# Clickjacking

Credit: paper (“Clickjacking: Attacks and Defenses” Huang et al.) and most slide content (Vern Paxson)

# Misleading users

- Browser assumes that clicks and keystrokes = *clear indication* of what the user wants to do
  - Constitutes part of the user's *trusted path*

- Attacker can meddle with integrity of this relationship in all sorts of ways

# Misleading users

- Browser assumes that clicks and keystrokes = *clear indication* of what the user wants to do
  - Constitutes part of the user's *trusted path*

- Attacker can meddle with integrity of this relationship in all sorts of ways

- Recall the power of Javascript
  - **Alter page contents (dynamically)**
  - **Track events (mouse clicks, motion, keystrokes)**
  - Read/set cookies
  - Issue web requests, read replies

# Using JS to Steal Facebook *Likes*


User intent


Actual outcome

Bait and switch

User tries to claim their free iPad, but
you want them to click your Like button

(Many of these attacks are similar to TOCTTOU vulnerabilities)

# Clickjacking

When one principal tricks the user into
interacting with UI elements of another principal

An attack application (script) compromises the ***context integrity***
of another application's User Interface when the user acts on the UI

# Clickjacking

When one principal tricks the user into
interacting with UI elements of another principal

An attack application (script) compromises the ***context integrity*** of another application's User Interface when the user acts on the UI

Context Integrity

1. **Visual context**: what a user should see right before the sensitive action. Ensuring this = the sensitive UI element and the cursor are both visible
2. **Temporal context**: the timing of a user action. Ensuring this = the user action at a particular time is what the user intended

# Compromising visual integrity of the *target*

- Hide the target element
  - CSS lets you set the opacity of an element to zero (clear)

# Compromising visual integrity of the *target*

- Hide the target element
  - CSS lets you set the opacity of an element to zero (clear)
- Partially overlay the target
  - Or *crop* the parts you don't want to show


To: Bad guy

From: Victim

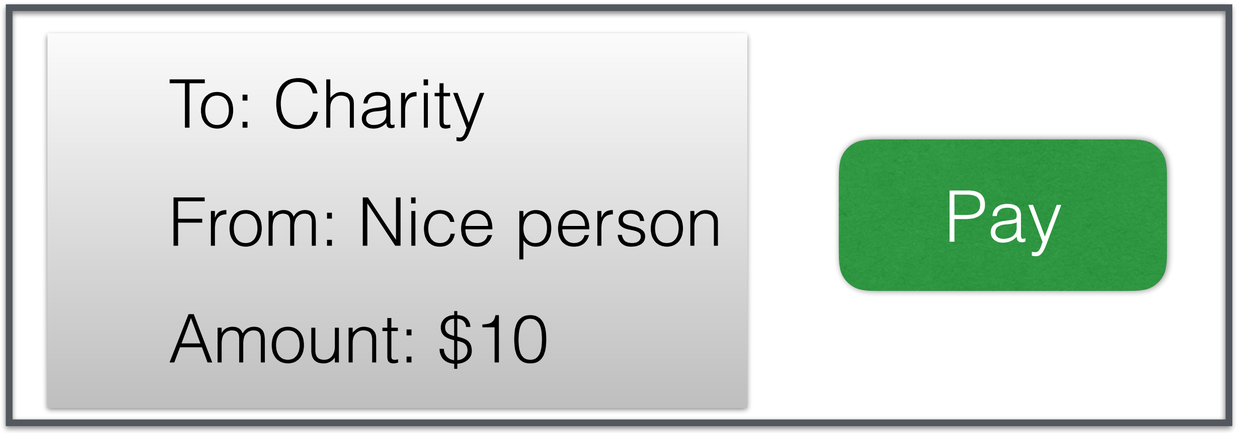
Amount: $1000

Pay

# Compromising visual integrity of the *target*

- Hide the target element
  - CSS lets you set the opacity of an element to zero (clear)


- Partially overlay the target
  - Or *crop* the parts you don't want to show



# Compromising visual integrity of the *pointer*

Claim your free iPad!


