Google Gears for Attackers

Data Theft and Backdoor Placement Attacks on Google Gears' Users

By,

Lavakumar Kuppan
www.andlabs.org
March 2, 2010

Table of Contents

Abstract: **2**

Google Gears: **3**

DATABASE: 3
LOCALSERVER: 3

Google Gears and Security: **4**

SAME ORIGIN POLICY: 4
USER PERMISSION: 4

Popular Sites using Google Gears: **5**

The attack: **6**

Stealing Data: **8**

STEALING DATA FROM THE DATABASE MODULE: 8
STEALING DATA FROM THE LOCALSERVER MODULE: 8

Setting Backdoors: **9**

BACKDOOR USING DATABASE: 9
The MySpace 0-day: *10*
BACKDOOR USING LOCALSERVER: 12
Click-activated: *12*
JavaScript Based Click-activated backdoors: 13
Flash based Click-activated backdoors: 13
Self-activated Backdoors: *14*
Cache malicious copies of common files: 14
Replace existing cached files with malicious ones: 14
Overwrite: 14
Override: 15
Override with 'Required Cookie' precedence: 15
Override ignoreQuery entries with HasNone: 16

Imposter: **17**

References: **20**

Abstract:

Google Gears is a free and open source project from Google that provides some powerful features to both web applications and site users. Browsers like Google Chrome and SRWare Iron come with Gears pre-installed. Users of other browsers can install Google Gears on their systems to experience these features.

This paper describes multiple stealthy and remote attacks against users of Google Gears which could have impacts ranging from stealing the entire Gmail Inbox of the victim to setting permanent backdoors in popular sites like Gmail, MySpace, WordPress, Google Docs etc.

For a website to make use of Google Gears, the user should explicitly permit the site to make use of Gears. Once this is done the site can store data on the user's hard disk, in the form of SQLite databases. The site can read, write and alter this database. Gears also lets the site to save and serve pages locally from the user's system, in effect, creating a web server on the user's system. All of Google Gears' features are accessible using the Gears API from JavaScript.

If an attacker controls the DNS server or is able to inject his data in to the user's HTTP traffic then an attacker can serve content for a site that has been permitted by the user to use Gears. When this happens the attacker can exploit the features provided by Google Gears to cause serious and long-lasting damage to the victim.

In traditional attacks in this scenario, the attacker would get the credentials of the user directly if the login is on HTTP. If it is on HTTPS then the attacker could perform a SSL MITM which is noisy or could strip off the SSL layer which is stealthier. A smart user might detect these attacks so alternatively the attacker could let the authentication happen securely on HTTPS and capture the post authenticated cookie, since most sites switch back to HTTP after authentication. All of these attacks depend entirely on the user entering his credentials which takes the control away from the attacker.

The attacks described in this paper could result in long-term compromise and do not require user interaction, the attack duration is typically a second or two and is virtually undetectable by the user. Gears is a new technology which creates possibilities for new types of attacks to be carried out against the user. This paper discusses in details the various attacks that can be performed using the Database and LocalServer modules of Google Gears.

Interestingly the local database and content caching features are also part of HTML5. Attacks of similar nature to the ones described in this paper does work on HTML5 as well. However, no popular site has been found to use these features of HTML5 yet, hence this paper only focuses on Google Gears.

Google Gears:

Google Gears has two very powerful modules which form the focus of this paper, they are:

1) Database

2) LocalServer

Database:

This module lets a website create a database on the user's computer and access it through JavaScript. Gears creates a SQLite file on the user's computer and lets the JavaScript from the site access it through the API provided by Gears. The API provides the ability to perform the following actions:

- Create and delete databases
- Create, delete and alter tables
- Read, insert and delete data in tables
- Search through the data in the database

LocalServer:

This module allows a site to save resources locally on the user's system. These resources could include HTML files, JavaScript files, SWF files, Image files etc. It is a more powerful and advanced form of caching pages. When the user requests for these resources, the locally saved copy is served by Gears instead of fetching it from the server. This serves two main purposes:

1. Enable Offline Web applications – By saving and serving all pages related to a web application locally the user can access the site even when not connected to the internet
2. Save bandwidth and improve speed - When static content like images, flash files etc are stored and served locally the site's loading time significantly increases and the bandwidth consumption reduces.

Google Gears and Security:

Security in Google Gears relies on two important factors:

1) Same Origin Policy
2) User Permission

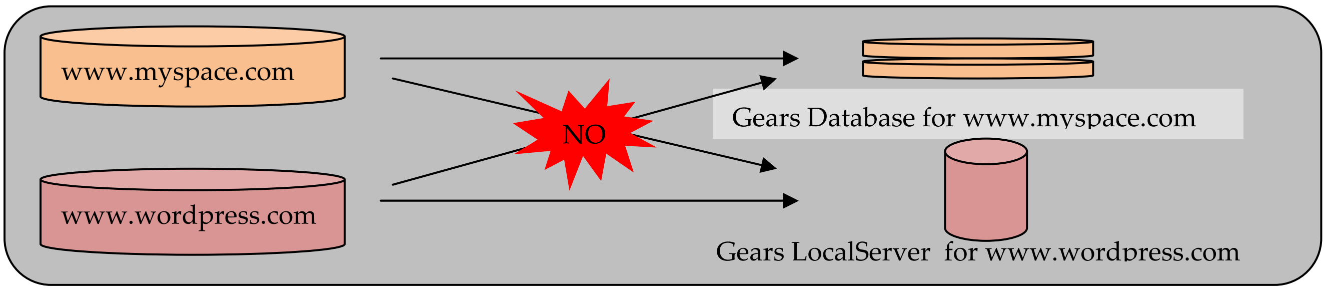
Same Origin Policy:

In Google's words, "Gears uses the same origin policy as its underlying security model. A web page with a particular *scheme, host, and port* can only access resources with the same*scheme, host, and port*.

This means a site using Gears:

- Database: Can only open databases created for that site's origin.
- LocalServer: Can only capture URLs and use manifests from the site's origin."

