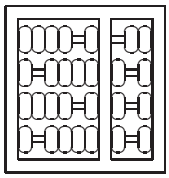

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Javascript na Web 2.0:
ameaças e prevenções no lado do cliente

Vagner Figuerêdo de Santana
Maria Cecília Calani Baranauskas
Marco Aurélio Amaral Henriques

Technical Report - IC-10-12 - Relatório Técnico

April - 2010 - Abril

Javascript na Web 2.0: ameaças e prevenções no lado do cliente

Vagner Figuerêdo de Santana [*] Maria Cecília Calani Baranauskas [†]
Marco Aurélio Amaral Henriques [‡]

Resumo

A evolução da Web 1.0 para a Web 2.0 trouxe possibilidades de compartilhar e distribuir informações com poder de alcance antes nunca visto. No entanto, este poder trouxe consigo possibilidades de que códigos maliciosos pudessem ser disseminados usando o mesmo potencial que o conteúdo possui na Web 2.0. Este trabalho aponta as ameaças mais comuns no desenvolvimento de aplicações Web 2.0 que usam scripts (e.g., Javascript). Complementarmente, são apresentadas questões relacionadas às ferramentas de avaliação como estudo de caso sobre como lidar com alguns pontos fracos da utilização de scripts no que diz respeito à segurança. Por fim, são propostas diretrizes a serem seguidas por desenvolvedores de websites de forma a evitar os problemas identificados, mas sem abrir mão do potencial de difusão de conteúdo e de colaboração entre usuários existente na Web 2.0.

1 Introdução

A Web pode ser definida como um sistema de documentos disponíveis na Internet, que podem conter fotos, vídeos e áudios e que podem ser buscados a partir de um assunto específico [Onl09]. Ela é composta por software, conjunto de protocolos e convenções, que através do uso de hipertexto e técnicas de multimídia, permite que pessoas naveguem, procurem e contribuam com o conteúdo nela disponível [W3C01]. A Web e suas tecnologias estão em constante evolução e isso se mostrou na transição da Web 1.0 para Web 2.0.

Na Web 1.0 o paradigma de um produtor de conteúdo para vários consumidores passivos que podiam acessar qualquer conteúdo disponível a qualquer momento já foi uma revolução quando comparado aos meios de comunicação em massa síncronos. O paradigma evoluiu devido a vários fatores, como tecnologia e o uso que as pessoas vinham fazendo das tecnologias usadas na Web. Segundo Castells [Cas99], as pessoas moldam a tecnologia para adaptá-las a suas necessidades.

[*]Instituto de Computação, Universidade Estadual de Campinas (UNICAMP), suporte financeiro da FAPESP, processo 2009/10186-9

[†]Instituto de Computação, Universidade Estadual de Campinas (UNICAMP)

[‡]Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas (UNICAMP)

No final dos anos 90 a Web mudou para um modelo de comunicação em que todos podem produzir, consumir e contribuir com diferentes tipos de mídia, a qualquer momento. Segundo Noureddine e Damodaran [ND08], a Web 2.0 possui uma arquitetura focada no usuário final. Dessa forma, o uso da Web mudou de uma forma significativa a ponto de levar à criação do termo Web 2.0.

Segundo Castells [Cas99], "o surgimento de um novo sistema eletrônico de comunicação caracterizado pelo seu alcance global, integração de todos os meios de comunicação e interatividade potencial está mudando e mudará para sempre nossa cultura". Pode-se dizer, entretanto, que a Web 2.0 é apenas um passo em direção a esse sistema.

Com relação à segurança, a Web 2.0 faz uso de diversas tecnologias que desenvolvedores tentaram evitar devido à preocupação com segurança e confiabilidade, o que está levando a Internet a se tornar a "terra dos sonhos"para hackers [ND08]. Enquanto a troca maciça de dados estáticos pode ser aceitável, quando se considera segurança, restrições devem ser impostas na troca de conteúdo executável [OWvOS08].

Na Web, o desenvolvimento server-side lida com a filtragem de requisições entre navegadores e clientes, usando autenticação e outros recursos para evitar execução de código malicioso no servidor. Neste caso, as variáveis relativas à segurança e compatibilidade são grandes, mas o desenvolvedor foca suas preocupações nas configurações do servidor e no formato das requisições. Ao considerar a Web pelo lado do cliente, a heterogeneidade de dispositivos e navegadores torna a tarefa de produzir código client-side compatível e seguro (i.e., que evite execução de código malicioso) um desafio à parte.

1.1 A tríade HTML, CSS e Javascript

Os documentos disponíveis na Web são comumente construídos usando três tecnologias:

- Linguagem de marcação para etiquetar os dados contidos em documento, por exemplo, HyperText Markup Language (HTML);
- Folha de estilo para controlar a aparência dos dados, por exemplo, Cascading Style Sheets (CSS);
- Scripts para adicionar comportamento dinâmico à página, por exemplo, Javascript.

Considerando esta tríade de tecnologias comumente utilizada em páginas Web, é possível identificar pontos de ataques no código HTML, CSS ou Javascript. Este trabalho tem como foco apresentar as ameaças mais comuns no desenvolvimento Web considerando scripts, neste caso, Javascript. No entanto, da mesma forma que desenvolvimento Web integra essas tecnologias, ataques também costumam integrá-las. O objetivo deste trabalho é auxiliar desenvolvedores Web a lidar com essas ameaças sem abrir mão das funcionalidades da Web 2.0.

1.2 Javascript e AJAX (Asynchronous Javascript And XML)

Segundo Yue e Wang [YW09], Javascript está presente em 96,9% das homepages mais populares do mundo. Segundo Crockford [Cro08], dado que Javascript é a linguagem usada

em navegadores Web e que navegadores se tornaram a aplicação receptora de conteúdo dominante, ela se tornou a linguagem de programação mais popular do mundo. No entanto, seu potencial foi subutilizado por muito tempo. Crockford [Cro08] comenta que houve diversos motivos para Javascript ter sido mal entendida e mal utilizada no início da Web, mas o AJAX (Asynchronous Javascript And XML) deu à linguagem Javascript uma segunda chance.

AJAX não é uma linguagem, mas uma maneira de usar padrões existentes [WAR08, W3S10]. AJAX se tornou popular em 2005 pelo Google (com Google Suggest) [W3S10]. É um acrônimo em inglês para Asynchronous Javascript And XML, que é uma técnica de transmissão assíncrona de dados em XML utilizando a linguagem Javascript. Os dados transmitidos podem seguir outros formatos (e.g., texto ou JSON), resultando em outros acrônimos (e.g., AJAT, AJAJ) [WAR08]. O objeto comumente utilizado para troca de informações via AJAX é o XMLHttpRequest [SB08a].

