

Affaire suivie par :
CERTA

NOTE D'INFORMATION DU CERTA

Objet : Les attaques de type « cross-site request forgery »

Conditions d'utilisation de ce document : <http://www.certa.ssi.gouv.fr/certa/apropos.html>
Dernière version de ce document : <http://www.certa.ssi.gouv.fr/site/CERTA-2008-INF-003>

Gestion du document

Référence	CERTA-2008-INF-003
Titre	Les attaques de type « cross-site request forgery »
Date de la première version	17 décembre 2008
Date de la dernière version	–
Source(s)	–
Pièce(s) jointe(s)	Aucune

TAB. 1 – Gestion du document

Une gestion de version détaillée se trouve à la fin de ce document.

1 Introduction

L'objet de ce document est d'expliquer le principe des injections de requêtes illégitimes par rebond (*cross-site request forgery* ou CSRF), les risques qu'elles peuvent poser et les moyens de s'en protéger.

2 Le principe

Les attaques de type *cross-site request forgery* (ou CSRF) sont mal connues et trop souvent non prises en compte par les développeurs de sites Internet. Elles sont également souvent confondues avec les attaques de type injection de code indirecte (*cross-site scripting* ou XSS) alors que le principe est quelque peu différent (voir CERTA-2002-INF-001).

En français, un *cross-site request forgery* est une injection de requête(s) illégitime(s) par rebond. Concrètement, pour un attaquant, cela consiste à effectuer des opérations sur un site sans le consentement d'un utilisateur.

L'attaque consiste à provoquer l'envoi de requêtes (par exemple, GET ou POST) par une victime vers un site vulnérable (V), à son insu et en son nom (cf. figures 1 et 2). L'envoi des requêtes se fait lorsque la victime visite un site malveillant ou compromis (M), ou clique sur un lien. L'attaque cible toujours un ou plusieurs sites en particulier qui sont vulnérables. Le CSRF se fait toujours en aveugle, car l'attaquant provoque l'envoi d'une ou plusieurs requêtes sans obtenir de réponse.

On peut distinguer deux types d'attaques : celles qui profitent d'une authentification déjà établie sur le site (V) (*session riding* ou détournement de session) et celles qui vont forcer une authentification sur ce site.

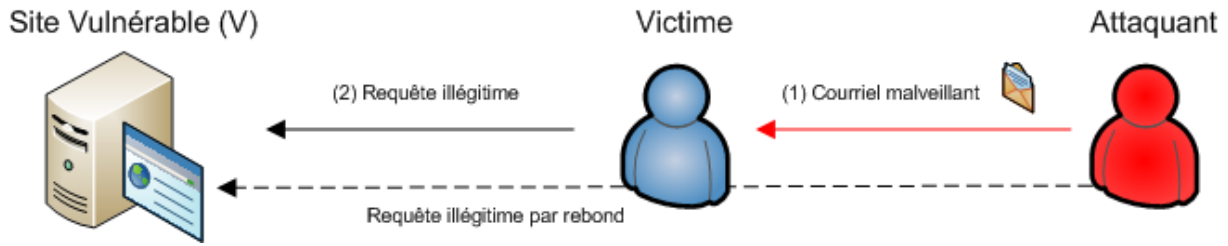


FIG. 1 – L'attaquant envoie un courriel qui provoque l'envoi d'une requête illégitime

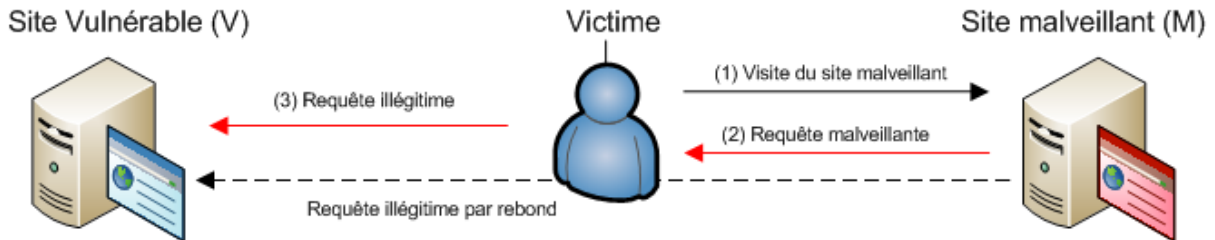


FIG. 2 – La visite d'un site malveillant provoque l'envoi d'une requête illégitime

2.1 Le détournement de session

Le détournement de session profite de l'authentification déjà établie par l'utilisateur sur le site (V).

