XSS Test Driver et les navigateurs web mobile

Erwan Abgrall, Yves Le Traon, Sylvain Gombault, Alain Ribault

XSS Test Driver et les navigateurs web sur mobile

Erwan ABGRALL[1], Yves LE TRAON[2], Sylvain GOMBAULT[3], and Alain RIBAULT[4]

[1] KEREVAL erwan.abgrall@kereval.com
[2] University of Luxembourg yves.letraon@uni.lu
[3] Telecom Bretagne sylvain.gombault@telecom-bretagne.eu
[4] KEREVAL alain.ribault@kereval.com

[5]

Résumé Abstract. Les attaques de type Cross Site-Scripting (XSS) constituent l'une des principales menaces pour les applications web. Ces attaques se propagent au travers des différents composants web : le site web, les composants de sécurité internes et externes, et sont in fine exécutées par le navigateur. Tester le degré d'exposition des applications web ou évaluer la nocivité d'un XSS n'est pas simple, dans la mesure où cela dépend beaucoup du navigateur et des vecteurs XSS utilisés pour l'exploiter. Pour rendre un diagnostic précis de l'impact d'un XSS donné, il convient d'identifier précisément la sensibilité de chaque composant à l'attaque, en s'assurant que l'environnement de test n'est pas perturbé les vecteurs XSS eux-mêmes. Dans cet article, nous proposons une méthodologie reposant sur un nouvel outil pour évaluer l'impact des XSS sur les navigateurs : les vecteurs d'attaques XSS sont présentés comme des cas de tests unitaires dont on observe l'effet sur les différents navigateurs web. Notre outil a été testé et validé sur un grand échantillon de navigateurs. Nous présentons ici les résultats de l'étude sur des navigateurs mobiles, elle démontre la pertinence de tester chaque version d'un navigateur avec notre outil, dans la mesure où une nouvelle version peut présenter des vulnérabilités XSS absentes de la version précédente.

1 Introduction

Le Cross-Site Scripting (XSS) est une catégorie d'attaques polymorphiques qui peuvent infecter les applications web comme leurs clients, de façon directe ou indirecte. Beaucoup de contre-mesures sont (ou devraient être) déployées pour contrer cette menace : ces mécanismes de sécurité peuvent être localisés au sein même de l'application (p.ex. des contrôles de validation), ou sur des composants de sécurité externes (reverse-proxies, web application firewalls (WAF) comme ModSecurity[1]), ou encore côté navigateur. Comme le montre la figure 1, un scenario typique d'attaque XSS se déroule en deux étapes principales : insérer un code XSS sur un serveur web, puis propager l'attaque sur les clients en faisant exécuter du code malicieux par leurs navigateurs. Dans ce papier nous allons nous intéresser à la seconde étape et particulièrement à l'exposition des navigateurs

aux attaques XSS. Une attaque XSS est composée d'un vecteur (pour pénétrer dans le système) et d'une *payload* qui constitue l'attaque effective. De part la nature dynamique des applications web actuelles, de part la grande variété des navigateurs et de leurs techniques d'interprétation HTML et JavaScript (JS), il est très complexe d'identifier si un vecteur *passant*(qui entre effectivement dans l'application web) représente une menace ou non pour l'utilisateur.

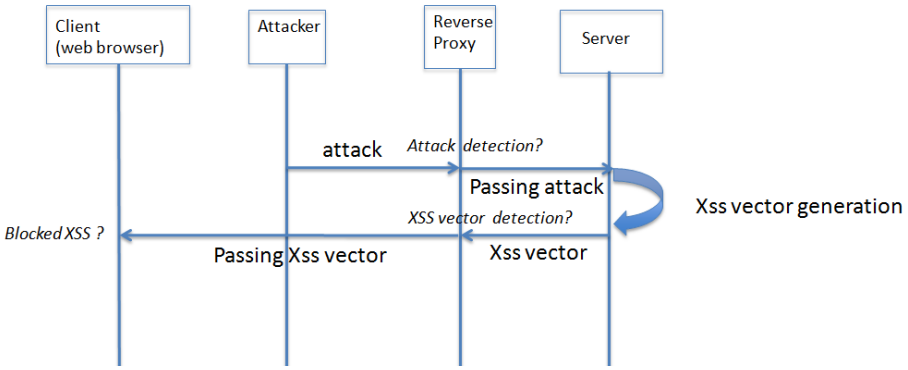

Figure 1. Déroulement d'une attaque XSS

La plupart des mécanismes de sécurité jouent leur rôle face aux attaques XSS basiques, mais peuvent échouer face à des attaques plus sophistiquées. Ces XSS évolués exploitent des comportements rarement connus issus d'interprétations spécieuses de l'HTML ou d'autres ressources web. Pour échapper à la détection d'attaque par recherche de signatures connues, un attaquant cherche à insérer des appels JavaScript (JS) dans des propriétés inattendues de certaines balises telles que :

```
<DIV STYLE="width:expression(eval(
String.fromCharCode(97,108,101,114,116,40,39,120,115,115,39,41,32)
));">
```

Pour comprendre cet exemple, il est important de noter que l'expression d'une propriété CSS (Cascading Style Sheet) exécute du code JS sur un navigateur Internet Explorer (IE). L'expression CSS appelle la fonction JS eval(), qui elle-même appelle la fonction de conversion d'une donnée ASCII en chaîne de caractères et génère la charge utile simple et non destructive suivante :

```
<script>alert(xss)</script>
```

Deux techniques sont principalement utilisées pour bloquer la propagation d'une attaque XSS d'un serveur vers le client : l'approche basée sur des signatures où l'on recherche le pattern du XSS et l'approche utilisée par SWAP [2] faisant

analyser les pages web par un navigateur pour en identifier les scripts s'exécutant et les éventuels XSS. La limite connue de la première approche est qu'elle ne peut bloquer les nouvelles attaques ou un nouveau mécanisme d'évasion (pas de signatures connues). SWAP n'est pas parfait non plus car il ne peut détecter les attaques spécifiques à un navigateur donné, comme l'attaque présenté en [3] ne pouvant être exécutée que par Internet Explorer (IE). En particulier, un moteur de détection XSS basé sur un analyseur HTML et JS donné, ne peut détecter un XSS spécifique à un autre navigateur, dans la mesure où il n'utilise probablement pas exactement le même analyseur HTML que le navigateur ciblé. En conséquence, il est nécessaire de d'étudier l'ensemble des navigateurs web et chacun de manière spécifique pour connaître leur degré d'exposition aux XSS.

Comme le montre notre exemple, une difficulté majeure pour protéger les clients web contre les attaques XSS est la sophistication technique de chaque XSS : des mécanismes HTML et JS peu usuels sont en effet déclenchés. Savoir prévoir quel mécanisme peut être exploité par une attaque n'est pas trivial, la démarche de test prend alors toute sa dimension pour estimer la sensibilité et l'exposition aux attaques XSS de façon systématique. Cependant le test doit répondre à trois exigences : la sélection des cas de tests pertinents, la définition de l'environnement d'exécution des tests et la mesure de la menace réelle portée par le vecteur XSS. La contribution de ce papier est une méthodologie, basée sur l'outil XSS Test Driver, pour tester de façon systématique l'impact d'un large jeu de vecteurs XSS sur les navigateurs web, y compris les clients sur téléphones ou tablettes mobiles.

