Parameter cloaking

Executive Summary

This report outlines the discovery of a **web cache poisoning vulnerability via parameter cloaking** on the target application. The vulnerability arises from inconsistent parameter parsing between the caching layer and the back-end server, along with the exclusion of specific query parameters (like `utm_content`) from the cache key. This flaw enables attackers to serve malicious cached responses, such as JavaScript payloads, to unsuspecting users. The impact is **high**, as it can lead to client-side code execution (XSS) across multiple user sessions.

Introduction

The objective of this engagement was to identify vulnerabilities in the web application's cache behavior, specifically those related to unkeyed parameters and inconsistent parsing. This is crucial as improperly keyed cache systems can be leveraged by attackers to poison shared caches and deliver harmful content to multiple users.

Methodology

The following approach was used to identify and exploit the vulnerability:

1. Burp Suite was used to intercept and analyze HTTP requests and responses.
2. Param Miner extension was used to detect unkeyed and cloaked parameters.
3. Manual testing was performed to observe caching behavior and reflected JavaScript.
4. Cache behavior was exploited to overwrite the callback parameter in a JSONP response.
5. A malicious `alert(1)` JavaScript payload was injected and cached for all users.

Vulnerability Findings

- **Type**: Web Cache Poisoning via Parameter Cloaking
- **Location**: `/js/geolocate.js` endpoint
- **Severity**: High
- **Impact**: Arbitrary JavaScript execution in victim browsers via poisoned cache

Description

The application included an analytics-related query parameter `utm_content`, which is excluded from the cache key. Additionally, the application fetches `/js/geolocate.js` using a `callback` parameter to execute a function with geolocation data in JSONP format.

The cache processes:

bash

CopyEdit

`GET /js/geolocate.js?callback=setCountryCookie&utm_content=foo;callback=alert(1)`

as:

- Cache key: `/js/geolocate.js?callback=setCountryCookie`

- Executed script: `alert(1)({"country":"United Kingdom"})`

Because the second `callback` is cloaked within `utm_content` using a semicolon (`;`), the back-end uses it, but the cache ignores it, resulting in a poisoned cache where every user receives a response with `alert(1)` executed in their browser.

Proof of Concept (PoC)

1. Request to poison the cache:

   vbnet

   CopyEdit

   `GET /js/geolocate.js?callback=setCountryCookie&utm_content=foo;callback=alert(1) HTTP/1.1 Host: website.id.net`

2. Poisoned response:

   javascript

   CopyEdit

   `alert(1)({"country":"United Kingdom"})`

Impact Verification

When the homepage is loaded by another user (e.g., the victim using Chrome), the cached `/js/geolocate.js` is loaded, triggering `alert(1)` in their browser.

Impact Assessment

- Cross-site Scripting (XSS): Executing scripts in the victim's browser.

- High Exploitability: Requires no authentication or user interaction beyond visiting the homepage.

- Persistent Across Sessions: Cached responses persist for multiple users until manually flushed or expired.

- Abuse Potential: Can be escalated to steal session tokens, deface pages, or load malicious scripts.

Recommendations

1. Include all parameters in cache keys, including those considered innocuous (e.g., UTM parameters).

2. Sanitize or avoid reflecting user-supplied input, especially in dynamic JavaScript responses.

3. Avoid JSONP where possible; use secure alternatives such as CORS with strict policies.

4. Regularly scan for parameter cloaking vulnerabilities using automated tools like Burp Param Miner.

5. Implement Content Security Policy (CSP) to mitigate XSS impact.

Conclusion

This assessment revealed a critical flaw in the application’s caching strategy. Through parameter cloaking, an attacker can poison shared cache entries with malicious JavaScript that executes in victim browsers. This illustrates the importance of rigorous cache key management and input sanitization. Immediate remediation is necessary to prevent exploitation at scale.