AJAX é um componente-chave de aplicações da Web 2.0 [O'R05]. Dessa forma, Javascript se tornou um elo importante em aplicações na Web 2.0, o que trouxe uma visibilidade para ataques relacionados às brechas existentes na utilização insegura de Javascript.

1.3 Widgets e Mashups

Na Web 2.0 está se tornando popular a utilização de widgets. Eles são componentes oferecidos por diversos websites que adicionam funcionalidades e conteúdos multimídia. Alguns desses widgets utilizam Javascript para enviar/receber conteúdo ou até mesmo executar scripts no lado do cliente. No entanto, há uma questão a ser considerada, pois ao incluir um script no corpo do documento principal de uma página, esse programa fica no mesmo escopo que qualquer outro script que esteja funcionando nesta página, possibilitando acesso às informações restritas (e.g., cookies), sobrescrita de código, acesso a variáveis, entre outras. Este tipo de acesso abre possibilidades para diversos tipos de ataque e serão apresentados na seção seguinte. Segundo Yue e Wang [YW09], incluir scripts de uma fonte externa no corpo do documento principal de uma página Web é perigoso, pois ele fica no mesmo escopo que outros scripts rodando na página e permite controle total sobre a página e sobre a janela do navegador.

Mashup é uma nova forma de construir aplicações [Cro06b]. Mashup pode ser definida como um website ou aplicação Web que combina conteúdo de mais de uma fonte de maneira harmoniosa, resultando em uma experiência integrada para os usuários [JW07]. Mashup pode combinar programas e serviços de dados de múltiplas fontes em uma página, usando Javascript para reunir componentes e usar conexões combinadas de maneira diferenciada [Cro06b]. Segundo Crockford [Cro06b], Mashups usam o navegador como se fosse uma plataforma Lego®, permitindo a construção de coisas novas e interessantes a partir de coisas existentes. Mashup também pode ser visto como website ou aplicação Web que integra widgets de diversas fontes.

Widgets são cada vez mais compartilhados entre websites e páginas Web de domínios diferentes, o que propicia compartilhamento de conteúdo e colaboração entre os usuários, mas traz questões delicadas em relação à segurança e à privacidade. Ponto em que, comumente, desenvolvedores são levados a escolher entre segurança e funcionalidade [JW07]. O

ideal é equilibrar a utilização das tecnologias com segurança, sem abrir mão do melhor dos mundos.

A Web 2.0 usa compartilhamento e distribuição de arquivos, sendo que em alguns casos esses arquivos podem ser scripts. Atacantes usam esta característica para distribuir código malicioso. Adicionalmente, tecnologias relacionadas à Web 2.0 possibilitam exploração de falhas não apenas na máquina local, mas também podem forçar navegadores a se conectarem com outras máquinas [ND08].

Este trabalho está organizado da seguinte forma: a seção seguinte apresenta pontos em que atacantes podem agir no contexto da Web 2.0 no nível de aplicação, alguns ataques mais comuns e técnicas existentes para lidar com essas questões; a seção 3 mostra como esses pontos são importantes, considerando um estudo de caso envolvendo ferramenta de avaliação de websites; por fim, são apresentadas medidas de prevenção para desenvolvedores Web que desejam lidar com as ameaças apresentadas, mas sem abrir mão dos potenciais de difusão de conteúdo e de colaboração existentes na Web 2.0.

2 Segurança na utilização de Javascript

Na Web 2.0 os alvos principais são os computadores cliente e servidor [ND08]. Atualmente é importante prevenir que usuários tenham seus computadores comprometidos via páginas Web maliciosas ou ataques que possuem como alvo a máquina do cliente [SAN07]. Alguns ataques possíveis no cliente são: Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) e ataques à privacidade [YW09].

Considerando a popularidade da linguagem Javascript, tem-se que 74,9% das homepages mais populares do mundo usam Javascript de maneira insegura. Essas práticas inseguras estão relacionadas à inserção insegura de Javascript e à geração insegura de Javascript [YW09]. Exemplo de inserção insegura é a prática de usar o atributo src da tag <script> para incluir (in)diretamente um arquivo Javascript de um domínio externo no corpo do documento principal de uma página Web [YW09]. Exemplo de geração insegura é utilizar dados recebidos de outras fontes/domínios para gerar código via funções document.write() ou eval().

Mais de 43% das homepages mais populares usam código Javascript de três ou mais domínios externos [YW09]. Segundo Oda et al. [OWvOS08], entre as 500 homepages mais populares do mundo a média é que cada uma inclui conteúdo de cinco ou mais domínios externos. Estes dados mostram o quão potencialmente alarmante é o cenário de aplicações que usam Javascript na Web 2.0 com relação à segurança.

Além disso, outra característica a ser considerada é a troca de informação entre cliente e servidor. Segundo Yue e Wang [YW09], 3,6% dos servidores de arquivos Javascript disponibilizam arquivos via HTTPS (HyperText Transfer Protocol Secure) e 5,8% das homepages mais populares do mundo usam arquivos Javascript disponibilizados via HTTPS.

2.1 Entrada de conteúdo

Um ponto sensível em relação à segurança em páginas Web refere-se à inserção de conteúdo. No lado do servidor é comum haver a preocupação com inserção de código para ataque a

bancos de dados (e.g., SQL Injection). No lado do cliente a preocupação também deve existir tanto quando usuários apenas inserem informações que não são armazenadas em banco (e.g., um termo de busca mostrado na página de resultados), como quando o conteúdo inserido via CMS (Content Management System) é armazenado em banco de dados.

Este ponto é significativo para Web 2.0, uma vez que a falta de filtragem cria vetores para diversos tipos de ataque. Uma forma de verificar se uma filtragem é feita adequadamente em formulários é inserir códigos e tags HTML e observar como a página mostra o termo buscado, ou seja, se as tags são filtradas, convertidas, interpretadas, etc. Se as tags inseridas forem interpretadas, uma porta para ataques está aberta e possibilita a inserção, por exemplo, da tag <script>, que delimita scripts em páginas Web.

Se um atacante conseguir inserir código Javascript malicioso, então ele pode facilmente iniciar ataques como XSS e CSRF [YW09]. Esses ataques podem ser usados para obter dados de acesso, rastrear comportamento de usuário, ataque de denial of service (DoS) e defacing [YW09].