La suite de cet article s'articule en trois parties. Le chapitre 2 présente les travaux relatifs aux tests XSS et les limitations des solutions existantes, puis décrit la nouvelle approche et la « bonne pratique »proposée. Le chapitre 3 décrit notre outil XSS Test Driver et montre qu'il s'agit d'un environnement sécurisé et isolé, permettant de tester un large jeu de vecteurs que la communauté des utilisateurs peut enrichir. Le chapitre 4 valide la pertinence de l'outil et de son approche en comparant le degré de nocivité des différents jeux de tests de XSS ainsi que le degré d'exposition des navigateurs mobiles, permettant ainsi de déterminer si un vecteur XSS passant au travers d'une application web représente une réelle menace pour les clients habituels de l'application web.

2 Tests de vecteurs de XSS et sécurité

Les tests de sécurité sont un vaste sujet, tant par le large spectre de domaines à prendre en compte qu'en fonction du type de propriété ciblée. Bien que les attaques XSS soient un sujet souvent abordé, la dimension test l'est, elle, beaucoup moins. L'analyse d'une application web conduit à considérer 3 couches pour le déploiement des contre-mesures : le navigateur web client, la sécurité côté serveur et les composants intermédiaires. A l'intérieur de chacun de ces niveaux, différents composants de sécurité peuvent être déployés pour examiner le trafic. La combinaison des protections sur chaque couche permet d'améliorer la sécurité globale : toutefois, chaque couche constitue un élément qui peut présenter à lui

seul des vulnérabilités face aux scenarii XSS. Chaque couche doit donc être testée indépendamment des autres. Le niveau de menace d'un scenario d'attaque XSS donné étant directement lié à la nature et à la distribution des navigateurs sur le site web client, il est très important d'évaluer chaque navigateur. Dans la suite de ce chapitre, nous détaillons la méthodologie proposée : elle repose sur l'utilisation de vecteurs XSS comme cas de test unitaire et a pour objectif d'étudier tous les navigateurs connus.

2.1 Études relatives

D'après nos recherches, aucune étude antérieure ne traite la façon de sélectionner et de comparer automatiquement un jeu de tests XSS. Toutefois plusieurs travaux, dont certains des auteurs eux-mêmes, proposent des méthodes et des outils pour tester automatiquement les politiques de sécurité (politiques de contrôle d'accès) [4],[5],[6],[7]. D'autres proposent des frameworks et des techniques pour tester le système à partir de ses interfaces [8],[9]. Offut propose une approche de test par bypass[4],[10] plus proche des techniques de tests XSS que nous présentons. La démarche décrite dans notre article en suit les grandes lignes, mais en apportant un éclairage spécifique sur la sélection des tests XSS et un comparatif systématique au travers de ces tests (et ce, sans évincer le navigateur des tests, dans la mesure où il s'agit d'une cible XSS à part entière). De façon similaire à la méthode proposée par Su's [11], Yao-Wen et al. [12] proposent de muter et d'injecter des entrées erronées, dont des injections SQL et XSS dans les applications web (outil WAVE), mais ils ne fournissent pas de techniques de diagnostic pour distinguer l'impact sur les différentes couches et valider la capacité d'un vecteur XSS à les traverser jusqu'au navigateur web.

La seule méthodologie d'évaluation de cas de test XSS que nous avons trouvée est basée sur du test de mutation [13] : un jeu de données de test est qualifié en mutant le code PHP de cinq applications web. Les attaques XSS sont alors utilisées pour tuer les mutants. Mais cette étude ne tient pas compte de l'impact du navigateur sur l'efficacité d'un vecteur XSS, introduisant de ce fait un biais dans l'expérimentation. Des sources similaires aux nôtres sont utilisées pour les vecteurs XSS, mais sans adaptation à un point d'injection spécifique. Cette pratique introduit un biais dans l'efficacité de l'attaque. Les attaques doivent en effet être ajustées à chaque point d'injection. Un même vecteur peut n'avoir aucun effet sur un point d'injection donné, et réussir sur un autre. La plupart des études XSS se concentrent, soit sur la détections d'attaques XSS [2], [3], [14], soit sur la recherche de vulnérabilités XSS [15], [**?**]. D'autres études se penchent sur les vulnérabilités ou les vers XSS [16], [17]. Enfin, une étude très complète de l'état de l'art sur les problématiques XSS et les parades est disponible en [18].

2.2 Méthodologie de test

Comme le montrent la figure 1 et notre premier exemple, un scénario d'attaque XSS se divise en deux étapes : l'attaquant adapte son attaque pour exploiter une vulnérabilité située sur un serveur web ou une application web pour

envoyer au navigateur un vecteur XSS avec une charge utile (payload). La charge utile contient en général du code JavaScript à exécuter par le navigateur. Elle peut être inoffensive, comme dans notre exemple, ou nocive, en redirigeant vers un site malveillant, pour exploiter une faille dans le navigateur, menant ainsi à l'exécution de code arbitraire sur le système client, comme pendant l'attaque Aurora visant les employés de Google [19]. La charge utile est exécutée si le navigateur comprend le vecteur XSS, le succès d'une attaque par XSS dépend donc aussi du navigateur utilisé par le mobile, téléphone ou tablette.

Comme nous voulons évaluer la capacité de n'importe quel navigateur à exécuter ou non la charge utile contenue dans un vecteur XSS, nous avons décidé de supprimer la première partie d'une attaque XSS complète, notre outil jouant le rôle de serveur web. Dans notre contexte, un cas de test est composé d'un vecteur d'attaque transportant une charge utile non destructive.

Comme présenté figure 2, un cas de test échoue (FAIL) si le navigateur n'exécute pas la charge utile, ou s'il attend indéfiniment, empêchant l'exécution du JavaScript et donc l'attaque. Un cas de test réussit (PASS) si le navigateur exécute la charge utile. Cela signifie qu'un cas de test qui réussit reflète une réelle menace pour le navigateur. PASS veux donc dire faille de sécurité (alors qu'en général dans le domaine du test PASS signifie qu'il n'y a pas de problème détecté).

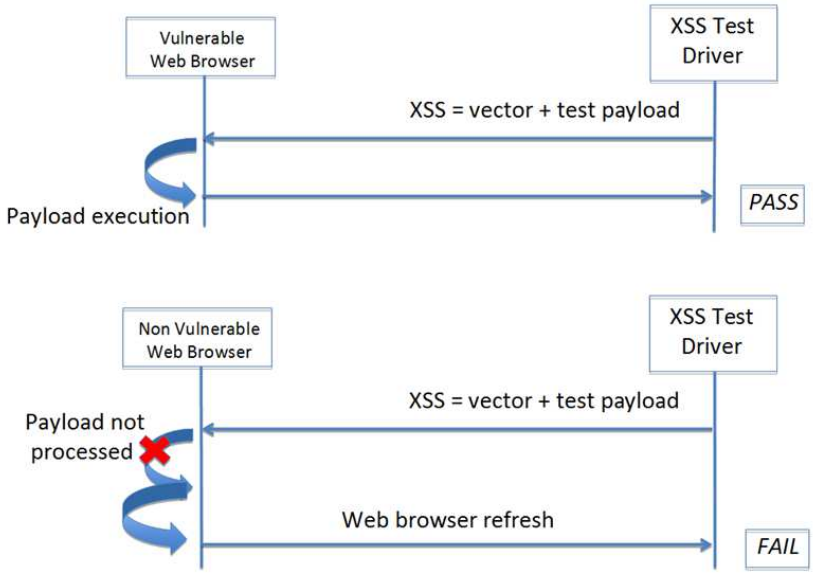

Figure 2. Logique d'un test unitaire de XSS Test Driver

Si l'on exécute les cas de tests sur un jeu prédéfini de navigateurs, le résultat est une liste contenant les vecteurs qui s'exécutent (tests qui réussissent) par navigateur. Comme XSS Test Driver agit comme un serveur web, les navigateurs doivent s'y connecter pour lancer la suite de tests. Les tests unitaires XSS ont été recueillis sur divers sites web de la communauté sécurité [20],[21],[22].

