Clickjacking Revisited
A Perceptual View of UI Security

Devdatta Akhawe, Warren He, Zhiwei Li, Reza Moazzezi, Dawn Song
UC Berkeley

Abstract

Clickjacking is a powerful attack against modern web applications. While browser primitives like X-Frame-Options provide a rigorous defense for simple applications, mashups such as social media widgets require secure user interaction while embedded in an untrusted webpage. Motivated by these application scenarios, the W3C UI safety specification proposes new browser primitives to provide a strong defense against clickjacking attacks on embedded widgets. We investigate whether these proposed primitives provide requisite security against clickjacking. We observe that UI security attacks such as clickjacking are fundamentally attacks on human perception. Revisiting clickjacking from a perceptual perspective, we develop five novel attacks that completely bypass the proposed UI safety specification. Our attacks are powerful with success rates ranging from 20% to 99%. However, they only scratch the surface of possible perceptual attacks on UI security. We discuss possible defenses against our perceptual attacks and find that most defenses either have an unacceptable usability cost or do not provide a comprehensive defense. Finally, we posit that a number of attacks are possible with a more comprehensive study of human perception.

1 Introduction

User Interface (UI) security attacks, or UI redressing attacks, including clickjacking [11, 14] and tapjacking [37], present a difficult problem for modern web applications. Such attacks are a threat whenever mutually distrusting principals (e.g., cross-origin iframes on the web) share the screen canvas or interface. Using the shared canvas, the attacker principal (i.e., malicious page) can trick the user into clicking on a third-party application frame (e.g., a Facebook Like button). For example, typical clickjacking attacks rely on making the third-party frame (containing the Like button) transparent so that the user does not realize she is clicking on it. These attacks are particularly difficult for the application to defend against, because they look like legitimate user actions to the application backend.

Current clickjacking defenses such as X-Frame-Options sacrifice functionality for security. Using X-Frame-Options, an application can request the browser never to load a particular page in a frame. This is not a practical solution for third-party mashup applications such as Facebook Like buttons. Instead, Facebook mitigates such attacks by requiring a second user click on a new window;[1] again, sacrificing functionality for security.

Motivated by the absence of defenses that preserve functionality, the W3C recently started work on a UI safety specification [25]. At its core, the specification relies on the InContext defense previously proposed by Huang et al. [14]. In the proposed specification, the browser permits cross-origin user interaction events (such as mouse clicks, touch events, or keyboard events) only if the event target (e.g., the Like button) was fully visible for a minimum amount of time. The aim of the proposal is to help applications defend against clickjacking attacks with minimal impact on functionality.

We investigate whether the proposed UI safety specification will protect applications against clickjacking attacks. Our key observation is that UI security attacks are fundamentally attacks on limitations of human perception. User interfaces strive for the integrity of user actions as their invariant. In other words, any user interaction with sensitive UI elements should succeed only if the user perceived, understood, and made a conscious decision to take the particular action. This process falls under the domain of human perception and action.

We evaluate the proposed W3C specification against a simple model of human perception and find a number of limitations. Requiring that a UI element remain visible for a certain duration is a necessary but not sufficient condition for perception. For example, the W3C specification

[1]This defense is only turned on if the Facebook Site Integrity team suspects fraud, presumably due to usability concerns.

does not adequately consider an attacker's ability to direct user attention. Instead, it implicitly assumes that users perceive any interface visible for a minimum length of time. In reality, a distracted user might not even notice the presence of a sensitive UI element.

We develop five novel attacks that bypass the proposed UI safety specification. Other than our first attack, all our proposed attacks also work on touch devices. We present our attacks and explain the underlying perceptual mechanisms in play. We also make all our attacks available online [6].

The attacks we develop rely on multiple, distinct limitations of human perception. Our intention is to demonstrate the breadth of attack possibilities when we take a perceptual view of UI security. For example, our attacks vary from destabilizing pointer perception to attacking peripheral vision to even attacking limitations of the human motor system.

To evaluate the practicality of these perceptual attacks, we conduct a study with 568 participants on Amazon Mechanical Turk. We find that our attacks have a success rate ranging from 20% to 99%. This is despite the fact that our attacks are prototypes and, as we discuss, a number of improvements can make our attacks far more powerful. Our experience developing these attacks also suggests that a number of attacks remain unexplored.

Contributions. We make the following contributions:

- We revisit UI security attacks as attacks on human perception.
- We present a simple model of human perception and find a number of limitations in the UI safety specification (Section 3).
- We systematically identify a number of novel UI security attacks that bypass the defenses proposed in the W3C UI safety specification (Section 4.
- We evaluate our attacks on 568 participants on Amazon Mechanical Turk and find that our attacks have success rates ranging from 20% to 99% (Section 5).

2 Background and Related Work

Ruderman first warned of UI redressing attacks as early as 2002 [31–33].[2] Zalewski [45] as well as Hansen and Grossman [10, 11] drew further attention to the problem by applying the attack to the Web, in an attack they call Clickjacking. UI attacks are especially powerful since the server sees the attacker's fraudulent messages as legitimate user actions.

[2]Note to reviewers: We gave a presentation on some preliminary results on these attacks at Blackhat USA 2013 and no paper was published. We believe it is acceptable to give presentations on work-in-progress, similar to giving talks at workshops.

In a typical clickjacking attack, the attacker convinces a victim to click a seemingly innocent UI element, such as a "skip ad" button. Unbeknownst to the user, the attacker overlays the button with a transparent `iframe` containing a sensitive button (such as a Facebook Like button or a PayPal "Pay" button). The web browser delivers any user clicks to the overlaid, transparent target.

Unknowingly interacting with a sensitive page causes unintended actions, such as undesired sharing of information [1, 9, 38, 42] or placing an order [14]. A recent study of clickjacking vulnerabilities on top web sites showed that three of the Alexa Top 10 web sites and 70% of the top banking sites have no countermeasures against clickjacking attacks [16, 44].

Firefox introduced time delays to installation of addons and activation newer features (e.g., geolocation) to protect against UI attacks. The browser disables the dialog buttons until the dialog is visible for a fixed amount of time. Other than in Firefox, time delays have not been adopted, possibly due to the adverse impact on usability. Browsers like Google Chrome instead switched to relying on two clicks to increase the difficulty of mounting a successful clickjacking attack [7].

Initial web application defenses against clickjacking included framebusting [36], in which a site uses JavaScript to prevent other pages from including it as a frame. Rydstedt et al. identified a number of bugs in popular framebusting code [36]. Browser vendors added support for HTTP header based mechanisms to prevent framing, including X-Frame-Options [19] and its successors [25].

Mechanisms such as X-Frame-Options and Framebusting protect only standalone applications and not mashups that rely on a shared canvas. For example, X-Frame-Options fails to enforce UI security for applications such as Facebook Like buttons that *need* to embed inside third party frames. Similarly, these defenses do not affect attacks that rely on cross-origin, overlapping windows [47, 48].