Actual cursor

- Manipulating cursor feedback

# Compromising visual integrity of the *pointer*

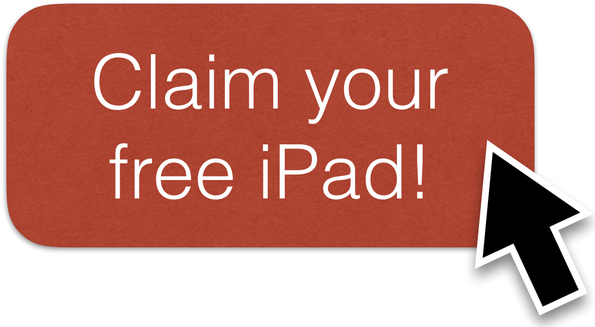
Claim your free iPad!


Displayed cursor

Actual cursor

- Manipulating cursor feedback

# Compromising visual integrity of the *pointer*




Displayed cursor


Actual cursor

- Manipulating cursor feedback

# Clickjacking to access a user's webcam

# Some clickjacking defenses

- Require confirmation for actions
  - Annoys users

- **Frame-busting**:  Website ensures that its "vulnerable" pages can't be included as a *frame* inside another browser frame
  - So user can't be looking at it with something invisible overlaid on top…
  - …nor have the site invisible above something else


The attacker implements this by placing Twitter's page in a "Frame" inside their own page, otherwise they wouldn't overlap

# Some clickjacking defenses

- Require confirmation for actions
  - Annoys users

- **Frame-busting**: Website ensures that its "vulnerable" pages can't be included as a *frame* inside another browser frame
  - So user can't be looking at it with something invisible overlaid on top…
  - …nor have the site invisible above something else

- Conceptually implemented with Javascript like

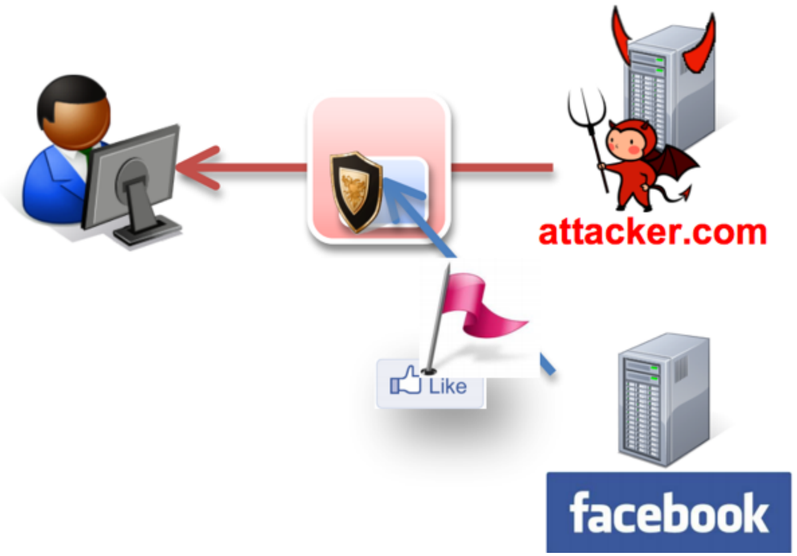
```
if(top.location != self.location)
    top.location = self.location;
```

  (actually, it's quite tricky to get this right)
- Current research considers more general approaches