Ayant plus de 80 vecteurs d'attaques XSS, les principales difficultés rencontrées dans le développement de XSS Test Driver ont été :

1. l'adaptation de la charge utile à tous les navigateurs ;
2. l'adaptation de la charge utile à tous les vecteurs ;
3. l'automatisation des tests, afin d'exécuter tous les tests en une session pour chaque navigateur.

3 XSS Test Driver : un outil léger pour la sélection et l'exécution de vecteurs de XSS

Ce paragraphe présente dans le détail XSS Test Driver. Pour atteindre notre but, nous avons développé un environnement léger, mais non moins efficace, en python. Celui-ci fournit les vecteurs XSS aux navigateurs, et peut ainsi obtenir un verdict sur l'exécution du XSS. XSS Test Driver permet de mesurer l'exposition d'un navigateur à un ensemble donné de vecteurs XSS. Côté navigateur, l'exécution d'un XSS se fait en deux temps : le navigateur analyse le code HTML, permettant ainsi de reconnaitre les parties du Document Object Model et d'en construire une représentation interne. Il appelle ensuite le code JavaScript identifié (situé entre des balises ¡script¿ ou dans les propriétés de certaines balises) et l'exécute si nécessaire (ce n'est pas toujours le cas, par exemple les propriétés onevent comme onload ou onmouseover ne sont pas exécutées à chaque fois). Une démonstration de XSS Test Driver est disponible sur internet [23], vous pouvez y tester le navigateur de votre téléphone ou de votre tablette.

3.1 Exigences dans le déroulement des tests

L'objectif principal d'une attaque XSS est d'exécuter du JavaScript dans une page web, et l'une des difficultés est de s'assurer que les mécanismes sous-jacents déclenchés par les vecteurs XSS ne peuvent avoir un effet de bord sur ceux utilisés pour les tests ou le monitoring d'un système sous test.

Pour répondre aux problématiques de test standard et à l'objectif spécifique des tests de XSS, notre outil doit répondre à trois exigences. La première est que le JavaScript doit s'exécuter à l'intérieur du navigateur testé, ce qui semble évident, car le vecteur XSS a pour cible le navigateur, mais certains outils de test incorporent leur propre environnement d'exécution JavaScript pour les tests, comme RhinoUnit[24] par exemple. Un environnement de test basé uniquement sur un tel moteur néglige l'interprétation du HTML faite par le navigateur, et n'est pas capable d'analyser les vecteurs XSS basé sur HTML. Le deuxième prérequis est la capacité à exécuter une fonction de rappel qui sera déclenchée

à partir de la charge utile pour valider l'exécution du XSS par le navigateur : cela est nécessaire pour informer l'oracle du succès de l'attaque. Enfin, dernière exigence, l'outil doit avoir un contrôle total sur le DOM fourni au navigateur, car l'exécution du XSS dépend de la manière dont le HTML est analysé et interprété. Cette dernière exigence est atteinte par la conception des cas de tests qui spécifient le vecteur XSS qui est fourni au navigateur.

3.2 Logique de test

Pour éviter l'utilisation de bibliothèques JavaScript, ou toute interaction avec le DOM, nous avons utilisé la logique suivante pour enchaîner les tests et récupérer les résultats :

1. chaque attaque XSS est servie par une URL différente, avec une charge utile indépendante de toute bibliothèque JavaScript ;
2. la charge utile d'une attaque XSS contient une routine de validation JavaScript et un mécanisme de redirection pour passer au test suivant ;
3. quand le mécanisme de validation est déclenché, le test est marqué comme réussi ;
4. quand l'URL d'un test est rafraîchie, le serveur vérifie si l'une des routines de validation a été exécutée, sinon le test est marqué comme échoué ;
5. dans les deux cas (réussi/échoué), un message 302 Redirect est envoyé au navigateur pour le rediriger vers le test suivant.

3.3 Format de test et charge utile

Les cas de test sont fournis sous forme de tuples python, constitués du vecteur avec le format de la charge utile en paramètre et de sa description :

Nous utilisons deux types de charges utiles : celles que nous avons déjà décrites et une générique contenant `alert('xss')`. Cette dernière est envoyée pour pouvoir confirmer manuellement l'exécution d'un vecteur.

Plusieurs formats de charge utile sont disponibles pour couvrir les besoins de vecteurs spécifiques : quelques attaques XSS nécessitent de présenter le JavaScript dans un fichier spécifique pour tromper certains navigateurs (IE6 dans ce cas) :

```
<LINK REL="stylesheet" HREF="http://ha.ckers.org/xss.css">
```

Les formats supportés sont :

- payload : code JavaScript à exécuter
- jscript : adresse pour charger un .js contenant la charge utile
- eval_payload : charge utile XSS fournie sous la forme d'une fonction `eval(String.[fromCharCode'](XX,X`
- scriptlet : XSS dans une petite page HTML
- css : XSS dans un css
- htc : XSS dans un fichier .htc
- jpg : code JavaScript dans un `.jpg` servi comme un fichier jpeg

3.4 Automatisation des tests

Les cas de tests peuvent s'enchaîner automatiquement : une balise

```
<meta http-equiv="refresh" content="5" />
```

est ajoutée au vecteur pour rafraîchir la page. L'enchaînement des tests peut néanmoins s'interrompre si ce mécanisme entre en conflit avec quelques rares vecteurs XSS, comme par exemple :

```
<META HTTP-EQUIV="refresh" CONTENT="0;url=javascript:alert('XSS');">
```

Quand cette attaque XSS échoue, le navigateur peut être bloqué avec une adresse invalide, nécessitant alors une intervention humaine : naviguer vers une URL de reprise des tests. En fait, le W3C déconseillant l'usage de la méthode meta-refresh, le seul mécanisme permettant d'enchainer les tests n'est donc pas compatible avec tous les navigateurs [25] (alors que la navigation vers une URL de reprise l'est).

3.5 Identification des navigateurs et collecte des résultats

En tant que serveur web, notre outil peut identifier chaque navigateur connecté via son champ user-agent : Cette information permet à un serveur web d'adapter son comportement à chacun de ses clients ; elle va permettre à notre outil de reconnaître chaque type de navigateur et ses multiples versions.