For scenarios *requiring* a shared canvas, applications can use two defenses which work on existing browsers: confirmation dialogs and UI randomization. In the former, the application asks the user for confirmation with a second dialog (a popup) before performing any sensitive action [24, 43]. This impacts usability and Facebook currently deploys this defense only if it already suspects fraud. Further, it is unclear whether the confirmation dialogs are also vulnerable to UI security attacks. Alternatively, applications can randomize the location of the sensitive buttons within their frame (as suggested by Hill [12]) to reduce the success rate of attacks and make large scale attacks easier to detect. We are not aware of any application relying on this defense. ClearClick, part of the NoScript Firefox extension, aims to protect its users from clickjacking attacks by comparing a screenshot of the target element rendered in isolation with a screenshot of the displayed page [24]. No browser deploys a ClearClick like defense by default, possibly due to false-positives [14].

To best of our knowledge, Zalewski was the first to investigate impact of human perception on UI security [47]. We share Zalewski's motivation but provide a more comprehensive investigation of perceptual attacks and defenses. Huang et al. presented a comprehensive study of clickjacking attacks and defenses [14]. After identifying a number of novel attacks, Huang et al. present InContext, a model and a system to formalize the notion of UI attacks in terms of "contextual integrity" of the user interface. It relies on an invariant-based design requiring distinct preconditions before allowing a user action.

InContext provides a defense against nearly all the attacks previously discussed in the literature, including all the attacks mentioned in the W3C's list of clickjacking threats [39]. Since the W3C UI safety specification adopted the core ideas of InContext, we discuss them together below. Huang et al. also suggested defenses that they did not implement such as a "lightbox" around sensitive UI elements. The W3C has not adopted the lightbox defense either, possibly because it is not applicable to touch devices.

2.1 The UI Safety Specification

The W3C UI Safety specification aims to provide a comprehensive defense against UI redressing attacks such as clickjacking. First, it subsumes the X-Frame-Options header into a `frame-option` Content Security Policy (CSP) directive. As we discussed, this does not provide protection to mashup applications such as Facebook Like buttons and PayPal "Pay" buttons.

For mashup applications, the specification defines a new `input-protection` directive in the Content Security Policy header of a page. The application defines a secure UI element via CSS selectors (with the `input-protection-selectors` directive) or a secure area with the `input-protection-clip` directive. These input-protection directives provide protection similar to the InContext defense, which Huang et al. describe as three components of "contextual integrity": display integrity, temporal integrity, and pointer integrity. Briefly, the browser needs to ensure that a secure UI element/area be fully visible (display integrity) for a sufficient amount of time (temporal integrity) before delivering the input event (e.g., mouse click).

Display Integrity. Display integrity requires that a sensitive HTML element (such as the `div` defining a Facebook button) appear fully visible for a secure click. For example, if the button is part of a transparent iframe, then it violates display integrity and the browser should not deliver any clicks to the application. Another attack that violates display integrity includes an attacker that covers the sensitive button with a "clickthroughable" image [2]. Other variations include overlaying only part of the sensitive target [15,40,46].

The UI safety specification requires the browser to take two screenshots: one only of the protected frame as if viewed alone and one of what the user sees. If in the two screenshots, the secure element (defined via the `input-protection-selectors` or `input-protection-clip` directives) does not look the same (i.e., it is not fully visible in the second screenshot) the browser should not deliver the input (mouse/keyboard) event. The application can specify a `tolerance` value to allow minor obstructions for usability.

Temporal Integrity. Temporal integrity requires that the sensitive UI element be visible on screen for a minimum amount of time before a user can interact with it. Enforcing temporal integrity presumably allows a user sufficient time to perceive visual changes. In the absence of temporal integrity, an attacker can make the sensitive UI button transiently visible just before the user clicks on it [13,14,34,35,47,49].

The UI safety specification requires the browser to check for repaint events for the secure UI element within a time period specified by the application using the `display-time` directive.

Pointer Integrity. Another class of attacks occurs due to the attacker tricking the user's perception of the pointer [3, 18, 28]. Pointer integrity requires that all pointer feedback be visible and authentic to the user. By hiding the real pointer and rendering a fake pointer vertically displaced from the correct location, the attacker can trick the user into clicking on a sensitive button unintentionally. The user does not see the hidden real pointer, so he follows the fake one. For touch devices, pointer integrity is not required since the user's finger serves as the pointing element. Huang et al. also did not implement any defense or detection mechanism for pointer integrity as part of their browser implementation. The UI Safety specification also only mentions that the browser should check for hidden cursors before delivering the event. In all our attacks, the cursor is fully visible before the click.

3 Perception and UI Safety

In this section, we present a simple model of human perception and revisit the UI safety specification with a perceptual view. In our model, human perception and interaction is a three-step process: humans give attention to a visual stimulus, come to understand what it means, and then take an action. We do not claim that our model is complete in any way—human perception is a deep and complex topic, studied in the areas of neuroscience, psychophysics, and human–computer interaction. Our model of human perception, in comparison, is simple (crude, almost), but suffices for our needs.

3.1 Attention

Attention enables us to zoom in on a specific part of the visual scene [27] and increases the signal to noise ratio of the cortical channels that transmit and process that information [26].

The W3C UI Safety specification does not sufficiently consider the importance of human attention.[3] Invariants such as display and temporal integrity ensure that the sensitive UI element is visible for some duration, but this requirement is moot if the user's attention is on a completely different area of the screen. In such cases, the user does not notice the sensitive UI element at all.

We make this idea concrete with our "Peripheral Vision" attack (Section 4.2): we trick the user into focusing on one area of the screen while interacting with another. In fact, directing user attention is a critical step in all our attacks on user interfaces (Section 4). For example, in our "Destabilizing Pointer Perception" attack (Section 4.1), we rely on a number of distractors to reduce the likelihood of the user noticing a possible attack. We discuss these along with our attacks.

3.2 Visual Perception and Illusions

In our model, once a user places attention on a particular area of the visual scene, she proceeds to process the image of that area in detail. This provides another example of the disconnect between human perception and current defenses: current defenses assume that, if the sensitive element was visible to the user, the user also correctly perceived it. Unfortunately, visibility does not always equate to the user correctly processing the scene.

For example, a visual illusion could cause a mismatch between the user's understanding of the scene and the UI drawn by the browser. Our "Fast Motion Mislocalization" attack (Section 4.4) relies on the flash-lag illusion to trick the user into unintentionally clicking on a sensitive UI element. A wide literature exists on human illusions, providing a rich source of possible UI security attacks.

3.3 Taking Action: The Motor System

Once the user perceives a particular UI/visual stimulus, she proceeds to interact with it via moving the mouse and possibly clicking on it. The motor system is the part of the central nervous system responsible for any actions/movement. Current defenses assume a synchronous perception and motor system: if a user performs an action, she must have processed the visual scene before the action (the check-and-use paradigm).

[3]InContext [14] proposes a lightbox effect to grab attention, but only for pointer integrity. We demonstrate that attention is necessary not just for pointer integrity.

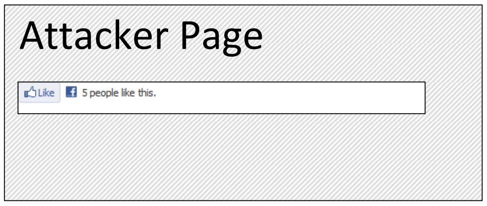

Figure 1: Illustration of our threat model. The attacker controlled page frames a page with a sensitive action, in this case, a Facebook Like button. The attacker has full control over the page (marked with grey background) except for the Facebook Like button frame (clear background), which must be fully visible for at least a second.

