Leaking Browser URL/Protocol Handlers

Rotem Kerner, Research Team

Affected platforms:	Windows, Linux
Impacted parties:	Any
Impact:	Leaking sensitive data
Severity level:	High

An important step in any targeted attack is reconnaissance. The more information an attacker can obtain on the victim the greater the chances for a successful exploitation and infiltration. Recently, we uncovered two information disclosure vulnerabilities in Firefox, Edge and Chrome which can be leveraged to leak out a vast range of installed applications on a victim machine, including major security vendors, allowing a threat actor to gain critical insights on the target.

In this post we will discuss what are protocol handlers and disclose two information disclosure vulnerabilities in Firefox, Edge and Chrome which could enable an attacker to leak what Protocol (or URL) handlers, and thus applications, exist in a targeted system. Additionally, we will discuss how attackers can leverage such vulnerabilities.

Overview - What Are Protocol Handlers?

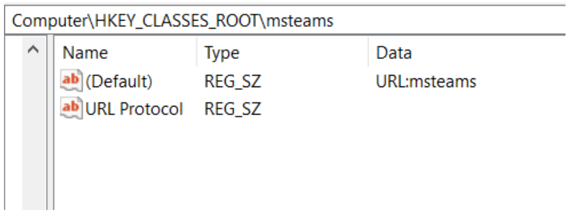
Generally speaking when talking about Protocol Handlers we are referring to a mechanism which allows applications to register their own URI scheme. This enables the execution of processes through the use of URI formatted strings.

The Windows OS manages custom URL handlers under the following key-

- HKEY_CURRENT_USER\SOFTWARE\Classes\*
- HKEY_LOCAL_MACHINE\SOFTWARE\Classes\*
- HKEY_CLASSES_ROOT*

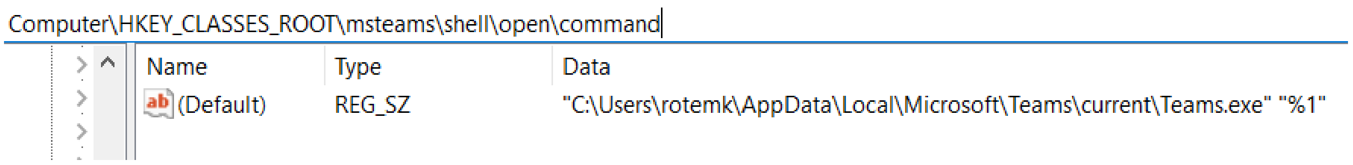
When a URL Handler is invoked the OS is searching within those locations for keys containing values with the name *"URL Protocol"*.

For instance, we can use *regedit* to inspect the path at *HKEY_CLASSES_ROOT\msteams* and see that it contains the special Value of *"URL Protocol"*.


Computer\HKEY_CLASSES_ROOT\msteams

Name	Type	Data
(Default)	REG_SZ	URL:msteams
URL Protocol	REG_SZ	

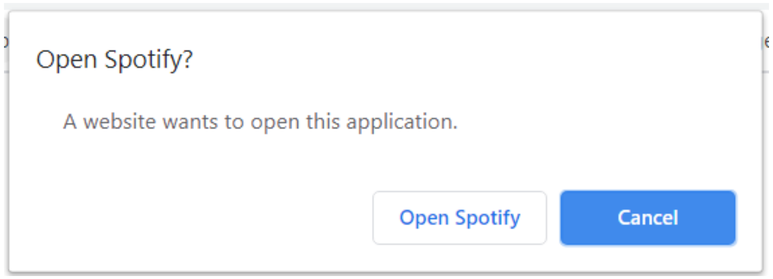
Looking further into *HKEY_CLASSES_ROOT\msteams\shell\open\command\* we can see the actual command that gets invoked -



```
Computer\HKEY_CLASSES_ROOT\msteams\shell\open\command
Name        Type      Data
(Default)   REG_SZ    "C:\Users\rotemk\AppData\Local\Microsoft\Teams\current\Teams.exe" "%1"
```

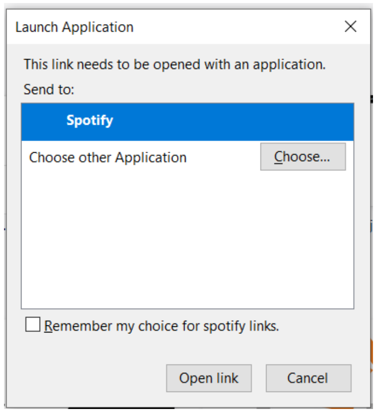
In this example the browser will launch Teams.exe when a url that starts with “msteams” is clicked.

Web browsers will enable their users to click on links with non-http schemes which will result in prompting the user with a message message box asking them if they want to let another application handle this URL.



Open Spotify?

A website wants to open this application.



Launch Application

This link needs to be opened with an application.

Send to:

Spotify

Choose other Application

Remember my choice for spotify links.