4 Expérimentations et détermination de l'exposition des navigateurs webs face à un ensemble de vecteurs de XSS

Nous avons choisi d'illustrer les résultats bruts de notre outil avec deux indicateurs pour simplifier l'analyse des résultats. Il s'agit du niveau de nocivité de chaque cas de tests XSS à l'encontre d'un ensemble de navigateurs. En complément nous calculons le niveau d'exposition d'un navigateur à l'ensemble des cas de tests.

4.1 Définitions des indicateurs

Soit *Ts* une suite de tests. Soit *Verdict(Tc, Wb)* le résultat de l'exécution du cas de test *Tc* provenant de *Ts*, contre le navigateur web *Wb*. *Verdict(Ts, Wb)* retourne soit *PASS* (le XSS a fonctionné) ou *FAIL*.

Le *niveau de nocivité Nox(Ts, Wb)* de la suite de tests *Ts*, contre un ensemble de navigateurs web *WB*, est défini par le pourcentage de tests retournant *PASS* sur un ensemble de navigateurs web testé.

$$Nox(Ts, Ws) = |Verdict(Tc, Wb) = Pass, Wb \in Wb|/|Wb| \qquad (1)$$

Table 1. User Agents Identification

Browser	User Agent String
Chrome 11.0.696.68	`Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.24 (KHTML, like Gecko) Chrome/11.0.696.68 Safari/534.24`
Firefox 7	`Mozilla/5.0 (X11; Linux i686; rv:7.0.1) Gecko/20100101 Firefox/7.0.1`
Safari Mac OS X Leopard	`Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_8; fr-fr) AppleWebKit/533.21.1 (KHTML, like Gecko) Version/5.0.5 Safari/533.21.1`
IE 8.0.6001.19048	`Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; WOW64; Trident/4.0; SLCC1; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET4.0C; .NET4.0E; .NET CLR 3.0.30729)`

La surface d'attaque *ThExp(Wb, Ts)* d'un navigateur web à la suite de tests XSS Ts est le pourcentage de résultats *PASS* lorsque nous l'utilisons avec l'ensemble des éléments de Ts :

$$ThExp(Wb, Ts) = |verdict(Tc, Wb) = Pass, Tc \in Ts|/|Ts| \quad (2)$$

Ces indicateurs sont utilisés pour simplifier l'analyse des résultats et déterminer quels vecteurs sont les plus efficaces pour une catégorie de navigateurs web, et d'identifier rapidement parmi ceux étant les plus exposés. Néanmoins, ces estimations ne remplacent pas une analyse plus précise des attaques affectant un navigateur web précis. Par exemple, nos résultats montrent que certaines attaques ne réussissent que sur certains navigateurs, qui ne sont pas nécessairement les plus exposées.

4.2 Conditions d'expérimentation

Classificaton des vecteurs d'attaque Les sources de vecteurs d'attaque sont variées. Beaucoup proviennent de l'étude des normes et des comportements de navigateurs, mais une partie d'entre elles est directement issue d'éléments non documentés, tels que des comportements particuliers comme l'interprétation

spécifique de l'échappement d'une chaine de caractère, etc Donc si nous utilisons comme référence une norme, nous pouvons classer les vecteurs par rapport à celle-ci, permettant ainsi la génération de vecteurs XSS basés sur des modèles. Les autres vecteurs ne respectant pas la norme, tombent finalement dans la catégorie des bugs et défauts applicatifs.

Génération de nouveaux vecteurs d'attaques XSS Afin de découvrir de nouveaux vecteurs, nous avons combiné des ensembles de balises et propriétés d'HTML4 avec des appels JavaScript, et utilisé le produit scalaire de ces ensembles (balise, propriété, appel) en vue de générer des vecteurs XSS. Avec cette approche, sur les 44000 cas de tests générés, nous n'avons trouvé que quelques vecteurs fonctionnels, ainsi que des variations de quelques vecteurs déjà connus. Les interdépendances entre les balises et les propriétés ne sont pas prises en compte dans cette génération, et quelques contraintes se doivent d'être respectées pour obtenir un vecteur valide, certaines propriétés étant simplement ignorées par les analyseurs syntaxiques de par leur contexte d'utilisation (par exemple l'utilisation de balises svg ou HTML5 sans le bon type mime spécifié dans le content-type).

Avec une modélisation correcte de ces contraintes, il serait possible de générer tous les vecteurs d'attaques pour une norme donnée, et de s'en servir pour évaluer les techniques de détections de XSS. Il serait également possible d'utiliser ce modèle pour détecter des attaques, ou pour améliorer les IDS actuels. Ces travaux feront l'objet d'un autre article.

4.3 Choix des cas de tests

Du point de vue d'un navigateur, la sécurité de l'ensemble des composants du navigateur importe, alors que d'un point de vue XSS, seul l'analyseur syntaxique du code HTML importe : ainsi Safari, Chrome et les autres navigateurs bases sur webkit ont globalement le même comportement avec nos cas de tests. Les navigateurs testés furent : Internet Explorer, Netscape, Mozilla, Firefox, Opera, Safari and Chrome, dans des versions publiés entre juillet 1998 et mars 2011.

Les vecteurs XSS furent construit à partir du « XSS Cheat Sheet »[20], du « HTML5 Security Cheat Sheet »[21] and « UTF-7 XSS Cheat sheet » [26], et quelques vecteurs découverts en générant les triplettes balise/propriété/appel JavaScript qui firent leur apparition plus tard dans [21].

Les cas de tests XSS représentent un grand nombre de vecteurs d'attaques. Nous les avons adaptés pour avoir une charge utile dédiée à l'interprétation des résultats. Dans le but de reproduire ces essais, tous les cas de tests sont disponibles en ligne. Une nouvelle version de l'outil est disponible sur `http://xss.labosecu.rennes.telecom-bretagne.eu` et est en cours d'amélioration avec pour objectif de simplifier le traitement des résultats.

Certains vecteurs ont réussi sur la majorité des navigateurs, alors que d'autres ne fonctionnent que sur des versions bien précises. Ceci est dû à la qualité de l'implémentation des normes (et de l'analyseur syntaxique sous-jacent). Par exemple, il y a eu un effort important de respect de la norme entre IE6 et IE7.

Nous pouvons aisément imaginer une charge utile ne visant qu'une version spécifique d'un navigateur, l'identification du navigateur se faisant par l'interprétation (ou non) du vecteur d'attaques XSS. Le comportement de l'analyseur syntaxique devient alors une empreinte, de la même manière que l'empreinte d'une pile réseau peut être obtenue et servir de préliminaire à une attaque

4.4 Étude des résultats de tests

Les tableaux suivants (Table 2) listent les résultats de 84 vecteurs XSS de test pour 3 familles de navigateurs. La table de droite est la suite de la table de gauche. Dans la table de gauche, nous présentons les résultats des cas de tests 3 à 45, la seconde table présentant quant à elle les résultats des tests 45 à 87 (le résultat 45 étant répété pour des raisons de présentation). En ordonnées nous trouvons les 3 familles de navigateurs, et pour chacun d'eux nous affichons en première ligne le niveau d'exposition à la menace (ex : 33 pour IE8 signifiant 33 % de niveau d'exposition à la menace). Le niveau de nocivité pour chaque cas de tests est donné dans la dernière colonne à droite. Nous fournissons ces niveaux pour tous les navigateurs testés. La table montre donc à la fois le niveau d'exposition de chaque navigateur et le niveau de nocivité potentiel de chaque vecteur. La table montre un moyen simple de sélectionner un sous-ensemble de navigateurs permettant un maximum d'attaques. Nous pouvons utiliser cette matrice pour sélectionner les cas de tests qui peuvent être utilisés pour tester une certaine catégorie de navigateurs web. Par exemple les cas de tests 7, 8, 9, 10, 14, 18 ne sont pas nocifs pour les navigateurs web modernes. Quelques cas de tests ont un niveau de nocivité de 0 %, ce qui signifie qu'ils sont inutiles pour l'ensemble des navigateurs choisis pour cet article. Si nous corrélons ces résultats avec les statistiques provenant de sites web (comme celles publiées sur W3School [26]), nous pouvons aisément déterminer le niveau de menace d'un vecteur XSS donné transitant par un site web. Le développement rapide des navigateurs actuels rend complexe la tâche de suivi de l'efficacité d'un vecteur XSS, et lorsqu'un nouveau vecteur est découvert, il est fastidieux de le tester sur de nombreux navigateurs. XSS Test Driver répond à ces problématiques tout en facilitant les comparaisons.

