Blob URI Phishing Attack

Introduction

Blob URI-based phishing attacks represent a sophisticated and stealthy method of credential theft that bypasses traditional security mechanisms. By leveraging browser-native features like blob and URL.createObjectURL, attackers can craft convincing fake login pages that never touch the network, making detection significantly harder. This article outlines the technical mechanics, real-world applications, detection challenges, and defensive strategies against Blob URI–based phishing attacks.

Phishing remains one of the most effective tools in a cybercriminal's arsenal. While traditional phishing involves redirecting users to lookalike domains, modern attackers are increasingly adopting in-memory attack vectors using Blob URIs. These techniques exploit native browser features to present fake but convincing login pages without relying on external hosting.

What Is a Blob URI?

A Blob URI is a special URL generated in the browser using JavaScript:

```
const blob = new Blob([htmlContent], { type: 'text/html' });
const url = URL.createObjectURL(blob);
```

The resulting URL looks like:

blob:https://legitdomain.com/550e8400-e29b-41d4-a716-446655440000

This URI is generated locally in the browser and does not correspond to any real network resource. The origin shown in the blob URI (e.g., https://legitdomain.com) is simply the origin of the page that created the blob — not where the data is hosted. This makes blob URIs appear trustworthy even though their contents are entirely controlled by JavaScript running in the browser.

Anatomy of a Blob URI Phishing Attack

A typical attack sequence looks like this:

- The attacker injects JavaScript into a trusted page (via XSS or extension abuse).
- The script constructs a fake login page using HTML and JavaScript.
- A Blob object is created and converted into a blob URI.
- The fake page is opened in a new window/tab using window.open(blobUrl).
- Credentials are captured and sent via fetch() to the attacker's server.
- Optionally, the user is redirected to the real login page to reduce suspicion.

Why It’s Dangerous

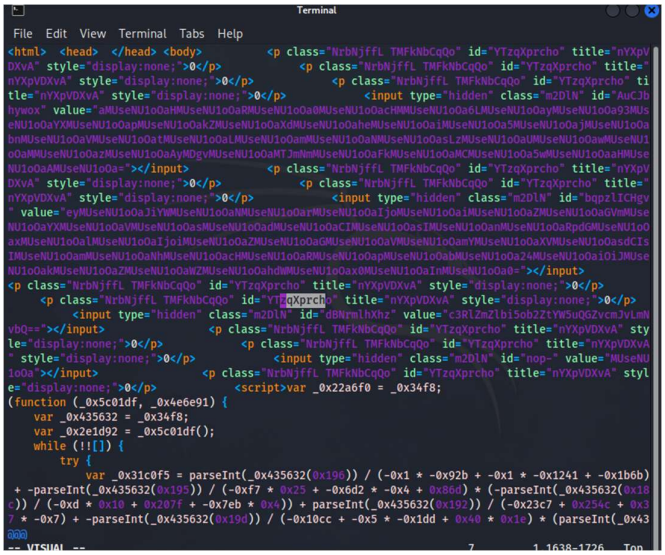
In-memory execution: The payload never touches the disk or network.

- Origin spoofing: The blob URI shows the origin of the parent page.
- No DNS or certificate indicators: Users can’t rely on padlocks or URL bars.
- Hard to detect with traditional AV/WAFs: There’s no external domain to flag.

Real-World Examples

- Malicious Chrome extensions have used blob URIs to load phishing forms in-memory.
- Researchers have seen phishing kits delivering Office 365 and Google login clones via blob-based popups.
- Advanced malware loaders use encrypted payloads stored in Blob objects to bypass disk-based scanners.

Real Example of a Phishing Email based on HTML with JavaScript (obfuscated):



Use of Encrypted and Obfuscated JavaScript

While not required, encryption and obfuscation are often used together to:

- Obfuscate the creation of the Blob URI itself (e.g., hiding new Blob(...) and URL.createObjectURL)
- Encrypt the phishing HTML or script content
- Delay detection and slow reverse engineering
- Bypass static analysis tools

Common techniques include:

- Base64 encoding of the phishing payload (atob(...))
- AES or XOR encryption with decryption done at runtime
- Use of dynamic code execution via eval(), Function(), or setTimeout("...")
- Obfuscation tools that rename variables, compress code, and flatten logic

Example:

```
let e = atob("PGZvcm0+PHVzZXI+PC9mb3JtPg==");
let b = new Blob([e], { type: 'text/html' });
let u = URL.createObjectURL(b);
window.open(u);
```

This makes analysis and automated detection significantly harder.

Detection Challenges

- blob: URLs aren't logged by firewalls or antivirus software.
- Browser telemetry may not reveal the blob's contents.
- Users can't see anything suspicious unless trained to examine full URLs.

Defense Strategies

a. Technical Controls:

Enforce a strict Content Security Policy (CSP):

```
Content-Security-Policy: default-src 'self'; script-src 'self'; object-src
'none'; base-uri 'none'; worker-src 'none'; child-src 'none'; frame-src
'none';
```

Disallow blob: and data: URIs in script-src, frame-src, etc.

Monitor browser extensions and enforce enterprise policies.

b. Endpoint Detection and Response (EDR):

- Look for suspicious browser activity and outbound POSTs to unknown domains.
- Flag use of eval(), Function(), or atob() inside browsers.

c. User Awareness:

- Train users to avoid entering credentials unless the URL is clearly HTTPS with a known domain.
- Encourage password manager use (they won't autofill on fake pages).

d. Advanced Hunting (for Microsoft Defender):

Sample Kusto Query:

```
DeviceNetworkEvents
| where InitiatingProcessFileName in~ ("chrome.exe", "msedge.exe")
| where RemoteUrl has_any (".php", "/login", "/submit")
| where RemoteUrl !contains "yourdomain.com
```

Conclusion

Blob URI phishing is a stealthy, sophisticated technique that takes advantage of browser trust and in-memory execution. While not yet mainstream, its usage is growing in targeted attacks and advanced phishing kits. Organizations must respond with a mix of technical controls, user education, and active monitoring to defend against this modern phishing vector.