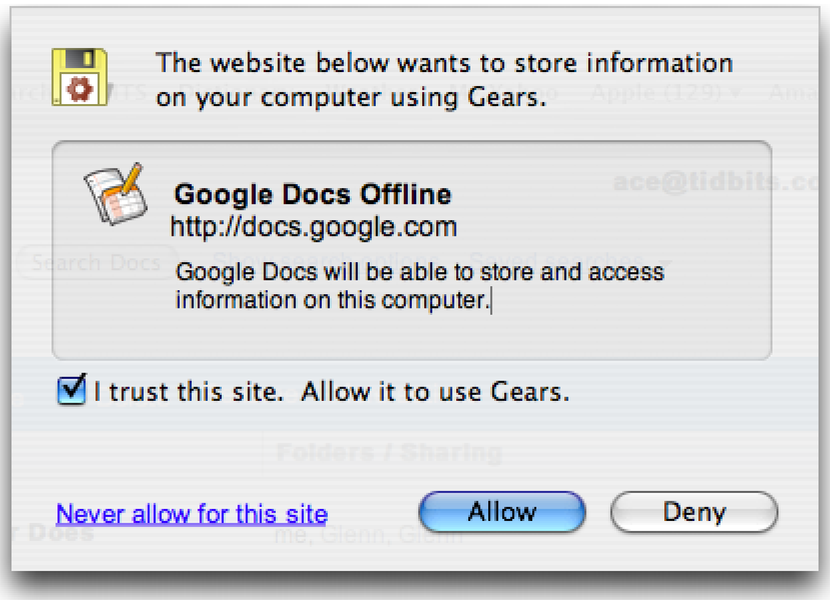
Same Origin Policy Access Control in Google Gears


www.myspace.com
www.wordpress.com
NO
Gears Database for www.myspace.com
Gears LocalServer for www.wordpress.com

User Permission:

In Google's words, "To protect users, Gears shows a warning dialog when a site first attempts to use the Gears API. User opt-in is important because Gears allows applications to store data on the user's hard disk. Users can grant or deny access for each security origin. When a user grants access to Gears for a particular origin, Gears remembers this decision for future visits. Denying access is only until the page is reloaded, though users can also choose to never allow a particular site to access Gears. Remembered decisions can later be changed using the Gears Settings dialog, located in the browser's Tools menu."

Google Gears' User Permission dialog for Google Docs


The website below wants to store information on your computer using Gears.

Google Docs Offline
http://docs.google.com
Google Docs will be able to store and access information on this computer.

☑ I trust this site. Allow it to use Gears.

In summary this means that a site can only use Google Gears after the user permits the it to do so by clicking 'Allow' on a dialog box. Once the user permits a site, the user is not asked for permission on subsequent visits to the same site. After permitting a site, the data stored in the local database and the files in the local server can only be accessed by the same site, this is due to Same Origin Policy restrictions.

Popular Sites using Google Gears:

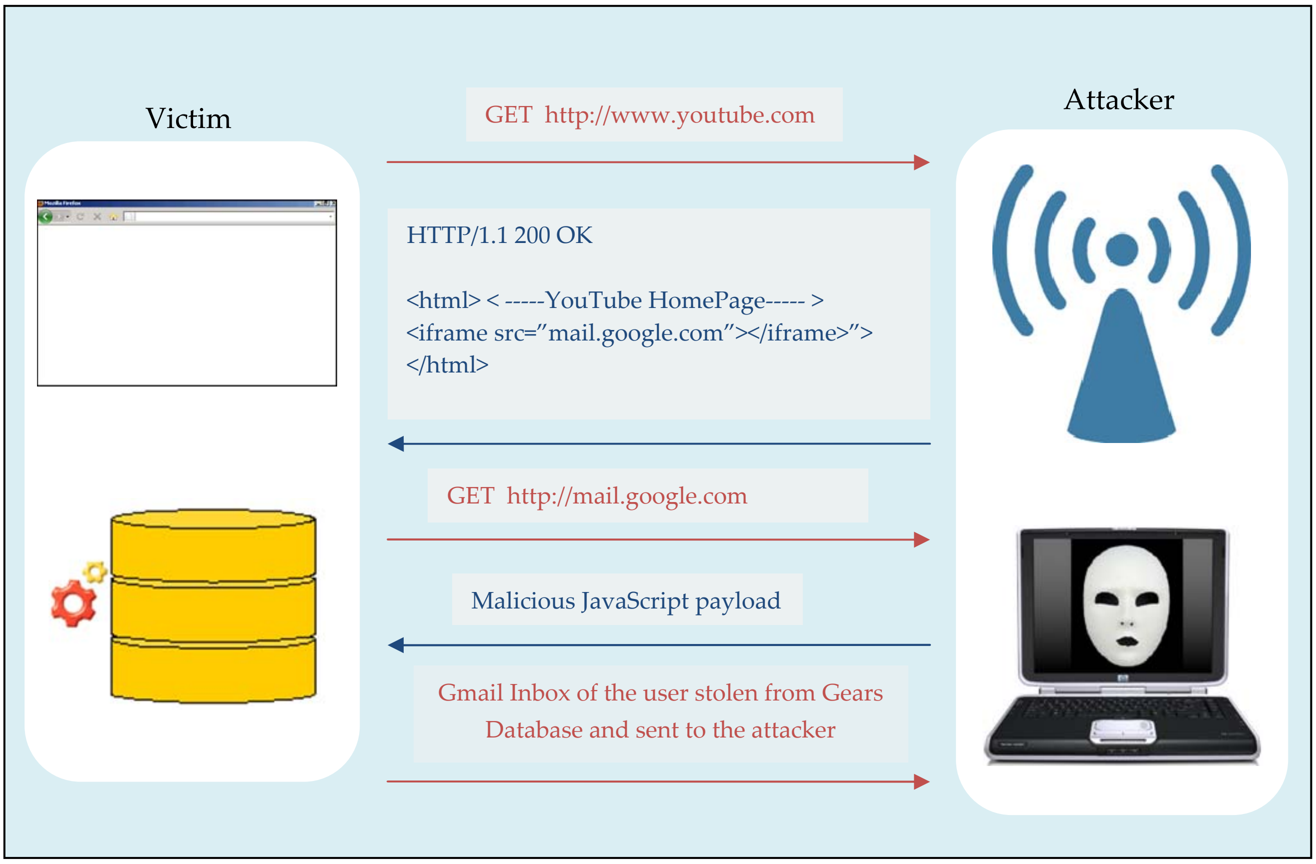
Google Gears is used by many popular sites that rely on it for very important functions. The nature of data that they store offline using Gears is very sensitive, still majority of these sites use Google Gears over HTTP instead of HTTPS.
Gmail has an 'offline' mode which when enabled stores the user's Inbox offline and also stores multiple HTML and JavaScript files in the LocalServer to facilitate offline browsing. Interestingly, Gmail now uses HTTPS by default for all users, but enabling the 'offline' mode pulls Gmail down to HTTP.

The table below lists some of the popular sites using Google Gears along with the function it is being used for.

