BROWSER ATTACK SURFACE

HADESS

WWW.HADESS.IO

INTRODUCTION

The web browser, often viewed as a mere portal to the vast digital world, has transformed into one of the most critical software interfaces in our daily lives. As a bridge between users and the boundless troves of web content, it's equipped with the power to both enrich and compromise the digital experience. However, as web technologies have grown in complexity and utility, so too have the security threats that aim to exploit them. This combination has placed browsers on the frontline of cybersecurity battles, making understanding browser security risks more essential than ever.

Browsers interface with diverse content, pulling data from multiple sources, rendering pages, managing sessions, and running complex web applications. These processes, while integral to the browser's functionality, present numerous points of potential exploitation. Malicious entities have continually found creative ways to manipulate these features, whether through cross-site scripting, drive-by downloads, or cookie theft, among other techniques.

Yet, the onus of security doesn't lie with browsers alone. Users often inadvertently expose themselves to risks through poor browsing habits, failure to update software, or mismanagement of plugins and extensions. For instance, outdated browser versions may lack the latest security patches, making them susceptible to known vulnerabilities. Similarly, rogue extensions can act as Trojans, appearing legitimate but harboring malicious intentions.

In the ensuing sections, we will delve deep into these risks, exploring the myriad ways browsers can be compromised, the potential impacts of such breaches, and the measures that can be taken to fortify browser security. As we move further into an interconnected era, the proverbial "locks" on our digital "doors" – our browsers – demand our keen attention and understanding.

DOCUMENT INFO

HADESS

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.

At Hadess, our mission is twofold: to unleash the power of white hat hacking in punishing black hat hackers and to fortify the digital defenses of our clients. We are committed to employing our elite team of expert cybersecurity professionals to identify, neutralize, and bring to justice those who seek to exploit vulnerabilities. Simultaneously, we provide comprehensive solutions and services to protect our client's digital assets, ensuring their resilience against cyber attacks. With an unwavering focus on integrity, innovation, and client satisfaction, we strive to be the guardian of trust and security in the digital realm.

Security Researcher
Negin Nourbakhsh
Fazel Mohammad Ali Pour

TABLE OF CONTENT

Executive Summary

Web browsers are more than just software applications; they are the portals through which most of us access the digital universe. With the escalating digitization of our day-to-day activities, from banking to entertainment to business operations, browsers have never played a more critical role. This centrality underscores the need for robust browser security, as vulnerabilities can disrupt not just individual users, but entire organizational infrastructures.

The Ubiquity of Browser Security Risks:
As the frontlines of our online interactions, browsers are continually targeted by a myriad of security threats. These range from sophisticated cross-site scripting attacks to deceptive phishing scams, to more covert drive-by downloads. Each of these threats presents its unique set of challenges, with potential ramifications that can severely compromise user data and privacy.

The Human Element:
While technological solutions are essential, it's paramount to acknowledge the role of the user. Often, a browser's security posture is influenced by human actions, such as the timely updating of software, prudent management of browser extensions, and discernment against malicious links. The human-user interaction with the browser often becomes the weakest link, even with advanced security protocols in place.

A Deep Dive into the Firefox Vulnerability:
Amid this backdrop of prevalent browser threats, the article delves into a specific and intriguing vulnerability linked to Firefox. This flaw, stemming from a misconfiguration with 'xdg-mime'—a MIME type management tool in Linux environments—leads to an anomalous behavior where Firefox can be trapped into opening tabs ad infinitum until it crashes. It's a stark illustration of how even renowned browsers aren't immune to subtle glitches.

Implications and Lessons:
While the Firefox misconfiguration might seem relatively benign at first glance, its implications are profound. It exemplifies how intricate software ecosystems can become susceptible to unexpected behaviors due to minor oversights. The vulnerability underlines the need for rigorous, continual software testing, prompt patching, and the active management of configurations across platforms.

Towards a Secure Browsing Future:
As we increasingly intertwine our lives with digital platforms, the security of our browsers isn't just an IT concern—it's a societal one. Recognizing and addressing vulnerabilities, while also educating users on best practices, will be instrumental in ensuring that our digital gateways remain both functional and secure. This article aims to illuminate these challenges and advocate for a comprehensive, proactive approach to browser security.

Key Findings

Pervasive Security Threats: Browsers, as the primary interface for online interactions, are constant targets for myriad security threats. From advanced tactics like cross-site scripting to deceptive phishing, browsers face a diverse array of challenges.

