La sécurité des sites Web mobile

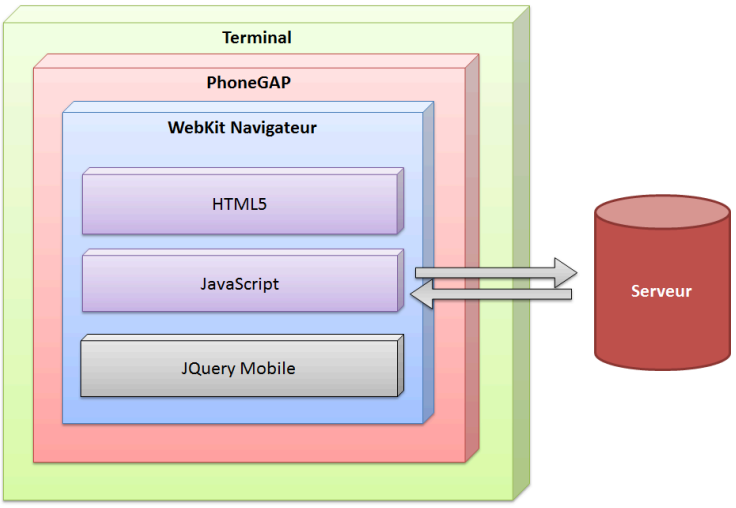
Plusieurs technologies permettent de réaliser des applications pour les mobiles. Les applications peuvent être codées à l'aide du langage de développement de la plate-forme (Objective-C, C, Java ou .Net) ou intégrer une application Web (locale ou distante), éventuellement enrichie de composants pour étendre les capacités du couple HTML5/Javascript. Quel est l'impact sur la sécurité ?

Par Philippe PRADOS - 2012
www.prados.fr

Site web mobile

Les applications utilisant un composant HTML (WebKit) ou un sites web mobile sont vulnérables à toutes les attaques classiques du Web coté client, et en particulier les attaques XSS (Cross Site Scripting).

Généralement, les applications utilisent le framework « JQuery for Mobile », combiné au composant PhoneGap[1]. Elles peuvent utiliser d'autres framework ou leurs propres librairies javascript.



Les applications exploitant ces technologies doivent respecter certaines règles, pour ne pas être vulnérable aux attaques, ou pour ne pas fragiliser le domaine web classique en exposant de nouveaux services.

JQuery mobile utilise la technologie AJAX pour récupérer des informations ou des pages complémentaires, avant de les injecter dans la page courante à l'aide de Javascript.

Ajax est construit sur les technologies DHTML (Dynamic HTML) les plus courantes :

- JavaScript: JavaScript est un langage de script utilisé couramment dans les applications Web coté client.

[1] http://phonegap.com/

- Document Object Model (DOM): DOM est un modèle objet standard pour représenter des documents HTML ou XML. Tous les navigateurs d'aujourd'hui supporte le DOM pour permettre à du code JavaScript de lire et modifier dynamiquement le contenu HTML.
- Cascading Style Sheets (CSS): CSS est un langage de style utilisé pour décrire la présentation des documents HTML. JavaScript peut modifier la feuille de style au moment de l'exécution, permettant la présentation à la page Web de se mettre à jour dynamiquement.

En Ajax, JavaScript côté client met à jour la présentation d'une page Web en modifiant dynamiquement l'arbre DOM et la feuille de style. La communication asynchrone met à jour dynamiquement la page sans avoir besoin de la recharger intégralement.

XMLHttpRequest est une API qui permet au JavaScript côté client d'établir des connexions HTTP vers des serveurs distants et d'échanger des données, telles que du texte brut, XML, JavaScript ou des objets Javascript sérialisé (JSON).

JSON (RFC 4627) est un format d'échange, indépendant de la langue. Il est basé sur un sous-ensemble du langage JavaScript. Il définit un petit ensemble de règles de formatage pour créer une représentation portable de données structurées.

Comprendre la politique de « même origine »

Lorsqu'un site intègre des contenus d'origine multiples dans une seule application, une partie pourrait avoir un niveau de confiance différents des autres. Les parties ne se font pas forcément mutuellement confiance. Il est naturel d'isoler le contenu suivant leurs sources afin de minimiser les interférences.