Résultats pour les navigateurs modernes Avec les navigateurs récents considérés, 32 des 84 cas de test sont concluants, c'est-à-dire que les vecteurs XSS sont effectivement exécutés au sein de l'outil.

Le comportement de Safari et Chrome pour les 84 cas de test est identique excepté pour les tests #16 et #83. Ceci s'explique par l'utilisation de Apple Webkit 534.3 comme moteur de rendu pour Chrome et l'utilisation de la version 531.22.7 (version indiquée par le User-Agent) pour Safari. Confirmant la prépondérance du moteur HTML dans l'execution des vecteurs de XSS.

Peu de cas de test sont efficaces sur tout le panel des navigateurs. Les vecteurs 3 à 6 sont des balises `<script>` standard basées XSS contenant différentes charges utiles. Les tests #12,#13 et #15 sont des balises `<body>` basées XSS

avec des OnLoad events pour exécuter les charges utiles. Le test #17 est une balise avec double chevrons pour échapper aux filtres classiques. Enfin, les données du test #19 offrent une forme très intéressante d'évasion basée sur une balise semi-ouverte `<iframe>` chargeant la payload d'une page HTML dédiée : `<iframe src=/inc/16/payload.html <`

Résultats pour les navigateurs mobiles Pour les navigateurs mobiles, 43 des 84 cas de test sont concluants, avec cependant des comportements sensiblement différents des navigateurs web modernes. Si l'on compare les résultats des 2 navigateurs basés sur Webkit, Safari Mobile et Chrome, les résultats sont similaires car ils sont tous deux basés sur webkit.

Comparaison des versions mobile et standard

La comparaison de la version mobile et standard d'une même famille de navigateurs fait apparaître de légères différences comme entre Opera Mobile et Desktop ou Firefox 4 Mobile et Desktop (Table 3) par exemple.

Entre Opera mobile et standard, seule l'exécution du vecteur n°53 suivant met en évidence une différence de comportement :

```
<input onfocus=javascript:eval(String['fromCharCode']
(97,108,101,114,116,40,39,120,115,115,39,41,32)) autofocus>
```

Dans la mesure où ils embarquent le même moteur presto, ils reconnaissent les mêmes vecteurs mais la navigation mobile induit une implémentation différente des evénements Javascript, onfocus dans notre cas. On remarquera qu'une légère différence est aussi présente entre la version desktop et la version « mobile emulator ».

On peut observer la même différence de comportement entre les versions mobile et standard de Firefox.

Résultats pour les anciens navigateurs Comme nous pouvons le constater, bien qu'encore largement utilisé dans le monde de l'entreprise, IE6 présente la plus haute exposition aux menaces, avec 45 % des vecteurs interprétés. L'exposition spécifique d'une entreprise peut être calculée à partir de sa population de navigateurs web (obtenue à partir d'un inventaire software ou de statistiques internes) et pondérée par la répartition de ses navigateurs. Sachant cela, et en utilisant le bon sous-ensemble de vecteurs d'attaques XSS, il est facile d'évaluer un intranet face aux menaces XSS et de déterminer précisément les risques associés à une vulnérabilité XSS donnée. Cette démarche apporte des informations concrètes pour prioriser les solutions à mettre en place pour les vulnérabilités détectées. Enfin, les vecteurs basés sur des propriétés et balises HTLM5, sont de fait, sans effet sur les navigateurs anciens.

Pourquoi certains vecteurs ne sont jamais exécutés ? Les résultats permettent d'observer que 16 vecteurs ne sont exécutés par aucun des navigateurs testés, ce qui s'explique par les raisons suivantes :

- des vecteurs très spécifiques à des versions précises des navigateurs, comme le test #7 issu de Xss Cheat Sheet [23], ciblant seulement les versions Firefox 2.0 et Netscape 8.1 basées sur le moteur Gecko. Firefox 2.0.0.6 et Netscape 8.1.2 avaient fait l'objet de la dernière correction.
- des erreurs dues à des contextes de test inappropriés, comme le charset utilisé pour la suite de test ou une mauvaise DTD ou encore un content-type mal défini, montrent que selon les cas, les vecteurs peuvent être dépendants ou non du contexte de test.
- des vecteurs pouvaient rendre les navigateurs instables, voire conduire au crash, comme
  `<DIV STYLE="width:expression(eval(String['fromCharCode']`
  `(97,108,101,114,116,40,39,120,115,115,39,41,32)));">`
  , qui plongeait IE dans une boucle d'attente du serveur.

Les travaux sur l'amélioration de la gestion des contextes des tests ont rapidement fait apparaître la problématique d'explosion combinatoire des cas de test. L'utilisation de stratégies de génération de cas de test, dont la méthode Pair Wise a permis de restreindre le jeu de vecteurs de test.

4.5 Tests de non-régression sur les appels JavaScript

Pour le W3C, les efforts de validation se font sur le formatage du HTML, mais très peu sur le comportement des navigateurs. La raison principale vient du coût des tests sur les navigateurs, lié à la difficulté d'automatisation. XSS Test Driver répond à cette problématique en proposant un outil et une méthodologie de test de non régression des appels à JavaScript, permettant de en évidence pour chaque navigateur :

- L'évolution dans le temps de sa robustesse face aux XSS
- La stabilité de son comportement d'une version à l'autre

La figure 3 présente l'évolution de ses caractéristiques pour Opera, qui propose une nouvelle version de son navigateur tous les six mois. Le nombre des vecteurs XSS exécutés est indiqué dans les colonnes sombres. Le nombre d'évolutions entre deux versions est indiqué dans les colonnes grises et est construit de la façon suivante :

PASS(n) représente le jeu de vecteurs XSS passant pour une version n d'un navigateur Web *wb*, *n-1* représentant la version précédente de *wb*.

$$PASS(n) = tc/verdict(tc, wb_n) = Pass, tc \in TS, \tag{3}$$

Delta représente le nombre d'évolutions entre deux versions de *wb* :

$$Delta(n, n-1) = |PASS(n) \cup PASS(n-1) - PASS(n) \cap PASS(n-1)| \tag{4}$$

Entre Opera 10.50 *(n)* et 10.10 *(n-1)*, alors que le nombre de vecteurs passant est proche (23 et 17), les évolutions *Delta (10.50, 10.10)* sont importantes (17). Ceci met en évidence une forte instabilité entre ces deux versions mineures plutôt qu'un comportement stabilisé. Cela révèle aussi un manque de tests de

non régression systématique d'une version à l'autre. Ces évolutions ne peuvent s'expliquer par les seules nouveautés d'implémentation de la norme HTML. XSS Test Driver offre donc une solution pour systématiser les tests de non régression et de nouvelles opportunités de recherche dans le domaine du test et du diagnostic des navigateurs web avec un historique des différentes versions.