# InContext Defense (recent research)

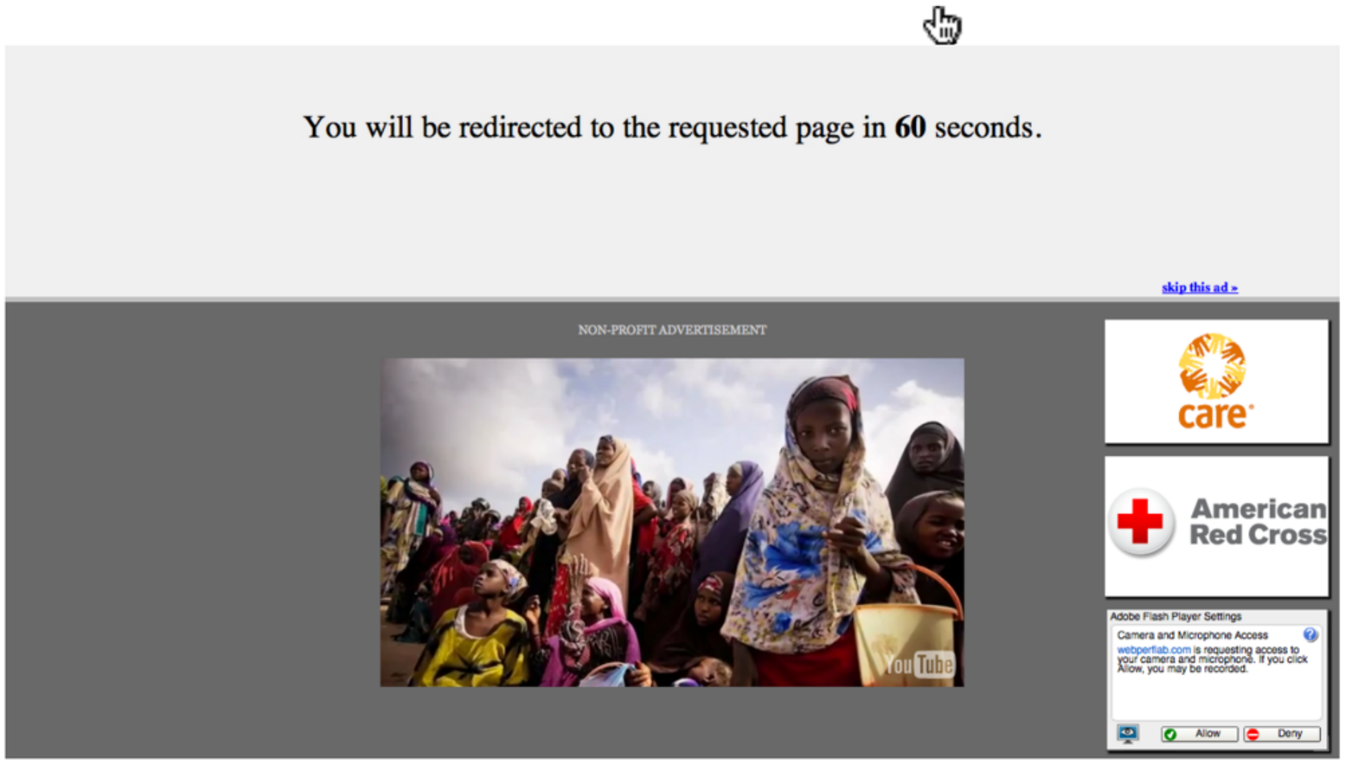
- A set of techniques to ensure context integrity for user actions

- Servers opt-in
  - Let the websites *indicate* their sensitive UIs
  - Let browsers *enforce* context integrity when users act on the sensitive UIs