Website	Module Type	Function of Google Gears
http://mail.google.com	Database	The entire 'Inbox' of the user is stored offline on the user's computer in the Gears database.
http://mail.google.com	LocalServer	Multiple JavaScript, Flash, CSS, Image and HTML files from the site and email attachments are cached and served from the LocalServer
http://<BlogName>.wordpress.com	LocalServer	Multiple JavaScript, Flash, HTML, CSS and Image files belonging to the administrative interface of WordPress are cached and served from the LocalServer
http://messaging.myspace.com	Database	MySpace stores all the private messages of the user in the Gears database
http://docs.google.com	Database	Contents of all documents created by the user
http://docs.google.com	LocalServer	Multiple JavaScript, CSS, Image and HTML files from the site are cached and served from the LocalServer

The attack:

Gears uses Same Origin Policy as the access control for all its data. If an attacker can control the DNS used by the victim or perform a MITM on the victim's HTTP traffic then the attacker can serve content as any website. This lets the attacker to access the Google Gears' features specific to any site that he wants to.
A typical situation where such an attack is possible is in Airports and cafes with free WiFi access. The attacker can set-up his access point and serve his own malicious content as any site to the victims connecting to his access point. This attack can be performed without any alerts on the victim's browser as long as the site is on HTTP and not HTTPS. But as shown in the table in the previous section, most of the sites use HTTP and not HTTPS to store critical data using Gears.
For example if the victim connects to the attacker's access point and requests for http://www.youtube.com the attacker can inject an iframe pointing to http://mail.google.com in the response and send in malicious JavaScript payload for http://mail.google.com which will read the contents of the user's Gmail Inbox from the Gears database and the send it back to the attacker.


Victim
Attacker
GET http://www.youtube.com
HTTP/1.1 200 OK
<html> < -----YouTube HomePage----- >
<iframe src="mail.google.com"></iframe>">
</html>
GET http://mail.google.com
Malicious JavaScript payload
Gmail Inbox of the user stolen from Gears Database and sent to the attacker

Attacker stealing the user's Gmail Inbox from Gears Database

This attack is completely invisible to the user because by using the Google Gears API, the attacker can confirm if Gears is installed on the victim's machine and if he has permitted a particular site to use it before trying to access it. This prevents an alerts or pop-ups from being displayed to the user.

The attacks against Google Gears users can be split in to two categories:

- Stealing data
- Placing backdoors

Stealing Data:

Data can be stolen from both the Gears modules:

Stealing data from the Database module:

The Database module is designed to store data in a relational database that can be queried from JavaScript. In the attack described above the attacker can serve JavaScript as a particular site, read the data stored in the database and transfer it over in the body of a POST request.
All sites that make use of the Gears database module use it to store sensitive data and most of them serve the content over HTTP, making it vulnerable to this attack. Sensitive data like the Gmail Inbox, MySpace private messages, Google Docs files etc are stored in the Gears database over HTTP.
The only information that the attacker would require to perform this attack is the name of the database.
MySpace uses the same database name for all users, it is 'myspace.messaging.database'.
Gmail however uses a different name for each user. The database name is derived from the Gmail ID, '<email ID>@gmail.com-GoogleMail' .
This makes it slightly more complicated to steal the Gmail database as the attacker should know the email ID of the person as against MySpace where it can be performed against random unknown victims. On the flip side mapping of the email ID to the database name enables an attacker to perform targeted attacks against a particular user.
Having a random and unguessable database name which varies for each user makes this attack harder to perform.

Stealing data from the LocalServer module:

Sites could cache sensitive files in the LocalServer. For example, when offline mode is enabled Gmail saves the email attachments in the LocalServer. By knowing the name of this file the attacker can steal this just like he would steal the contents of a cached file.
When an XMLHttpRequest is made to this file, it is served form the LocalServer. Its contents can now be grabbed from the 'responseBody' property.
Though it might be harder to guess the exact name of the file where the attachment might be stored, this attack could be easier to perform in certain situations.
For example consider a social networking site that stores the users' profile picture offline to speed up the site, the file name could be static like 'http://socialnetworking.site/profile.jpg'.
Now the attacker would not have to guess the filename anymore as it would be well known and remain constant across multiple users.

Setting Backdoors:

Backdoors can be placed on the victim's computer specific to any site that has enabled Google Gears over HTTP. This backdoor is an HTML, JavaScript or XML file that can give the attacker continued access to the victim's data or session of that site even after the victim has moved out of the attacker control network.
Essentially attacking with the backdoor has two parts to it:

- **Placing the backdoor:**
  The backdoor is placed when the victim is still in the attacker controlled network. This is done by serving malicious JavaScript to the victim as a legitimate Google Gears enabled site. A backdoor can be placed in different ways and in different places which are explained in detail in later sections.

- **Activating the backdoor:**
  Once the victim connects to a safe network at home or at office with the same laptop and visits the site that has been infected with a backdoor, the backdoor could get activated. Once the backdoor is activated, depending on how it has been designed by the attacker, it could either steal data, send it to the attacker regularly or it could send the current session ID to let the attacker take over the session.
  The exact actions that required by the victim to activate the backdoor could vary depending on how the backdoor is set. More details are provided in later sections.

Backdoor using Database:

Under certain circumstances backdoors can be placed using the database module. The data stored in the Gears database is sometimes, when requested for, displayed to the user making it a potential source of XSS attacks. If this data is displayed to the user without proper sanitization then Persistent XSS attacks are possible. This technique will only work on sites that have the vulnerability of displaying data from the Gears database in the unencoded form.

Placing the backdoor:

Injecting malicious HTML or JavaScript in the vulnerable fields of the site's Gears database. This can be done even over the internet if the site has a reflected Cross-site scripting vulnerability. Once exploited even if the original Cross-site scripting vulnerability is fixed the backdoor will remain active.

Activating the backdoor:

When the victim performs the action that will fetch this data from the database and display on the screen without sanitization. The exact action would depend on how and where the site is making using of this data.

The MySpace 0-day:

A live example of this is the persistent XSS vulnerability discovered by the author in the 'Search' feature of the MySpace Inbox, where the data from the Gears database is displayed to the user without proper encoding.