User Influence on Security: Often overlooked, the role of the end-user is a significant factor in a browser's overall security. Practices like timely software updates, judicious use of extensions, and cautious browsing habits can substantially reduce potential threats.

Firefox xdg-mime Misconfiguration: A notable vulnerability was identified in Firefox related to its interaction with 'xdg-mime' in Linux systems. The misconfiguration can cause Firefox to enter an infinite loop of opening new tabs, eventually leading to a crash.

Subtleties in Software Interactions: The Firefox misconfiguration emphasizes the fragility of software interactions. What might appear as minor oversights in configuration or code can lead to unintended and disruptive behaviors.

Imperative for Comprehensive Testing: The identified vulnerability underscores the importance of thorough software testing across different environments and configurations. Assumptions about software behavior in varied environments can lead to oversights.

Abstract

Web browsers, as our primary gateways to the vast digital universe, have become intricate software tools with multifaceted features and functionalities. They are continuously evolving, not just in terms of features, but also in addressing the ever-growing list of vulnerabilities and threats. This paper dives deep into the challenges surrounding patch management in browsers.

Patch management, the systematic process of deploying updates to software applications, is crucial for browsers given their direct exposure to external threats. However, it's not as straightforward as it appears. Factors like diverse user bases, multiple operating platforms, and backward compatibility can complicate the rollout of crucial updates.

Furthermore, the urgency to address critical vulnerabilities often pushes developers into a rushed patching cycle. This urgency can sometimes compromise the quality of patches, leading to new, unforeseen issues or even reintroducing old vulnerabilities. Such scenarios emphasize the need for a more refined, yet agile, patch management strategy.

HADESS.IO

Browser Attack Surface

01

Attacks

Browser Security Risks

Web browsers are more than just tools for accessing information; they are the gateways to our digital identities, the portals through which we engage with the vast universe of the Internet. As our dependency on these platforms grows, so too does the attention they receive from those with malicious intent. Recent years have seen a disturbing surge in browser-based vulnerabilities, a phenomenon not merely incidental but emblematic of the evolving cybersecurity landscape. This spike in potential threats isn't just a matter of numbers but speaks to a deeper, more insidious trend. Cyber adversaries are no longer merely exploiting known vulnerabilities—they are crafting complex, adaptive strategies designed to probe, test, and breach browser defenses. As we peel back the layers on this issue, we delve into an environment marked by subterfuge, innovation, and a constant game of cat and mouse between defenders and attackers. This article seeks to illuminate the intricate dance between browser security mechanisms and the sophisticated techniques employed by cybercriminals, with a spotlight on the revealing world of adversary simulation operations.

The Ever-Present Threat Landscape
Browser vulnerabilities are not a novel threat; they've been a persistent concern since the dawn of the internet era. However, the surge by over 20% in just the last year, as highlighted in the 2021 report, emphasizes an acceleration of risk. This rapid increase raises immediate questions about the evolving nature of these vulnerabilities and the broader implications for digital security.

The Pandemic's Digital Pivot
The COVID-19 pandemic dramatically altered the global digital landscape. With lockdowns and remote work becoming the new norm, there was a notable surge in online activity. Personal, professional, and educational interactions shifted to the web, leading to a more extensive and more varied user base. This uptick in traffic, combined with users perhaps less familiar with online best practices, created a vast playground for cybercriminals.

Cybercriminal Evolution
While the pandemic broadened the attack surface, it was the evolution in cybercriminal strategies that exploited it. Gone are the days of rudimentary phishing attacks. Today's hackers employ sophisticated methods, combining multiple vulnerabilities, deploying zero-day exploits, and leveraging advanced persistent threats (APTs) to penetrate defenses.

Proliferation of Web Technologies
The web ecosystem has witnessed a diversification of technologies and platforms, such as WebAssembly, progressive web apps, and single-page applications. While these innovations enhance user experience and functionality, they also introduce novel vulnerabilities, broadening the attack spectrum.

Attack Sophistication and Economic Incentives
The dark web and cybercrime forums have become hotbeds for exchanging tools, strategies, and even zero-day exploits. With ransomware attacks becoming increasingly profitable, there's a heightened economic incentive for hackers to identify and exploit browser-based vulnerabilities.

