Humberto Borgas Bulhões

Metodologia da segurança da informação na linguagem Javascript nos ambientes de execução em navegadores

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
Fevereiro/2017

Humberto Borgas Bulhões
Metodologia da segurança da informação na linguagem Javascript nos ambientes de execução em navegadores/ Humberto Borgas Bulhões. – São Paulo, 2017-
16 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

1 Estado da arte

1.1 Introdução

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 (GOGUEN; MESEGUER, 1982) (DENNING, 1976) entre os sistemas de informação 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.

1.2 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., 2015), problemas para os quais os navegadores não oferecem ainda proteção total.

1.2.1 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.

1.2.2 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

1.3 Abordagens tradicionais para contenção de ataques

1.3.1 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.

1.3.2 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 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] "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

1.3.3 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).

1.4 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. 33). 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 confiabilidade 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.

Abordando a segurança da informação no navegador de forma holística, mecanismos experimentais têm sido propostos para implementar *controle do fluxo de infor-*

mação (IFC, na sigla em inglês) como estratégia de segurança não-discricionária e dinâmica.

1.4.1 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 controle de fluxo de informações como um caminho a ser seguido. Diversas iniciativas posteriores, algumas revisadas neste documento, seguem nessa direção.

1.4.2 Security of web mashups: A survey (De Ryck 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.

1.4.3 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 Javascript 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.

1.4.4 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.

1.4.5 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

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

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 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.

1.4.6 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

[5] JSFlow – <http://www.jsflow.net/>

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

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 prototipal do artefato de software derivado do trabalho, os autores expõem com 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.

1.4.7 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

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

uma API padrão pelo W3C[12].

1.4.8 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>`).

1.4.9 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).

[12] <https://w3c.github.io/webappsec-cowl/>

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.

De Ryck, 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.

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>.

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>.

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.

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.

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

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

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

____. **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>.

YANG, E. Z. et al. Toward Principled Browser Security. **Workshop on Hot Topics in Operating Systems**, p. 1–7, 2013.