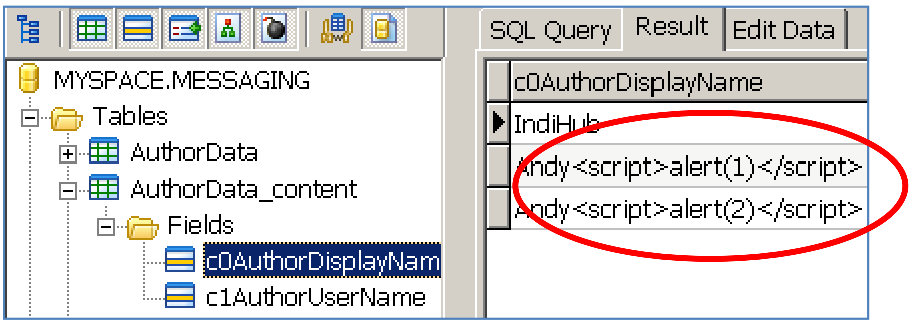
MySpace stores all the user's private messages in the Gears database to perform offline search of the Inbox, reducing load on MySpace servers. When an user searches the Inbox using a keyword, a SQL query is run on the local database for this keyword and the matching results are displayed back. The search results contain the subject of the mail and the sender's name. Both of these are taken from the local database and displayed to the user without any form of encoding or filtering.

Placing the backdoor:

Injecting malicious HTML in to the 'c0AuthorDisplayName' field of the 'AuthorData_content' table or in to 'c0Subject' field of the 'MessageData_content' table of MySpace's Gears database.

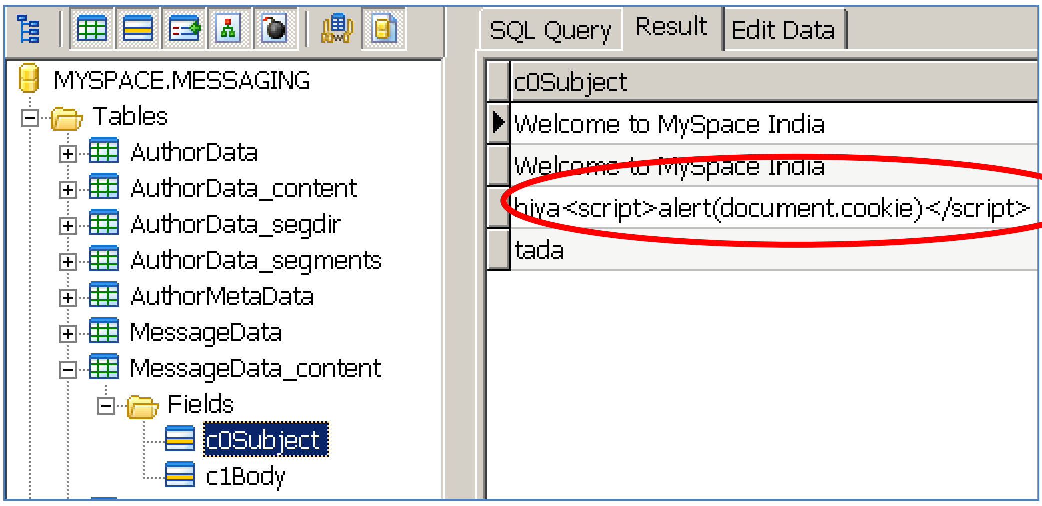
Activating the backdoor:

When the victim searches for the mail using a keyword and the entries with the injected payload show up in the search result.



c0AuthorDisplayName
IndiHub
Andy<script>alert(1)</script>
Andy<script>alert(2)</script>

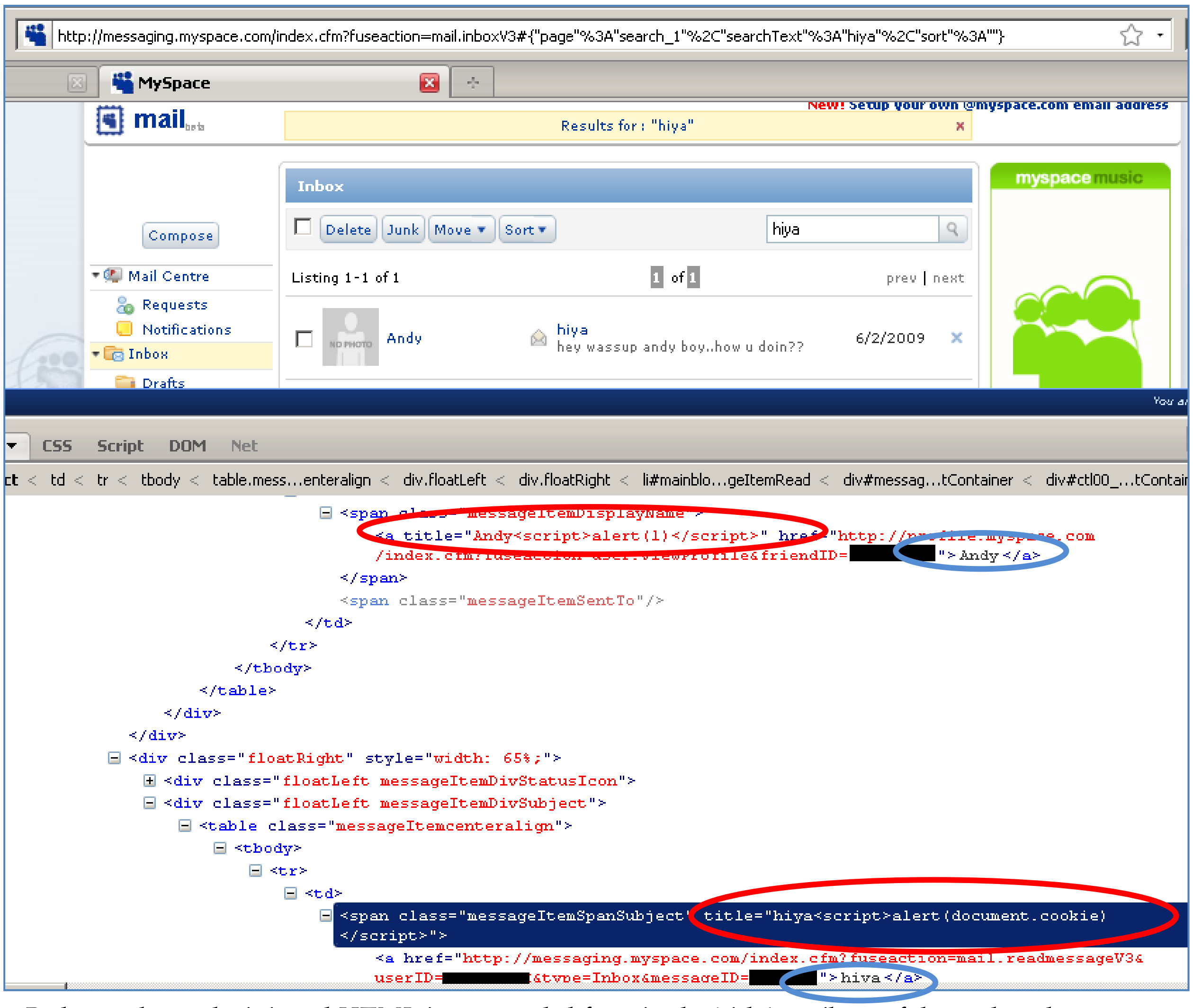
HTML Injected in to the 'AuthorData_content' table which contains the mail sender's name



```
MYSPACE.MESSAGING
  Tables
    AuthorData
    AuthorData_content
    AuthorData_segdir
    AuthorData_segments
    AuthorMetaData
    MessageData
    MessageData_content
      Fields
        c0Subject
        c1Body
```

c0Subject
Welcome to MySpace India
Welcome to MySpace India
hiya<script>alert(document.cookie)</script>
tada