The Challenge of Patch Management
One key factor contributing to the vulnerability surge is the challenge of timely patch management. Software developers constantly race against time, striving to identify and rectify vulnerabilities. Yet, even when patches are available, a significant proportion of users delay or even neglect updates, leaving themselves exposed.

The Intersection with Other Technologies
Modern browsers integrate with a plethora of other applications and technologies, from plugins to cloud services. Each integration point potentially introduces new vulnerabilities, making browsers a nexus of multiple security concerns.

The Road Ahead
The current threat landscape, marked by an ever-increasing number of browser vulnerabilities, demands vigilance. As the web continues to evolve and become more complex, a multifaceted, proactive approach to browser security will be paramount in safeguarding users and organizations.

The Rise of Browser-Based Attacks in Adversary Simulations
In adversary simulations conducted over the last year, a significant 30% focused on exploiting browser-based vulnerabilities. This underscores the importance and susceptibility of browsers in the current threat landscape.

Scripting Attacks: A Key Concern
Cross-site scripting (XSS) remains a dominant concern. This type of vulnerability allows attackers to inject malicious scripts, with 40% of browser-related breaches in simulations attributed to XSS exploits.

Phishing and Credential Harvesting
Phishing attacks are another major player. Cybersecurity firms noted a 25% increase in simulated phishing operations targeting browser vulnerabilities, aiming to deceive users into providing sensitive information.

Third-party Plug-ins: A Double-edged Sword
One notable vulnerability stems from third-party plug-ins. While they enhance browser functionality, they also introduce potential weak points. An estimated 15% of browser-related breaches in adversary simulations can be traced back to third-party plug-in vulnerabilities.

The Impact of Browser Sandboxing
Modern browsers use sandboxing techniques to isolate web processes, preventing malicious code from accessing critical system resources. However, sandbox escape techniques have been employed in 10% of adversary simulations, revealing gaps in this security measure.

Browser Capabilities

- User Interaction and Behavior
  - Whether it's the websites a user visits, the bookmarks they save, or the search queries they perform, browsers have an extensive record of user interaction. Queries like **SELECT url, title, visit_count FROM visits WHERE visit_count > 100;** can provide insights into the most frequently visited websites, while **SELECT keyword, COUNT(*) AS query_count FROM search_engines GROUP BY keyword ORDER BY query_count DESC LIMIT 10;** would yield the most common search terms. This data, in the wrong hands, can be used to profile users, making it a potential privacy concern.

- Sensitive Data Storage
  - Browsers often offer to save user inputs to enhance the user experience. This includes form submissions, credit card details, passwords, and even auto-fill data. However, if this data is compromised, it could have disastrous consequences for the user. For instance, **SELECT formSubmitURL, encryptedUsername, encryptedPassword FROM moz_logins WHERE formSubmitURL IS NOT NULL;** provides access to login details, and a query like **SELECT name_on_card, card_number, expiration_month, expiration_year FROM credit_cards;** can potentially expose saved credit card details.

- Engagement Metrics and Preferences
  - Browsers not only store raw user data but also compile engagement metrics. Queries like **SELECT origin, SUM(count) AS total_engagement_count, MAX(last_engagement_time_usec) AS last_engagement_time FROM media_engagement GROUP BY origin;** offer insights into user engagement with media content. These metrics can be leveraged for targeted advertising or content recommendations, but they can also be a privacy intrusion if accessed without consent.

- Potential Threat Vectors
  - The data that browsers hold can be a goldmine for malicious actors. Whether they're targeting saved passwords, looking for patterns in visited URLs, or trying to exploit downloaded files, the potential threat vectors are vast. For instance, a query like **SELECT url, target_path, start_time, end_time FROM downloads WHERE url LIKE '%malware%';** might expose downloaded files from suspicious URLs, and **SELECT guid, manufacturer, product FROM usb_devices WHERE manufacturer LIKE '%unknown%' ORDER BY connection_timestamp DESC LIMIT 5;** could unveil potentially harmful USB device connections.