2.2 Troca de conteúdo entre domínios diferentes (cross-domain)

Comunicação usando AJAX foi projetada para que a troca de dados ocorra somente entre a página Web vista no cliente e o servidor Web que a disponibilizou, ou seja, comunicação envolvendo o mesmo domínio. No entanto, conforme apresentado, em muitos casos é desejável contar com mashups que enviem/recebam informações para/de diferentes domínios (ou cross-domain).

A restrição que evita conexões usando XMLHttpRequest cross-domain é chamada Same Origin Policy (SOP). Ela previne que documentos ou scripts carregados em um local obtenham ou alterem propriedades de um documento de outra origem [Rud09].

Permitir que scritps acessem qualquer domínio a partir de uma página Web abre um canal potencial para ataques [Lev05]. Ao incluir um script de outro domínio, não há como validar e garantir que o script está usando de maneira adequada o acesso ao documento principal [JW07].

Parte do desafio em lidar com código malicioso em Javascript é que eles podem ser dinamicamente gerados, ou seja, a análise do código estático é dificultada, uma vez que é difícil determinar precisamente quais scripts serão gerados e executados [YW09]. Um exemplo apresentado por Phung et al. [PSC09] que ilustra esta questão:

```
var url = { toString: function(){
 this.toString() = function(){ return "bad" ; } ;
 return "good" ;
 }
};
```

Neste exemplo o código retorna valores diferentes para o mesmo método toString(), uma vez que o método se sobrescreve após ser executado.

Dado que mashups reúnem dados de diversas fontes, eles devem contornar de alguma forma o modelo de segurança SOP para obter dados de terceiros [JW07]. SOP não se aplica

às tags <script>, que podem ser baixadas de outros domínios e executadas com privilégio da página que a inclui; outros exemplos de tags cross-domain são: <style> e <img> [JW07].

Uma forma de lidar com essas restrições seria utilizar um proxy no lado do servidor, uma vez que linguagens de programação no lado do servidor permitem conexões cross-domain. Esta solução é eficaz, mas conta com dois pontos negativos: 1) ao considerar o contexto de Web 2.0 em que um widget é disponibilizado para vários websites diferentes, todos os administradores desses websites precisariam configurar o proxy disponibilizado pelo mantenedor do widget; 2) o servidor do website utilizando widgets contaria com uma sobrecarga significativa, uma vez que toda comunicação a ser enviada para o servidor do widget passaria antes por ele [SB08a].

A figura 1 mostra, em linhas gerais, diferentes formas de construir um mashup. Uma diferença significativa das duas abordagens está na assincronicidade, uma vez que um mashup construído no servidor requer que todos widgets sejam carregados e montados; já o mashup construído no cliente permite que o conteúdo seja carregado em partes e atualizado independentemente, o que reduz o tempo de resposta e consequentemente aumenta a usabilidade.

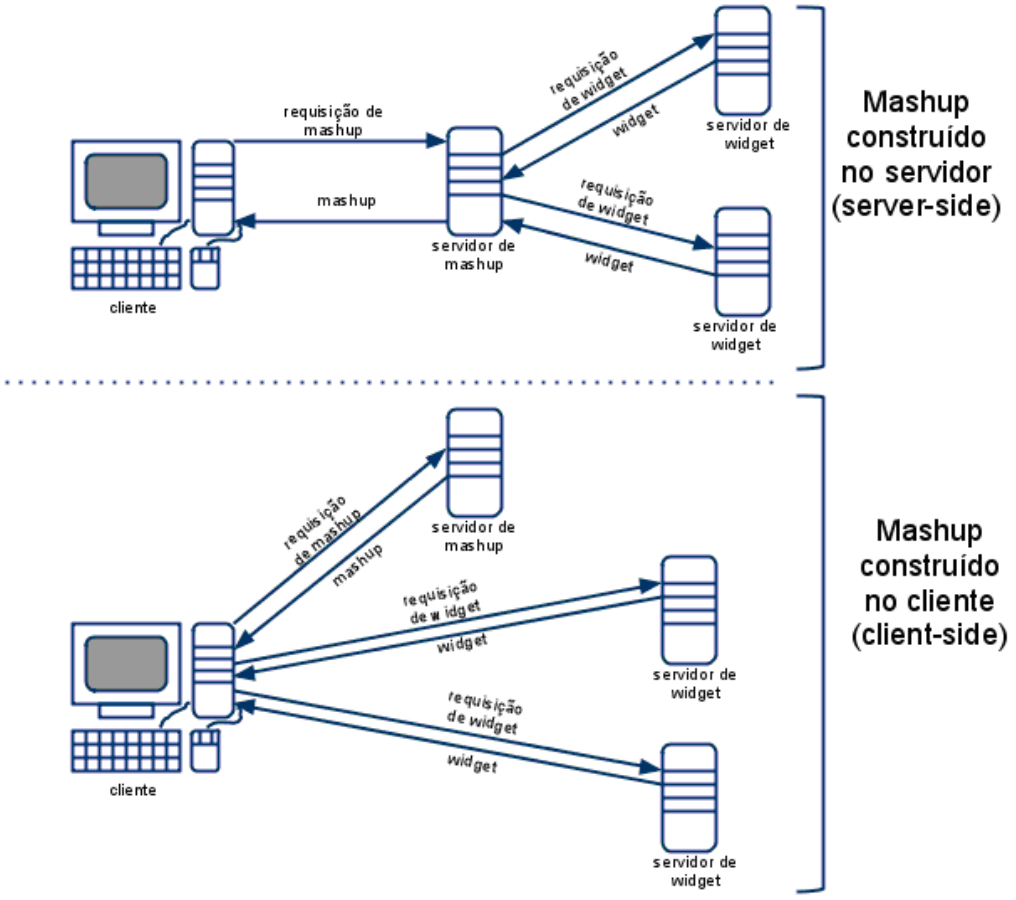

Figura 1: Tráfego de mensagens em mashups construídos no servidor e no cliente.

Uma forma simples de implementar troca de conteúdo cross-domain é através da utilização de uma técnica chamada de Iframe proxy [Doj06] ou Fragment Identifier Messaging [JW07]. A técnica se baseia no fato de que uma página que conta com um elemento <iframe> tem acesso ao objeto relacionado à URL do <iframe>, ou seja, o documento principal pode inserir informações, por exemplo, após o símbolo "#". Dessa forma, o documento carregado no <iframe> pode recuperar informações em sua URL e guiar seu funcionamento com base nelas. Esta solução é interessante e simples, mas requer cuidados no sincronismo e controle na troca de mensagens; dependendo da técnica utilizada no <iframe> para atualizar as mensagens (e.g., refresh) um som pode ser tocado pelo navegador, o que acaba reduzindo a usabilidade do mashup [SB08a].