HTML Injected in to the 'MessageData_content' table which contains the mail's subject



http://messaging.myspace.com/index.cfm?fuseaction=mail.inboxV3#{"page"%3A"search_1"%2C"searchText"%3A"hiya"%2C"sort"%3A""}

New! Setup your own @myspace.com email address

Results for : "hiya"

Inbox

hiya

Listing 1-1 of 1 1 of 1

Andy hiya hey wassup andy boy..how u doin?? 6/2/2009

ct < td < tr < tbody < table.mess...enteralign < div.floatLeft < div.floatRight < li#mainblo...geItemRead < div#messag...tContainer < div#ctl00_...tContai

```
            <span class="messageItemDisplayName">
              <a title="Andy<script>alert(1)</script>" href="http://profile.myspace.com
              /index.cfm?fuseaction=user.viewProfile&friendID=[redacted]">Andy</a>
            </span>
            <span class="messageItemSentTo"/>
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
<div class="floatRight" style="width: 65%;">
  <div class="floatLeft messageItemDivStatusIcon">
  <div class="floatLeft messageItemDivSubject">
    <table class="messageItemcenteralign">
      <tbody>
        <tr>
          <td>
            <span class="messageItemSpanSubject" title="hiya<script>alert(document.cookie)
            </script>">
              <a href="http://messaging.myspace.com/index.cfm?fuseaction=mail.readmessageV3&
              userID=[redacted]&type=Inbox&messageID=[redacted]">hiya</a>
```

Red spot shows the injected HTML in unencoded form in the 'title' attribute of the anchor element Blue spot shows the same value in the anchor text section. Since this view is from firebug, the injected script element is not visible

The first two screenshots show the contents of the Gears database created by MySpace. It can be seen that HTML containing some simple JavaScript has been injected in to the vulnerable fields. The third screenshot shows the results of the search made through the Inbox. The entry that was injected with HTML shows up in the search result, it can been seen from the FireBug interface that the injected HTML shows up in unencoded form.

This vulnerability was reported to MySpace, after a prompt response asking for more details and some investigation from their side, they have decided to consider this a non-issue. They see this as a problem with Google Gears and not with MySpace and hence feel it's beyond their area of control. This vulnerability still existed at the time of this writing.

Backdoor using LocalServer:

The very purpose of LocalServer is to cache pages locally and serve them when requested by the user. This makes it an ideal place to set backdoors. The attacker could cache a page containing malicious script in the LocalServer. When the victim requests for this page the backdoor is activated.

Placing the backdoor:
The attacker creates caches of malicious files for legitimate sites in the victim's LocalServer. These could either be files that have not been cached legitimately by the site. Or could instead be overwriting or overriding existing files stored by that site. Both of these approaches are discussed in further sections.

Activating the backdoor:
When the victim's browser requests for the malicious file that has been cached then the backdoor is activated. The different activation categories are also discussed in further sections.
Depending on how the backdoor will be activated LocalServer backdoors can be split in to two categories:

- Click-activated backdoors
- Self-activated backdoors

Click-activated:

Once these backdoors are placed on the victim's computer, these backdoor are activated by the attacker. To do this the attacker must force the victim to click on a link or visit a site that he controls. There are two types of Click-activated backdoors:

- JavaScript based
- Flash based

JavaScript Based Click-activated backdoors:

This is the most simple and basic form of LocalServer based backdoor. When the victim is connected to the attacker's network, the attacker caches a single page in the LocalServer of the site that he wants to backdoor. This page could be an HTML page with embedded JavaScript to steal the victim's session data and send to the attacker or to do something equally malicious.
To activate this backdoor the attacker could send a mail to the victim with a link pointing to the page that has been placed by the attacker. When the victim clicks on the link the cached page is served and the malicious JavaScript on the page is executed.

Example:
An attacker could cache a page named 'backdoor.html' in the LocalServer of http://mail.google.com, which has permission to use Google Gears.
To activate this, the attacker could send a mail to the victim's Gmail ID and use social engineering to force the victim to click on a link pointing to http://mail.google.com/backdoor.html , or to a site controlled by the attacker where http://mail.google.com/backdoor.html is called in an iframe invisible to the user. Since the user is logged in to Gmail the backdoor can potentially send over session related data to the attacker.

Flash based Click-activated backdoors:

In this technique the attacker places a 'crossdomain.xml' file in the LocalServer of the Gears-enabled site that he wants to backdoor. This 'crossdomain.xml' will either have a universal wildcard (*) 'allow-access-from' value or will have an 'allow-access-from' value set for the attacker's site. To activate the backdoor the attacker will lure the victim in to visiting his website, this site will contain a flash file which can make cross-domain calls to the site which contains the 'crossdomain.xml' backdoor.

Example:
An attacker could place the following 'crossdomain.xml' file in the LocalServer of http://messaging.myspace.com:

```
<cross-domain-policy>
<allow-access-from domain="www.attacker.site" />
</cross-domain-policy>
```

Now the attacker can mail the link http://www.attacker.site to the victim on MySpace. When the victim visits http://www.attacker.site, flash file from this site can make calls to http://messaging.myspace.com and access the victim's MySpace mail functionality.

Self-activated Backdoors:

These backdoors get activated automatically when the victim logs in and uses the 'backdoored' site. Unlike the Click-activated backdoors, the attacker does not have to lure the victim's in to clicking any links or visiting websites.

There are two ways to do this:

Cache malicious copies of common files:

Websites make use of many standard JavaScript files whose name and path might remain constant. A site may have a file named 'browserCheck.js' containing the code to check the type of the browser, such a file would be called each time a user visits the site. The name and path of such a file might remain constant. If the site is Gears-enabled and if this file is not currently cached in the LocalServer, the attacker can cache his own malicious version of this file in the LocalServer. Now each time the victim visits this site, 'browserCheck.js' is automatically called and the attacker's malicious code is executed.
Sometimes a file can have a static filename but could have a constantly changing querysting, like 'browserCheck.js?version=00234'. This can be easily handled by caching it in ManagedResourceStore and by setting the 'ignoreQuery' option in the manifest file.