In reality, our motor system is often asynchronous—we can take an action and process the visual scene *after* the action. Our "Motor Adaptation" attack (Section 4.3) trains the user's motor system to perform a series of actions asynchronously and then sets up an attack to take place during the actions but before the visual processing.

We stress that this asynchronous behavior goes beyond the time-of-check to time-of-use vulnerability discussed in the literature [14] as well as in the W3C specifications. Previous work focused on the time window between the user perceiving a visual scene and the motor system taking an action, pointing out that an attacker could potentially switch the visual scene in this small window. Our attacks do not involve such a bait-and-switch; instead, our attack trains the user to perform an action much further ahead in the future and rely on the inability of the motor system to easily cancel commands once given.

4 Perceptual UI Attacks

The discussion above suggests that the contextual integrity conditions, while necessary, are not sufficient for ensuring UI security. In this section, we systematically investigate different, orthogonal dimensions of human perception and identify new UI attacks that bypass current defenses.

We present five new attacks that do not violate contextual integrity, while still tricking the user into unintentionally clicking on a Facebook Like button. We also discuss the perceptual model we use to develop and refine each attack. Our explanations are not comprehensive; instead, they aim to provide a simple model to help understand our attacks.

Threat Model. Figure 1 illustrates our threat model. We focus on web applications that need to share part of the screen real estate with a third party, possibly untrusted application. For our attacks, we use the example of Facebook Like buttons, but the same concept applies for other sensitive targets that *need* to be embedded in third party pages (e.g., Twitter Follow buttons). In our threat model, the whole Like button frame must remain fully visible for

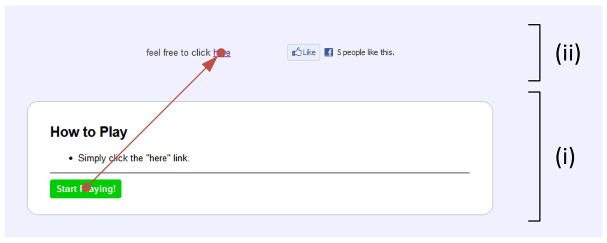

Figure 2: Workflow for our pointer destabilization attack. At first, (i) displays the instructions, and (ii) is hidden. When the player clicks the green button in the lower left, (i) disappears and (ii) appears. This ensures that the cursor approaches from a controlled direction (superimposed as the red arrow).

at least a second before a jacked click. We believe that this is a stronger model than what applications will adopt. For example, Facebook is likely to turn on UI safety only for the Like button and not the whole frame.

We do not consider applications that use X-Frame-Options to disable embedding. We believe these applications could also be vulnerable to related attacks via windows and popups, although, the window decorations around such popups will likely reduce the success rate.

Our investigation in this section is nascent. The attacks we present are only prototypes. A real attacker can likely create powerful, convincing attacks. Further, the large number of effects observed by studies of human perception suggests that other attacks are possible. Our aim is to focus on the breadth of possible attacks and we investigate limitations of human perception along orthogonal dimensions. For instance, we investigate limitations of the motor system in Section 4.3 and limitations of the visual system in Section 4.2.

4.1 Destabilizing Pointer Perception

The key idea of this attack is destabilizing the user's perception of the pointer by transiently showing an image of a mouse-pointer moving in a different direction from the real cursor. Our attack qualitatively differs from the fake pointer attacks discussed by Huang et al. [14] and the similar phantom cursor attacks discussed in the W3C wiki [39]. Previous work focused on a fake (or phantom) pointer that followed the hardware pointer but at a constant displacement.

Instead, our attack creates a pointer that moves in a direction different for a transient duration. This momentarily confuses the user who makes a sudden correction; our attack relies on this sudden correction. We present it as an example of how a better understanding of human perception can help guide the design of strong UI attacks.

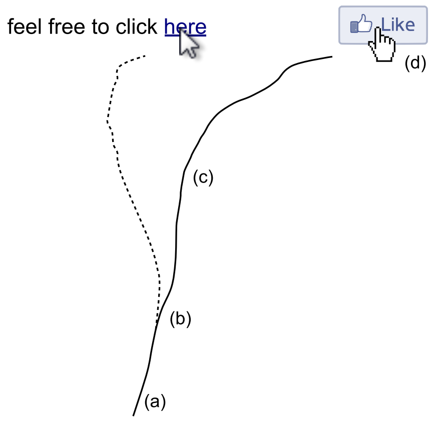

Figure 3: Pointer movement in our pointer destabilization attack. The solid line depicts the path of the hardware cursor, while the dashed line depicts the path of the fake cursor. Section 4.1.1 provides details about each labeled point.

4.1.1 Attack Setup

Figure 2 depicts the attack setup. The attacker positions the user's pointer at the bottom left of the screen via a green 'Start Playing' button. Once the user starts playing, the attacker requests that the user click on a link ("click here" in Figure 3). The previous click ensures that the user's mouse starts its movement toward the specified link from the bottom left of the screen. When the pointer (traced with a solid line in Figure 3) is close to the web link (starting at point (a) in Figure 3), the attacker hides it and shows a fake image of a pointer. This image only remains visible for a short time, while the user closes the final distance to the link. This fake pointer image moves with the hidden real pointer, but in a different direction (path of fake pointer traced with dotted line in Figure 3, starting at point (b)).

The user's natural reaction is to correct for this leftward error, and the user moves the mouse to the right in a sudden movement (point (c) in Figure 3). This moves the hardware pointer over the Like button, where the real pointer reappears. Since our attack page does not receive any pointer movement events after this, we extrapolate the fake pointer's last few pixels of movement. Since the user's original intent was clicking on the link, continuing that task results in a mistaken click on the Like button (point (d) in Figure 3).

4.1.2 Understanding the Attack

The attacker-controlled fake pointer image appears near the link target ("click here") and smoothly replaces the original pointer. This makes it difficult for the user to realize that a replacement pointer exists. Seeing the (replaced) pointer not following the intended path, the user naturally tries correcting the pointer and ends up taking the real pointer over the Like button.

One concern to a successful attack is the appearance of the real pointer when it reaches the like button. A user may notice the appearance of the real pointer and abort. To improve chances of success, our attack relies on a number of small moving objects near the Like button to steal user attention away from the appearance of the real pointer. We discuss these in more detail since such attention grabbing techniques are useful in a wide number of UI attacks.

Filtering. Suppose that when the user starts her task, a moving background near the like button catches her attention but carries no relevant information. Quickly, the brain primes itself to ignore the area around the Like button as a source of unnecessary distractions, a phenomenon known as filtering [8]. Filtering is critical for conserving human attention. Thus, when the real pointer suddenly appears on top of the like button, the user does not notice it.

Crowding. Background motion additionally helps reduce the chances of the user *identifying* the real pointer, despite detecting it, due to a phenomenon referred to as "crowding" [20,30]. (This is in contrast to filtering, which reduces the chances of the user *detecting* the real pointer.) In crowding, stimuli surrounding a target make the target stimulus less visible. Crowding is most effective in peripheral vision.