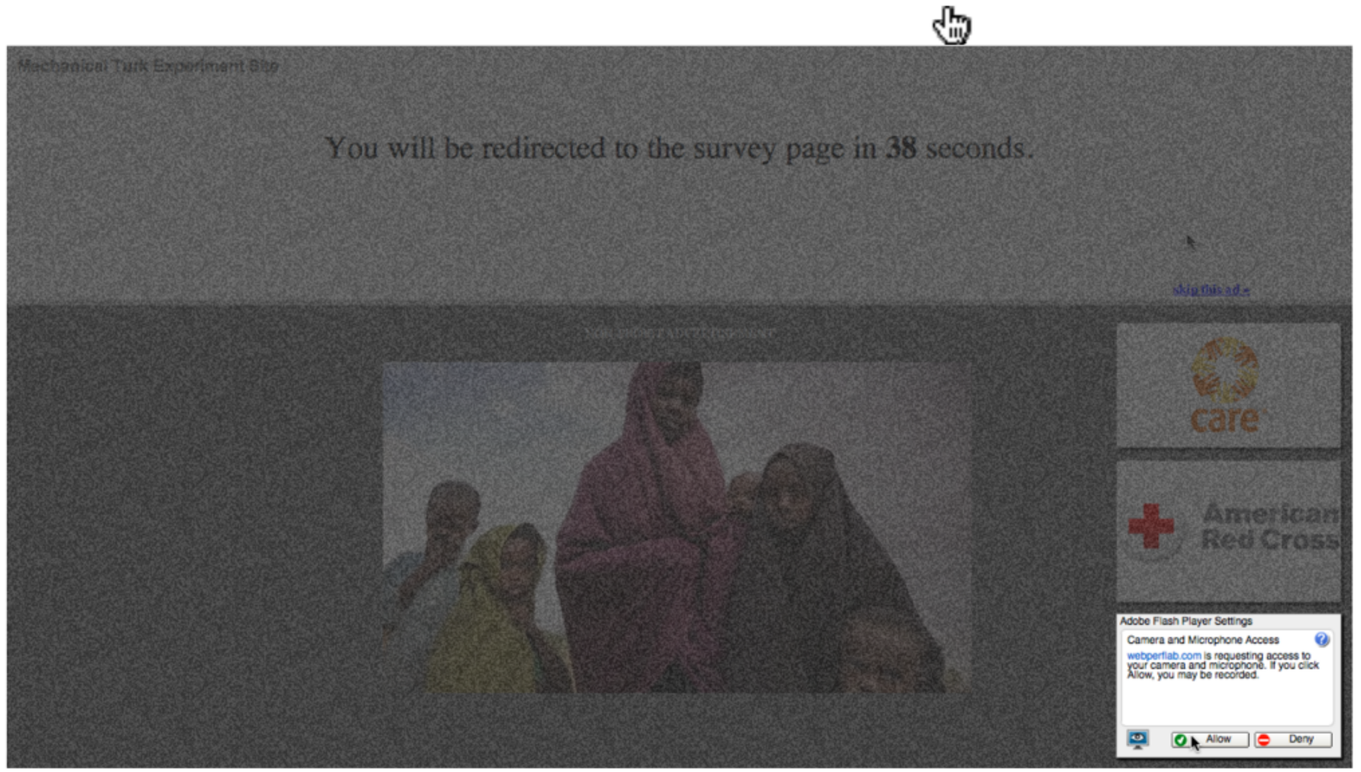
# Ensuring visual integrity of pointer

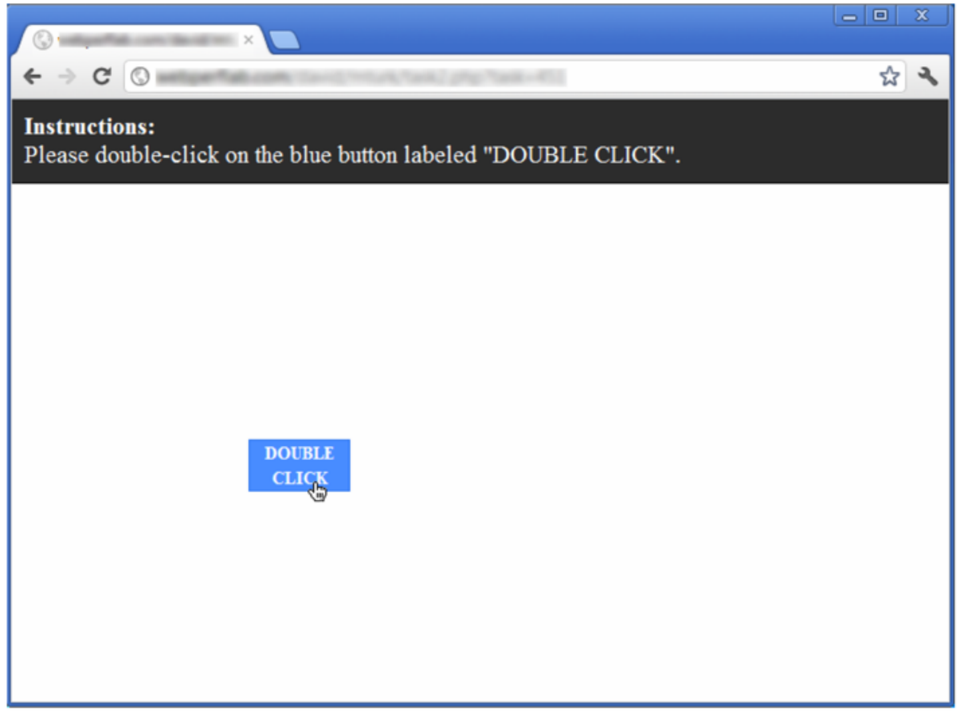
- Remove cursor customization
  - Attack success: 43% -> *16%*