Replace existing cached files with malicious ones:

Sites like Gmail, Google Docs, WordPress etc that make use of the LocalServer module cache many JavaScript and HTML files. When the user logs in and works on these sites, these locally cached files are requested for. If an attacker is able to replace a legitimate cached file with his own malicious file then every time the user logs in to these sites the malicious is executed.

There are two approaches to this:

Overwrite:

This involves opening an existing ResorceStore(RS) or ManagedResourceStore(MRS) and explicitly overwriting the legitimate file with a malicious equivalent. This approach works 100% of the time and is very easy to perform for RS.
But it is a lot trickier with MRS. Because to force an update of the MRS, the attacker must serve a manifest file with a version number different than the current one. Once the update begins the attacker should be able to serve all files in the manifest or the update fails. After a successful update the attacker must do an update again, this time to set the version of the manifest file to its earlier value. If this is not done, when the victim connects to the actual site the MRS will get updated due to version mismatch and the attacker's backdoor will get overwritten. Even if the attacker sets the manifest file's version to the current value, success is not guaranteed as the version of the manifest file can be changed by the site owner anytime.

Even with RS, since the filename is known by the site owner he can update it to a newer file anytime, thereby replacing the malicious copy with a safe one.

Override:

In this approach a new RS or MRS is created where a file with the same name as an already cached file is saved. Now the LocalServer would have two files with the same name, in this situation if all properties of these two files are the same then the file that is cached first (i.e. the legitimate file) would be taking precedence. However under certain circumstances the malicious file that is cached later by the attacker can take precedence over the earlier created legitimate file. This is achieved by tweaking certain parameters of the file that the attacker caches. The different techniques to do that are discussed in detail below:

Override with 'Required Cookie' precedence:

'Required Cookie' is an optional property of both RS and MRS. If this property is set then the files saved in that store are only served if the browser has the cookie value specified.

For example, if a resource store is created this way:

```
var rs = localServer.createStore('demo','loggedin=1');
```

Then the files from this store are only served when the browser has the 'loggedin=1' key-value pair in the cookie.

The interesting behavior with this feature is that if there are two cached copies for the same file where one of them has the 'Required Cookie' property set then that entry takes precedence when the browser has the required value in its cookie. This happens irrespective of which of these two entries were set first.

Any site that uses RS or MRS without the 'RequiredCookie' property but sets a cookie with a static key-value pair becomes vulnerable.

WordPress for example does not use the 'RequiredCookie' property in its MRS. However it sets a cookie containing the static key-value pair "wordpress_test_cookie=WP+Cookie+check".

If an attacker creates a malicious MRS with similar files as the legitimate MRS but has the 'RequiredCookie' property set to "wordpress_test_cookie=WP+Cookie+check" then the attacker's files would be served instead of the legitimate files.

This technique works for both RS and MRS.

Override ignoreQuery entries with HasNone:

MRS provides greater flexibility in saving files. If a file's QueryString parameter would continuously change then it can be cached with the 'ignoreQuery' property set. This would make Gears ignore the query value and serve the same file irrespective of its QueryString value. This can be a very useful feature sometimes but the same can create an opportunity for abuse as well.
MRS files have another property called 'hasNone' which would serve a file when the QueryString does not have the value mentioned in the 'hasNone' property.

If two files of the same name are cached where one has the 'ignoreQuery' attribute set and another has the 'hasNone' attribute set then the 'hasNone' entry take precedence as long as the QueryString does not contain the specified value.

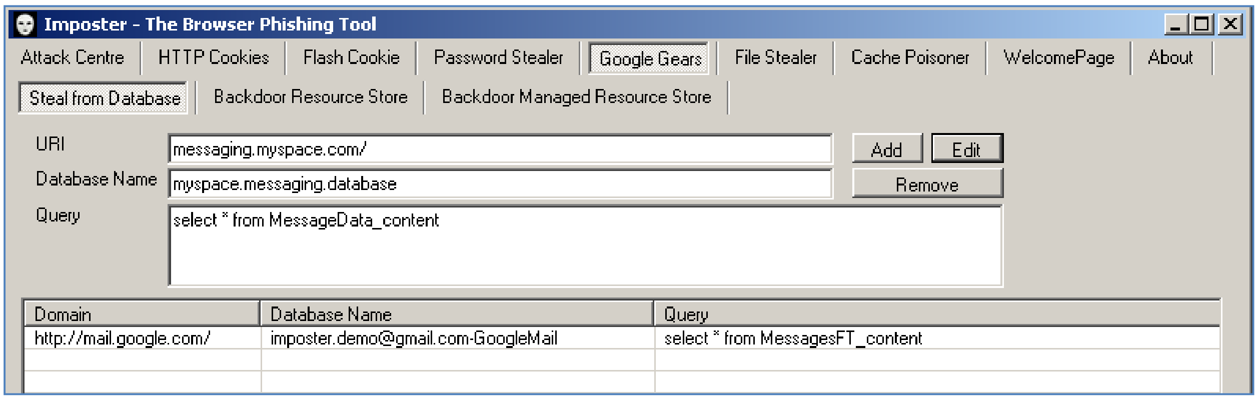
For example if there are two entries for the file http://www.andlabs.org/report.php.
The first entry has the 'ignoreQuery' option set. The second one has the 'hasNone' property set to "zzz".
When the request for http://www.andlabs.org/report.php?hitcount=34545665 is made then the second entry is served since the QueryString does not contain "zzz".

This can be exploited by an attacker to set malicious copies of legitimate files with the 'hasNone' property set with a value that is unlikely to be part of query.

Imposter:

Imposter is a tool that has been designed to perform multiple browser phishing attacks against the browser. The 'Google Gears' module of Imposter can be used to perform the database stealing and backdoor placement attacks explained in this paper.

