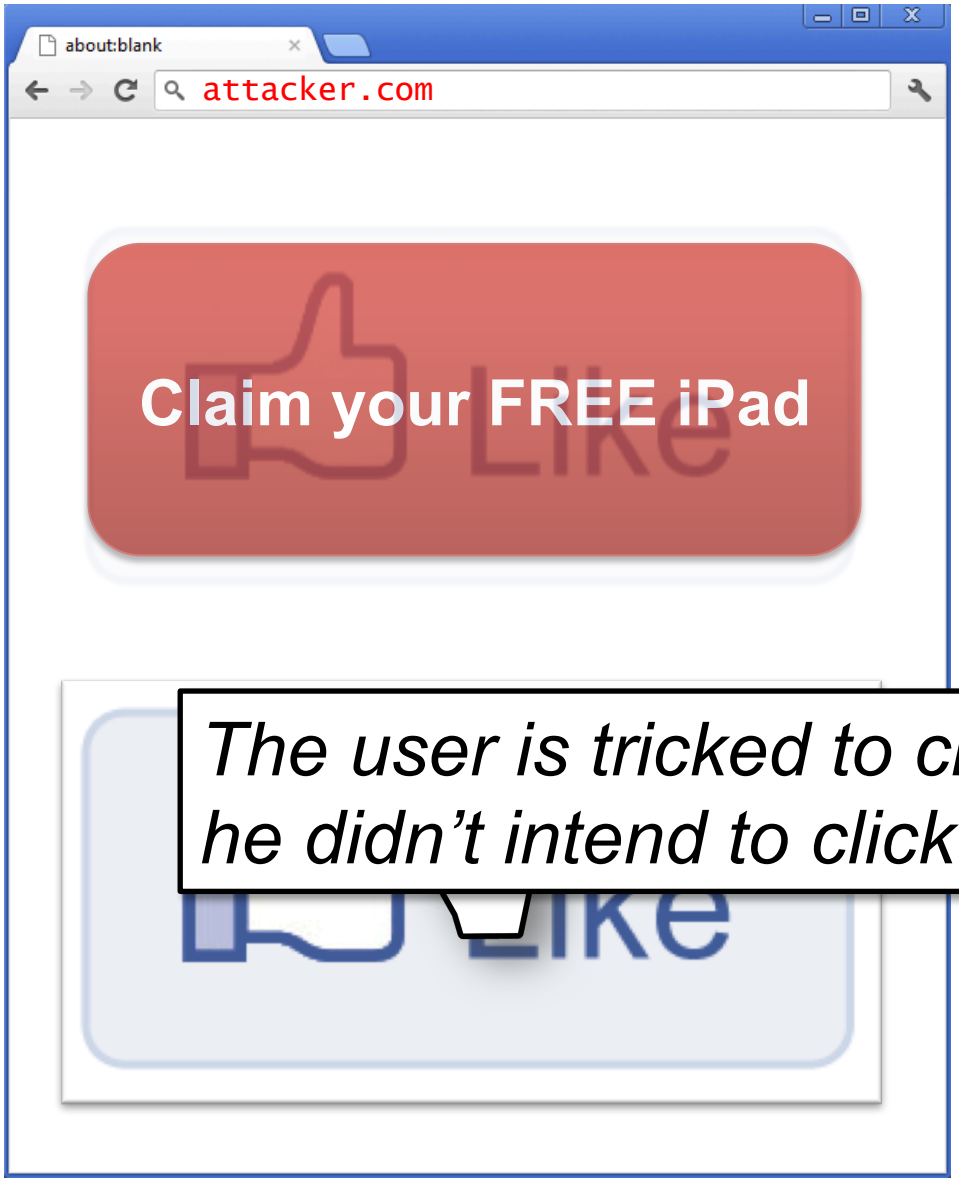
Clickjacking: Attacks and Defenses

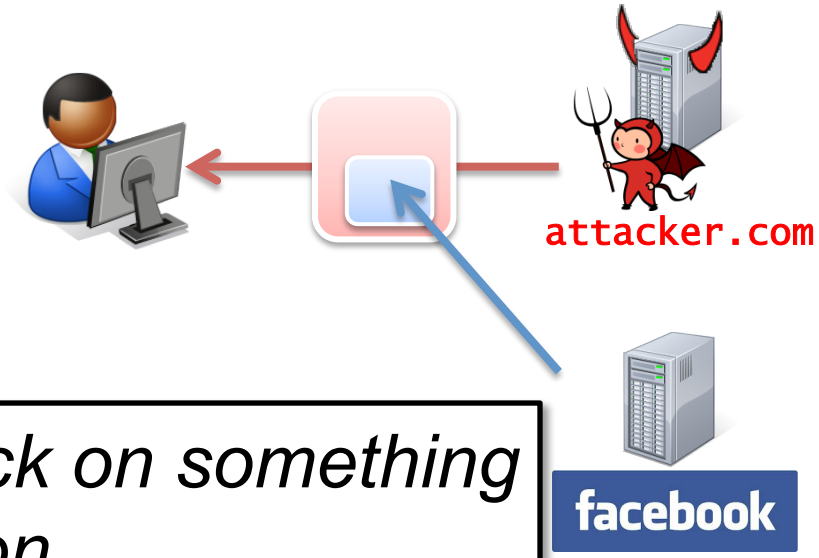
Lin-Shung Huang[1], Alex Moshchuk[2], Helen Wang[2], Stuart Schechter[2], and Collin Jackson[1]