- **User Sessions and Interactions**
  - Monitoring the length and frequency of user sessions can shed light on browsing habits.

```
SELECT session_id, start_time, end_time, total_duration FROM user_sessions ORDER BY
total_duration DESC LIMIT 10;
```

Extensions and Plugins Data

- Extensions can access a lot of user data. Understanding which extensions are frequently used and what permissions they have is essential.

```
SELECT extension_id, name, permissions, last_access_time FROM extensions_data WHERE permissions
LIKE '%readData%';
```

Ad Interaction and Tracking

- Advertisers heavily track user interactions with ads for targeted marketing.

```
SELECT ad_id, click_count, hover_duration FROM ads_interactions WHERE click_count > 10;
```

Accessed Files and Local Data

- Some websites allow or require users to access local files, and this can be a potential vector for vulnerabilities.

```
SELECT file_path, last_access_time FROM local_files_access WHERE file_path LIKE '%.exe%';
```

Pop-up Interactions

- Pop-ups can sometimes be a front for malicious activities.

```
SELECT pop_up_url, interaction_type, interaction_time FROM pop_ups_history WHERE interaction_type
= 'allowed';
```

Browser Notifications and Permissions

- Notifications require permissions, and tracking these can prevent potential misuse.

```
SELECT origin_url, notification_type, permission_status FROM notifications_data WHERE
permission_status = 'granted';
```

=Camera and Microphone Access Logs

- Unauthorized access to hardware components can be a serious breach.

```
SELECT origin_url, hardware_component, access_time FROM hardware_access_logs WHERE
hardware_component IN ('camera', 'microphone');
```

Google Chrome:
Popular worldwide, Google Chrome's data management has become a reference point for many. The browser stores various user-specific settings, bookmarks, extensions, and importantly, login credentials, in a "Profile" directory. These credentials are stored in a file named "Login Data".
Location of Chrome's Profile data:
- **Windows: C:\Users\<YourUsername>\AppData\Local\Google\Chrome\User Data\Default\Login Data**
- **macOS: ~/Library/Application Support/Google/Chrome/Default/Login Data**
- **Linux: ~/.config/google-chrome/Default/Login Data**

Mozilla Firefox:
Mozilla Firefox, an open-source favorite, similarly organizes its data. Firefox segregates its user data into various profiles, each containing a unique set of user data. The "logins.json" file within each profile directory holds the login credentials.
Location of Firefox's Profile data:
- **Windows: C:\Users\<YourUsername>\AppData\Roaming\Mozilla\Firefox\Profiles\<ProfileName>\logins.json**
- **macOS: ~/Library/Application Support/Firefox/Profiles/<ProfileName>/logins.json**
- **Linux: ~/.mozilla/firefox/<ProfileName>/logins.json**

Brave:
Brave Browser, recognized for its privacy-focused features, also keeps its user data in a profile directory. Like Chrome, it uses a "Login Data" file to store credentials, given that it's built on the same Chromium platform.
Location of Brave's Profile data:
- **Windows: C:\Users\<YourUsername>\AppData\Local\BraveSoftware\Brave-Browser\User Data\Default\Login Data**
- **macOS: ~/Library/Application Support/BraveSoftware/Brave-Browser/Default/Login Data**
- **Linux: ~/.config/BraveSoftware/Brave-Browser/Default/Login Data**

Opera:
Opera, while not as widely adopted as some others on this list, has been a long-standing player in the browser market. Its profile data storage, like the others, includes a specific file, "Login Data," where credentials are stored.
Location of Opera's Profile data:
- **Windows: C:\Users\<YourUsername>\AppData\Roaming\Opera Software\Opera Stable\Login Data**
- **macOS: ~/Library/Application Support/com.operasoftware.Opera/Login Data**
- **Linux: ~/.config/opera/Login Data**

Paths and Profiles Across Platforms

Windows: A Haven of Browser Diversity
Windows, with its vast user base, naturally supports a multitude of browsers. Here's where each browser stashes its user data:

1. **Google Chrome & Its Siblings**:
   - Chrome: **homeDir + "/AppData/Local/Google/Chrome/User Data/Default/"**
   - Chrome Beta: **homeDir + "/AppData/Local/Google/Chrome Beta/User Data/Default/"**
   - Chromium (Open-source variant of Chrome): **homeDir + "/AppData/Local/Chromium/User Data/Default/"**
2. **Microsoft Edge**:
   - **homeDir + "/AppData/Local/Microsoft/Edge/User Data/Default/"**
3. **Brave** (Privacy-focused, built on the same engine as Chrome):
   - **homeDir + "/AppData/Local/BraveSoftware/Brave-Browser/User Data/Default/"**
4. **Asian Market Dominants**:
   - 360 Speed Browser: **homeDir + "/AppData/Local/360chrome/Chrome/User Data/Default/"**
   - QQ Browser (Popular in China): **homeDir + "/AppData/Local/Tencent/QQBrowser/User Data/Default/"**
   - Sogou (A notable Chinese search engine's browser): **homeDir + "/AppData/Roaming/SogouExplorer/Webkit/Default/"**
5. **Opera & Its Gaming Variant**:
   - Opera: **homeDir + "/AppData/Roaming/Opera Software/Opera Stable/"**
   - Opera GX (A version of Opera designed for gamers): **homeDir + "/AppData/Roaming/Opera Software/Opera GX Stable/"**
6. **Others**:
   - Vivaldi (A highly customizable browser): **homeDir + "/AppData/Local/Vivaldi/User Data/Default/"**
   - Coc Coc (Tailored for the Vietnamese audience): **homeDir + "/AppData/Local/CocCoc/Browser/User Data/Default/"**
   - Yandex (Russian multinational specializing in Internet-related products): **homeDir + "/AppData/Local/Yandex/YandexBrowser/User Data/Default/"**
   - DC Browser: **homeDir + "/AppData/Local/DCBrowser/User Data/Default/"**
7. **Mozilla Firefox**:
   - Unlike other browsers which have a single profile directory, Firefox organizes its data into various profiles. The profile root is located at **homeDir + "/AppData/Roaming/Mozilla/Firefox/Profiles/"**

Linux: The Open-Source Paradise
Linux, known for its customizability and open-source nature, also supports a myriad of browsers.

1. **The Chrome Family**:
   - Chrome: **homeDir + "/.config/google-chrome/Default/"**
   - Chrome Beta: **homeDir + "/.config/google-chrome-beta/Default/"**
   - Chromium: **homeDir + "/.config/chromium/Default/"**
2. **Brave**:
   - **homeDir + "/.config/BraveSoftware/Brave-Browser/Default/"**
3. **Microsoft Edge**:
   - **homeDir + "/.config/microsoft-edge/Default/"**
4. **Opera**:
   - **homeDir + "/.config/opera/Default/"**
5. **Vivaldi**:
   - **homeDir + "/.config/vivaldi/Default/"**
6. **Mozilla Firefox**:
   - **homeDir + "/.mozilla/firefox/"**

Darwin (macOS): Apple's Unix-Based OS
macOS, with its unique blend of user-friendliness and Unix power, places user data within the "Library" directory.

1. **The Chrome Lineage**:
   - Chrome: **homeDir + "/Library/Application Support/Google/Chrome/Default/"**
   - Chrome Beta: **homeDir + "/Library/Application Support/Google/Chrome Beta/Default/"**
   - Chromium: **homeDir + "/Library/Application Support/Chromium/Default/"**
2. **Brave & Edge**:
   - Brave: **homeDir + "/Library/Application Support/BraveSoftware/Brave-Browser/Default/"**
   - Edge: **homeDir + "/Library/Application Support/Microsoft Edge/Default/"**
3. **Opera Variants**:
   - Opera: **homeDir + "/Library/Application Support/com.operasoftware.Opera/Default/"**
   - Opera GX: **homeDir + "/Library/Application Support/com.operasoftware.OperaGX/Default/"**
4. **Others**:
   - Vivaldi: **homeDir + "/Library/Application Support/Vivaldi/Default/"**
   - Coc Coc: **homeDir + "/Library/Application Support/Coccoc/Default/"**
   - Yandex: **homeDir + "/Library/Application Support/Yandex/YandexBrowser/Default/"**
   - Arc Browser: **homeDir + "/Library/Application Support/Arc/User Data/Default"**
5. **Mozilla Firefox**:
   - **homeDir + "/Library/Application Support/Firefox/Profiles/"**

Analyzing Important Tables and Columns

Google Chrome
Google Chrome, being the world's most popular browser, has a myriad of tables that store user data. Here are the most pivotal ones:

- **Logins**: This table stores saved website login credentials. Columns such as **action_url**, **username_value**, and **password_value** provide the website's URL, the saved username, and the saved password, respectively.
- **Autofill**: As the name suggests, this table contains data related to the browser's autofill functionality. The **name** and **value** columns capture the autofill data for forms and fields.
- **Cookies**: It captures stored browser cookies. The **host_key**, **name**, and **value** columns contain details about the cookies' origin website, their names, and values.
- **Bookmarks**: This table contains information on user bookmarks. **url** and **title** columns provide the URL and title of the bookmarked page.
- **History**: Holds browsing history data. The **url** and **title** columns detail the websites visited and their respective titles.
- **Downloads**: A repository of downloaded file records. The **url** and **target_path** columns shed light on the source URL of the download and the location it was saved to.
- **Extensions**: Lists the browser extensions installed. The **name** and **permissions** columns describe the extension's name and the permissions it has.
- **Media Engagement**: Stores data regarding media engagement. The **origin** and **last_engagement_time_usec** columns highlight the website's origin and the last time media was engaged.

...and many more. For brevity, not all tables are detailed, but Chrome has tables capturing data from USB devices, search engines, form data, local storage, etc.

Firefox
Firefox, an open-source browser by Mozilla, similarly has numerous tables critical to forensic investigations:

- **moz_logins**: Contains saved website logins. Columns like **formSubmitURL**, **hostname**, **encryptedUsername**, and **encryptedPassword** provide details about the website and encrypted login credentials.
- **moz_autofill**: Houses autofill data. The **name** and **value** columns depict the autofill form data.
- **moz_cookies**: Contains stored browser cookies. **host**, **name**, and **value** columns describe the cookie's host website, name, and value.
- **moz_bookmarks**: Holds bookmark data. **url** and **title** columns detail the bookmarked URL and title.
- **moz_historyvisits**: Focuses on user browsing history. **from_visit**, **place_id**, and **visit_date** provide data on website visits, the place ID, and the visit date.

...among others. Firefox tables also contain data on user extensions, search history, downloaded files, etc.

Microsoft Edge
Microsoft's Edge browser, though it has a foundation in Chrome's Chromium project, has its unique tables:

- **Logins**: Similar to Chrome, it contains saved login credentials. The **action_url**, **username_value**, and **password_value** columns provide data on the website's URL and saved login details.
- **Autofill**: Stores the browser's autofill data. Columns **name** and **value** depict the autofill data for forms.
- **Cookies**: Like other browsers, it captures stored browser cookies. Columns **host_key**, **name**, and **value** offer insights into the cookie's host, name, and value.

...and more. Edge, similar to Chrome, captures data on user bookmarks, browsing history, extensions, and other user activities.

Browser	Table Name	Column Name(s)	Description	Potential Attack Surface
Chrome	logins	action_url, username_value, password_value	Saved website logins and passwords	Credential theft, impersonation
Chrome	autofill	name, value	Autofill data for forms and fields	Data harvesting
Chrome	cookies	host_key, name, value	Stored browser cookies	Session hijacking
...	...	...	...	...
Firefox	moz_logins	formSubmitURL, hostname, encryptedUsername, encryptedPassword	Saved website logins and passwords	Credential theft, impersonation
Firefox	moz_autofill	name, value	Autofill data for forms and fields	Data harvesting
Firefox	moz_cookies	host, name, value	Stored browser cookies	Session hijacking
...	...	...	...	...
Edge	logins	action_url, username_value, password_value	Saved website logins and passwords	Credential theft, impersonation
Edge	autofill	name, value	Autofill data for forms and fields	Data harvesting
Edge	cookies	host_key, name, value	Stored browser cookies	Session hijacking
...	...	...	...	...

Table 1 - Possible Attacks

The Challenge of Patch Management

This report provides a technical analysis of a vulnerability identified in Firefox version 102.8 on Linux where the browser goes into an infinite loop of opening tabs, leading to a potential Denial-of-Service (DoS) scenario.

Vulnerability Overview
Name: Infinite Tab Loop Vulnerability
Affected Version: Firefox 102.8 on Linux
Impact: Browser crash, potential data loss
Vulnerability Type: Denial-of-Service (DoS)

Technical Details
The vulnerability manifests itself when the firefox-trunk launcher file, provided by Ubuntu, is set as the default opener application. If a user is tricked into opening a file with a specific pattern, such as a .patch file (though other file types might also be vulnerable), the browser goes into an infinite loop, continuously opening tabs.

Vulnerability Root Cause Analysis
The primary cause of this vulnerability seems to reside in how the Ubuntu-specific firefox-trunk launcher script interacts with Firefox's file handling. When a file is attempted to be opened, the script may unintentionally invoke a new instance of Firefox, rather than passing the file to an already opened instance.