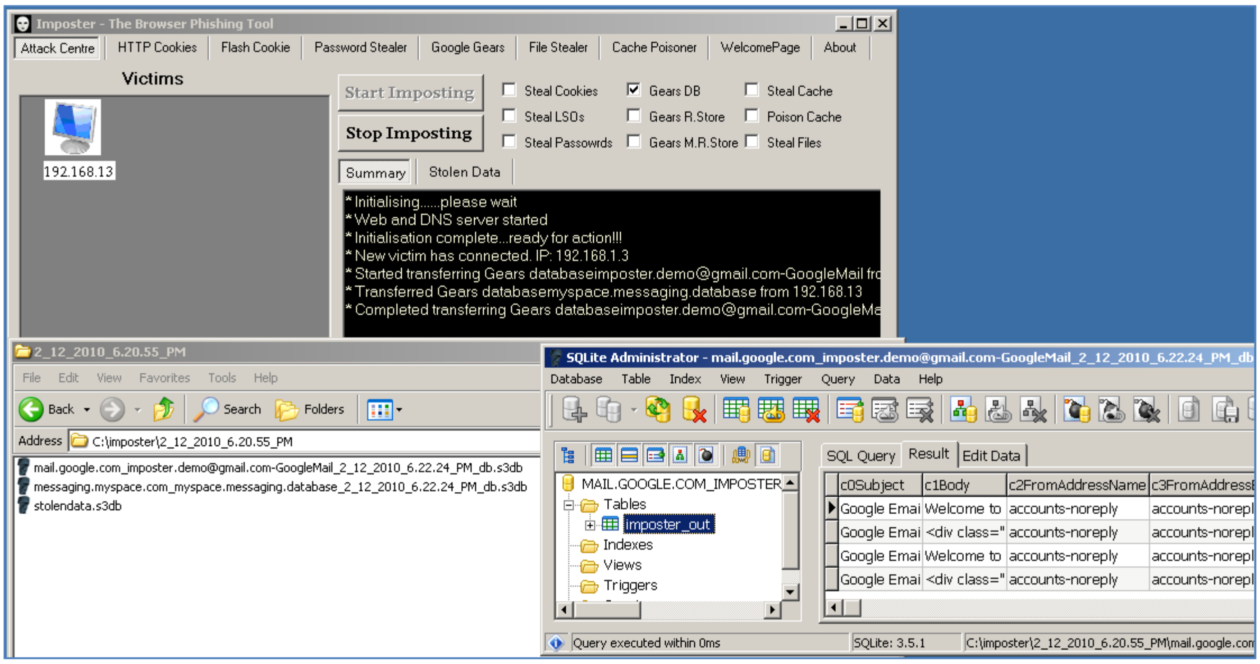
Database Query:
To interact with the database the attacker has to enter the URL of the website, the name of the database and the SQL Query. Imposter generates the JavaScript required to run this query and transfer its output. The results are stored in a SQLite database on the attacker's system.



URI	messaging.myspace.com/
Database Name	myspace.messaging.database
Query	select * from MessageData_content

Domain	Database Name	Query
http://mail.google.com/	imposter.demo@gmail.com-GoogleMail	select * from MessagesFT_content

Section in Imposter where the attacker enters the database name and query



Steal Cookies [ ] Gears DB [✓] Steal Cache [ ]
Steal LSOs [ ] Gears R.Store [ ] Poison Cache [ ]
Steal Passowrds [ ] Gears M.R.Store [ ] Steal Files [ ]

```
* Initialising......please wait
* Web and DNS server started
* Initialisation complete...ready for action!!!
* New victim has connected. IP: 192.168.1.3
* Started transferring Gears databaseimposter.demo@gmail.com-GoogleMail fro
* Transferred Gears databasemyspace.messaging.database from 192.168.13
* Completed transferring Gears databaseimposter.demo@gmail.com-GoogleMa
```

Address C:\imposter\2_12_2010_6.20.55_PM

mail.google.com_imposter.demo@gmail.com-GoogleMail_2_12_2010_6.22.24_PM_db.s3db
messaging.myspace.com_myspace.messaging.database_2_12_2010_6.22.24_PM_db.s3db
stolendata.s3db

SQLite Administrator - mail.google.com_imposter.demo@gmail.com-GoogleMail_2_12_2010_6.22.24_PM_db

MAIL.GOOGLE.COM_IMPOSTER
Tables
imposter_out
Indexes
Views
Triggers

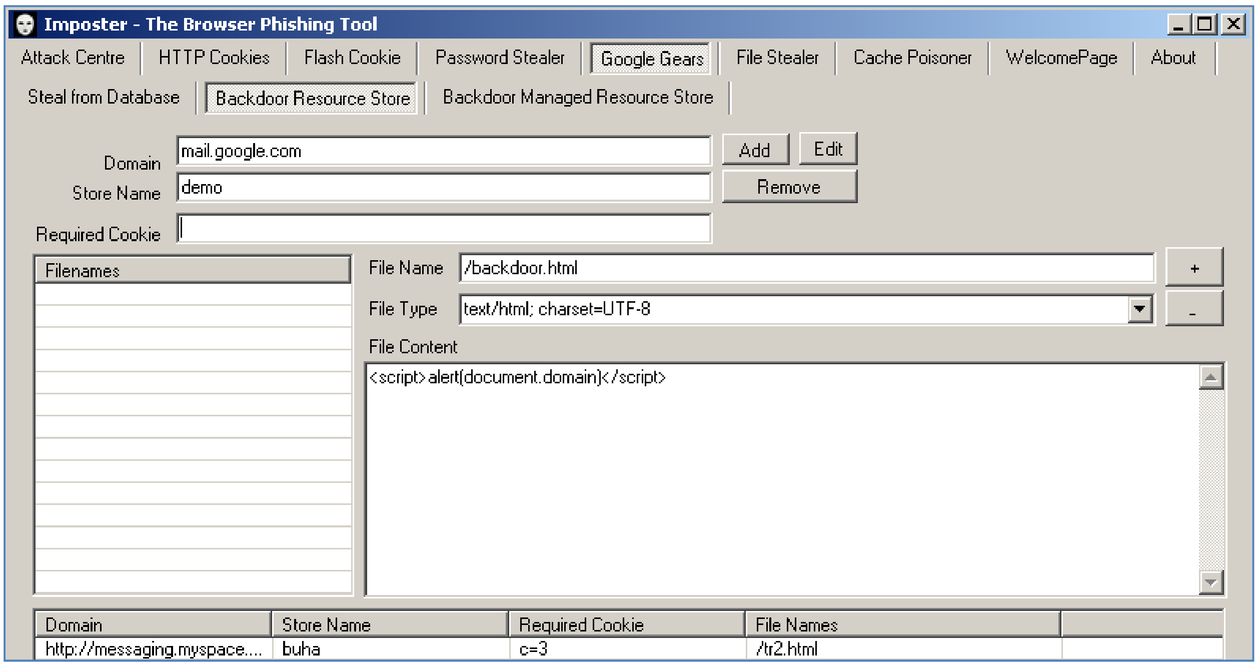
c0Subject	c1Body	c2FromAddressName	c3FromAddress
Google Emai	Welcome to	accounts-noreply	accounts-norepl
Google Emai	<div class="	accounts-noreply	accounts-norepl
Google Emai	Welcome to	accounts-noreply	accounts-norepl
Google Emai	<div class="	accounts-noreply	accounts-norepl

Query executed within 0ms | SQLite: 3.5.1 | C:\imposter\2_12_2010_6.20.55_PM\mail.google.com

The results of the query are transferred over and stored in the attacker's system.