Script assinado é uma iniciativa do projeto Mozilla e envolve a geração de uma assinatura digital associada a um determinado script para obter privilégios como acesso ao sistema de arquivos ou envio de requisições cross-domain usando XMLHttpRequest [SB08a]. Apesar de ser sugerido por Noureddine e Damodaran [ND08] como boa prática, o uso de assinatura tem um ponto desfavorável com peso significativo. Políticas de assinatura de scripts não são consolidadas por diversos navegadores e, portanto, em um contexto heterogêneo como a Web não se pode considerar como solução prática.

JSONRequest é uma proposta baseada na troca de informações usando a notação JSON (Javascript Object Notation), que é um subconjunto de Javascript utilizado para trocar dados estruturados. Assim, JSONRequest é uma proposta de Crockford [Cro06a] para permitir a troca de dados entre domínios diferentes, mas sem expor usuários às questões relacionadas à execução de Javascript, uma vez que trafegam somente dados em JSON, e evitam ataques CSRF, uma vez que não trafegam cookies e outras credenciais juntamente com requisições HTTP.

Outra proposta de Crockford é a tag <module>, que propõe dividir uma página em coleção de módulos [Cro06b]. A troca de informações entre os módulos e a página que os contém é feita via troca de dados usando JSON.

Podemos agrupar as técnicas relacionadas à utilização de <iframe>, Iframe proxy e a tag <module> sob o conceito de sandbox, em que um mashup é formado de diversos componentes com escopo controlado, mitigando diversos ataques relacionados à inserção insegura de código Javascript, uma vez que, segundo Jackson e Wang [JW07], sandboxed cross-domain frames não permitem comunicação com o documento principal. Sandbox é um conceito de execução originado em Java Applets que previne que o código Javascript de um determinado sistema afete o sistema operacional ou outras instâncias de navegadores, incluindo suas abas [OWvOS08].

Dadas as restrições, políticas e propostas de solução ainda não implementadas, uma das técnicas usadas é a Dynamic Script Tag. Ela parte do princípio de que o código contido em uma tag <script> é executado assim que o navegador a carrega. Dessa forma, a técnica envolve a criação dinâmica de tags <script>, o que permite a troca de conteúdo cross-domain. No entanto, ela pode ser caracterizada como uma forma de inserção insegura de Javascript e traz consigo potenciais vetores de ataques que serão apresentados nas próximas seções.

2.3 XSS (Cross-Site Scripting)

XSS pode ser definido como um tipo de ataque em que é possível executar um script dentro do documento principal de uma página que não se tem acesso. O script executado possuirá acesso a todas variáveis da página, assim como à árvore DOM, cookies, etc.

Tipicamente, ataques de XSS envolvem código Javascript que consegue enviar dados para um servidor Web malicioso [OWvOS08].

Buraco de XSS pode ser definido como o ponto de entrada para um ataque de XSS. Assim, se inserir o conteúdo <script>alert("Buraco de XSS")</script> em uma página Web e a página retornar uma janela com a mensagem entre aspas, é um sinal de que encontrou um Buraco de XSS. Dessa forma, é possível inserir código Javascript na página em questão, mudar sua aparência, alterar destino de formulários, registrar o que é feito na página, etc. Para uma lista com diversos exemplos de como explorar códigos para efetuar ataques de XSS, veja [RSn09].

Um exemplo de utilização insegura de Javascript que permite ataques de XSS é a utilização da função eval(), que executa o código passado por parâmetro. Assim, se o parâmetro não for corretamente filtrado é possível inserir código Javascript malicioso. A função eval() é usada em 44,4% das homepages mais populares [YW09].

Contra-exemplo de utilização da função eval: var response = eval(o.responseText) ;

Neste contra-exemplo o conteúdo da resposta está sendo executado e seu resultado armazenado na variável response. O problema é que se houver injeção de código malicioso, o script inserido estará no mesmo escopo que o código que usa a função eval e, dessa forma, ganha acesso às variáveis, cookies e etc.

2.4 CSRF (Cross Site Request Forgery)

CSRF (Cross-Site Request Forgery) é uma técnica usada por atacantes que usam vulnerabilidades existentes tanto do navegador quanto do website alvo do ataque [ND08]. Ela se baseia no fato de que em alguns casos navegadores incluem credenciais como cookie, IP, domínio, etc. Dessa forma, se uma aplicação autenticar um usuário apenas com essas credenciais presentes no lado do cliente, sem verificações no servidor, ela permite ataques deste tipo. Um atacante que obtiver acesso ao código da página pode usar credenciais no lado do cliente para efetuar requisições passando-se pelo usuário que está autenticado.

CSRF também ocorre quando um usuário visita uma página Web que se baseia na URL para executar determinadas ações e que permite que um atacante faça requisições usando os privilégios dos usuários para acessar determinadas URLs sem que o usuário perceba [OWvOS08].

Exemplo CSRF via atributo src de uma imagem:

```
var image = new Image();
image.src = "http://www.target.com/reportabuse/1234";
```

Neste exemplo a criação de uma imagem resulta em uma requisição usando as credenciais de um usuário autenticado. Assim, em uma rede social, por exemplo, um código inserido em diversas contas poderia prejudicar um usuário específico, sugerindo que várias pessoas

reportaram abuso do uso do sistema, quando na verdade a pessoa foi prejudicada por um ataque coordenado usando CSRF.

