

# URL Hijacking

line 1: Matteo Ciavaglia  
line 2: *College of Computing*  
line 3: *Grand Valley State*  
University  
line 4: Allendale MI, USA  
line 5: ciavaglm@mail.gvsu.edu

line 1: Hunter Sutton  
line 2: *College of Computing*  
line 3: *Grand Valley State*  
University  
line 4: Allendale MI, USA  
line 5: suttonhu@mail.gvsu.edu

line 1: Jack Wolak  
line 2: *College of Computing*  
line 3: *Grand Valley State*  
University  
line 4: Allendale MI, USA  
line 5: wolakj@mail.gvsu.edu

line 1: Andrew Slayton  
line 2: *College of Computing*  
line 3: *Grand Valley State*  
Univesity  
line 4: Allendale MI, USA  
line 5: slaytona@mail.gvsu.edu

**Abstract**—Ad fraud and artificial traffic present significant challenges in the digital advertising landscape, undermining trust and transparency within the industry. Click hijacking, a form of ad fraud, redirects users to unintended websites when clicking on ads, leading to monetary losses for advertisers and deceptive metrics. Current solutions, such as click redirection validation, URL monitoring, and referrer verification, aim to mitigate click hijacking but face limitations, particularly with dynamic URLs. This project proposes an enhanced solution focusing on improving click redirection validation, specifically addressing the challenge posed by dynamic URLs. By converting dynamic URLs into strings and comparing them against a whitelist, the proposed solution aims to increase the effectiveness of detecting and preventing click hijacking.

**Keywords**—clickjacking, validation, dynamic, parse.

## I. INTRODUCTION

In the ever-expanding digital advertising realm, the threat of ad fraud and artificial traffic undermines the integrity and effectiveness of online advertising campaigns. Click hijacking, a prevalent form of ad fraud, deceives users by redirecting them to unauthorized websites upon clicking on legitimate ads, leading to financial losses for advertisers and degraded trust in the digital advertising industry. To combat this growing problem, innovative solutions are imperative.

This project focuses on enhancing click redirection validation, a key strategy in mitigating click hijacking. Leveraging the insights from research and informed by the challenges posed by dynamic URLs, our proposed solution aims to fortify the existing defense mechanisms against ad fraud. By parsing and comparing URLs against a whitelist, our solution aims to bolster the accuracy and efficacy of click redirection validation.

With a team committed to a specific timeline, doing research, development, testing, debugging, and documentation, this project aims to deliver solution to tackle one of the problems of ad fraud in the digital age. Through collaborative efforts and innovative approaches, we aspire to contribute towards a safer and more trustworthy digital advertising landscape.

## II. EASE OF USE

### A. Simplified URL Parsing

The `urlparse` function provided by the `urllib.parse` module offers a straightforward approach to parsing URLs. By passing a URL string to `urlparse`, we can extract various components of the URL, such as the scheme, netloc (domain), path, query parameters, and fragment. This streamlines the process of working with URLs, as developers don't need to manually parse strings or handle complex regular expressions to extract URL components. Instead, they can rely on the `urlparse` function to provide a structured representation of the URL.

### B. Efficient URL Reconstruction

On the other hand, the `urlunparse` function enables us to efficiently reconstruct URLs from their individual components. By providing the scheme and netloc (domain) components obtained from parsing, along with empty strings for the other components, we can generate clean and concise URLs. This streamlined approach eliminates the need for manual string concatenation or manipulation, reducing the likelihood of errors and enhancing code readability. Additionally, the simplicity of `urlunparse` contributes to improved maintainability, as we can easily comprehend and modify URL reconstruction logic as needed.

## III. UNDERSTANDING URL COMPONENTS

### A. Abbreviations and Acronyms

These are the abbreviations and acronyms that we used throughout the project. Understanding what these abbreviations and acronyms means will help to understand how our solution works.

URL: Uniform Resource Locator

HTTP: Hypertext Transfer Protocol

DNS: Domain Name System

IP: Internet Protocol

URL Hijacking: Unauthorized manipulation of a URL to redirect users to a different destination

URL Validation: Process of verifying whether a URL adheres to certain rules and standards

URL Whitelisting: Allowing access to only specified URL's while blocking others

#### B. Units

- URL is broken up into different components. The components are scheme, netloc, path, params, query, and fragment.

#### C. Method

A parser is used to break down each URL into it's different components. Scheme is the protocol or scheme used in the URL. Netloc is the domain name or IP address associated with the URL. Path is the component of the URL that shows the specific resource on the server. Parameters are any parameters that are passed into the URL query string. Fragment is often used to specify a section within an HTML document.