La politique de « même origine » est une approche de protection des navigateurs qui isole les applications Webs en provenance de différents domaines. Autrement dit, si les applications dans des fenêtres multiples ou cadres sont téléchargés à partir de serveurs différents, ils ne seront pas en mesure d'accéder aux données des autres.

Notez que la politique de la même origine ne s'applique qu'aux documents HTML. Les fichiers JavaScripts qui sont importés dans un document HTML par <script src ="..."> sont considérés comme faisant partie de la même origine, comme le document HTML.

Dans le contexte de XMLHttpRequest, la même politique d'origine est destiné à contrôler l'interaction d'une application avec des serveurs distants. Cependant, la politique de même origine n'a qu'un impact limité sur les applications Web 2.0 pour plusieurs raisons :

- Il est possible de contourner la politique de même origine, de plusieurs façons.
- Une caractéristique majeure des applications Web 2.0 est la contribution des utilisateurs au contenu. Autrement dit, le contenu n'est généralement pas assurée par un service de confiance, mais souvent par des utilisateurs anonymes à travers les blogs, les wikis, etc. Par conséquent, même le contenu d'un seul serveur peut effectivement provenir de sources multiples.
- Le navigateur applique la politique de même origine par le nom de domaine du serveur. Par exemple, http://www.abc.com/ et http://12.34.56.78/ sont traités comme des domaines différents, même si l'adresse IP de www.abc.com est réellement 12.34.56.78.
- De plus, toute expression de chemin de l'URL est ignoré. Par exemple, http://www.abc.com/~Alice est identifié comme étant de la même origine que http://www.abc.com/ ~Paul, ignorant le fait que les deux répertoires appartiennent probablement à des utilisateurs différents.

Contourner la politique de même origine.

JSON est un format texte avec une structure simple entre accolades. De nombreux canaux peuvent publier un message JSON. En raison de la politique de la même origine, vous ne pouvez pas utiliser XMLHttpRequest pour communiquer avec des serveurs externes à votre

demaine.

Pour contourner cela, il est possible d'utiliser un JSON encapsulé (JSONP[2]) pour contourner la politique de même origine. Il s'agit de combiner le flux JSON avec la balise <script>.

```
<script type="text/javascript"
 src="http://travel.com/findItinerary?
 reservationNum=1234&output=json&callback=showItinerary" />
```

Lorsqu'un JavaScript insère du code dynamiquement via la balise <script>, le navigateur accède à l'URL de l'attribut src. Il en résulte l'envoi des informations dans les paramètres de l'URL, comme le numéro de la réservation. En outre, la requête contient le format de sortie demandé au serveur et le nom de la fonction de rappel (showItinerary). Lorsque le script est produit, la fonction de rappel s'exécute et les informations renvoyées par le serveur sont transmises à travers ses arguments.

Une autre approche pour contourner la politique de même origine consiste à utiliser un reverse-proxy. C'est un service proposé par le domaine, qui permet de router une partie des requêtes vers un autre serveur cible. Le serveur sert d'intermédiaire entre la page Web et le site cible d'une autre origine. Le reverse-proxy peut s'appuyer sur une URL cible indiquée en paramètre, ou être paramétré pour modifier l'URL demandée en intervenant sur le nom de domaine avant d'effectuer la requête. Le résultat est alors transmis au navigateur.

Attention, un reverse proxy peut être utilisé pour anonymiser les connexions vers un site.

Sécurité côté terminal

Ce chapitre reprend les principales vulnérabilités présentes coté terminal :

- Cross site scripting
- Injection SQL
- Vulnérabilité JSON
- Composant WebKit

Attaques de type Cross site scripting (XSS)

XSS est une attaque classique consistant à injecter un morceau de code qui sera exécuté par le composant WebKit à l'insu de l'utilisateur. Certaines attaques sont temporaires, elle ne sont valides que pour un utilisateur particulier ; d'autres sont permanentes et impactent tous les utilisateurs du site.