The problem is exacerbated by potential misconfigurations in the xdg-mime system, a MIME type database for desktop environments on Linux. If the MIME type for .patch files is set to open with Firefox by default, it triggers the infinite loop.

The xdg-mime utility is a part of the xdg-utils suite on Linux systems, which assists in managing MIME types and their associated default applications. When a file type, like an RSS feed, is to be opened, xdg-mime determines the default application set to handle it.

Under specific circumstances, when Firefox is set as the default handler for certain RSS or Atom files and such a file is malformed or not correctly validated, an infinite loop scenario is triggered. When attempting to process the file, Firefox refers to xdg-mime, which in turn redirects back to Firefox, leading to endless tab openings until Firefox becomes unresponsive.

Firefox is set (either by user action or misconfiguration) as the default handler for .rss or .atom files.
A user tries to open a malformed or unvalidated .rss or .atom file.
Firefox defers to xdg-mime to determine the file's handler.
xdg-mime identifies Firefox as the handler.
Firefox attempts to open the file in a new tab.
Due to the file's malformed nature, Firefox again queries xdg-mime.
Steps 3-6 repeat indefinitely.

Here's a simplified, conceptual assembly snippet illustrating the loop:

```
start:
  CALL load_file ; Load the RSS or Atom file
  CALL check_file_validity ; Validate the file format
  CMP AL, invalid ; Check if file is invalid
  JZ query_xdg_mime ; If file is invalid, query xdg-mime

query_xdg_mime:
  CALL check_xdg_mime ; Ask xdg-mime for file handler
  CMP AL, firefox ; Check if Firefox is the handler
  JZ open_tab ; If yes, jump to open_tab

open_tab:
  OPEN new_tab ; Open the file in a new tab
  JMP start ; Loop back to start
```

Here are some commands illustrating the interaction:

Set Firefox as the default handler for .rss files:

xdg-mime default firefox.desktop application/rss+xml

Attempt to open a malformed RSS file:

xdg-open malformed.rss

Kill Chain
Reconnaissance: Attacker identifies the victim is using the vulnerable version of Firefox on Linux.
Weaponization: Prepare a .patch file, potentially named sample.atom either with malicious content or leave it empty.
Delivery: Send the .patch file to the victim via email, chat, or any other medium.
Exploitation: Instruct or trick the victim into opening the sample.atom file using Firefox.
Installation: Not applicable for this attack.
Command & Control: Not applicable for this attack.
Actions on Objectives: The browser crashes due to resource exhaustion.

Exploitation
The attacker needs to:

Prepare a file, like "sample.atom" or a .patch file.
Trick the victim into downloading the file.
Instruct the victim to open this file using their Firefox browser.
Command to check the current default handler for .patch files:

xdg-mime query default text/x-patch

And for reproduce the exploitability:

```
# Create a sample .patch file
echo "This is a sample patch file." > sample.atom

# Instruct the victim to open this file using Firefox
firefox sample.atom
```