Though it requires user interaction and thus poses a limited risk, it expands the attack surface beyond the browser borders. An attacker could craft a special web page which triggers another potentially vulnerable application. In some cases such attacks may bypass protection measures such as Smart Screen and other security products.

While exploring the potential of attacking the browsers through the different protocol handlers I got curious as to whether web browsers somehow disclose what protocols handlers exist on a targeted system. The short answer is yes.

Leaking Protocol Handlers

In this section we disclose how both Chrome and Firefox were circumvented in order to disclose which protocol handlers exist on a targeted system. It's worth mentioning that these findings are the result of manually playing with HTML/CSS components with the emphasis on finding a difference in behaviour when referring(using some elements) to existing and non-existing URL handlers.

The environment I’ve been testing on is Windows 10 but it is fair to assume that the same vulnerabilities exist on other platforms (such as Linux and Mac).

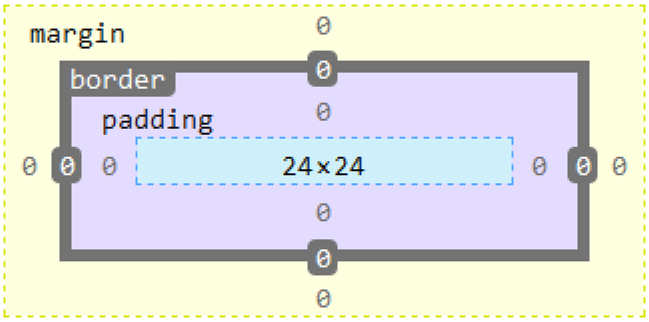
Leaking Firefox protocol handlers (CVE-2020-TBD)

This vulnerability has been tested on Firefox 78.0.1 (64-bit) under Windows 10. To leak the protocol handlers in Firefox we leverage differences in the way firefox renders images sourced from existing and non-existing protocol handlers.

For example, if we will try to load a web page containing the following element -

`<img src=”search-ms://abc”>`

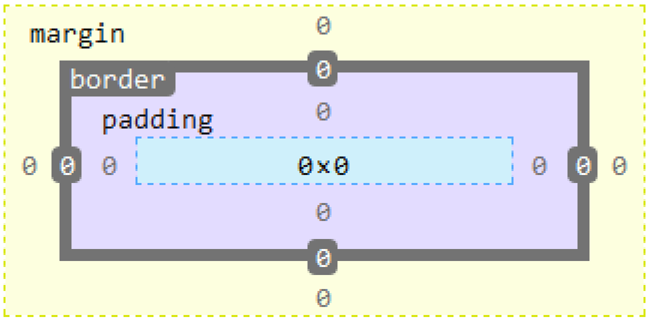
And observe the elements styling using developer tools we would see that the default styling for broken images generate element with size of 24x24 as can be seen in Figure-TBD



Unlike the example above, if we try and create an image element and set source to some non-existent handler like the following -

`<img src=”NonExistingHandler://abc”>`

This will result with an element with different sizing of 0x0 as can be seen in Figure-TBD



This difference can be measured using a simple JS script Basing on this a malicious actor may perform a brute-force attack to disclose the different protocol handlers on a targeted system.

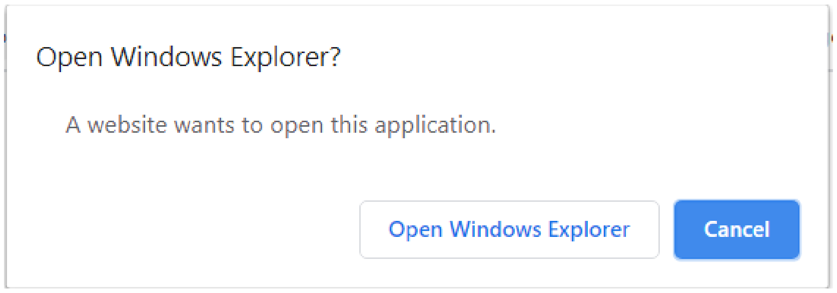
The following example code will print whether a handlers exists or not on a targeted system -

```
known_handlers = [
      ...
]

for (var i = known_handlers.length - 1; i >= 0; i--) {
       handler_id = 'handler_' + i
       $('body').append('<img id="' + handler_id +'" src="'+known_handlers[i] +
      '://192.168.133.142/"></img>')
       if($('#' + handler_id).css('width') == "24px"){
            $('<p>Handler ' + known_handlers[i] + ' 
      Exists</p>').appendTo('#logbox');
       }
       else{
            $('<p>Handler ' + known_handlers[i] + ' Does not 
      exists</p>').appendTo('#logbox');
       }

       $('#' + handler_id).remove()

}
```

Leaking Chrome & Edge protocol handlers (CVE-2020-TBD, CVE-2020-TBD)

This vulnerability has been tested on Chrome 83.0.4103.116 under Windows 10. The exploitability of this vulnerability may be less stealthy but still yields equivalent results as the Firefox vulnerability.