2.5 Propostas para lidar com ataques via Javascript

Além da utilização de tecnologias que mitigam ataques, há propostas de bibliotecas e ferramentas que auxiliam desenvolvedores a evitarem brechas de segurança, por exemplo:

- Subspace é uma primitiva de comunicação entre componentes de uma página e depende da manipulação e controle da propriedade document.domain com o aninhamento de tags <iframe> de forma que a página principal se comunica com um <iframe> mediador, que por sua vez se comunica com um <iframe> externo [JW07];

- SOMA (Same Origin Mutual Approval), uma política para controlar fluxo de informações prevenindo vulnerabilidades comuns como XSS e CSRF. A ideia é deixar SOP mais restritivo para que um mashup funcione somente se todos websites envolvidos permitirem explicitamente sua participação. SOMA pode prevenir diversas formas de vulnerabilidade em aplicações Web. No entanto, exige uma conexão HTTP a mais para cada requisição e a definição de arquivos de configuração para a política de troca de conteúdo entre domínios, no provedor de informações e no domínio que as utiliza. A proposta considera que tanto o website que vai usar um conteúdo quanto o website que fornece o conteúdo em questão devem aprovar a comunicação para que ela ocorra. Para tanto, os autores implementaram um add-on para o navegador Firefox [OWvOS08]. Apesar de interessante, na prática, esta proposta precisaria ser implementada nos navegadores ou ser usada via add-on. Adicionalmente, para obter todos seus benefícios, desenvolvedores que usam/mantêm mashups deveriam definir a lista de domínios confiáveis. SOMA não previne ataques relacionados à inserção de código como defacing a partir de um Buraco de XSS. No entanto, evita que dados vazem para fontes não confiáveis, uma vez que seu foco é evitar o acesso não autorizado. Por fim, dada a heterogeneidade da Web, pode-se considerar que esta solução deve ser somada às outras propostas que buscam eliminar os ataques comuns de vazamento de dados;

- NoScript é um add-on para o Firefox que tem um mecanismo de bloqueio de conteúdo executável ou interpretável (e.g., Javascript, Java) [Mao10]. Ele permite que o usuário indique de quais domínios é permitido rodar código, de maneira semelhante aos bloqueadores de popup. A proposta é interessante, no entanto, a responsabilidade de configuração cai sobre os usuários. Dado que homepages populares incluem em média conteúdo de cinco ou mais domínios externos [OWvOS08], na prática a solução pode ter impactos negativos na satisfação dos usuários. Por fim, é uma solução isolada para o navegador Firefox e não uma proposta definitiva para todos os navegadores.

- Lightweight Self-protecting Javascript, uma abordagem que busca controlar e modificar o comportamento do código Javascript para fazer com que ele se proteja. Ela se baseia no conceito de reference monitor, um método para especificar e implementar

sistemas usando um componente que intercepta requisições relevantes para questões de segurança e aplica políticas para decidir quais requisições devem ser atendidas. Em relação à implementação, ela usa o AOP (Aspect Oriented Programming) para definir os pontos de interceptação em relação às chamadas às funções monitoradas [PSC09]. Um ponto interessante da técnica é que mesmo que haja problemas na codificação e seja possível aplicar ataques de XSS, as políticas de segurança ainda podem ser aplicadas. No entanto, devido às atuais implementações da linguagem Javascript, a técnica é vulnerável a um ataque utilizando a função delete, em que as funções wrapper definidas podem ser apagadas e as funções do núcleo da linguagem ficam expostas novamente.

2.6 Privacidade

A privacidade é uma preocupação cada vez mais presente na utilização da Web. Muito se discute sobre os traços ou footprint digital que deixamos enquanto navegamos e sobre quem é o dono desses dados, que podem indicar preferências, perfis de consumo, entre outras coisas.

Um exemplo de como acessos simples a dados do cliente são potenciais para ataques é o acesso indireto ao histórico de navegação dos usuários via CSS (Cascading Style Sheets). Um e-mail contendo uma lista de links de webmails, tal que o atributo onclick de cada um remete para um proxy que grampeia a comunicação entre o cliente e o servidor de e-mail. Se a lista de links for mostrada para todos os usuários, o ataque seria pouco direcionado. No entanto, se a lista por default tivesse o estilo CSS para ficar invisível e a pseudo-classe :visited torná-los visíveis apenas para links visitados, tornaria o ataque muito mais direcionado, utilizando indiretamente uma informação sobre o histórico.

Um exemplo simples de ataque à privacidade com vazamento de informações é a obtenção de conteúdo gravado em cookie usando atributos de tags cross-domain para enviar dados para outro domínio, bastando para isso ter acesso ao script da página principal, como em:

```
var image = new Image() ;
image.src = "http://www.exemplo.com.br/log.php?cookie=" +
  encode(document.cookie) ;
```

Qualquer informação que possa ser lida do documento pode ser enviada de maneira similar, incluindo informações confidenciais como senhas, cartões de crédito, etc. [OWvOS08].

Conforme comentado, se um website conseguir ficar no meio da comunicação entre o usuário e o servidor, como um proxy, ele conseguirá grampear toda a comunicação, o que pode resultar em um problema de privacidade/segurança. Algumas ferramentas de avaliação usam essa técnica e serão comentadas na seção seguinte. Complementarmente, a transferência de dados de utilização representa outro ponto crítico em ferramentas de avaliação.

3 Estudo de caso: ferramentas de avaliação

Ferramentas de avaliação são pertinentes neste trabalho, pois ao utilizar módulos em Javascript para capturar informações sobre utilização, surgem questões em relação à privacidade e como essas ferramentas são alvos potenciais de ataques. Segundo Yue e Wang [YW09], este caso é especialmente atrativo quando websites fornecem trechos de Javascript que são utilizados em diversos websites.

Se um atacante conseguir obter acesso com privilégios para alterar um Javascript utilizado por diversos websites, ele pode coordenar um ataque de DoS, manipular dados estatísticos de audiência, capturar eventos relacionados ao teclado e obter senhas, entre outros. Em 2005, o Google Analytics tinha mais de 230 mil contas cadastradas [Rog05]. O que mostra que um ataque bem sucedido ao Google Analytics possibilitaria ataques coordenados de grandes proporções.

Apesar do cenário preocupante, este tipo de utilização de Javascript é inevitável para servidores de publicidade ou ferramentas de avaliação [OWvOS08]. É possível que, da mesma forma que bloqueadores de popup se tornaram parte do núcleo dos navegadores atuais, filtros de código baseados em domínios sejam incorporados em seus núcleos seguindo o exemplo do add-on NoScript do Firefox. Da mesma forma que antivírus verificam se um determinado website tem ou não alguma ameaça, poderiam manter listas de domínios com código Javascript malicioso (i.e., red list) e lista de domínio sem código malicioso e sem utilização insegura de Javascript (i.e., green list). Assim, usuários teriam uma configuração padrão com um nível razoável de segurança.

3.1 Exemplos de ferramentas

MouseTrack é uma ferramenta de avaliação de usabilidade que usa abordagem baseada em proxy para efetuar captura e análise automáticas. Ela fornece configuração e visualização on-line mostrando caminhos que o ponteiro do mouse percorre. A ferramenta processa as páginas Web solicitadas pelos clientes e as modifica inserindo o código Javascript responsável por capturar informações relacionadas aos movimentos do mouse. Então, quando um usuário clica em um link de uma página processada pela ferramenta, as coordenadas de movimento do ponteiro do mouse gravadas até o momento são enviadas para o proxy, que grava as informações e retorna a nova página Web requisitada pelo usuário [ASW06]. A ferramenta depende do suporte à linguagem Javascript e do acesso inicial a um link que passe primeiro pelo proxy.