Summary. Adding background motion provides two advantages to the attacker, filtering and crowding. Filtering reduces the chances of the user detecting the real pointer on the Like button. Even if the cortex detects the appearance of the real pointer, crowding causes the cortex to fail to identify it as such. Instead, the cortex interprets the real pointer as another moving, distracting stimulus.

4.2 Attacking Peripheral Vision

Our second attack targets the low resolution of peripheral vision. The key idea of the attack is to convince the user to focus on one area of the screen, while interacting with another. This attack is a simple demonstration of the ability of an attacker to direct user attention for clickjacking attacks.

4.2.1 Game Setup

Figure 4 depicts a frame in our attack and Figure 5 illustrates the attack in a timeline. To play this game, a player must leave her mouse in a designated sensor area in the bottom left corner of the screen (Figure 4) and watch a main game area in the top right corner of the screen. A series of blocks drops through the instruction area towards two holes (Figure 5 (a)). The player must click the left or right mouse button in the sensor area as the blocks reach the left or right hole.

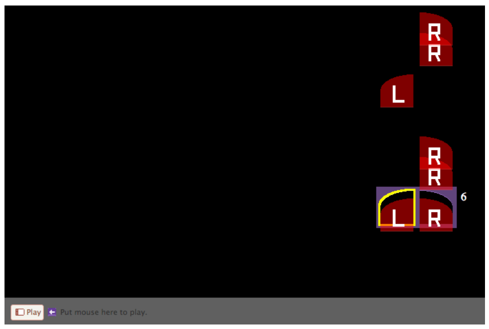

Figure 4: A screenshot of our peripheral vision attack game.

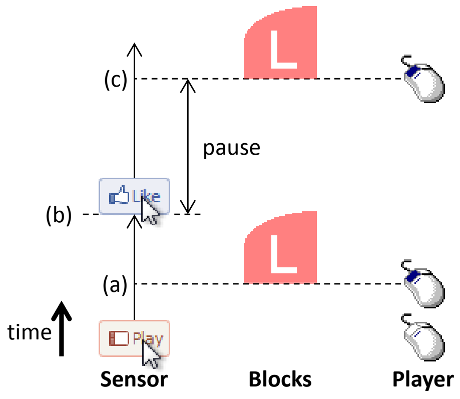

Figure 5: Timeline of the peripheral vision attack game. In normal operation (a), the player interacts with a click sensor. For the attack, the page replaces the sensor with a Like button at time (b) leading up to a left-click instruction (c).

This game trains the user to focus on one corner of the screen, while clicking in another. After some time playing the game, the attacker replaces the sensor with a Like button (Figure 5 (b)). This replacement takes place during a pause in the block sequence followed by a block heading to the left window (Figure 5 (c)). The user naturally clicks the left mouse button.

4.2.2 Understanding the Attack

Peripheral vision refers to the part of the field of view that occurs outside the center of gaze, particularly at the edges of the human field of view. Spatial resolution is low in peripheral vision, and we present an attack utilizing this limitation. The key to our attack is training the user to carry out actions in her peripheral vision, while relying on the stimulus in the center of vision.

The pause after the replacement allows the attacker to maintain the UI safety invariants. Since the user focuses her gaze on the main game area, she does not notice the replacement in the sensor area and clicks on it when she

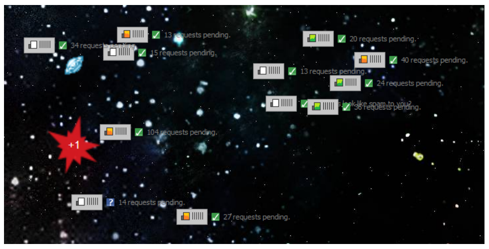

Figure 6: A screenshot of our adaptation attack game. We include additional distractors (discussed in Section 4.1.2) to make the attack hard to notice.

sees the next block dropping to the left hole.

4.3 Motor Adaptation

Next, we discuss attacks that rely on motor system limitations. We focus on attacks made possible due to the phenomenon of motor adaptation. Adaptation, in general, refers to our sensor and motor systems optimizing for repeated stimuli and actions. Motor adaptation trains the user to perform a sequence of repetitive actions on cue, which makes us vulnerable to clickjacking attacks (e.g., the click-move-click pattern we explain below).

4.3.1 Game Setup

We create a game that adapts players to a repeated stimulus and trains them to perform a sequence of actions. Figure 6 depicts a screenshot of our game. When the game starts, we present the player with an asteroid, which the player can destroy for one point by clicking it. Clicking on the asteroid also creates a mineral object at a constant displacement above the former location of the asteroid. This mineral automatically disappears after a short time. If the player clicks the mineral before it disappears, she gets three additional points.

We make the lifetime of the mineral short, so that it will disappear if the player waits for multiple round trips between the sensor and motor systems. A short time after the mineral disappears, another asteroid appears in a random location.

The game continues this way for the duration of the experiment. As the player adapts to this repeated asteroid-mineral stimulus, it becomes much easier for her to execute a practiced click-move-click sequence of actions to achieve success. As the player slowly builds up her ability with the click-move-click sequence, the game increases in difficulty by gradually reducing the time for which each mineral stays visible.

4.3.2 Understanding the Attack

Every movement we make corresponds to a population of neurons in our motor system. Activation of such a population causes us to execute the corresponding movement. In motor adaptation, the populations that execute the different actions in the sequence get connected through a chain. Neurons controlling individual, sequential movements not only encode a particular movement, but also the next movement in the sequence [17, 22].

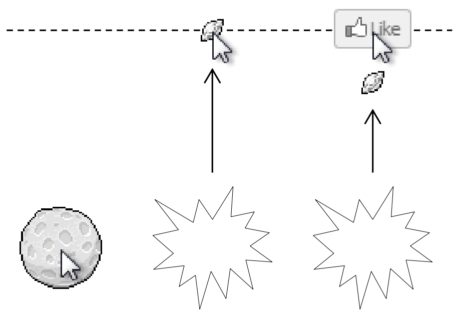

Figure 7: Stages of the adaptation game. **Left:** player clicks on asteroid. **Middle:** asteroid explodes and a mineral appears at a constant displacement above. **Right:** (attack) Like button appears at usual displacement; mineral appears unusually low.

This is a form of redundant coding in our brain: the earlier commands (click) also encode information about later commands (move-click). This form of encoding makes our command system more robust against noise. On the other side, this makes canceling mid-way through a sequence of actions difficult, since the command for the first action carries information for the next action.

In our game setup, it is difficult, if not impossible, for a player to click the asteroid, notice the mineral, point to the mineral, and click again as separate steps. After a couple trials of performing the click-move-click sequence, the brain starts to adapt a new chain to encode this sequence.

Our game design aims to ease such an adaptation. For example, the mineral always appears at a fixed displacement from the asteroid. Given the time delay between the brain issuing the commands and our motor system executing it, our brain creates a chain such that the entire sequence executes before the second target disappears.

