda fora do escopo desta presente pesquisa, exploram essa alternativa.

2.1.3.7 Information Flow Control in WebKit's JavaScript Bytecode (BICHHAWAT et al., 2014)

Levando a análise do fluxo de informações a um patamar ainda mais profundo, este trabalho introduz um monitor de segurança integrado ao compilador de Javascript do mecanismo WebKit de navegação[6]. Os autores discorrem sobre os desafios de se implementar um monitor dinâmico operando sobre o *bytecode* gerado pelo compilador, enfatizando a dificuldade de tratamento de fluxos não-estruturados, porém válidos, na linguagem Javascript – especificamente, programas que fazem uso de instruções como `break`, `throw`, `continue`, `return` etc. Os autores demonstram como a análise estática é mais apropriada que a análise dinâmica para a avaliação de *bytecode*. Preocupações com o desempenho do monitor e seu *overhead* comparado às implementações padrão do WebKit são endereçadas com uma bateria de testes realizada através da suíte SunSpider[7].

Contribuição. O trabalho deixa evidente a complexidade e o esforço necessário para a implementação de um projeto dessa envergadura. Desconsiderando a natureza prototípica do artefato de software derivado do trabalho, os autores expõem com

[6] O projeto de código aberto WebKit serve como base para a construção de navegadores como o Safari (MacOS e iOS).

[7] SunSpider – <https://webkit.org/perf/sunspider/sunspider.html> (descontinuado; sucedido pela suíte JetStream, disponível em <http://browserbench.org/JetStream/>)

clareza o funcionamento do *bytecode* gerado a partir de Javascript sob o ponto de vista da segurança da informação, apontando, como (HEDIN et al., 2014), para a análise híbrida como o meio mais adequado para a avaliação de níveis de segurança em fluxos não-estruturados.

2.1.3.8 Protecting Users by Confining JavaScript with COWL (STEFAN et al., 2014)

O artigo argumenta que, face às dificuldades que os desenvolvedores encontram para aderir aos mecanismos tradicionais SOP, CSP e CORS, acaba-se optando pela funcionalidade em detrimento da segurança. Isto se manifesta em extensões de navegador solicitando mais permissões do que o necessário, em *mashups* que requerem autorizações desnecessárias para o usuário, e em notificações de segurança tão constantes que se tornam efetivamente invisíveis. Entendendo que o estado-da-arte da análise do fluxo de informações em navegador é deficiente – seja porque as ferramentas são incompletas ou porque degradam desempenho –, os autores apresentam o projeto COWL[8], implementanto o conceito de "controle de acesso mandatório"[9]onde os desenvolvedores definem quais informações são restritas e quem são os atores que podem acessar essas informações, relegando ao COWL o monitoramento da política de segurança. A esse estilo de IFC os autores denominam "granularidade ampla"[10] uma vez que a política de segurança se aplica a contextos de execução inteiros, em contraste com a "granularidade fina"[11] em que a aplicação da política recai sobre objetos específicos.

Contribuição. A separação das iniciativas de IFC por níveis de granularidade efetivamente recontextualiza o conceito de controle de fluxo de informação. Além disso, a iniciativa COWL é um projeto em andamento, documentado e em vias de se tornar uma API padrão pelo W3C[12].

---

[8] COWL: Confinement with Origin Web Labels – <http://cowl.ws/>
[9] Tradução livre para o termo *mandatory access control*.
[10] Idem para o termo *coarse-grained*.
[11] Idem para o termo *fine-grained*.
[12] <https://w3c.github.io/webappsec-cowl/>

2.1.3.9 Architectures for inlining security monitors in web applications (MAGAZINIUS et al., 2014)

Este trabalho avalia diferentes arquiteturas que podem ser aplicadas para efetuar checagem de segurança *inline* (incorporado ao código fonte previamente sua execução). Os autores listam quatro arquiteturas – extensões de navegador, proxies HTTP, proxies por prefixo e através de integradores. O artigo explora prós e contras de cada uma, considerando as garantias de segurança envolvidas. O trabalho é complementado pela implementação experimental das arquiteturas, empregando como monitor o JSFlow (HEDIN et al., 2014).

Contribuição. O artigo é inovador ao propor uma diversidade de arquiteturas e, consequentemente, de *stakeholders* para controle de fluxo de informações. Partindo disso, os autores concluem que existem diversas oportunidades para avanço e funcionalidades ainda vulneráveis a ataques por não se enquadrarem no foco das pesquisas nesta área, como scripts de origens heterodoxas (fora de elementos `<sctipt>`).

2.1.3.10 Information Flow Control for Event Handling and the DOM in Web Browsers (RAJANI et al., 2015)

Resumo. O trabalho explora vulnerabilidades no fluxo de informações em scripts acionados por eventos do navegador (tecnicamente, eventos do DOM). Tais vulnerabilidades são inerentes ao modo como os navegadores executam eventos e como isso se reflete, negativamente, nos monitores de IFC. Os autores partem da abordagem híbrida descrita em (BICHHAWAT et al., 2014) para criar um monitor à prova de vazamento de informação, com baixo *overhead*, e o colocam em comparação com iniciativas como (STEFAN et al., 2014) e (HEDIN et al., 2014).

Referências

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/>.

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/>.

HEDIN, D. et al. JSFlow: Tracking information flow in JavaScript and its APIs. **Proceedings of the 29th Annual ACM Symposium on Applied Computing**, p. 1663–1671, 2014.

HEDIN, D.; SABELFELD, A. Information-Flow Security for a Core of JavaScript. In: **2012 IEEE 25th Computer Security Foundations Symposium**. [S.l.]: IEEE, 2012. p. 3–18. ISBN 978-1-4673-1918-8.

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

____. IFC Inside: A General Approach to Retrofitting Languages with Dynamic Information Flow Control. **Post**, n. Section 2, 2015. Disponível em: <http://web.mit.edu/{~}ezyang/Public/IFCInside.p>.

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.

KINLAN, P. **Detecting injected content from third-parties on your site**. 2015. Disponível em: <https://paul.kinlan.me/detecting-injected-content/>.

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)>.

RAJANI, V. et al. Information Flow Control for Event Handling and the DOM in Web Browsers. **Proceedings of the Computer Security Foundations Workshop**, v. 2015-Septe, p. 366–379, 2015. ISSN 10636900.

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/>.

____. **Shadow DOM**. 2017. Disponível em: <https://www.w3.org/TR/shadow-dom/>.

____. **Web Application Security Working Group**. 2017. Disponível em: <https://www.w3.org/2011/webappsec/>.

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>.

YANG, E. Z. et al. Toward principled browser security. In: **Proceedings of the 14th USENIX Conference on Hot Topics in Operating Systems**. Berkeley, CA, USA: USENIX Association, 2013. (HotOS'13), p. 1–7. Disponível em: <http://dl.acm.org/citation.cfm?id=2490483.2490500>.