[1]Carnegie Mellon University

[2]Microsoft Research

Example: Likejacking





The user is tricked to click on something he didn't intend to click on

Outline

- Defining clickjacking
- Existing defenses are insufficient
  - We evade them with three new attack variants
  - Our user study on Amazon Mechanical Turk shows that people fall for these attacks
- New defense to address root causes
  - Our user study demonstrates its effectiveness

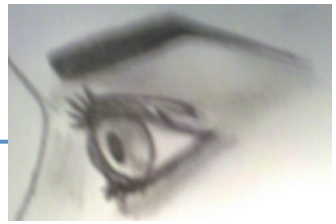
Defining clickjacking

- Prerequisite: multiple mutually distrusting applications sharing the same display
- An attack application compromises *context integrity* of another application's UI when the user acts on the UI

Visual integrity
Target Like is visible
Pointer is visible

Context integrity consists of visual integrity + temporal integrity

1. Target checked



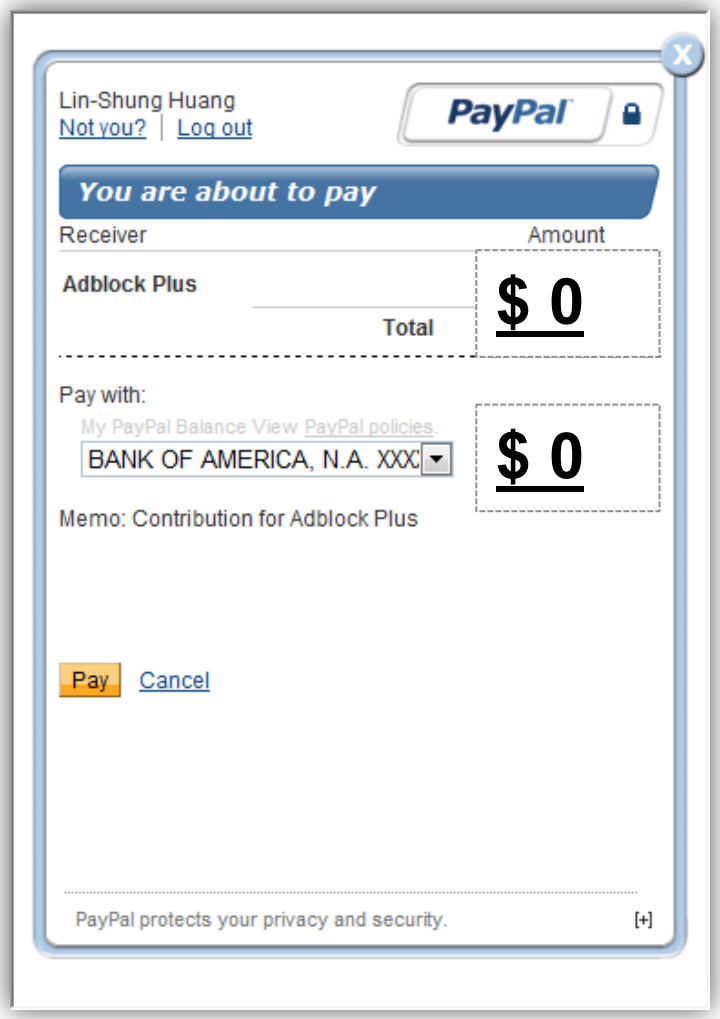
2. Initiate click

3. Target clicked

Temporal integrity
$\text{Target}_{\text{clicked}} = \text{Target}_{\text{checked}}$
$\text{Pointer}_{\text{clicked}} = \text{Pointer}_{\text{checked}}$