4.3.3 Attack

As the player adapts to this repeated asteroid-mineral stimulus, it becomes much easier for her to execute a practiced click-move-click sequence of actions to achieve success. At this point, the game executes the attack, which is a variation on the normal behavior. The game places a special asteroid under a Facebook Like button, such that the normal mineral displacement would place the mineral where the Like button is.

When the player clicks this special asteroid, it creates a mineral at a smaller displacement (i.e., positioning it between the asteroid and the Like button, illustrated on the right of Figure 7). In a successful attack, the player executes the click-move-click sequence starting with this

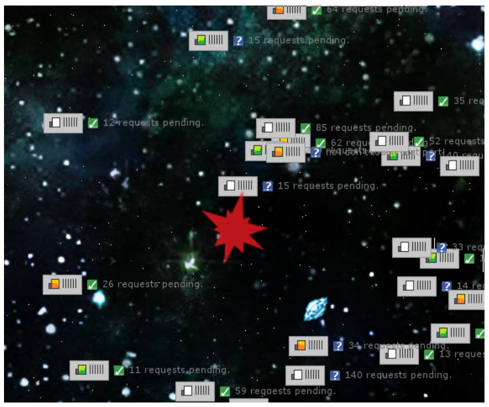

Figure 8: A screenshot of our flash-lag attack game. As before, we also include additional distractors on the page to reduce the likelihood of the user noticing the attack.

special asteroid. The movement places the cursor on the Like button, and the player clicks the Like button with the second click.

Our attack exploits the fact that the human brain adapts to repeated actions, as discussed above, and there is a known lag between the brain sending a command and our motor system carrying it out. The success of the attack requires that the command sequence ("click-move-click") leave the brain before the player notices the Like button. On noticing the Like button, the brain sends a "stop" command to our motor system, but it often cannot stop the sequence in time.

The UI safety specification cannot protect against this attack, since we make the sensitive target visible for a while. When the Like button appears in advance for the attack, the user is focused on playing the game in another part of the screen. When the user's attention does switch towards the sensitive click target, the motor system adaptation gives the user no time to correct.

4.4 Fast Motion Mislocalization

Human perception inherently includes a model of inertia. For example, flash lag is a visual illusion in which a moving object appears displaced from its true location, as we perceive it will move further [23]. The key idea of our attack is instructing the user to click a moving object; due to the flash lag illusion, the player can overshoot the moving object and instead click on a Like button beyond the object.

4.4.1 Game Setup

This game starts with an asteroid in the center of the play area. An arrow near the asteroid spins around and stops in a random orientation. When the arrow stops, the asteroid explodes in a flash and a fast-moving mineral shoots out of the asteroid in the direction of the arrow, travels for a fixed distance, and disappears shortly after reaching its destination.

The player must click on the mineral at its destination to score points. After the mineral disappears (either shortly after stopping or when the player clicks it), a new asteroid appears, and the arrow starts spinning around it again. This process repeats itself for the duration of the experiment.

The key to our attack is the user over-shooting the fast moving mineral due to the flash-lag illusion. Our game adjusts itself to the player's skill level. The speed of the mineral adjusts itself based on the results of previous trials: it speeds up every time the player successfully clicks it, and it slows down if it disappears without the player clicking it.

4.4.2 Understanding the Attack

A simple explanation of motion mislocalization is that processing of the motion signal in our brain takes time. Such a delay would result in the brain seeing the objects lag behind their true location: the object had already moved by the time the brain processed its current location.

Since humans obviously do not face this issue, Nijhawan [29] suggested that the brain compensates for the delay by extrapolation: the brain reconstructs the location of moving objects by representing them ahead of the location measured by cortical processing.

One issue with our game setup is the interplay of the motor and visual system. Flash-lag is a *visual illusion*; experiments investigating flash-lag asked the user to recall visually the position of the moving object (see [5] for an online demo).

Our game involves tricking the user's motor system to take an *action*. Visual illusions do not always trick our motor system. To mitigate this concern, our attack relies on fast clicking tasks where the brain generates motor commands purely based on inaccurate visual information. Once the commands are generated, the motor system does not have any chance to correct them based on visual feedback.

4.4.3 Attack

Due to the flash lag illusion, the player tends to overshoot the mineral and click ahead of it. We create an attack out of this by placing a Like button just beyond the point where the mineral would disappear (as illustrated on the bottom of Figure 9), so that the overshoot results in the player clicking the sensitive target.

Our attack does not violate any of the integrity constraints from the UI safety specification. The Like button is visible for a noticeable amount of time, but the user is focused on the game and does not notice the appearance of the Like button. The attack is successful when the user wants to click on the mineral, but overshoots due to the flash-lag illusion.

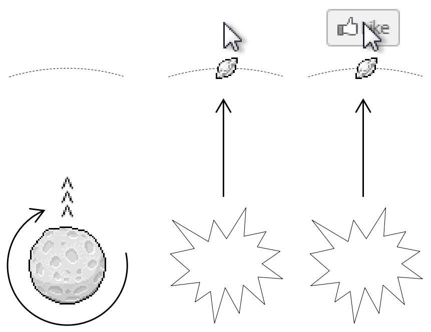

Figure 9: Stages of the flash-lag game. **Left:** arrow spins around asteroid. **Middle:** asteroid explodes and a mineral shoots out to a constant distance in the direction of the arrow; the player tends to overshoot this mineral. **Right:** (attack) Like button appears just beyond mineral destination; mineral moves as usual.

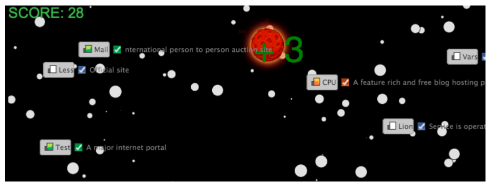

Figure 10: A screenshot of our click prediction attack game.

4.5 Visual Cues and Click Timing

Another possible attack is to set up a visual cue to control the timing of a user click. Combined with an appropriate position of the mouse, this cue allows an attacker to trick the user into clicking the target (Like button).

4.5.1 Game Setup

Figure 10 depicts our game setup. An asteroid moves randomly around the play area and flashes red on a regular interval. The game instructs the player to click on the asteroid while it is red. The game's scoring rewards clicking on the asteroid while it is red. This setup aims to adapt the player to click the asteroid as soon as the color of the asteroid changes.

Attack. After some time spent training the user, the attacker moves the moving object across the like button. To ensure visual integrity, the asteroid moves under the Like button. The attacker then changes the color of the asteroid while it moves across and under the Like button. The user's brain, adapted to clicking on the color change, clicks on the Like button by mistake.

4.5.2 Understanding the Attack

A successful attack is complicated by the presence of the Like button, which acts as an inhibitory signal—on noticing it, the user tries to cancel the click command

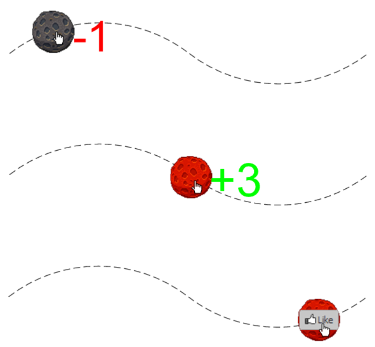

Figure 11: Events in the click timing game. **Top:** Clicking on the asteroid while it is not red penalizes the player. **Middle:** Clicking on the asteroid while it is red rewards the player. **Bottom:** (attack) The asteroid passes through a Like button while it flashes red.

cued by the color change via a "stop" signal that we discussed earlier. We rely on the race model of response inhibition [21] to understand and improve success of our attack.