LocalServer backdoors:

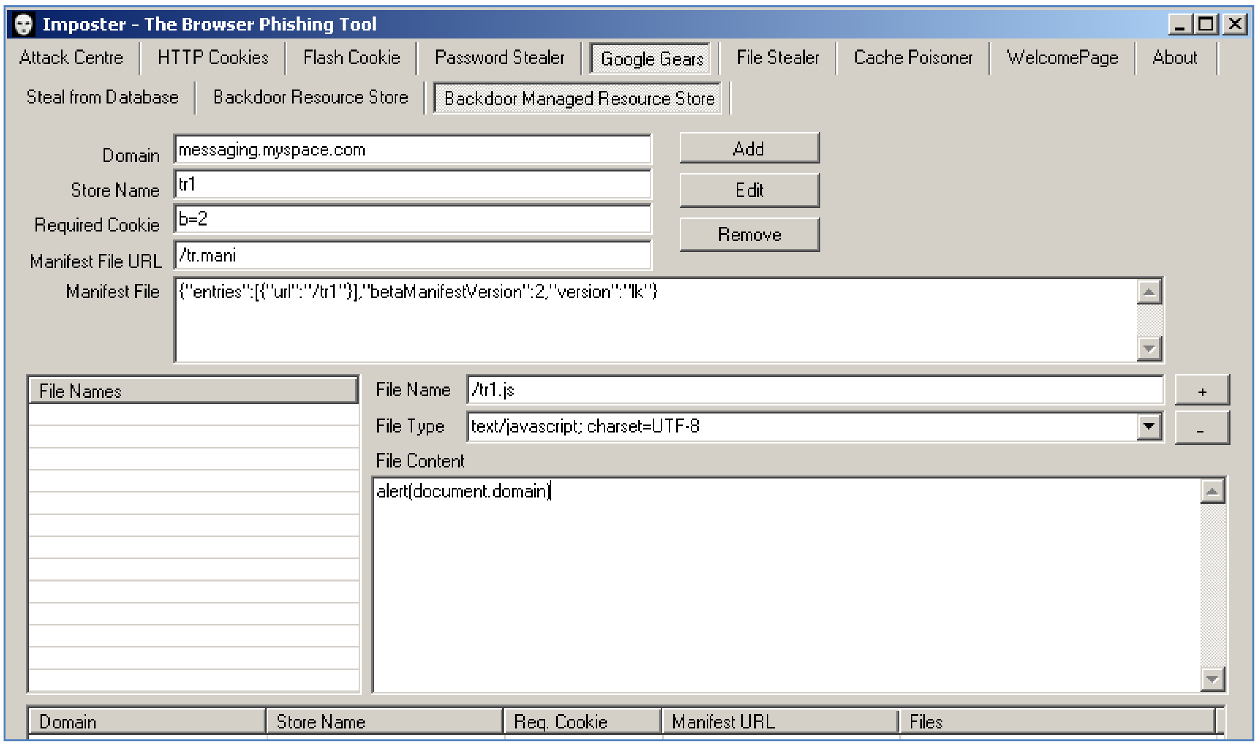
Imposter can set both RS and MRS backdoors. The attacker must specify the name of the URL, name of the store, RequiredCookie value, Manifest file details and the name and content of the files to be cached.



Domain	mail.google.com
Store Name	demo
Required Cookie	
File Name	/backdoor.html
File Type	text/html; charset=UTF-8
File Content	<script>alert(document.domain)</script>

Domain	Store Name	Required Cookie	File Names
http://messaging.myspace....	buha	c=3	/tr2.html

Section where details to create a RS backdoor are entered

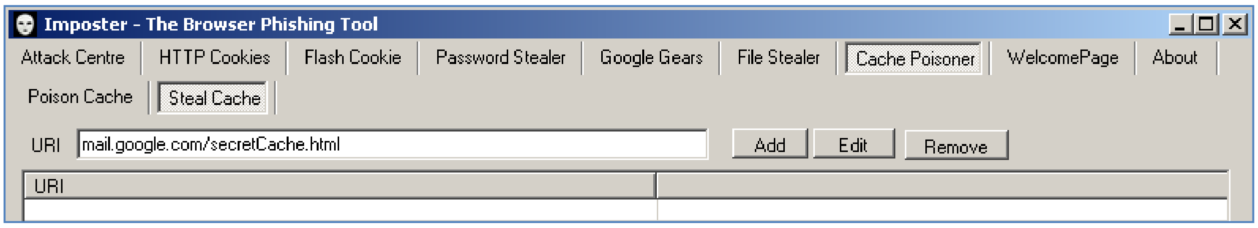


Domain	messaging.myspace.com
Store Name	tr1
Required Cookie	b=2
Manifest File URL	/tr.mani
Manifest File	{"entries":[{"url":"/tr1"}],"betaManifestVersion":2,"version":"lk"}
File Name	/tr1.js
File Type	text/javascript; charset=UTF-8
File Content	alert(document.domain)

Domain	Store Name	Req. Cookie	Manifest URL	Files

Section where details to create a MRS backdoor are entered

Cached RS and MRS files can be stolen from the victim using the 'Steal Cache' module of Imposter.



URI: mail.google.com/secretCache.html

Section where the URL of the cached file to be stolen are entered

Imposter uses two main components to perform these attack:

1. Custom DNS Server

   The DNS server sends fake DNS responses to redirect all HTTP requests to Imposter's web server.

2. Custom Web Server

   The web server reads the requests and sends the suitable payload to the clients. It also collects the stolen data from the client and saves it.

References:

- Lust 2.0-Desire for free Wi-Fi and the threat of the Imposter
  http://www.andlabs.org/presentations/lust_2.0.pdf

- Imposter, the browser phishing tool
  http://www.andlabs.org/tools/imposter/imposter.html

- Browser Phishing
  http://blog.andlabs.org/2009/11/browser-phishing-explained.html

- Official Google Gears API documentation
  http://code.google.com/apis/gears/

- Active Man in the Middle Attacks by Adi Sharabani and Roi Saltzman, IBM
  http://blog.watchfire.com/AMitM.pdf

- Client-side SQL Injection Attacks by pdp, Gnucitizen.
  http://www.gnucitizen.org/blog/client-side-sql-injection-attacks/

- Attacking Rich Internet. Applications by Alex kuza and Stefano Di Paola
  http://www.ruxcon.org.au/files/2008/Attacking_Rich_Internet_Applications.pdf