UsaProxy é uma ferramenta de avaliação baseada em proxy que executa captura e análise de eventos ocorridos no lado do cliente. Ela utiliza a linguagem Javascript e tem foco em testes de usabilidade [AS07]. A ferramenta traz questões interessantes relacionadas ao incentivo de usuários a participarem de avaliações remotas de usabilidade quando nenhuma alteração é necessária por parte dos usuários. A ferramenta também depende do suporte à linguagem Javascript e do acesso inicial a um link que passe primeiro pelo proxy.

Google Analytics é uma ferramenta de captura e análise de logs. Os dados utilizados pela ferramenta representam pageviews. Seus pontos positivos são os diversos tipos de relatório disponíveis e a possibilidade de ser configurado para que determinadas ações sejam

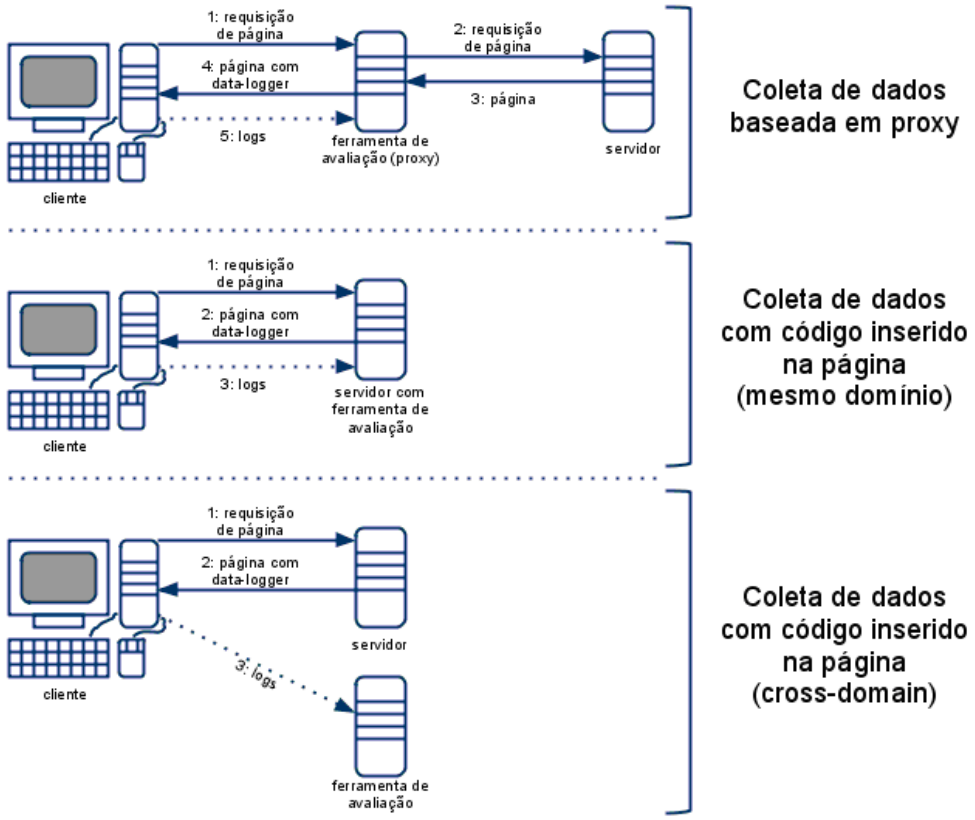

Figura 2: Diferentes formas de obter logs de utilização.

registradas como pageviews virtuais ou registrar eventos abstratos [Goo09]. A ferramenta requer um cadastro de um administrador do website e de que seja inserido o código Javascript nas páginas em que se deseja registrar pageviews e nos elementos em que se deseja capturar eventos abstratos.

WELFIT é uma ferramenta de avaliação que captura logs de eventos ocorridos no lado do cliente e analisa esses logs para apontar possíveis problemas de interface a partir de identificação de fluxos de utilização [San09]. A ferramenta segue os requisitos definidos para ferramentas de avaliação de websites definidos em [SB08b], levantados a partir da identificação de pontos fortes, fracos e lacunas das ferramentas de avaliação. Ela provê sumarização e representação dos logs de utilização possibilitando a identificação de padrões de utilização de maneira visual. A ferramenta requer um cadastro de um administrador do website e de que seja inserido o código nas páginas a serem avaliadas. Complementarmente, ela depende do suporte à tecnologia utilizada para desenvolver o módulo de captura, neste caso, Javascript, e da aceitação do usuário em participar da avaliação.

3.2 Avaliações e considerações

Ferramentas de avaliação de websites precisam capturar informações relacionadas ao uso. Para dados relacionados à utilização no lado do cliente normalmente é usado um data-logger automático. Este é o primeiro ponto relacionado à privacidade, uma vez que se deve considerar quais informações devem ser gravadas sem expor os usuários. Adicionalmente, é necessário lidar com a transferência desses dados, normalmente relacionada às questões cross-domain. A transferência deve ser segura e eficiente, sem influenciar na utilização. Mais requisitos para ferramentas de avaliação são apresentados por Santana e Baranauskas [SB08b].

Considerando as abordagens baseadas em proxy e em hard coded client-side data-logger é possível verificar que a utilização de um proxy sem o conhecimento ou consentimento do usuário pode caracterizar um ataque do tipo man-in-the-middle, em que a comunicação entre o a origem e o destino é grampeada (Figura 2). Um e-mail contendo links que remetam para o proxy antes de um serviço real possibilita que a comunicação seja grampeada sem o conhecimento dos usuários nem dos administradores desses serviços. Em contraste, a abordagem usando client-side data-logger possui alguns pontos positivos relacionados à segurança, são eles: o administrador do website em avaliação insere o código e está ciente da captura de dados, ou seja, aprova a captura; a ferramenta de avaliação normalmente mantém o cadastro dos usuários que a utilizam, ou seja, aprovam a utilização do data-logger; se o usuário aceita participar de uma avaliação, então nas sessões seguintes ele não necessita passar novamente pelo link que remete inicialmente para o proxy, o que torna a utilização mais próxima do real e, consequentemente, avaliações mais próximas do uso real do website em avaliação, eliminando possíveis tendências.