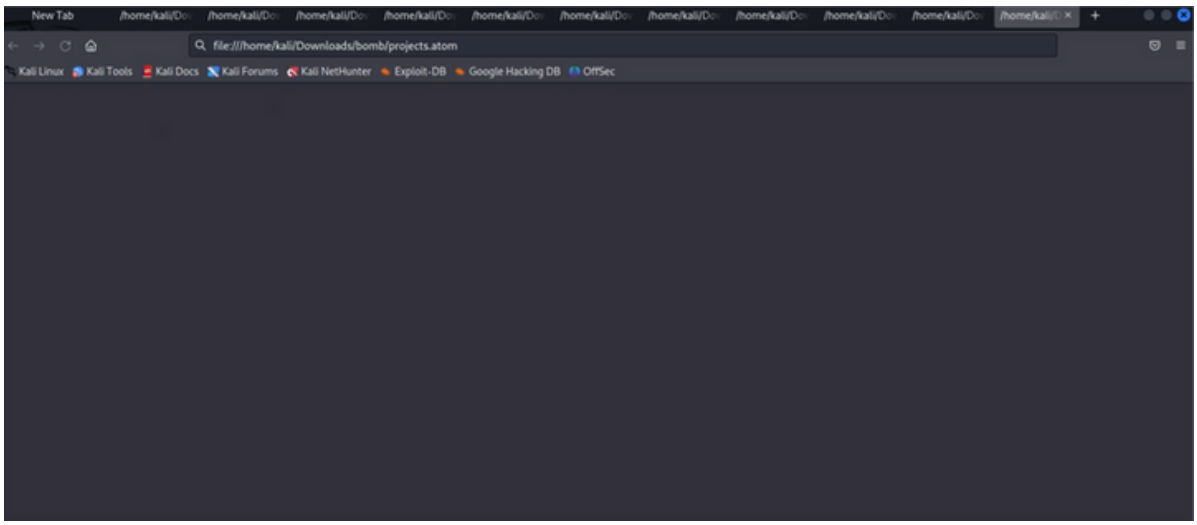

Image 1 - Infinite tab opening loops can happen on Linux

Remediation
To temporarily address this issue:

Open Firefox and navigate to about:preferences.
Under Applications, find the .patch file type or "differences between files".
Change the action from "Open in Firefox" to "Text Editor" or any other preferred application.

Defense in Depth

1. Continuous Discovery of Vulnerabilities
Vulnerabilities in browsers are discovered almost daily. These vulnerabilities can range from minor ones with little impact to severe zero-day vulnerabilities that can be exploited as soon as they're discovered.

Example: The command below demonstrates how to query the National Vulnerability Database (NVD) for known vulnerabilities related to Firefox:

```
curl https://nvd.nist.gov/feeds/json/cve/1.1/nvdcve-1.1-recent.json.gz | gunzip | jq
'.CVE_Items[] | select(.cve.Affects.sw[].sw_cpe.uri:contains("firefox"))'
```

2. Complexity of Browsers
Modern browsers are no longer just tools to view web pages; they are complex software that supports web apps, extensions, and plugins. This complexity increases the chances of vulnerabilities.

Example: To check for outdated plugins in Firefox, you can navigate to about:plugins. Any outdated plugin can be a potential security risk.

3. Dependency on Third-party Libraries
Browsers often rely on third-party libraries. If any of these libraries have vulnerabilities, it affects the browser too.

Command:
To check shared library dependencies of a program, such as Firefox:

```
ldd /path/to/firefox-bin
```

4. Diverse User Base with Different Needs
Not all users can apply patches immediately due to custom configurations, extensions, or integration with enterprise systems. Ensuring patches don't disrupt user configurations is challenging.

5. Deciding What to Patch
Sometimes, applying a patch to fix one problem might introduce another. Deciding what to patch and testing patches are resource-intensive tasks.

Example:
Before applying a patch, you might want to test it in a staging environment first. Using Docker can help:

```
docker run -d --name firefox-test -v /path/to/patched/firefox:/firefox ubuntu:latest
/firefox/firefox
```

6. Automatic Updates vs. User Consent
While automatic updates ensure users get the latest patches immediately, they may also disrupt work or change browser behavior. Getting the balance right between auto-updates and user consent is tricky.

Command:
To disable automatic updates in Firefox via about:config, you can set the app.update.auto preference to false.

7. Managing Legacy Systems
Older systems that don't support newer browser versions pose a significant challenge, as they might be left unpatched and vulnerable.

8. The Threat of Malicious Patches
There's always a risk that threat actors could introduce malicious patches. Ensuring the integrity of patches is crucial.

Example:
To verify the integrity of a downloaded Firefox patch:

```
sha256sum firefox-patch.tar.gz | grep <expected_checksum>
```

Conclusion

The vulnerability stemming from the interaction between Firefox and xdg-mime regarding malformed RSS and Atom files underscores the significance of rigorous MIME type management and inter-application coordination in modern computing systems. An infinite loop, as illustrated in this case, can not only disrupt the user experience but can also lead to potential exploitation avenues for malicious actors. Proper file validation, an understanding of the implications of default application settings, and periodic review of system configurations are paramount in mitigating such issues. This specific vulnerability serves as a poignant reminder of the intricate interdependencies within software ecosystems and the continuous vigilance required to maintain their security and stability.

HADESS

cat ~/.hadess

"Hadess" is a cybersecurity company focused on safeguarding digital assets and creating a secure digital ecosystem. Our mission involves punishing hackers and fortifying clients' defenses through innovation and expert cybersecurity services.

Website:

WWW.HADESS.IO

Email

MARKETING@HADESS.IO

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.