# Cross-site scripting e Cross-site request forgery

## Introdução

A web é ainda um mundo em expansão, todos os dias novas aplicações aparecem na web que servem para os mais variados objectivos, temos desde as redes sociais ao *home banking* e não só cada vez há mais aplicações como também existem novos utilizadores. Com o aumento da utilização da web também começou o aumento do número de ataques a estas aplicações, podendo estes ataques ter repercussões tanto para o utilizador como para a aplicação. As aplicações na web funcionam na sua maioria por confiando nos seus utilizadores e os utilizadores confiando nas aplicações, um bom exemplo é o *facebook*, milhões de pessoas confiam informações pessoais. O que aconteceria se as informações que as pessoas confiam no facebook, e que muitas vezes são privadas, pudessem ser descobertas e partilhadas sem o consentimento do utilizador. Esta confiança numa aplicação pode-se revelar perigosa se a aplicação contiver falhas que permitam que alguém use a aplicação como forma de distribuir um código malicioso. Neste documento vai-se falar de dois dos ataques mais utilizados na web e os danos que estes podem causar se as aplicações forem vulneráveis, estes são:

* Cross-site Scripting
* Cross-site Request Forgery

Estes tipos de ataques têm-se tornado cada vez mais utilizados e estima-se que mais de metade das aplicações web esteja vulnerável a este tipo de ataques.

## Cross-site scripting (XSS)

### O que é?

XSS é um ataque que consiste em o atacante injectar código script (ex. JavaScript) na aplicação, sendo que este irá mais tarde ser executado quando for enviado para algum cliente. Isto permite que código de um atacante seja executado como se fosse código da aplicação, tendo os mesmos privilégios e até mesmo podendo ultrapassar algumas protecções, tais como “*same-origin policy*”.

### Quando acontece?

Este tipo de ataques acontece quando é dada a possibilidade ao utilizador de introduzir algum tipo de informação, por exemplo, num registo quando se introduz os dados ou num motor de pesquisa quando se introduz a *query* de pesquisa. Se a aplicação não sanitizar os dados inseridos é dada a possibilidade a um atacante de injectar código malicioso.

|  |  |  |
| --- | --- | --- |
| **Cliente 1**  Utilizador insere código malicioso.  Utilizador submete código malicioso.  **Cliente 2**  Novo utilizador consulta informação.  *Browser* do utilizador corre código malicioso que rouba dados privados do utilizador e envia para um serviço do atacante. |  | **Aplicação/Servidor Web**  Página da aplicação a pedir *input* do utilizador.  Servidor guarda informação.  Servidor envia a página com informação, incluindo código malicioso. |

Fig. 1 – Esquema ilustrativo das interacções num ataque XSS.

Existem dois tipos de injecção, a injecção “para cima” e a injecção “para baixo”. A primeira é a mais comum e acontece quando o atacante consegue apenas fechar o contexto actual e começar um novo com as suas *tags* de HTML, ou seja, o atacante poderá injectar código inserindo novas *tags* podendo mesmo mudar o visual da página ou apresentar informação que não é da aplicação.

A injecção “para baixo” acontece quando o atacante não muda o contexto, mas consegue no entanto adicionar um novo subcontexto. Imaginemos que o programador da aplicação, como medida preventiva, não permite qualquer tipo de símbolo ‘<’ ou ‘>’ que são os símbolos usados em HTML. O atacante poderá ainda injectar código que indirectamente poderá então mudar o aspecto da página, por exemplo recorrendo a um evento ou um *url,* que quando carregadoirá executar esse código.

### Tipos

Não existe um consenso na divisão dos ataques em diversos tipos mas estes são normalmente divididos em dois tipos e duas categorias. Os tipos são, os ataques persistentes e os não persistentes, sendo que as categorias são ataques *server-side* ou *client-side*.

Os ataques do **tipo não-persistentes** acontecem quando a aplicação apresenta algo passado pelo utilizador, sendo o mais comum encontrar-se na *query string*. Este tipo de ataques só funciona quando o atacante consegue que outro utilizador consulte a página exactamente com os dados que este preparou para o ataque, sendo comum que este partilhe a hiperligação para o website atacado através de *email* ou até mesmo em redes sociais.

O **tipo persistente** acontece quando o atacante consegue que a informação seja guardada pela aplicação, sendo que mais tarde irá ser enviada para outros utilizadores. Este tipo torna-se mais perigoso pois não dependem que o utilizador aceda com uns parâmetros pré-determinados, que este pode facilmente evitar, neste caso o utilizador poderá nunca mesmo vir a saber que foi atacado, até que seja tarde demais.

Ambos os tipos de ataque podem ser de ambas as categorias, depende da forma como a aplicação foi consultada e como esta irá consultar os dados para apresentar ao utilizador.

### Como proteger?

Existem várias formas de se proteger uma aplicação contra ataques de XSS sendo que aquela que deve ser a primeira a ser tomada é a sanitização do output, ou seja, transformar símbolos que têm um significado enquanto código em algo, que embora quando mostrado pareça igual, o interpretador do script não reconhecerá como código. Isto pode ser feito pois todas as linguagens de *scripting* têm uma forma de *escaping* em que é possível transformar um símbolo reservado em algo passível de ser usado como apenas texto.

Existe mesmo assim formas de ultrapassar o *escaping* e injectar código, isto acontece quando existem diferentes tipos de *enconding* (UTF-8 e UTF-7, por exemplo) entre o *browser* e o servidor, pois o servidor não especificou qual o *charset* a usar, fazendo com que a informação transmitida seja interpretada pelo browser como algo diferente do transmitido e podendo então permitir o ataque.

Existem, no entanto, outras formas de protecção, as quais serão enumeradas em seguida:

* Validação de HTML de fontes não confiáveis
  + Caso a aplicação permita aos utilizadores algum tipo de formatação do seu *input*, esta deverá garantir que o código não é um perigo para a aplicação, sendo que existem bibliotecas para este efeito.
* Desactivação de scripts
  + Esta é feita pelo *browser* do utilizador, o browser tem uma opção que permite ao utilizador desabilitar todos os scripts permitindo assim que este esteja imune a XSS em todas as aplicações. No entanto, muitas aplicações funcionam à base de *scripting* e sem este não irão funcionar correctamente.

Por último, como forma de diminuir as consequências de uma vulnerabilidade é comum que as aplicações protejam os *cookies*, principalmente os de sessão, associando o cookie a um IP. Caso o atacante copie o *cookie* para mais tarde usar e ficar autenticado sem saber as credenciais este não será válido, a menos que o atacante esteja, ou simule, sobre o mesmo IP que o atacado. Existe também a opção de marcar um *cookie* como sendo impossível acedê-lo através de *scripts*.

# Bibliografia

*Cross-site scripting - Wikipedia*. (s.d.). Obtido em 9 de 1 de 2012, de Wikipedia: http://en.wikipedia.org/wiki/Cross-site\_scripting

*Cross-site Scripting (XSS)* . (s.d.). Obtido em 9 de 1 de 2012, de OWASP: https://www.owasp.org/index.php/Cross-site\_Scripting\_(XSS)

*The Cross-Site Scripting (XSS) FAQ*. (s.d.). Obtido em 9 de 1 de 2012, de cgisecurity: http://www.cgisecurity.com/xss-faq.html

*XSS (Cross Site Scripting)*. (s.d.). Obtido em 9 de 1 de 2012, de ha.ckers: http://ha.ckers.org/xss.html