O Google Analytics grava informações assim que o usuário acessa uma página em avaliação, já o WELFIT apenas grava informações se o usuário aceita participar da avaliação. Esta diferença traz duas questões importantes: 1) no caso do Google Analytics, vários possíveis vieses em relação aos resultados da avaliação são eliminados, como a seleção de sujeitos; no entanto, muitos usuários não têm conhecimento da captura de informações; 2) No caso do WELFIT, a granularidade dos dados capturados é mais fina, o que traria um impacto maior em relação à privacidade; a forma de lidar com isso é obter informações somente sob o consentimento do usuário; no entanto, alguns vieses em relação à avaliação surgem, como seleção e taxa de resposta, uma vez que o grupo de pessoas que participa da avaliação não é randômico e apenas uma parte das sessões é gravada pela ferramenta.

4 Recomendações

Aplicações estão significativamente à frente da tecnologia de navegadores Web [Cro06b]. Adicionalmente, propostas de solução para os ataques apresentados estão longe de se tornarem consenso, pois dependem de sua implementação em todos os navegadores mais comuns [JW07]. No entanto, é possível identificar uma tendência em relação à mudança do uso de SOP, uma vez que a versão recente do Firefox 3.5 possibilita a utilização de XMLHttpRequest entre domínios diferentes.

Noureddine e Damodaran [ND08] sugerem uma boa prática para usuários: que eles de-

sabilitem suporte à linguagem Javascript nos navegadores para evitar a execução de códigos maliciosos. No entanto, é um preço muito caro e que os usuários não devem pagar. Dado que AJAX é um componente-chave de aplicações da Web 2.0 [O'R05] e que ele usa Javascript. Pode-se dizer que Javascript é uma das tecnologias que proporcionam a interatividade da Web 2.0. Assim, desabilitá-la seria como evitar o uso dos websites da Web 2.0 nos moldes que temos hoje.

A seguir são apresentadas as recomendações reunidas a partir da literatura relacionada à administração de código Javascript para evitar a inserção insegura de código. Com ela espera-se que desenvolvedores tenham um roteiro breve e objetivo para evitar os pontos fracos apresentados, mas sem abrir mão das funcionalidades da Web 2.0.

4.1 Evitar inclusão de scripts externos

Se possível, evitar inclusão de Javascript externo. Ao incluir código Javascript via funções document.write() ou via innerHTML, o código em questão é executado assim que passa por estas funções. Dessa forma, sua utilização para escrita de código Javascript deve ser evitada [YW09].

A filtragem de tags e funções é uma forma de lidar com código malicioso, mas a natureza do Javascript dificulta esse tipo de filtragem, uma vez que se pode atribuir funções a outras variáveis ou objetos.

4.2 Trocar dados em vez de conteúdo executável

Sempre que possível, trocar dados em vez de conteúdo executável. Utilizar XML, JSON ou até mesmo texto para trocar informações, mas nunca executar essas informações baixadas diretamente no navegador, principalmente quando houver qualquer ponto fraco em relação à segurança na transmissão de dados, por exemplo, não utilizar HTTPS para enviar/receber dados. Segundo Oda et al. [OWvOS08], restrições devem ser impostas na troca de conteúdo executável.

4.3 Utilizar conteúdo inserido pelos usuários com mínimo de privilégio possível

Executar aplicações no servidor com o mínimo de privilégio possível [ND08]. Com isso, se um código malicioso for executado, em último caso, ele causará o menor dano possível. Para usuários isso pode ser adaptado de forma que nunca se deve navegar autenticado em uma conta de administrador no sistema operacional.

4.4 Utilizar HTTPS ao baixar/enviar conteúdo de/para outros domínios

Javascripts externos deveriam ser servidos usando HTTPS [YW09]. Sempre que for possível e a utilização do HTTPS não trouxer impactos notórios para os usuários, deve-se transmitir dados sempre usando HTTPS.

Segundo Oda et al. [OWvOS08], SSL e TSL fornecem autenticação, integridade e autenticação, enquanto assinatura de código previne a execução de código não autorizado, entretanto, não protegem contra a execução de código malicioso dentro do navegador. Dessa forma, a transferência segura trata de um ponto, mas o cuidado com brechas para execução de código no lado do cliente deve ser outro ponto a ser considerado. Ou seja, mesmo usando certificados confiáveis que possibilitam a identificação da fonte, se houver um Buraco de XSS e um atacante conseguir inserir um código Javascript malicioso, ele também será parte da página confiável.

4.5 Isolar scripts externos em componentes com alcance limitado (wrapping)

Caso haja a necessidade de executar scripts, restringir permissões de scripts externos a partir da sua utilização em tags como <frame> ou <iframe> no qual a origem é diferente do documento principal da página [YW09] ou utilizar bibliotecas/técnicas que fechem o acesso de scripts à página, conforme propostas de soluções apresentadas. Dessa forma, é possível isolar componentes externos de forma que a comunicação seja controlada e evite ataques de XSS, CSRF ou à privacidade.

Obter este isolamento e possibilitar a comunicação entre a página Web e seus widgets é uma tarefa ainda em aberto, uma vez que, conforme apresentado por Phung et al. [PSC09], o funcionamento de linguagem Javascript dificulta a filtragem e utilização segura de políticas de segurança para código utilização de widgets contendo código executável.

5 Conclusão

A integração de várias mídias está mudando significativamente as formas de interação entre pessoas e sistemas computacionais. Desenvolvedores estão contornando políticas e restrições para fazer uso de determinadas funcionalidades. Dessa forma, é possível verificar que os navegadores carecem de um modelo que concilie segurança a essas funcionalidades.

Este trabalho apresentou e discutiu ameaças mais comuns à segurança, no desenvolvimento de aplicações Web 2.0 que usam scripts (e.g., Javascript). Recomendações reunidas a partir da literatura relacionada à administração de código Javascript para evitar a inserção insegura de código foram apresentadas. Ainda, um estudo de caso mostrou como questões relacionadas à segurança são, em alguns casos, requisitos para determinadas aplicações, especialmente ferramentas de avaliação. Foram apresentadas quatro abordagens que consideram dados de alta granularidade, ou seja, mais detalhados do que logs de servidores Web. As abordagens foram divididas em duas técnicas principais: proxy e client-side data-logger. Abordagens baseadas em proxy buscam facilitar o estudo de websites mesmo que avaliadores não sejam administradores dos websites avaliados, mas trazem questões significativas em relação à segurança e pode até ser caracterizada como um ataque do tipo man-in-the-middle, dependendo da sua utilização. Já a abordagem baseada em client-side data-logger inserido no código do website a ser avaliado alcança um nível de aprovação mútua, em que o administrador tem conhecimento da captura e a ferramenta de avaliação captura apenas dados dos websites cadastrados. Questões sobre privacidade dos usuários e segurança da

ferramenta de avaliação são pertinentes e devem ser cuidadosamente levantadas. Complementarmente, a definição de requisitos para esse tipo de serviço é um tópico a ser endereçado no projeto de sistemas Web com essas características.