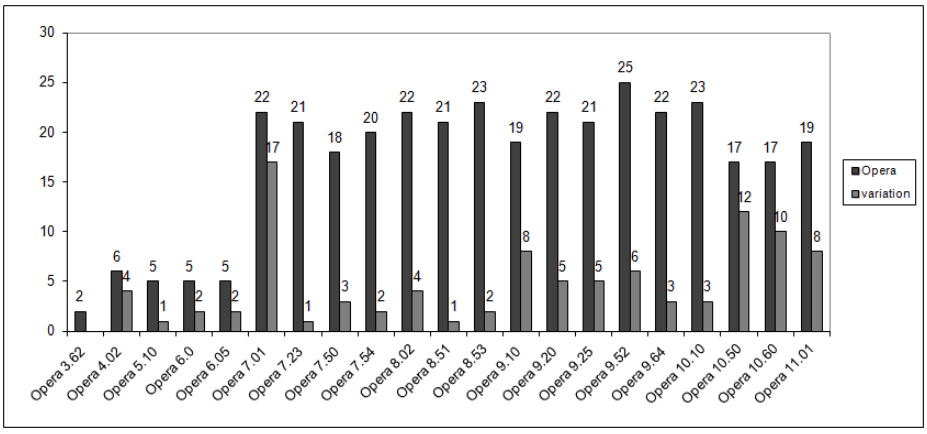

Figure 3. Tests de non regression des versions de Opera. passing vectors / *Delta(n, n-1)*

5 Conclusion et perspectives

Cet article a présenté une méthodologie et un outil pour évaluer de façon précise le comportement des navigateurs web sur mobiles face aux attaques XSS. XSS Test Driver est le Framework dédié à cette approche. La sélection des cas de test XSS pertinents pour chaque navigateur permet de lancer le bon vecteur XSS contre la bonne cible.

Pour démontrer la faisabilité de l'approche, un jeu de test XSS a été lancé contre trois types de navigateurs différents ; les anciens, les modernes et les navigateurs mobiles.

Les résultats montrent que le degré de nocivité d'un cas de test XSS varie en fonction de la vulnérabilité des navigateurs web. Concrètement, un navigateur avec un faible degré d'exposition aux menaces peut tomber face à un cas de test XSS faiblement nocif. La démarche proposée peut être précieuse pour sélectionner les tests XSS pertinents lors du déploiement d'une application web et de ses composants sécurité associés. Un axe fort de la poursuite de nos travaux s'intéressera à la construction méthodique d'une base enrichie de vecteurs XSS en utilisant des techniques telles que le *model based testing*, le *fuzzing* ou le *reverse-engineering* sur les analyseurs HTML des navigateurs web.

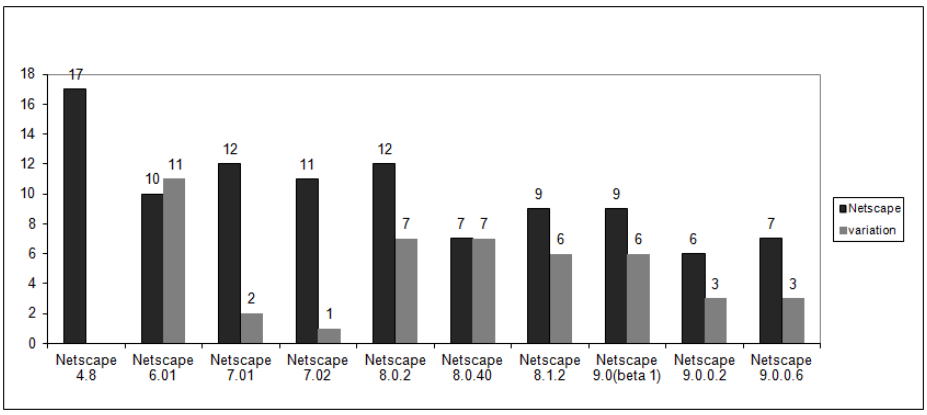

Figure 4. Tests de non regression des versions de Netscape. passing vectors / *Delta(n, n-1)*

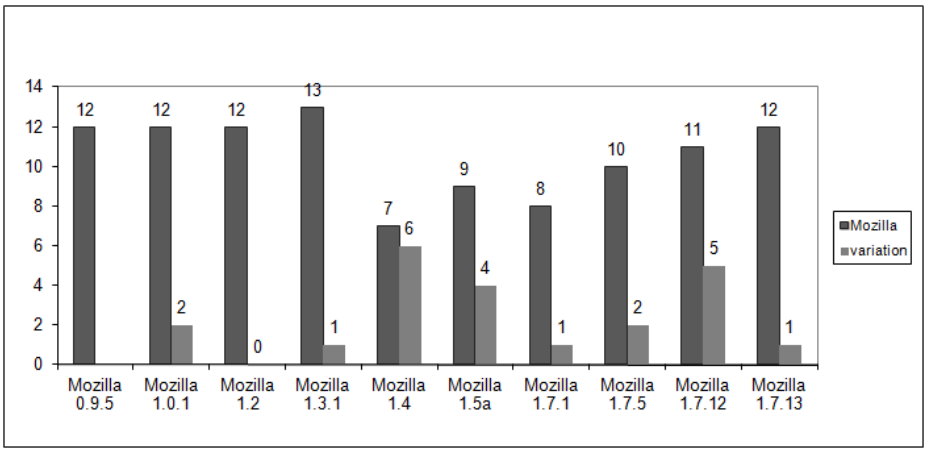

Figure 5. Tests de non regression des versions de Mozilla. passing vectors / *Delta(n, n-1)*

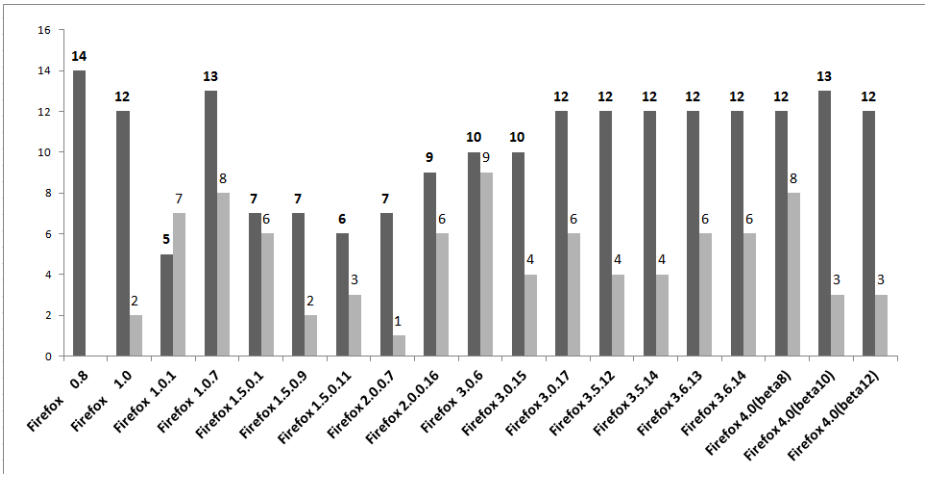

Figure 6. Tests de non regression des versions de Firefox. passing vectors / *Delta(n, n-1)*

