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-
18 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, a proposta deste trabalho é avaliar a viabilidade de uma abordagem que proporcione ao desenvolvedor uma barreira de proteção incorporada às aplicações, complementar aos recursos de segurança amplamente incorporados ao *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
DOM	Document Object Model
IFC	Information Flow 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., 2015a). 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. 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.

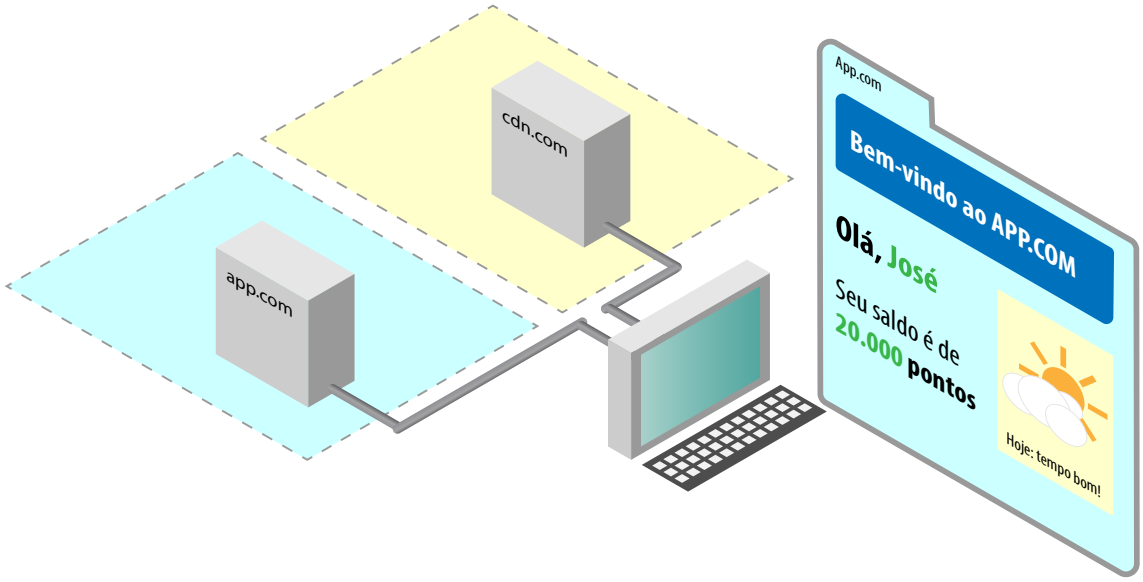

Figura 1 – Aplicação web composta por conteúdo proveniente de duas origens.

```
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)

(DIAGRAMA 2: COMPORTAMENTO DA APLICAÇÃO APÓS A PUBLICAÇÃO DO SCRIPT MALICIOSO)

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).

(DIAGRAMA 3: AÇÃO DE UMA EXTENSÃO COMPROMETIDA)

(Contextualizar: o que tem sido feito nessa direção, em termos de experimentos?)

(Contextualizar: como esta proposta contribui?)

1.2 Objetivo

O objetivo deste trabalho é propor uma abordagem para o confinamento de informação 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.

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/>.

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/>.

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.

____. 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.

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.

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/>.

____. **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>.