Trabalhos futuros envolvem o teste de diferentes técnicas apresentadas em uma ferramenta de avaliação considerando ataques apresentados e questões de desempenho, igualmente importantes no contexto apresentado. O foco é combinar segurança/privacidade e o potencial da Web 2.0, sem influenciar no uso de um website sendo avaliado, ou seja, após o usuário aceitar participar da avaliação, a captura da ferramenta deve ser a mais transparente possível.

Por fim, outro trabalho possível é o teste de comunicação cross-domain em novas versões dos navegadores mais populares e verificar tendências em relação ao modelo de segurança no lado do cliente, ainda dissonante.

Referências

[AS07] Richard Atterer and Albrecht Schmidt. Tracking the interaction of users with ajax applications for usability testing. In Mary Beth Rosson and David J. Gilmore, editors, *Proceedings of the 2007 Conference on Human Factors in Computing Systems, CHI 2007, San Jose, California, USA, April 28 - May 3, 2007*, pages 1347–1350. ACM, 2007.

[ASW06] Ernesto Arroyo, Ted Selker, and Willy Wei. Usability tool for analysis of web designs using mouse tracks. In *Proceedings of ACM CHI 2006 Conference on Human Factors in Computing Systems*, volume 2 of *Work-in-progress*, pages 484–489, 2006.

[Cas99] M. Castells. *A Sociedade em Rede: Era da Informação: Economia, Sociedade e Cultura*, volume 1. Editora Paz na Terra, 4 edition, 1999.

[Cro06a] Douglas Crockford. Jsonrequest. `http://json.org/JSONRequest.html`, 2006.

[Cro06b] Douglas Crockford. The <module> tag. `http://www.json.org/module.html`, 2006.

[Cro08] Douglas Crockford. The world's most misunderstood programming language has become the world's most popular programming language. `http://javascript.crockford.com/popular.html`, 2008.

[Doj06] Dojo Toolkit. Cross domain XMLHttpRequest using an IFrame Proxy. `http://dojotoolkit.org/node/87`, 2006.

[Goo09] Google. Google analytics. `http://www.google.com/analytics`, 2009.

[JW07] Collin Jackson and Helen J. Wang. Subspace: secure cross-domain communication for web mashups. In Carey L. Williamson, Mary Ellen Zurko, Peter F.

Patel-Schneider, and Prashant J. Shenoy, editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 611–620. ACM, 2007.

[Lev05] Jason Levitt. Fixing AJAX: XMLHttpRequest considered harmful. `http://www.xml.com/pub/a/2005/11/09/fixing-ajax-xmlhttprequest-considered-harmful.html`, 2005.

[Mao10] Giorgio Maone. Add-ons for firefox – noscript. `https://addons.mozilla.org/pt-BR/firefox/addon/722`, 2010.

[ND08] Adam A. Noureddine and Meledath Damodaran. Security in web 2.0 application development. In Gabriele Kotsis, David Taniar, Eric Pardede, and Ismail Khalil Ibrahim, editors, *iiWAS'2008 - The Tenth International Conference on Information Integration and Web-based Applications Services, 24-26 November 2008, Linz, Austria*, pages 681–685. ACM, 2008.

[Onl09] Cambridge Dictionary Online. The web. `http://dictionary.cambridge.org/define.asp?key=92500&dict=CALD`, 2009.

[O'R05] Tim O'Reilly. What is web 2.0: Design patterns and business models for the next generation of software. `http://oreilly.com/web2/archive/what-is-web-20.html`, 2005.

[OWvOS08] Terri Oda, Glenn Wurster, Paul C. van Oorschot, and Anil Somayaji. SOMA: mutual approval for included content in web pages. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 89–98. ACM, 2008.

[PSC09] Phu H. Phung, David Sands, and Andrey Chudnov. Lightweight self-protecting javascript. In Wanqing Li, Willy Susilo, Udaya Kiran Tupakula, Reihaneh Safavi-Naini, and Vijay Varadharajan, editors, *Proceedings of the 2009 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2009, Sydney, Australia, March 10-12, 2009*, pages 47–60. ACM, 2009.

[Rog05] Garett Rogers. Google analytics stops at 234,725 accounts. `http://blogs.zdnet.com/Google/index.php?p=36`, 2005.

[RSn09] RSnake. Xss (cross site scripting) cheat sheetesp: for filter evasion. `http://ha.ckers.org/xss.html`, 2009.

[Rud09] Jesse Ruderman. The Same Origin Policy. `http://developer.mozilla.org/En/Same_origin_policy_for_JavaScript`, 2009.

[SAN07] SANS. Top 20 internet security problems, threats and risks. `http://www.sans.org/top20/2007/`, 2007.

[San09] Vagner Figuerêdo de Santana. Identificação de padrões de utilização da web mediada por tecnologias assistivas. Master's thesis, Instituto de Computação - UNICAMP, Abril 2009.

[SB08a] Vagner Figuerêdo de Santana and Maria Cecilia Calani Baranauskas. An asynchronous client-side event logger model. Technical report, Institute of Computing (UNICAMP), 2008.

[SB08b] Vagner Figuerêdo de Santana and Maria Cecilia Calani Baranauskas. A prospect of websites evaluation tools based on event logs. In Peter Forbrig, Fabio Paternò, and Annelise Mark Pejtersen, editors, *Human-Computer Interaction Symposium*, volume 272 of *IFIP International Federation for Information Processing*, pages 99–104. Springer Boston, 2008.

[W3C01] World Wide Web Consortium W3C. About the world wide web. `http://www.w3.org/WWW/`, 2001.

[W3S10] W3Schools Online Web Tutorials. AJAX tutorial. `http://w3schools.com/ajax/default.asp`, 2010.

[WAR08] Websites Atendendo a Requisitos de Acessibilidade e Usabilidade WARAU. Glossário. `http://warau.nied.unicamp.br/?q=glossary`, 2008.

[YW09] Chuan Yue and Haining Wang. Characterizing insecure javascript practices on the web. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 961–970. ACM, 2009.