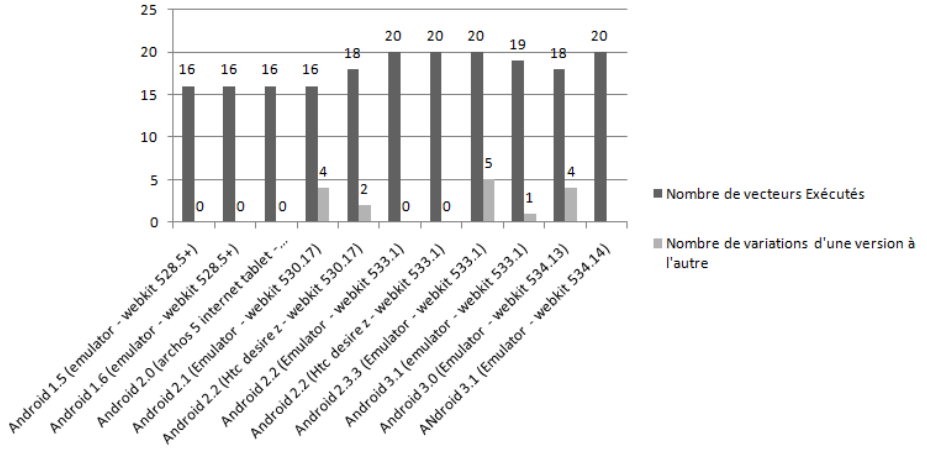

Figure 7. Tests de non regression des versions des navigateurs Android. passing vectors / *Delta(n, n-1)*

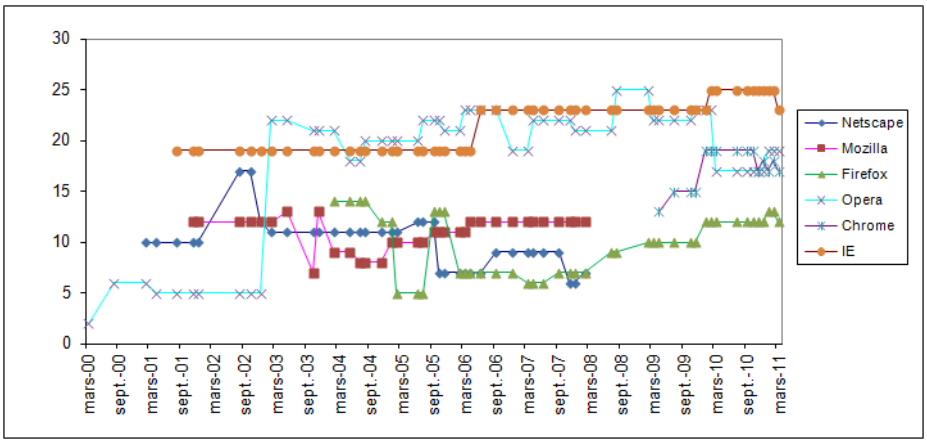

Figure 8. Evolution de la surface d'exposition des navigateurs aux XSS sur 10 ans

Remerciements Les auteurs tiennent à remercier Ingrid Kemgoum, Télécom Bretagne, campus de Rennes, pour avoir exécuté les tests sur autant de navigateurs, François Sorin pour ses commentaires avisés, Guillaume Couteau pour ses corrections. Les travaux de KEREVAL et Télécom Bretagne entrent dans le cadre du projet DALI (Design and Assessment of application Level Intrusion detection systems) supporté par le ministère français de la recherche (CNRS ACI-SI).

6 references

Références

1. Mod security. `http://www.modsecurity.org/`.
2. P. Wurzinger, C. Platzer, C. Ludl, E. Kirda, and C. Kruegel. Swap : Mitigating xss attacks using a reverse proxy. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems*, pages 33–39. IEEE Computer Society, 2009.
3. E. Nava and D. Lindsay. Abusing internet explorer 8s xss filters. *BlackHat Europe*, 2010.
4. J. Offutt, Y. Wu, X. Du, and H. Huang. Bypass testing of web applications. In *Proc. of ISSRE*, volume 4.
5. E. Martin and T. Xie. Automated test generation for access control policies via change-impact analysis. In *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, page 5. IEEE Computer Society, 2007.
6. Y. Le Traon, T. Mouelhi, and B. Baudry. Testing security policies : Going beyond functional testing. 2007.

7. T. Mouelhi, F. Fleurey, B. Baudry, and Y. Le Traon. A model-based framework for security policy specification, deployment and testing. *Model Driven Engineering Languages and Systems*, pages 537–552, 2008.
8. H. Liu and H.B. Kuan Tan. Testing input validation in web applications through automated model recovery. *Journal of Systems and Software*, 81(2) :222–233, 2008.
9. A. Tappenden, P. Beatty, and J. Miller. Agile security testing of web-based systems via httpunit. 2005.
10. J. Offutt, Q. Wang, and J. Ordille. An industrial case study of bypass testing on web applications. In *2008 International Conference on Software Testing, Verification, and Validation*, pages 465–474. IEEE, 2008.
11. Z. Su and G. Wassermann. The essence of command injection attacks in web applications. In *ACM SIGPLAN Notices*, volume 41, pages 372–382. ACM, 2006.
12. Y.W. Huang, S.K. Huang, T.P. Lin, and C.H. Tsai. Web application security assessment by fault injection and behavior monitoring. In *Proceedings of the 12th international conference on World Wide Web*, pages 148–159. ACM, 2003.
13. H. Shahriar and M. Zulkernine. Mutec : Mutation-based testing of cross site scripting. 2009.
14. F. Sun, L. Xu, and Z. Su. Client-side detection of xss worms by monitoring payload propagation. *Computer Security–ESORICS 2009*, pages 539–554, 2009.
15. J. Bau, E. Bursztein, D. Gupta, and J. Mitchell. State of the art : Automated black-box web application vulnerability testing. In *2010 IEEE Symposium on Security and Privacy*, pages 332–345. IEEE, 2010.
16. J. Shanmugam and M. Ponnavaikko. Xss application worms : New internet infestation and optimized protective measures. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on*, volume 3, pages 1164–1169. IEEE, 2007.
17. M.R. Faghani and H. Saidi. Social networks' xss worms. In *2009 International Conference on Computational Science and Engineering*, pages 1137–1141. IEEE, 2009.
18. D.M. Jayamsakthi Shanmugam1. Cross site scripting-latest developments and solutions : A survey. *Int. J. Open Problems Compt. Math*, 1(2), 2008.
19. détails de l'attaque aurora. `http://fr.wikipedia.org/wiki/Op%C3%A9ration_Aurora`.
20. Xss cheat sheet. `http://ha.ckers.org/xss.html`.
21. html5 security cheat sheet. `http://html5sec.org/`.
22. Sla.ckers web application security forum. `http://sla.ckers.org/forum/`.
23. Xss test driver demo. `http://xss.technomancie.net/`.
24. Rhinounit, framework de tests unitaires javascript. `http://code.google.com/p/rhinounit`.
25. W3c automatic page refresh. `http://www.w3.org/TR/WCAG10-CORE-TECHS/#auto-page-refresh`.
26. Utf7 xss cheat sheet. `http://openmya.hacker.jp/hasegawa/security/utf7cs.html`.