Compromise visual integrity – target

- Hiding the target
- Partial overlays



Compromise visual integrity – pointer

- Manipulating cursor feedback

Compromise temporal integrity

- Bait-and-switch

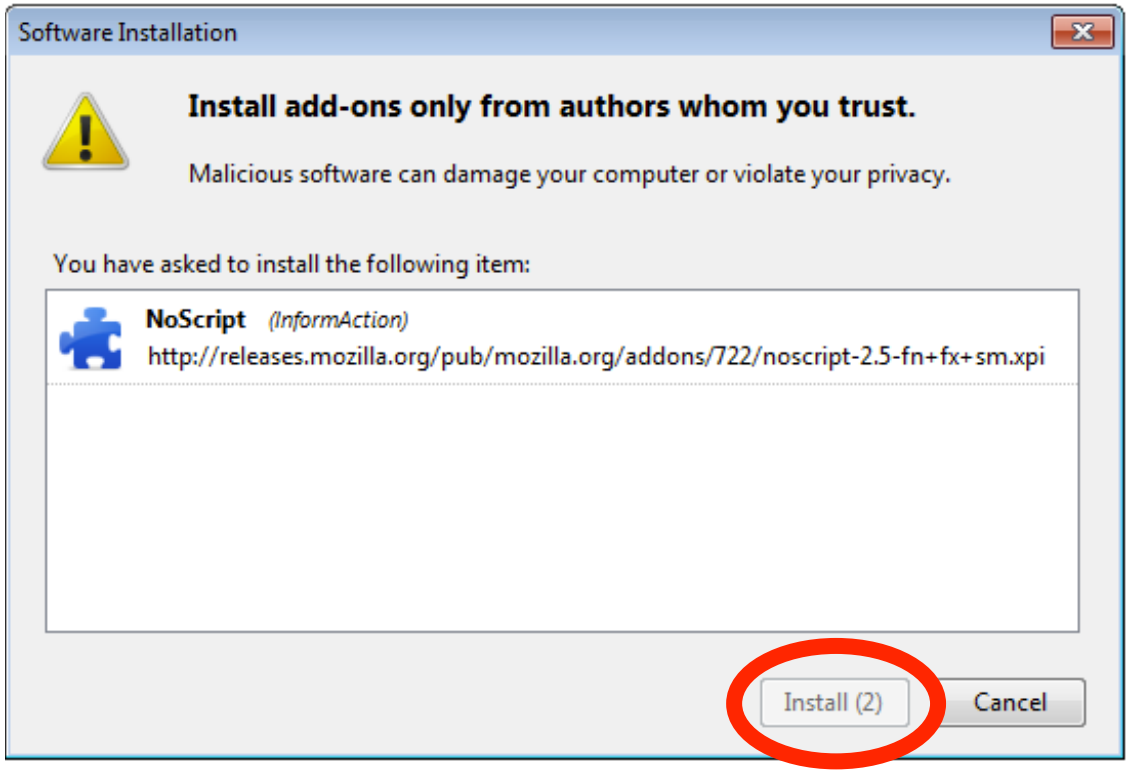
EXISTING DEFENSES

Existing defenses to protect visual integrity

- User confirmation
  - degrades user experience
- UI randomization
  - unreliable (e.g. multi-click attacks)
- Framebusting (`X-Frame-Options`)
  - incompatible with embedding 3rd-party objects
- Opaque overlay policy (Gazelle browser)
  - breaks legitimate sites
- Visibility detection on click (NoScript)
  - false positives

Protecting temporal integrity

- Imposing a delay after displaying UI
  - annoying to user



None of current defenses consider pointer

NEW ATTACK VARIANTS

1. Accessing user's webcam
2. Stealing user's email
3. Revealing user's identity

Evaluating attacks

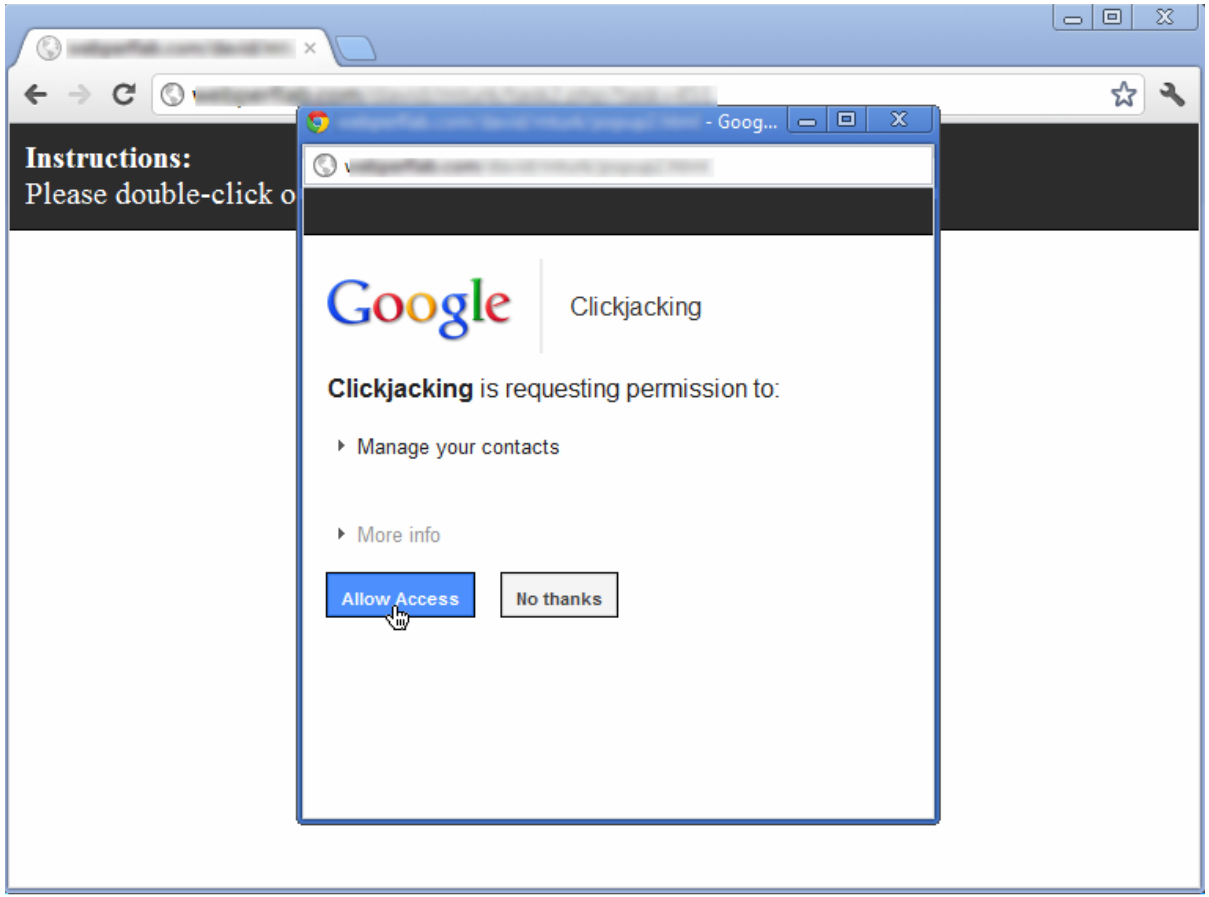
- 2064 Amazon Mechanical Turk web users
  - 25 cents per user
  - Users can only participate once, and only for one treatment