Voici le schéma global d'une attaque (cf. figure 3) :

- un utilisateur est authentifié sur un site vulnérable (par exemple, le site de sa banque) (V) ;
- il est incité à visiter une page malveillante d'un site (M) contenant une requête vers le site initial (V) ;
- la requête hérite automatiquement des propriétés d'authentification de l'utilisateur (par exemple, le *cookie* d'authentification est envoyé) ;
- la requête effectue une opération en lieu et place de l'utilisateur (par exemple, virement d'argent vers un compte).

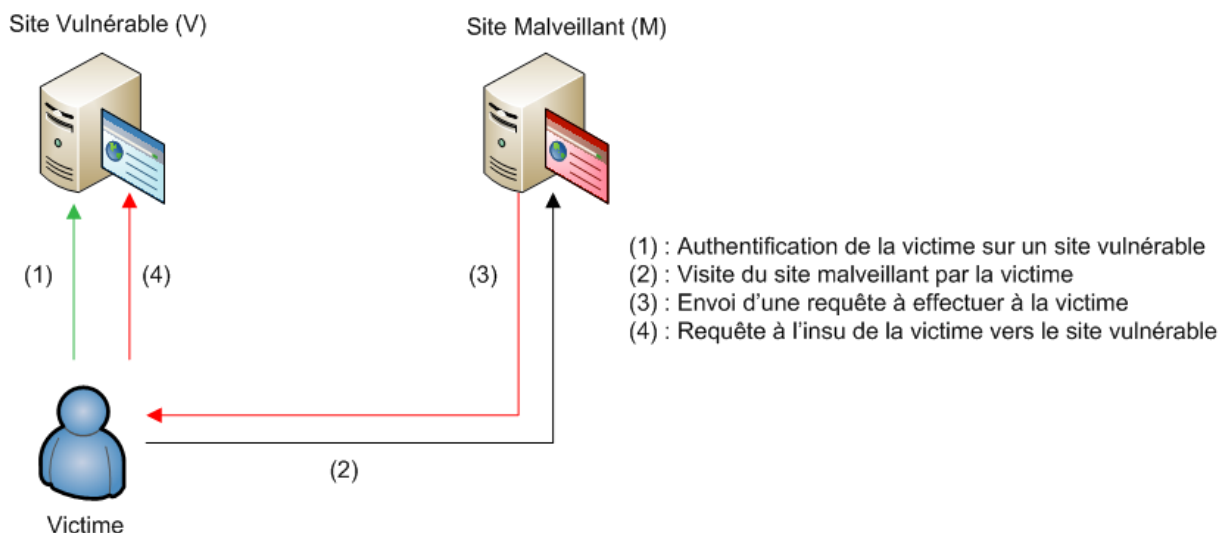


FIG. 3 – Principe d'un détournement de session

Il est important de noter qu'« authentifié » ne signifie pas nécessairement que la victime doit être en train de naviguer sur le site vulnérable. Par exemple, la personne peut choisir de sauvegarder un *cookie* de session valable plusieurs jours/mois/années et qui sera automatiquement envoyé par le navigateur (la date de péremption étant

fixée par les sites). Dans le cas d'une authentification par adresse IP, la requête est envoyée par la machine de la victime et sera donc toujours considérée comme valide.

Pour d'autres cas d'authentification (certificat, par exemple) ou si les *cookies* ou identifiants HTTP ne sont pas sauvegardés, l'attaque ne fonctionne que si la victime s'est préalablement authentifiée sur le site vulnérable dans la même fenêtre de navigateur ou dans le même processus¹ d'exécution.

Globalement, il faut comprendre qu'un site vulnérable au CSRF n'authentifie pas une personne mais son navigateur. En effet, il ne vérifie pas qu'une requête a été envoyée par l'utilisateur mais par son navigateur.

Un exemple simple d'attaque CSRF est une balise image comme suit sur un site malveillant :

```
<img src='http://sitevulnerable.tld/effectuer_action?action=changer_mdp&mdp=toto'>
```