Table 2. Résultats d'exécution de 84 vecteurs par des navigateurs Récents, Anciens et Mobiles

Vector / Browser	Chrome 11.0.696.68	IE 8.0.6001.19048	Opera mobile 11	Opera 11.11 rev2109	ie mobile	Safari Mac OSX	iPhone 3GS	Android 2.2	Firefox 5 Android	Firefox 8.0a1	IE 6.0.2900.2180	Firefox 2.0.0.2	Netscape 4.8	ie 4.01	opera 4.00
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
5	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0
8	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
9	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0
10	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
11	1	1	0	0	1	1	1	1	1	1	1	1	0	1	0
12	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
13	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
14	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
18	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0
21	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0
22	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
31	1	0	1	1	0	1	1	1	0	0	0	1	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	1	1	0	0	1	1	1	1	1	1	1	1	0	1	0
34	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0
35	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0
36	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0
37	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
38	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
41	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0
42	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0
43	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0
44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Vector / Browser	Chrome 11.0.696.68	IE 8.0.6001.19048	Opera mobile 11	Opera 11.11 rev2109	ie mobile	Safari Mac OSX	iPhone 3GS	Android 2.2	Firefox 5 Android	Firefox 8.0a1	IE 6.0.2900.2180	Firefox 2.0.0.2	Netscape 4.8	ie 4.01	opera 4.00
45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
46	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
47	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
49	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
50	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
51	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
53	1	0	0	0	0	1	1	1	1	1	0	0	0	0	0
54	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
56	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
58	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
59	1	0	1	1	0	1	1	1	1	1	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
61	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
63	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
64	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
65	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
66	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
67	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
68	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
70	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0
71	0	1	0	0	1	0	0	0	0	0	1	0	0	1	0
72	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
73	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
74	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
76	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0
77	0	1	0	0	1	0	0	0	0	0	1	0	0	1	0
78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
79	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
81	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
83	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0
84	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
85	0	0	0	0	0	0	0	1	1	1	0	0	1	1	0
86	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
87	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3. Comparaison Mobiles & Desktop

Comparatif entre Navigateurs Standards & Mobiles

Vector / Browser	Opera 11.11 windows	Opera mobile 11 Android	Opera mobile Emulator	Opera Archos edition
3	1	1	1	1
4	1	1	1	1
5	1	1	1	1
6	1	1	1	1
7	0	0	0	1
8	0	0	0	1
11	0	0	0	1
12	1	0	1	1
13	1	1	1	1
14	0	0	0	1
16	1	1	1	0
17	1	1	1	1
18	0	0	0	1
20	0	0	0	1
23	0	0	0	1
26	0	0	0	1
31	1	1	1	0
33	0	0	0	1
34	0	0	0	1
35	0	0	0	1
46	0	0	0	1
50	1	1	1	1
53	1	0	0	1
54	0	0	0	0
56	1	0	0	0
59	1	1	1	0
61	0	1	1	0
64	1	1	1	1
76	1	1	1	1
83	0	0	0	0
85	0	0	0	1

Vector / Browser	Firefox 4.0.1	Firefox 4.0.2 Android.....
3	1	1
4	1	1
5	1	1
6	1	1
7	0	0
8	0	0
11	1	1
12	1	1
13	1	1
14	0	0
16	1	1
17	1	1
18	0	0
20	0	0
23	0	0
26	0	0
31	0	0
33	1	1
34	0	0
35	0	0
46	0	0
50	1	1
53	1	1
54	0	1
56	1	1
59	1	1
61	0	0
64	1	1
76	1	1
83	0	1
85	0	1

Comparatif IE 6 mobile & desktop

Vector / Browser	IE mobile	IE 6.0.2900.2180
3	1	1
4	1	1
5	1	1
6	1	1
7	1	1
9	1	1
11	1	1
12	1	1
13	1	1
14	1	1
16	1	1
17	1	1
18	1	1
20	1	1
21	1	1
22	1	1
23	0	0
26	1	1
27	0	0
33	1	1
34	1	1
35	1	1
36	1	1
37	1	1
38	1	1
40	0	0
41	1	1
42	1	1
43	1	1
44	0	0
48	1	1
49	0	1
50	1	1
51	0	0
64	1	1
65	0	1
66	0	1
67	0	0
68	1	1
69	0	0
70	1	1
71	1	1
72	0	0
76	1	1
77	1	1
78	0	0
83	1	1

Comparatif entre Navigateurs Mobiles

Vector / Browser	Firefox 4.0.2 Android	Opera mobile 11 Android	ie mobile	n810 tablet browser	iPad 2	Nokia E65	archos 5 internet tablet	iPhone 3GS	Android 2.2 Htc desire z	Android 3.1 GALAXY TAB
3	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	0	0	1	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	0	0
11	1	0	1	1	1	0	1	1	1	1
12	1	0	1	1	1	1	1	1	1	1
13	1	1	1	1	1	1	1	1	1	1
14	0	0	1	0	0	0	0	0	0	0
16	1	1	1	1	0	0	0	0	0	1
18	0	0	1	0	0	0	0	0	0	0
20	0	0	1	0	0	0	0	0	0	0
21	0	0	1	0	0	0	0	0	0	0
22	0	0	1	0	0	0	0	0	0	0
26	0	0	1	0	0	0	0	0	0	0
31	0	1	0	0	1	1	1	1	1	1
33	1	0	1	1	1	0	1	1	1	1
34	0	0	1	0	0	0	0	0	0	0
35	0	0	1	0	0	0	0	0	0	0
36	0	0	1	0	0	0	0	0	0	0
37	0	0	1	0	0	0	0	0	0	0
38	0	0	1	0	0	0	0	0	0	0
41	0	0	1	0	0	0	0	0	0	0
42	0	0	1	0	0	0	0	0	0	0
43	0	0	1	0	0	0	0	0	0	0
46	0	0	0	0	0	0	0	0	1	0
48	0	0	1	0	0	0	0	0	0	0
50	1	1	1	1	1	1	1	1	1	1
53	1	0	0	0	1	0	1	1	1	1
54	1	0	0	0	1	0	1	1	1	1
56	1	0	0	0	0	0	0	0	0	1
59	1	1	0	0	1	0	0	1	1	1
61	0	1	0	0	0	0	0	0	0	0
64	1	1	1	1	1	1	1	1	1	1
68	0	0	1	0	0	0	0	0	0	0
70	0	0	1	0	0	0	0	0	0	0
71	0	0	1	0	0	0	0	0	0	0
74	0	0	0	0	0	0	0	0	1	0
76	1	1	1	1	1	1	1	1	1	1
77	0	0	1	0	0	0	0	0	0	0
83	1	0	1	1	1	0	1	1	1	1
85	1	0	0	0	0	0	0	0	1	0