Attack #1: Accessing User's Webcam

Attack technique: cursor-spoofing

Attack success: 43% (31/72)

Attack #2: Stealing User's Emails

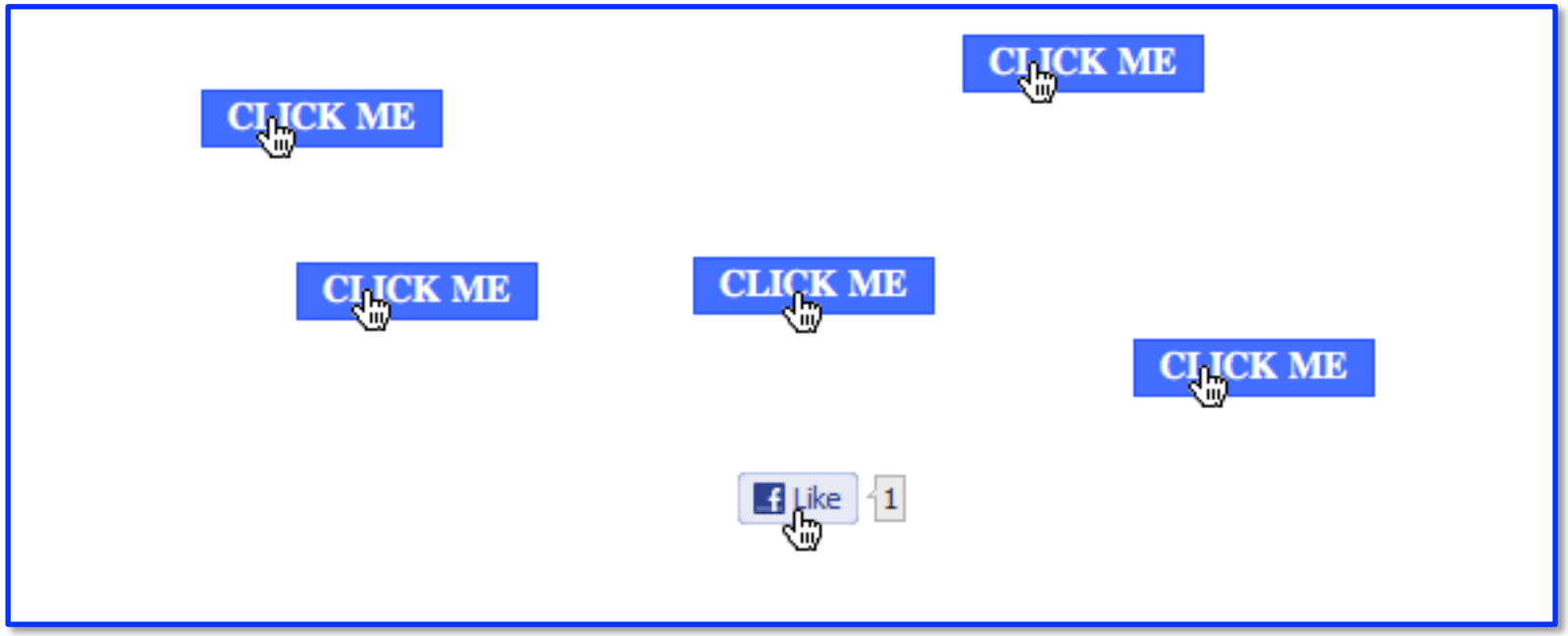


Attack technique: pop-up window

Attack success: 47% (43/90)

Attack #3: Revealing User's Identity

- Whack-a-mole game



Attack technique: cursor-spoofing + fast-paced clicking

Attack success: 98% (83/84)

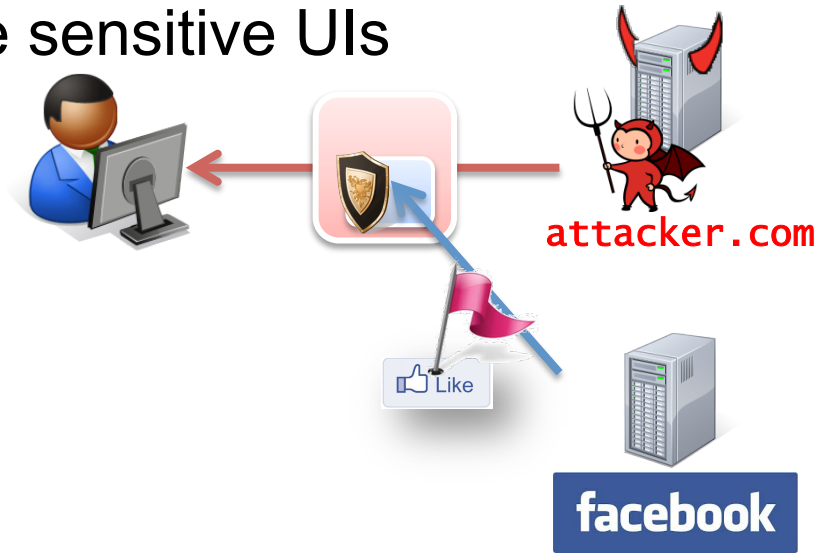
InContext Defense

Design Goals

- Should support embedding 3rd-party objects
- Should not prompt users for their actions
- Should not break existing sites
- Should be resilient to new attack vectors