When the user sees the like button (stop signal) and the flashing red asteroid (go signal), three factors determine the success of the attack: Stop signal delay, Go signal reaction time, and Stop signal reaction time.

- Stop signal delay refers to the temporal delay between the brain noticing the color change (go signal) and seeing the like button (stop signal).
- Go signal reaction time refers to the users reaction time to the Go signal in the absence of any stop signal. In this case, this corresponds to the time between the color change and the user deciding to click.
- Finally, the stop signal reaction time refers to the latency of the stop process—the time between the user noticing the like button and stopping the click. While we cannot directly measure the stop signal reaction time, we can infer it based on the race model.

The chance that our attack succeeds, i.e., the action escapes inhibition and the user clicks on the like button, increases with increasing stop signal delay and reaction time and decreases with increasing go signal reaction time. We can increase stop signal delay by adding changing stimuli around the like button, which results in attention diversion and delays in noticing the Like button. Training the user to click as soon as the color changes decreases the go signal reaction time.

5 Evaluation

We implemented prototypes of our attacks and evaluated their effectiveness. We have also made all our attacks available online [6]. We remind the reader that all the perceptual limitations we discussed are *statistical* by nature. While previous experiments found that these limitations affect a large number of humans, none of these limitations are certain to affect *all* humans.

Experimental Setup. To measure how effective our attacks are in tricking users into clicking on a Like button, we relied on an Amazon Mechanical Turk experiment similar to previous work [14]. In July of 2013, we recruited around 130 users for every attack discussed above. Our advertisement offered participants \$0.20–\$0.30 for a task described as "Play our 2min HTML5 game." We only allowed participants with Google Chrome or Mozilla Firefox browsers, since we tested our attacks on those browsers. We hosted our attacks on a third-party domain with no affiliation to our university. Our advertisement told participants that the task would take approximately 2 minutes. Participants had a chance to play one of our games for up to two minutes, or until they reached a game-specific goal condition. The HTML5 game simulated an attack with a fake Like button.

We then presented the users with a follow-up survey, with the same questions used in Huang et al.'s study:

1. Did you see the Like button in the game you played?
2. (If No to 1) Would you approve if your Facebook wall showed that you like this game?
3. (If Yes to 1) Did you click on the Like button?
4. (If Yes to 3) Did you intend to click on the Like button?

For our evaluation, we filtered out repeat participants between experiments. Second, using the survey, we also filtered out participants who knowingly clicked our fake Like button.

Results. Table 1 presents the attack success rates for each of our attacks. From the users who answered "No" to survey question 2, 3, or 4 (n_2), we computed the attack rate for each game as the fraction who ended the game by clicking the Like button. Our attack rates range from 20% to 99%. Other than the "Destabilizing Pointer Perception" (Section 4.1) attack, all our attacks are applicable to touch devices, which also lack a number of defenses available on pointer-based devices (Section 6.2).

Our follow-up survey also collected demographic information such as age, type of device, and sex. We did not find any correlation between these demographic factors and the success rates of our attacks so we do not discuss them any further.

Ethics. Our experiment was approved by our university's review board (IRB) with the protocol ID 2013-02-4988. Our experiment design aimed to pay participants at the maximum hourly minimum rate in California. Most experiments took much less than 2 minutes to finish. The user's private data is never at risk in our experiment: our fake Like button is non-functional and only visually similar to the Facebook Like button. If a participant clicked the fake Like button, we ended the experiment immediately and navigated to a debriefing page explaining the attack and that the user's data was never at risk. Thus, user anxiety, if any, was transient. Our experiment also did not collect any private data for the evaluation of our games.

Attack Name	n_1	n_2	Attacked
Destabilizing Pointer	122	71	99%
Peripheral Vision	103	86	55%
Adaptation	108	90	20%
Fast Motion	111	79	29%
Controlling the Timing	124	62	53%

Table 1: Success Rates of our attack. Starting from 150 participants for each attack, n_1 denotes number of participants after filtering for repeat participants; n_2 denotes the number of participants after we removed participants who reported in a follow-up survey that they knowingly clicked the Like button.

6 Discussion and Future Work

Our attacks demonstrate the practicality of perceptual attacks on user interfaces. In this section, we discuss possible defenses against perceptual attacks. We find that only a two-click requirement is able to provide a strong defense, although we recommend randomizing the location of the second click. Further, we also find that a number of previously proposed defense techniques do not work well for touch interfaces, suggesting more work on enforcing UI security on touch devices. We discuss this and other directions for future work below.

6.1 Architectural Defenses

A number of proposals already exist to defend against clickjacking attacks, only a few of which are in use [14, 39]. Our attacks bypass the key defense of visual and temporal integrity used in the W3C UI Safety specification (as well as InContext [14] and ClearClick [24]). We discuss other defenses below.

Disabling customization. One simple defense against our destabilizing pointer perception attack is to disallow cursor customization when any sensitive UI is present, a defense also presented in Huang et al. [14]. This defense also has a usability impact: video providers and game designers could no longer hide the pointer if their page has a Like button.

Lightbox. The browser could gray out everything but the sensitive UI (or otherwise try to draw attention to the sensitive UI) when the mouse pointer moves over the sensitive UI [14]. First, an attacker could desensitize users to this cue by using the same effect repeatedly before the attack. For example, in our peripheral vision game, an attacker could train the user to expect a lightbox like effect when a block starts falling. Second, this defense also has a tremendous usability impact, which could also explain why it has not achieved any adoption over the years.

Time Delay. Another defense requires that the cursor stay on the sensitive element for a fixed amount of time. This imposes a significant usability cost, because the user cannot do anything else while waiting for this timeout. We believe this usability cost is the reason Firefox is the only browser to uniformly implement this delay for security sensitive interactions. Additionally, this defense does not affect our attack on peripheral vision since the cursor stays on the Like button for the duration of the game.

Activation gestures. A similar defense requires that the cursor move in, out, and back into the sensitive element before enabling the element. We expect that this would be hard to explain to a first time user, since it is such a novel gesture. Furthermore, a variant of our motor adaptation game could adapt a player to performing this gesture in preparation for the attack.

Two Clicks. One defense that could protect against our attacks on touch interfaces is requiring, for every sensitive action, a second confirmation click. This is a defense already adopted by Chrome for its mixed-content dialogs [7]. Our motor adaptation attack could defeat this defense if the confirmation interface is in a predictable location. As a result, the second confirmation click should be in an unpredictable location. This negatively impacts usability.

Web services can already implement the two-click requirement with window popups. The fact that popular services like Facebook and Twitter have not implemented this defense suggests that they are not willing to accept the usability impact of requiring two clicks, let alone randomizing the location of the second click.