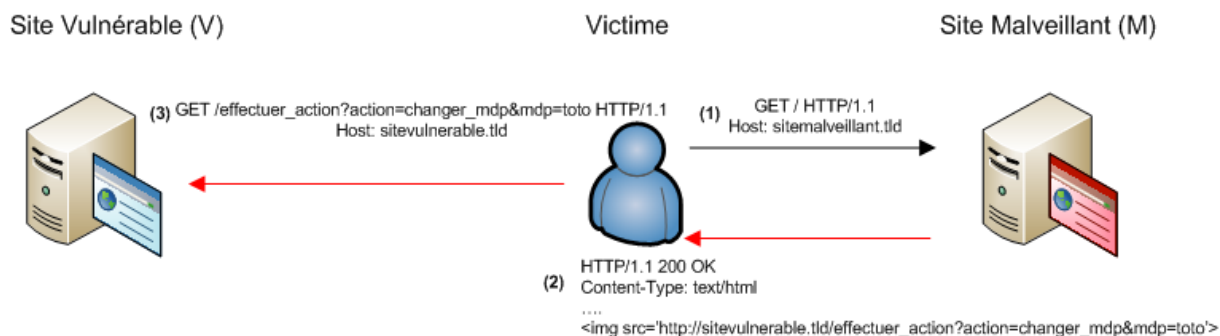


FIG. 4 – Exemple de CSRF

Dans cet exemple (cf. figure 4), une balise image sur un site malveillant contient en réalité une requête vers un site vulnérable changeant le mot de passe de l'utilisateur. Le navigateur ne vérifie pas si l'URL correspond à une image. Cet exemple est toutefois très simpliste. En effet, les attaques CSRF ne sont en aucun cas limitées aux requêtes de type GET et il est par exemple trivial avec du *javascript* d'envoyer automatiquement des requêtes POST vers d'autres sites.

Remarque : le principe de « same origin policy » qui est implémenté dans les navigateurs empêche un site tiers de lire ou manipuler du contenu provenant d'un autre site. Mais cela ne concerne pas l'inclusion de contenu ou l'envoi de requêtes.

2.2 CSRF d'authentification

Ce type d'attaque CSRF va forcer l'authentification d'un utilisateur sur le site vulnérable :

- soit, en utilisant un mot de passe par défaut qui n'a pas été changé pour ensuite effectuer des actions de détournement de session ;
- soit, pour que l'utilisateur effectue des actions à la place de l'attaquant voire d'une autre personne.

L'authentification elle-même est une attaque CSRF. La session peut ensuite être détournée au moyen d'autres requêtes.

Ainsi, le premier cas est équivalent au détournement de session vu en 2.1 sauf que l'on force l'authentification en plus.

Le deuxième cas est en revanche différent car l'utilisateur se connecte puis effectue ensuite des actions en tant qu'une autre personne. Il va ainsi, par exemple, sauvegarder le *cookie* de session d'un autre utilisateur sans s'en rendre compte.

3 Les risques

Le risque est inhérent au site visité par la victime puisque le CSRF consiste à effectuer des opérations sur un site sans le consentement de l'utilisateur et à son insu. Le site vulnérable peut très bien être celui d'une interface Web d'administration d'un matériel du réseau interne.

Voici trois exemples d'opérations qui étaient possibles par un attaquant sur des sites ou produits connus et vulnérables :

- opérations à l'insu de l'utilisateur sur un site bancaire (par exemple, virement d'argent vers un autre compte) ;

1. ce principe est différent selon le navigateur utilisé

- changement de la configuration du routeur WiFi de la victime (via l'interface d'administration web accessible en réseau local) ;
- changement de la configuration d'un *webmail* (notamment, l'ajout de filtres pour transmettre automatiquement les courriels reçus à une autre adresse).

Le risque principal est donc l'usurpation d'identité et l'exécution d'actions malveillantes sur un site.

Un CSRF d'authentification, pour lequel une victime se connecte pour le compte d'une autre personne, expose à un risque différent car c'est l'utilisateur qui usurpe une identité. L'intérêt ici pour l'attaquant est que la victime croit effectuer des requêtes en son nom alors qu'elle est connectée en tant qu'un autre utilisateur. Dans ce cas, le principal risque est donc la divulgation d'informations (si elle se connecte pour le compte de l'attaquant), mais d'autres scénarios peuvent être envisagés.

4 Les moyens de protection

4.1 Pour les développeurs

Le CSRF est une attaque contre les visiteurs d'un site et non contre le site lui-même. Les moyens de protection pour les développeurs servent donc à protéger leurs utilisateurs.

Le principe de protection est de s'assurer que l'application requiert une authentification qui n'est pas utilisée automatiquement par le navigateur. Cela consiste typiquement en l'ajout d'un élément aléatoire à la validation de formulaire de telle sorte qu'une requête ne sera considérée comme légitime que si elle contient cet élément aléatoire. En effet, les attaques CSRF ne sont possibles que si les requêtes sont prédictibles. Voici deux exemples :

- envoyer avec chaque formulaire une valeur aléatoire et limitée dans le temps. Le serveur doit vérifier dans une table d'états que cette valeur est bien renvoyée dans la requête de validation du formulaire ;
- créer un condensé (*hash*) avec une valeur secrète, une action définie (requête), l'identificateur de session et un horodatage, qui est envoyé avec chaque formulaire à l'utilisateur. Ce dernier doit renvoyer cette valeur lorsqu'il valide un formulaire, qui est recalculée par le serveur et vérifiée.