InContext Defense

- A set of techniques to ensure *context integrity* for user actions
- Server opt-in approach
  - Let websites *indicate* their sensitive UIs
  - Let browsers *enforce* context integrity when users act on the sensitive UIs



Ensuring visual integrity of target

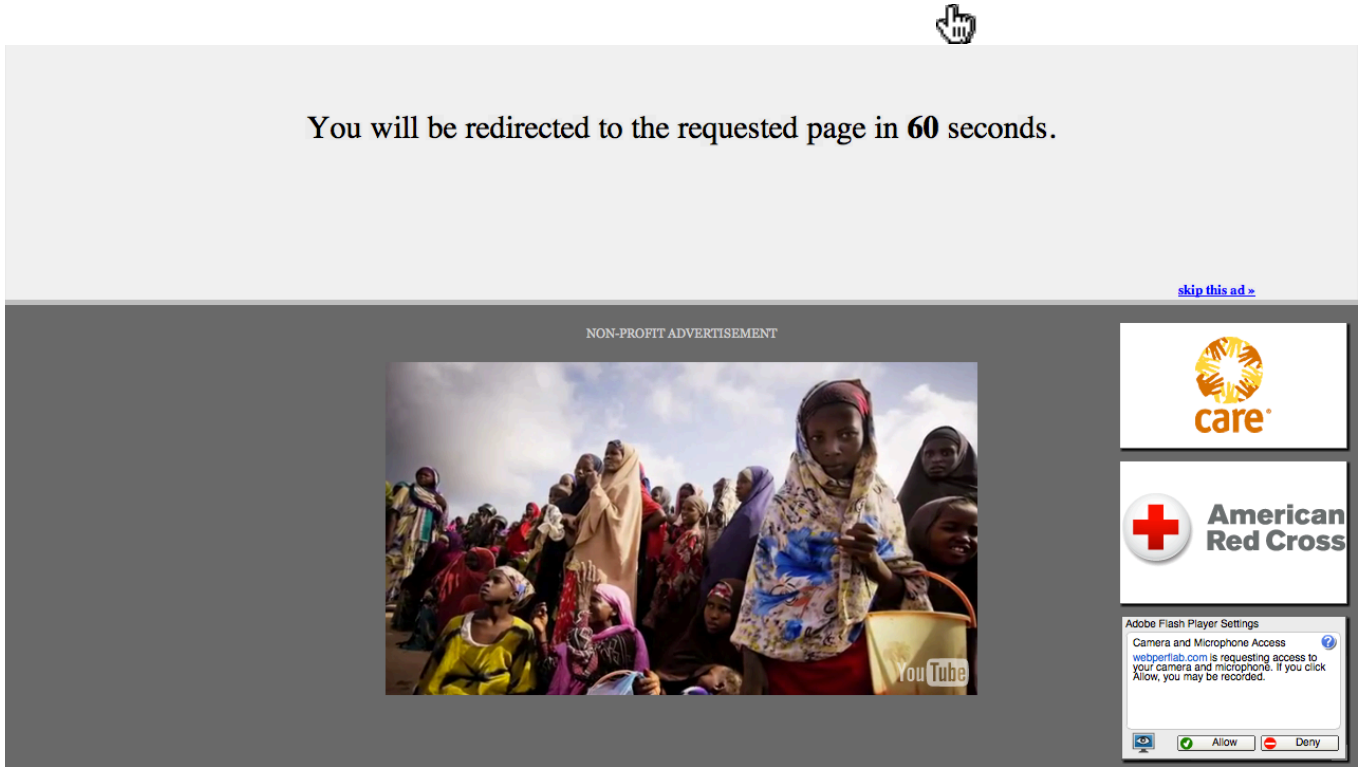
- Dynamic OS-level screenshot comparison
  - processing delay on click < 30ms (prototype on IE 9)

What is displayed
(OS screenshot)

What should be seen
(Reference bitmap)