Cela se présente généralement lorsqu'une information fournie par l'utilisateur revient dans une page, sans avoir été encodée. Par exemple, un utilisateur peut saisir comme nom d'utilisateur un morceau de code HTML permettant d'exécuter du code :

```
<script>alert(document.cookies)</script>.
```

Il est également possible d'avoir un XSS local, lorsque le code javascript utilise un paramètre pour produire dynamiquement du code et l'injecter dans la page.

```
<script language="JavaScript">
document.write("<a href=\""+document.location+"/image\">image</a>");
</script>
```

Une requête :

```
http://www.vulnerable.com/vulnerable.html?">Inserez l'écho ici<a%20href="
```

permet d'avoir un XSS.

---

[2]http://en.wikipedia.org/wiki/JSONP

Le code produit est alors celui-ci :

<a href="">Inserez l'écho iciimage

Il ne faut pas produire de code javascript en ligne pouvant exposer l'application à des attaques XSS.

Pour identifier cette vulnérabilité, il faut vérifier tous les usages des variables javascripts suivantes :

- document.URL
- document.URLUnencoded
- document.location
- document.referrer
- window.location

Une expression régulière peut les localiser :

document\.(write(ln)?|body\.innerHtml|.*\.action|attachEvent|create|execCommand|body)

Il faut également se méfier des variables et instructions suivantes :

Écriture du flux HTML :

- document.write(…)
- document.writeln(…)
- document.body.innerHtml=…
- Manipulation direct du DOM et des évènements DHTML :
- document.forms[0].action=… (et d'autres collections)
- document.attachEvent(…)
- document.create…(…)
- document.execCommand(…)
- document.body. …
- window.attachEvent(…)

Replacement de l'URL du document :

- document.location=…
- document.location.hostname=…
- document.location.replace(…)
- document.location.assign(…)
- document.URL=…
- window.navigate(…)

Ouverture et modification de la fenêtre :

- document.open(…)
- window.open(…)
- window.location.href=…

Exécution directe de script :

- eval(…)
- window.execScript(…)
- window.setInterval(…)
- window.setTimeout(…)

Injection SQL

HTML5 propose différentes extensions, absentes des versions précédentes. Par exemple, il est possible de manipuler une base de données SQL légère localement.

Lors de la production de requêtes par le code javascript localement, il est possible d'être confronté à une attaque d'injection SQL locale. Il s'agit pour le pirate, de livrer un extrait d'une requête SQL de telle sorte que la requête effectivement exécutée puisse retourner des informations confidentielles, présentes localement.

Généralement, ces informations sont connues de l'utilisateur, mais peuvent parfois appartenir à d'autre utilisateurs ayant utilisé le terminal.

Il est important de se prémunir des attaques d'injection SQL, lors de l'utilisation de la base de données SQL locale. Pour se prémunir des injections SQL il faut utiliser des requêtes "préparées" à l'aide de marqueurs sous forme de point d'interrogation.

La méthode transaction prend en paramètre une fonction de transaction tx qui va permettre d'exécuter des requêtes SQL avec executeSql. La méthode executeSql prend en paramètres une requête SQL, ses arguments, et en option les fonctions pour gérer le résultat et les erreurs

```
tx.executeSql(
'INSERT INTO entries (param) VALUES (?);' ,
  [param],
  sucessCallback,
  errorCallback
);
```

Notez que la base de donnée n'est pas chiffrée dans le téléphone.

Vulnérabilités liées à JSON

La plupart des réponses reçues d'un serveur sont au format JSON. Ce format est un sous-ensemble de Javascript.

Le flux reçu du serveur ne comporte normalement que des données, mais rien n'interdit qu'il contienne également des traitements, dans le cas par exemple d'une attaque de type « man of the middle » ou d'un mauvais encodage côté serveur.

JQuery mobile propose plusieurs approches pour analyser un flux JSON.

- Jquery.getJSon()
- Jquery.parseJSon()
- Jquery.ajax() qui permet d'analyser du JSON ou du JSONP.

Ces fonctions évitent les vulnérabilités évoquées.

N'utilisez que les fonctions de JQuery pour analyser les flux JSON. Il ne faut jamais utiliser eval() pour analyser un flux JSON.