6.2 UI Security of Touch Interfaces

From our examination of architectural defenses, we found that most defenses target pointer-based interfaces and do not work well with touch interfaces. For example, the Lightbox defense, in which the browser darkens out an area when the pointer moves onto a secure element, has no equivalent for touch devices. Adapting the Activation Gesture and Time Delay defenses is possible, but we think such an interface would be awkward on touch devices. In fact, we believe only the two-click defense works equally well in touch and pointer-based interfaces.

Consider again our model of human perception and interaction. The location of the pointer acts as a good indicator for the current state of the motor system. This signal is absent on touch interfaces, making defending against attacks harder.[4]

In fact, we conjecture that *in typical situations*, the current location of the pointer also acts as a good indicator of the user's attention. One class of UI security attacks relies on violating this assumption. For example, our peripheral vision game tries to make the user focus on an area of the screen far away from the location of the pointer. Pointer-based defenses such as Lightbox, Gestures etc. aim to enforce this invariant; namely, that the user's current pointer location be where he/she focuses his/her attention (e.g., by darkening the other areas of the screen).

6.3 Future Defenses

The relative inability of current and proposed defense techniques to work on touch devices suggests the need for new proposals for UI security on touch devices. Our attacks serve as a cautionary tale for the design of UI security mechanisms for touch interfaces. Similar to how pointer location serves as a crude, bypassable indicator of user perception, defenses for touch interfaces should not assume that a user pressing on a part of the screen implies the user paid attention and intentionally interacted with it.

A key reason for the low adoption of strong defenses is the usability impact of requiring more effort from the user to demonstrate that she is actually paying attention. A better model of human perception can reduce the number of times we require more effort from the user.

The UI Safety specification uses simple image comparison techniques to decide whether an image was fully visible or not. A natural next question is whether advanced computer vision techniques could mitigate the attacks we presented. We forsee two problems.

First, defending against our attacks would require capturing screenshots of the whole screen for (say) five seconds. On newer machines with massive resolutions (up to 2880x1800, or 5 million pixels), this would consume significant CPU and memory just to collect the data, let alone analyze it. In contrast, the UI Safety specification only required screenshots of the security sensitive element, which is far smaller.

Second, a simple image comparison would not suffice for protection against our attacks: the algorithm would have to attempt to recreate human perceptual limitations with sufficient precision. Given the complexity and depth of human perception, this is a challenging research area.

For instance, a vision algorithm could keep track of repeated actions and block clicks on a sensitive button if it detects the possibility of adaptation. While such techniques could work for specific attacks, it is not clear if a vision-based technique can reproduce all other limitations

[4]Of course, absence of a pointer makes attacks such as our first attack ("Destabilizing Pointer Perception") impossible too.

of human perception such as color cues and illusions. Additionally, humans will not notice transient changes in the scene (e.g., hiding a fake pointer for a few frames), while the same might confuse a vision algorithm.

6.4 Future Attacks

Our attacks are prototypes. In this section, we first discuss possible avenues that a malicious actor could rely on to increase success rates of the attacks we presented. Next, as an example of the breadth of human perception, we present two new ideas for attacking human perception.

6.4.1 Improving our Attacks

We believe that our experimental evaluation only provides a lower bound on the effectiveness of our proposed attacks. We present three reasons why we believe an attacker can achieve a far higher success rate with our attacks.

UI safety and Perceptual Attacks. We designed our attacks against a particularly strong form of the UI safety defense. The Like button in our attack was visible for a minimum of 1 second and up to 3 seconds in some cases. This is higher than the default `display-time` value of 800ms in the specification. In fact, due to usability concerns, it is unlikely that real-world web applications will rely on values above 500ms [14, 25, 35]. This suggests that attacks on real-world applications will achieve a higher success rate.

Combined Attack. Our attacks investigated limitations of human perception along orthogonal dimensions, namely, predicting click location due to limitations of motor system (Section 4.3), predicting click location due to limitations of the visual system (Section 4.4), predicting click times (Section 4.5), peripheral vision (Section 4.2), and a replaced pointer (Section 4.1). Our intention was to demonstrate the breadth of possibilities when we take a perceptual view of UI security. A malicious actor interested in relying on such attacks is likely to combine them into a powerful perceptual attack with a high success rate.

Complex Attacks. Our attack design is simple. For instance, we do not collect any data from the user as she plays the game. We believe such data collection can help dynamically tweak our attacks as well as help iterate over our designs to improve our success rates even more. An attacker could also use better models of pointer movement and click prediction (e.g., see Zalewski [47]) to improve attack accuracy. Collecting more data about our attacks and reasons for their success/failure could be a source of ideas for novel attacks, an idea we discuss next.

6.4.2 Novel Attacks

We stress that our exploration of the limitations of human perception is nascent: a number of other attacks are possible due to the breadth and complexity of human perception. We give a couple of ideas next.

Change Blindness. Change blindness is a well-studied psychological phenomenon in which a user fails to notice difference in two images [41]. An attacker can train a user to click on a particular button, move the mouse to another area, and move back and click on the same button again. The attacker can then switch the button with a sensitive UI element (e.g., a Like button). By setting up the attack webpage appropriately, the attacker can ensure that the user does not notice the switch, due to change blindness.

Background Motion. Background motion results in mislocalization of a target [4]. This effect works particularly well in goal-directed tasks, when a user is trying to move towards a particular goal. For instance, an attacker can place a simple asteroid on a page and entice a user to click on it. To the right of the asteroid is a Facebook Like button.

When the user's mouse approaches the asteroid, a mineral object appears near the asteroid and quickly moves away to the left. This creates an illusion that the asteroid has moved right, to compensate for the mineral's momentum. The asteroid actually remains stationary, as does the Like button. In a successful attack, the user corrects the mouse to accommodate the perceived rightwards motion and ends up clicking the Like button.

7 Conclusion

We presented a study of UI security on a perceptual level. A perceptual perspective on UI attacks helps identify limitations of current defenses against UI attacks as well as identify a number of novel UI attacks. Our attacks exploit diverse aspects of human perception such as adaptation, attention, and peripheral vision. An evaluation on Amazon Mechanical Turk showed success rates between 20% and 99%. We discussed the efficacy of current and proposed defenses and found them lacking, particularly for touch devices. Our work suggests the need for more work on improved defenses and mechanisms to better model the human perceptual system.

References

[1] ABOUKHADIJEH, F. How to: Spy on the webcams of your website visitors. http://feross.org/webcam-spy/, 2011.

[2] AUN, L. C. Clickjacking with pointer-events. http://jsbin.com/imuca, 2012.

[3] BORDI, E. Proof of concept - cursorjacking (noscript). http://static.vulnerability.fr/noscript-cursorjacking.html.

[4] BRENNER, E., AND SMEETS, J. B. Fast responses of the human hand to changes in target position. *Journal of motor behavior 29*, 4 (1997), 297–310.

[5] BUONOMANO, D. *Brain Bugs: How the Brain's Flaws Shape Our Lives.* WW Norton, 2011.

[6] Demo of perceptual ui attacks. http://wh0.github.io/safeclick-blast/.

[7] EVANS, C., AND SEPEZ, T. Ending mixed scripting vulnerabilities. http://blog.chromium.org/2012/08/ending-mixed-scripting-vulnerabilities.html.