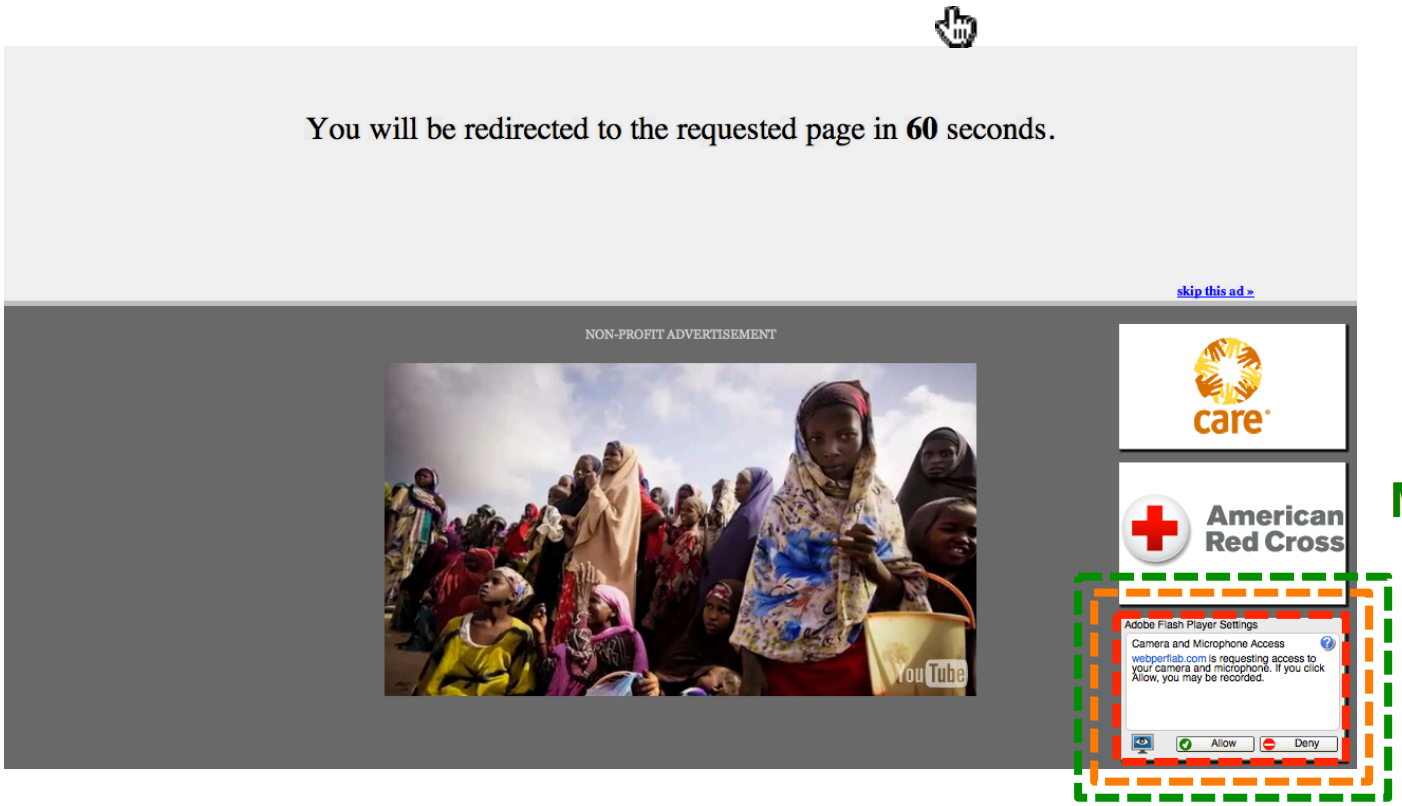
Ensuring visual integrity of pointer

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



Ensuring visual integrity of pointer

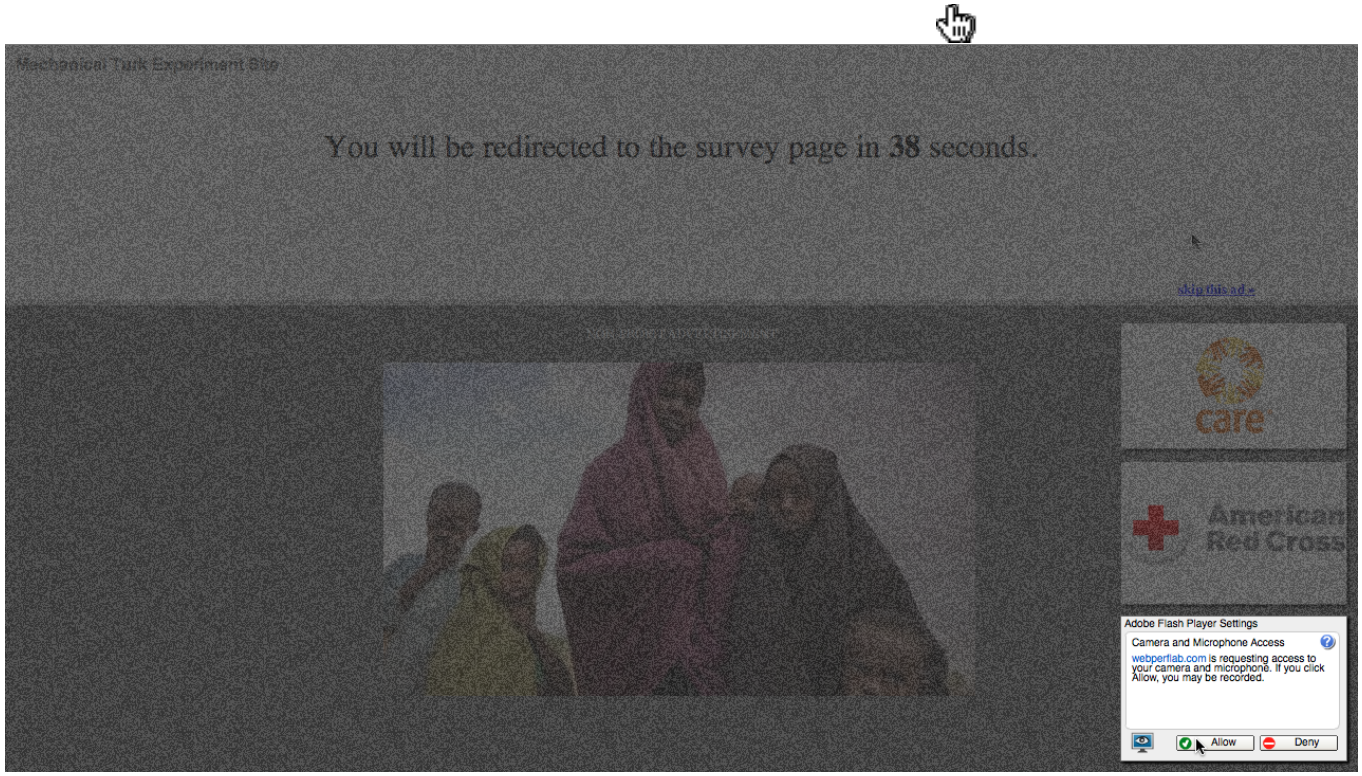
- Freeze screen around target on pointer entry
  - Attack success: 43% -> 15%
  - Attack success (margin=10px): 12%
  - Attack success (margin=20px): 4% (baseline:5%)