The mechanism here was different then the one in Firefox, here we leverage the fact that the window lose focus whenever the user is challenged with the message box as can be seen in figure-TBD


Open Windows Explorer?

A website wants to open this application.

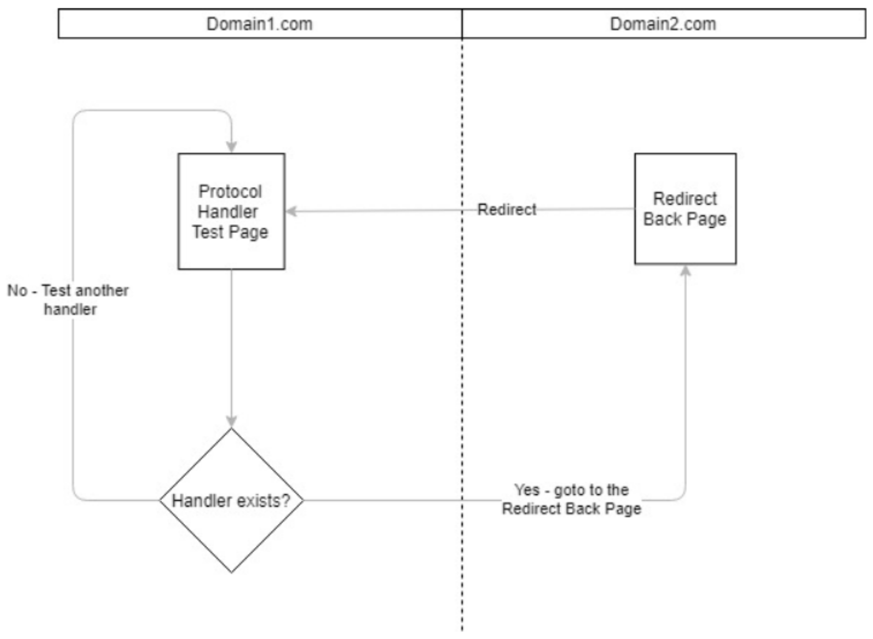
So in order to detect if a given handler exists on the victim we take the following steps.
First we dynamically generate a link that is made of the scheme we would like to detect like such -

```
<a href="MyHandler://test" id="handler-link">link</a>
```

Then we trigger the link and detect whether the document has focus:

```
document.getElementById('handler-link').click();
handler_exists = !document.hasFocus();
```

That will work for a one time check however if we would like to brute force an entire list of handlers we would have to get rid of the message box every time it pops up or else the `document.hasFocus()` will always return true.


Domain1.com
Domain2.com
Protocol Handler Test Page
Redirect
Redirect Back Page
No - Test another handler
Handler exists?
Yes - goto to the Redirect Back Page

The technique we came up with was to redirect the user to an entirely different domain/ip which will eliminate any previously opened message box.

Figure-TBD draws the general idea of how the flow should be carried out in order to work. *Protocol Handler Test page* performs the actual test and saves the results to the back-end. In case the handler exists it will redirect to "*Redirect-Back Page*" which exists on domain2.com. The redirection will get rid of the message box. Finally back to the *Protocol Handler Test Page* for the next handler test.

Vulnerabilities Impact

Such information disclosure vulnerability could be exploited in several different ways. Here are some examples:

- **Identifying communication channels**: By listing the handlers an attacker can get a hint to what platforms he may use for reaching the targeted user. For instance, detecting social applications such as slack, skype, whatsapp or telegram may be used for communicating with the target.
- **Pre-exploitation detection**: Exploit kits may leverage this information in order to identify if a potentially vulnerable application is present without exposing the vulnerability itself.
- **Detecting Security solutions**: Table-TBD contains a list of some security solutions whose presence can be exposed by leveraging the vulnerabilities because they have custom protocol

handlers installed. Attackers may use this to further customize their attack to be able to circumvent any protection mechanism set by those security solutions.

- **User Fingerprinting**: reading what protocol handlers exist on a system may also be used in order to improve browser/user fingerprinting algorithms.

Vendor	Protocol Handler
GData	GDataGDATAToastNews://
MalwareBytes	malwarebytes://
Avast	avastpam://
TrendMicro	vizorwebs://, tmtb://, titanium://, vizorweb://
BitDefender	bdlaunch://

Summary

In this post we uncovered a new type of information disclosure vulnerabilities in most major browsers and identified how an attacker can leverage them into valuable insights which could assist them in compromising their targets.

When browsers are enabling the interaction with other applications through URL handlers, they may be easing the engagement with third party software, but they also enable a wider attack surface giving the attacker a chance to attack the user through other applications.

Browsers security has advanced a lot in the last few years, making it harder for attackers to find vulnerabilities. Because of this, we anticipate that in the near future we shall see an increase in the number of attacks which exploit the different url handlers through the user’s web browser.