[8] FRIEDMAN-HILL, S. R., ROBERTSON, L. C., DESIMONE, R., AND UNGERLEIDER, L. G. Posterior parietal cortex and the filtering of distractors. *Proceedings of the National Academy of Sciences 100*, 7 (2003), 4263.

[9] GOODIN, D. Twitter attack exposes awesome power of clickjacking. http://www.theregister.co.uk/2009/02/13/twitter_clickjack_attack/, 2009.

[10] HANSEN, R. Clickjacking details. http://ha.ckers.org/blog/20081007/clickjacking-details/, 2008.

[11] HANSEN, R., AND GROSSMAN, J. Clickjacking. http://www.sectheory.com/clickjacking.htm, 2008.

[12] HILL, B. Adaptive user interface randomization as an anti-clickjacking strategy. http://www.thesecuritypractice.com/the_security_practice/papers/AdaptiveUserInterfaceRandomization.pdf, 2012.

[13] HUANG, L.-S., AND JACKSON, C. Clickjacking attacks unresolved. http://mayscript.com/blog/david/clickjacking-attacks-unresolved, 2011.

[14] HUANG, L.-S., MOSHCHUK, A., WANG, H. J., SCHECHTER, S., AND JACKSON, C. Clickjacking: attacks and defenses. In *Proceedings of the 21st USENIX conference on Security symposium* (Berkeley, CA, USA, 2012), Security'12, USENIX Association, pp. 22–22.

[15] IMPERVA. Clickjacking (aka. ui redressing). http://www.imperva.com/resources/glossary/clickjacking_ui-redressing.html.

[16] INFOSECURITY. Clickjacking threatens two-thirds of top 20 banking sites. http://www.infosecurity-magazine.com/view/29610, 2012.

[17] ISODA, M., AND TANJI, J. Cellular activity in the supplementary eye field during sequential performance of multiple saccades. *Journal of neurophysiology 88*, 6 (2002), 3541–3545.

[18] KOTOWICZ, K. Cursorjacking again. http://blog.kotowicz.net/2012/01/cursorjacking-again.html, 2012.

[19] LAWRENCE, E. Ie8 security part vii: Clickjacking defenses. http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx, 2009.

[20] LEVI, D. M. Crowding-an essential bottleneck for object recognition: A mini-review. *Vision research 48*, 5 (2008), 635.

[21] LOGAN, G. D., AND COWAN, W. B. On the ability to inhibit thought and action: A theory of an act of control. *Psychological review 91*, 3 (1984), 295.

[22] LU, X., MATSUZAWA, M., AND HIKOSAKA, O. A neural correlate of oculomotor sequences in supplementary eye field. *Neuron 34*, 2 (2002), 317–325.

[23] MACKAY, D. Perceptual stability of a stroboscopically lit visual field containing self-luminous objects.

[24] MAONE, G. Hello clearclick, goodbye clickjacking! http://hackademix.net/2008/10/08/hello-clearclick-goodbye-clickjacking/, 2008.

[25] MAONE, G., HUANG, D. L.-S., GONDROM, T., AND HILL, B. User interface safety. https://dvcs.w3.org/hg/user-interface-safety/raw-file/tip/user-interface-safety.html.

[26] MORAN, J., AND DESIMONE, R. Selective attention gates visual processing in the extrastriate cortex. *Frontiers in cognitive neuroscience 229* (1985), 342–345.

[27] MOTTER, B. C. Focal attention produces spatially selective processing in visual cortical areas v1, v2, and v4 in the presence of competing stimuli. *Journal of Neurophysiology 70*, 3 (1993), 909–919.

[28] NIEMIETZ, M. Cursorjacking. `http://www.mniemietz.de/demo/cursorjacking/cursorjacking.html`, 2011.

[29] NIJHAWAN, R. Motion extrapolation in catching. *Nature; Nature* (1994).

[30] PELLI, D. G., PALOMARES, M., AND MAJAJ, N. J. Crowding is unlike ordinary masking: Distinguishing feature integration from detection. *Journal of vision 4*, 12 (2004).

[31] RUDERMAN, J. Bug 56236 - possible to selectively allow chars into file upload control by disabling control onkeydown. `https://bugzilla.mozilla.org/show_bug.cgi?id=56236`, 2000.

[32] RUDERMAN, J. Bug 57770 - using styles, clipboard to confuse text entry into file upload control. `https://bugzilla.mozilla.org/show_bug.cgi?id=57770`, 2000.

[33] RUDERMAN, J. Bug 154957 - iframe content background defaults to transparent. `https://bugzilla.mozilla.org/show_bug.cgi?id=154957`, 2002.

[34] RUDERMAN, J. Bug 162020 - pop up xpinstall/security dialog when user is about to click. `https://bugzilla.mozilla.org/show_bug.cgi?id=162020`, 2002.

[35] RUDERMAN, J. Race conditions in security dialogs. `http://www.squarefree.com/2004/07/01/race-conditions-in-security-dialogs/`, 2004.

[36] RYDSTEDT, G., BURSZTEIN, E., BONEH, D., AND JACKSON, C. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. *IEEE Oakland Web 2* (2010).

[37] RYDSTEDT, G., GOURDIN, B., BURSZTEIN, E., AND BONEH, D. Framing attacks on smart phones and dumb routers: tap-jacking and geo-localization attacks. In *Proceedings of the 4th USENIX conference on Offensive technologies* (Berkeley, CA, USA, 2010), WOOT'10, USENIX Association, pp. 1–8.

[38] SCLAFANI, S. Clickjacking & oauth. `http://stephensclafani.com/2009/05/04/`, 2009.

[39] UHLEY, P. Clickjacking threats. `http://www.w3.org/Security/wiki/Clickjacking_Threats`, March 2012.

[40] VELA, E. About css attacks. `http://sirdarckcat.blogspot.com/2008/10/about-css-attacks.html`, 2008.

[41] WIKIPEDIA. Change blindness. `http://en.wikipedia.org/wiki/Change_blindness`.

[42] WIKIPEDIA. Likejacking. `http://en.wikipedia.org/wiki/Clickjacking#Likejacking`.

[43] WISNIEWSKI, C. Facebook adds speed bump to slow down likejackers. `http://nakedsecurity.sophos.com/2011/03/30/facebook-adds-speed-bump-to-slow-down-likejackers/`, March 2011.

[44] YANG, D. Clickjacking: An overlooked web security hole. `https://community.qualys.com/blogs/securitylabs/2012/11/29`, 2012.

[45] ZALEWSKI, M. Dealing with ui redress vulnerabilities inherent to the current web. `http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2008-September/016284.html`, 2008.

[46] ZALEWSKI, M. minor browser ui nitpicking. `http://seclists.org/fulldisclosure/2010/Dec/328`, 2010.

[47] ZALEWSKI, M. On designing uis for non-robots. `http://lcamtuf.blogspot.com/2010/08/on-designing-uis-for-non-robots.html`, 2010.

[48] ZALEWSKI, M. *The Tangled Web: A Guide to Securing Modern Web Applications.* No Starch Press, 2011.

[49] ZALEWSKI, M. X-frame-options, or solving the wrong problem. `http://lcamtuf.blogspot.com/2011/12/x-frame-options-or-solving-wrong.html`, 2011.