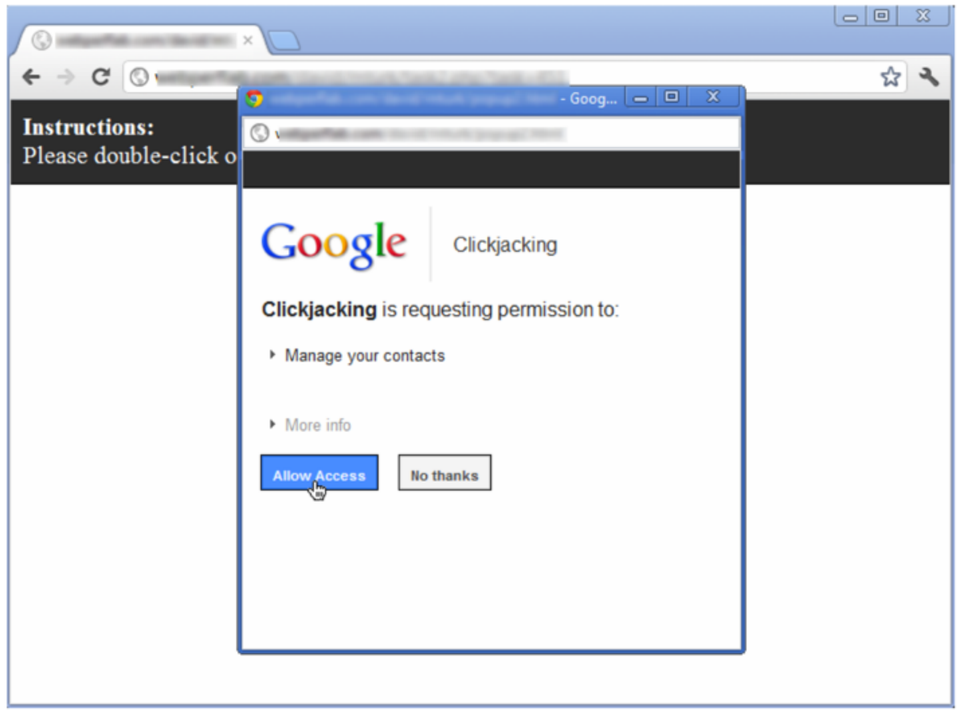


# Ensuring visual integrity of pointer

- Lightbox effect around target on pointer entry
  - Attack success (freezing + lightbox): 2%






# Enforcing temporal integrity

- UI delay: after visual changes on target or pointer, invalidate clicks for a few milliseconds
- Pointer re-entry: after visual changes on target, invalidate clicks until pointer re-enters target

# Other forms of UI sneakiness

- Along with stealing events, attackers can use the power of Javascript customization and dynamic changes to mess with the user's mind
- For example, the user may not be paying attention, so you can swap tabs on them
- Or they may find themselves "eclipsed"

# Browser in browser