La première méthode nécessite plus de mémoire (table d'états) tandis que la deuxième nécessite du temps de calcul supplémentaire de la part du serveur. D'autres implémentations similaires sont envisageables. Un élément aléatoire unique par requête peut être préférable à un élément aléatoire unique par session (s'il est compromis, l'impact sera limité à la seule requête). Ces solutions sont à renforcer par l'utilisation d'un moyen de chiffrement tel que SSL pour empêcher toute écoute du trafic.

Une protection à base de *captchas*² peut également être envisagée. Toutefois cette méthode est plus contraignante pour les utilisateurs et doit se limiter à des requêtes « importantes ».

Il est important de noter que toutes ces protections ne fonctionneront pas si le site est vulnérable à du *cross-site scripting*. En effet, il est alors possible de contourner le principe de *same origin policy*, le site attaquant pouvant alors « accéder » aux données renvoyées par le site vulnérable (notamment les *cookies*, ou les *captchas*). Un exemple concret est le ver Samy sur MySpace qui a pu contourner des protections similaires à base de jetons sur les requêtes POST pour « ajouter des amis ».

De même que le *captcha*, une demande de confirmation à l'utilisateur par mot de passe peut être envisagée pour des requêtes critiques. L'avantage étant que cela n'est pas contournable en cas de *cross-site scripting*.

Une protection vérifiant le *referer* est aussi possible, toutefois elle n'est pas conseillée car ce champ peut être volontairement filtré pour des raisons de sécurité.

Une bonne pratique à adopter est d'obliger les utilisateurs à effectuer des requêtes de type POST lorsque cela est attendu par le serveur. Par exemple, il ne faut pas utiliser l'expression `$_REQUEST['variable']` en PHP (qui accepte GET et POST) mais `$_POST['variable']`. De même, il faut préférer `Request.Form['variable']` à `Request.Params['variable']` en ASP.NET. Cela ne protège pas du CSRF mais empêche des attaques utilisant des requêtes GET.

Enfin, certaines bibliothèques de développement peuvent également contenir des protections contre les CSRF. On peut citer par exemple, *CSRF Guard* disponible pour de multiples langages. Le langage ASP.NET permet également l'utilisation de la variable *ViewStateUserKey* avec les requêtes de type POST (cf. section Documentation).

2. test de Turing permettant de différencier un humain d'un ordinateur (par exemple : série d'images représentant des caractères seulement reconnaissables par l'oeil humain)

4.2 Pour les utilisateurs

Les utilisateurs peuvent se protéger en suivant des bonnes pratiques de navigation :

- ne pas naviguer en parallèle sur des sites sensibles et sur d'autres sites (notamment, faire attention aux onglets ou autres fenêtres de navigateurs) ;
- si possible, configurer son navigateur pour qu'il efface les données de navigation (*cookies*, sessions authentifiées...) à chaque fermeture ;
- ne jamais stocker de donnée d'authentification plus longtemps que la session ;
- toujours se déconnecter « proprement » d'une session authentifiée si cela est proposé par le site (en cliquant sur « se déconnecter », par exemple).

Le `Javascript` facilitant ou étant nécessaire pour une majorité d'attaques CSRF doit également être désactivé par défaut et réactivé ponctuellement si nécessaire.

De plus, les mots de passe et autres paramètres de configuration par défaut de toutes les applications et matériels utilisés doivent toujours être changés pour éviter les CSRF profitant de ces faiblesses.

Enfin, certaines extensions pour navigateurs ont des mécanismes de protection contre le CSRF. Il convient cependant de ne pas se reposer uniquement sur ce type de service.

5 Documentation

- A. Barth, C. Jackson, J.C. Mitchell, "Robust Defenses for Cross-Site Request Forgery" :
<http://crypto.stanford.edu/websec/csrf/csrf.pdf>
- W. Zeller, E.W. Felten, "Cross Site Reference Forgeries: Exploitation and Prevention" :
<http://www.freedom-to-tinker.com/sites/default/csrf.pdf>
- Watkins, P. "Cross-Site Request Forgeries"
<http://www.tux.org/peterw/csrf.txt>
- Vulnérabilité de type « Cross Site Scripting » :
<http://www.certa.ssi.gouv.fr/site/CERTA-2002-INF-001/index.html>
- CSRF Guard :
http://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project
- Protections intégrées à ASP.NET contre les attaques Web :
<http://msdn.microsoft.com/en-us/library/ms972969.aspx>

Gestion détaillée du document

17 décembre 2008 version initiale.