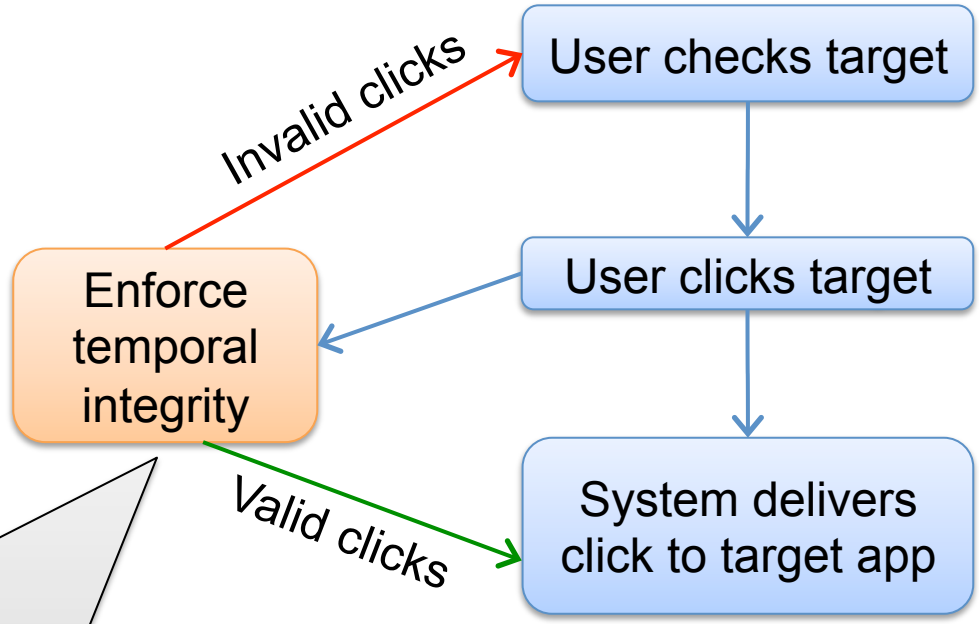
Margin=20px

Ensuring visual integrity of pointer

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



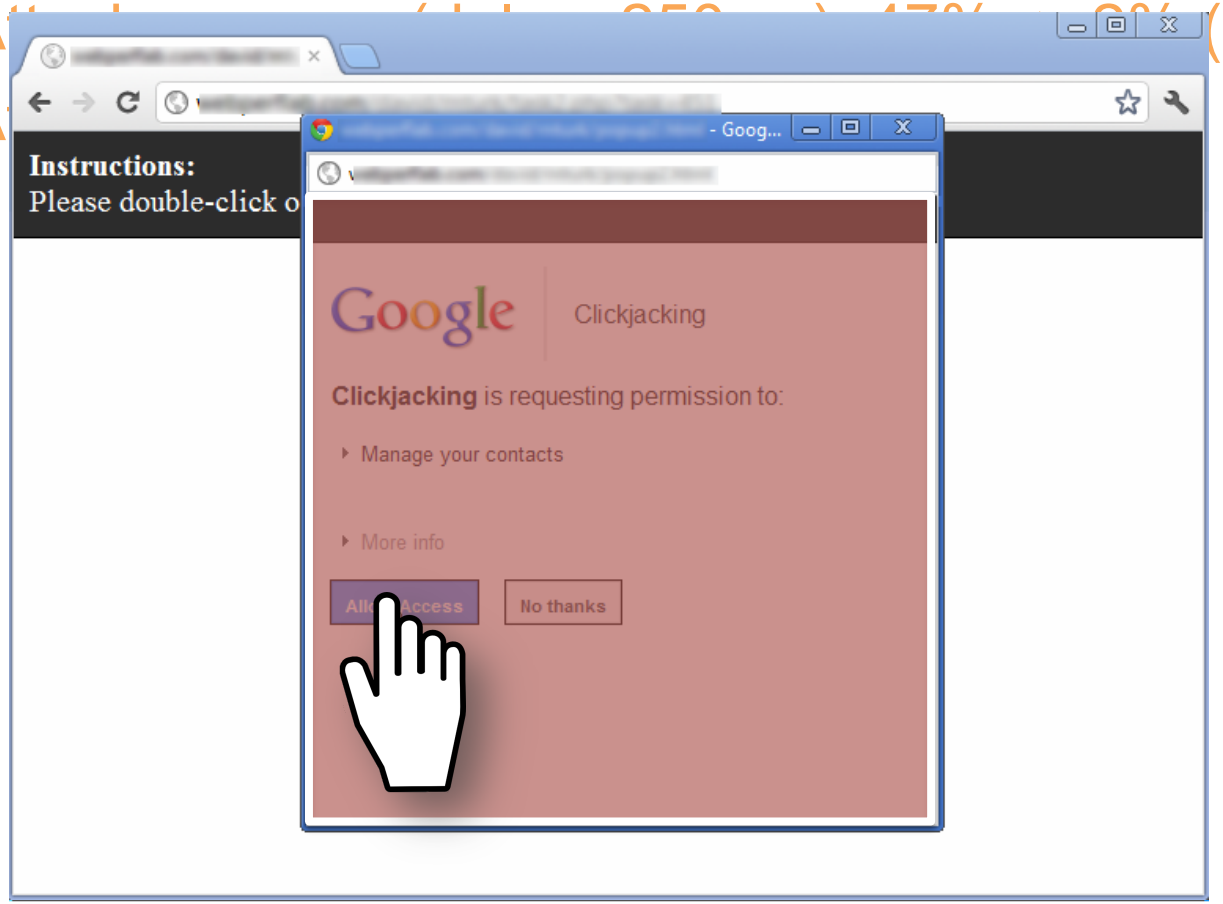
Enforcing temporal integrity



- **UI delay**: after visual changes on target or pointer, invalidate clicks for X ms
- **Pointer re-entry**: after visual changes on target, invalidate clicks until pointer re-enters target

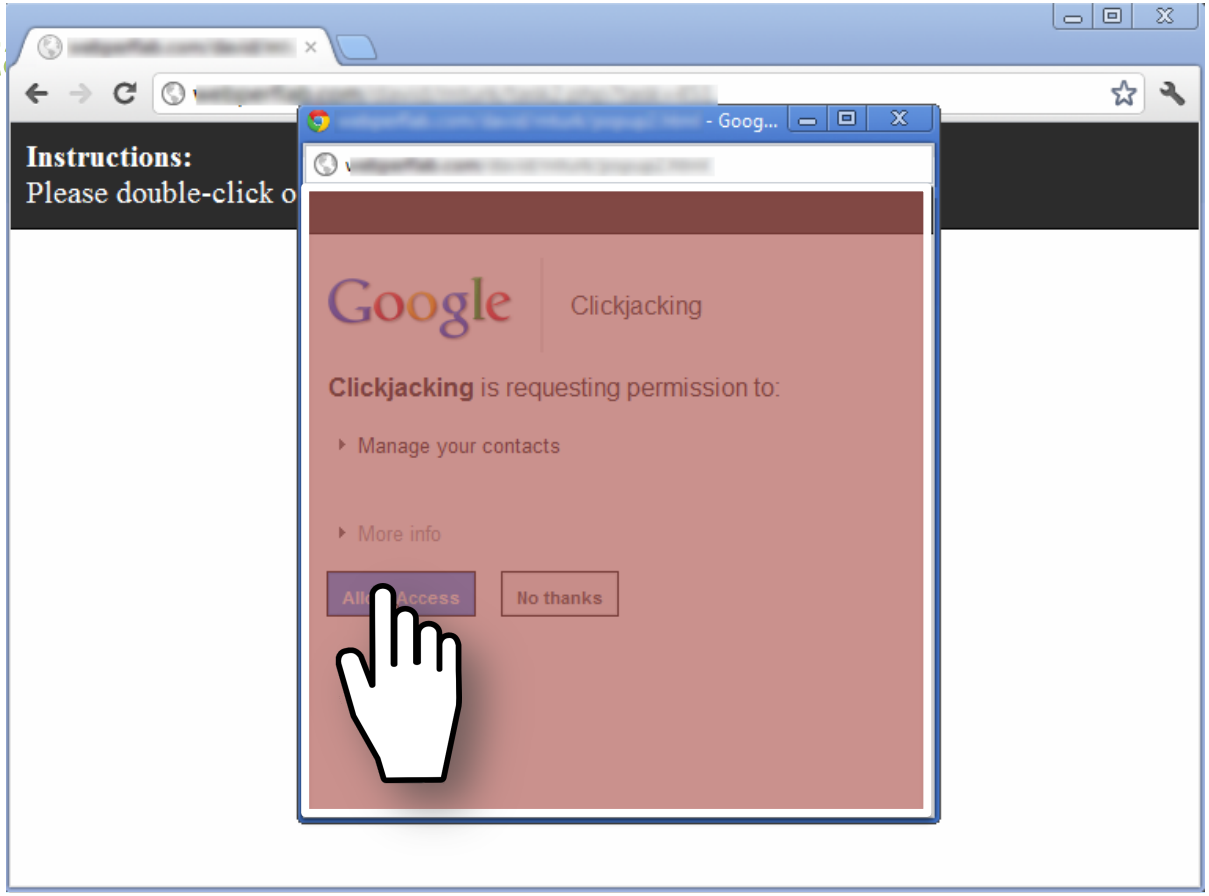
Enforcing temporal integrity

- UI delay: after visual changes on target or pointer, invalidate clicks for X ms
  - A (2/91)
  - A



Enforcing temporal integrity

- Pointer re-entry: after visual changes on target, invalidate clicks until pointer re-enters target
  - Att



Whack-a-mole attack

- Exclude victims who were moving their pointer around the Like button for many seconds, and deliberating whether or not to click
- Defense against clickjacking aspects
  - Screen freezing, margin=20px: 98% -> 16%
  - Screen freezing, margin=20px, pointer entry delay=500ms: 4%
  - Screen freezing, margin=20px, pointer entry delay=1000ms: 1%
- Social eng. aspects
  - 63% users intentionally clicked on Like button after our defenses made them fully aware of this

Conclusion

- We demonstrated new clickjacking variants that can evade current defenses
- Our user studies show that our attacks are highly effective (success rates 43% to 98%)
- Our InContext defense can be very effective against clickjacking
  - Ongoing efforts: UI Safety W3C proposal

QUESTIONS?

linshung.huang@sv.cmu.edu