Sécurisation du composant WebKit

Pour éviter l'exposition aux risques de sécurité, il est préférable de réduire au strict nécessaire les possibilités offertes à une application Web embarquée dans une application native. Pour cela, il faut désactiver dans le composant WebKit tous les services qui ne sont pas strictement nécessaires comme l'accès à la géolocalisation, aux bases de données locales, aux greffons, à JavaScript, etc.

Limiter au strict nécessaire l'activation des fonctionnalités avancées du composant WebKit.

Pour Android, afin de paramétrer au mieux le composant Webkit utilisé. (On peut aussi réécrire la méthode onCreate() de la classe DroidGap).

Il est préférable de ne pas autoriser la sauvegarde des mots de passe ou l'historique des formulaires, car ceux-ci ne sont pas chiffrés dans le téléphone.

Il ne faut pas autoriser le composant WebKit à sauvegarder les mots de passe Web. Utilisez la méthode WebSettings.savePassword(false).

L'utilisation d'un « User-Agent » spécifique peut limiter les attaques de type cross-site scripting, via un filtre côté serveur.

Utilisez un User-Agent spécifique pour l'application, et le vérifier côté serveur. Utilisez pour cela la méthode WebSettings.setUserAgentString(false). Il faut imposer la valeur de ce

User-Agent pour chaque requête, et le vérifier côté serveur lors de la réception de chaque requête. Un filtre JEE coté serveur est une bonne approche pour cela. Ainsi, les attaques XSS ne seront pas possibles avec un navigateur classique.

```
public final class MobileFilter implements Filter
 {
...
  public void doFilter(ServletRequest request,
     ServletResponse response, FilterChain chain)
     throws IOException, ServletException
  {
    if (request.getHeader("User-Agent").equals("XXX"))
     chain.doFilter(request, wrapper);
  }
}
```

Sécurité du composant PhoneGap

PhoneGap propose des extensions vers des composants ou des fonctions du téléphone, inaccessibles aux pages HTML5. Pour cela, des privilèges complémentaires doivent être accordés pour Android.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission
android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Sous iOS, une application PhoneGap (c'est valable pour toute application) est exécutée dans un espace isolé qui lui est propre appelé « Sandbox », limitant l'exposition de l'application au reste du système, ce qui définit un premier niveau de sécurité.

Les privilèges associés à l'application peuvent également être modifiés, via un fichier xml de type plist (<nom_projet>-Info.plist) contenant les droits ou entitlements associés à l'application, via un dictionnaire basé sur 3 clés : <application-identifier>, <get-task-allow> et <keychain-access-groups>.

Enfin, un profil associé à la sandbox dans laquelle s'exécute l'application permet de définir très précisément les opérations qui y sont autorisées.

Ces permissions ne sont nécessaires que si l'application utilise les services associés. Il est préférable de les ouvrir au fur et à mesure du développement.

De manière générale, on peut réduire les accès à l'API PhoneGap en modifiant la liste des modules (plugins) utilisés et en supprimant ceux qui ne sont pas nécessaires.

Sous iOS, le framework PhoneGap définit la liste des modules ou plugins utilisés dans le fichier PhoneGap.plist (rubrique Plugins), dans le répertoire Supporting Files du projet sous Xcode. Le fichier PhoneGap.plist décrit quels modules ou plugins sont autorisés à être appelés depuis JavaScript, en associant une clé avec le nom de la classe implémentant le composant. Exemple de fichier PhoneGap.plist :

```
<plist version="1.0">
        <dict>
                ...
                <key>Plugins</key>
                <dict>
                        ...
```

```
                <key>com.phonegap.accelerometer</key>
                <string>PGAccelerometer</string>
                ...
            </dict>
            ...
        </dict>
</plist>
```

À partir de la version 1.1 de PhoneGap, une liste blanche peut également être définie. Il faut configurer la clé OpenAllWhiteListURLsInWebView (toujours dans le fichier PhoneGap.plist, mettre la valeur false), puis dans la rubrique ExternalHosts, saisir les URLs dont l'appel est autorisé à partir du code JavaScript ou Objective-C.

Ceci résume les précautions à prendre pour une application Web mobile, embarquée dans une application native.

Philippe PRADOS article@prados.fr
Architecte Senior Smart Mobility - AtoS