Example:

Start URL: <https://www.example.com/page?param=value>

scheme: "https"

netloc: [www.example.com](http://www.example.com)

path: "/page"

params: ""

query: "param=value"

fragment: ""

End Result: <https://www.example.com/>

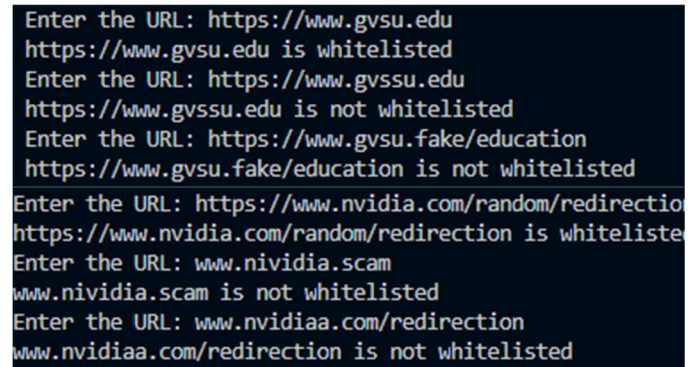
As in the example above the parser will take the original URL and then remove certain components to get the base URL. The resulting URL after parsing through it is just the base website with no parameters or fragments.

#### D. Some Common Mistakes

- Misinterpretation of components. In our program parsing might fail to handle edge cases.
- We also assume URL Validity which means we don't test to see if the URL endpoint is even reachable.
- With whitelisting there could be incomplete whitelists where you fail to include all good URL's.
- Being too laid back with whitelists may lead to malicious URL's.
- Some safe URL's may be blocked depending on how strict your criteria is.

## IV. DEMONSTRATION

#### A. Figure



```
Enter the URL: https://www.gvsu.edu
https://www.gvsu.edu is whitelisted
Enter the URL: https://www.gvssu.edu
https://www.gvssu.edu is not whitelisted
Enter the URL: https://www.gvsu.fake/education
https://www.gvsu.fake/education is not whitelisted
Enter the URL: https://www.nvidia.com/random/redirection
https://www.nvidia.com/random/redirection is whitelisted
Enter the URL: www.nvidia.scam
www.nvidia.scam is not whitelisted
Enter the URL: www.nvidiaa.com/redirection
www.nvidiaa.com/redirection is not whitelisted
```

a) *Breakdown of the figure above:* In the figure above this shows the output of the URLs being tested against the whitelist. The fake or non trusted URL's are being labeled as not whitelisted showing that it could be a malicious site, while the ones that are whitelisted are coming back as so.

## V. FINAL THOUGHTS

#### A. Conclusion

In conclusion, our project has delved into the realm of digital advertising security, particularly addressing the issue of click hijacking. By analyzing the URL handling and validation, we have figured out a potential solution to fortify click redirection mechanisms against fraudulent activities. Through the incorporation of URL parsing techniques and the implementation of dynamic URL validation, we have enhanced the effectiveness of click redirection validation. Moving forward, the insights gained from this project have benefited us in understanding this matter.

#### B. Problems Encountered

Some of the problems that we encountered was within our program we were originally using regular expressions to try and break up each component from the URL selected. But we kept getting errors with this method, so we changed the code to use a parse function built into python to get each of the components of the URL. On a previous implementation we hosted a proxy server that implemented our algorithm for cross checking the link with our whitelisted links. While the proxy server worked on occasion due to networking bugs it would consistently fail. However, from our proxy server implementation we discovered the applicability of the program in settings such as a workforce.

#### C. Future Work

In future implementation, our focus could extend towards integrating our solution into a mock website environment, ensuring seamless deployment and compatibility with existing web infrastructure. By developing a script to incorporate our click redirection validation mechanism into the website's backend, we can effectively intercept and validate outgoing click requests, thereby safeguarding users from potential click hijacking attempts.

## REFERENCES

- [1] A. C. D. Advocate and Andrea ChiarelliPrincipal Developer AdvocateI have over 20 years of experience as a software engineer and technical author. Throughout my career, “Clickjacking attacks and how to prevent them,” Auth0, <https://auth0.com/blog/preventing-clickjacking-attacks/> (accessed Apr. 16, 2024).
- [2] “What is clickjacking? the best attack prevention methods: Upguard,” RSS, <https://www.upguard.com/blog/what-is-clickjacking> (accessed Apr. 16, 2024).
- [3] “What is clickjacking? tutorial & examples: Web security academy,” What is Clickjacking? Tutorial & Examples | Web Security Academy, <https://portswigger.net/web-security/clickjacking> (accessed Apr. 16, 2024).
- [4] “Clickjacking,” Clickjacking | OWASP Foundation, <https://owasp.org/www-community/attacks/Clickjacking> (accessed Apr. 16, 2024).