mango.pdf.zone

hard mode

"work" by the hacker known as "Alex" / @mangopdf

Stealing Chrome cookies without a password

Sep 26, 2018

If you steal someone's Chrome cookies, you can log in to their accounts on **every website** they're logged in to.

Normally you need the user's password to do it, but I found a way to do it without the password. You just need to be able to execute code on their computer. It works by using Chrome's Remote Debugging Protocol.

If you wanna skip this dumb blog post and just get the demo code, here ya go: https://github.com/defaultnamehere/cookie_crimes. There's also a Metasploit module. Don't spend 'em all at once.

For how it works and how I found it, you need only *find within yourself the strength to scroll this page*.

Step 0: Reconsider your life choices

Imagine that, *for some reason*, you've hacked someone's computer. I dunno maybe you're like a spy or something?

Let's call your fictitious victim, uh, "Naruto".

Specifically, you've got the ability to execute code on Naruto's computer. That's like, probably hacking, and you're already going to be slam jammed into the shadow realm for your crimes, so ya may as well do some more.

One obvious crime you might want to do with this access is to steal Naruto's Chrome cookies. This would let you log in as him to anything he's logged in to. Oooooh yeah definitely. Imagine the trouble you could get up to with *that*.

How do cookies work?

You know how you don't have to log in every time you go to Facebook? How when you go to `facebook.com` it just shows you your timeline? How does Facebook know it's you?

It's because when you log in to Facebook, Facebook gives you a cookie, which lives on your computer. Next time you go to Facebook, you just *show* it that cookie, and it lets you in without having to type your password again.

It's like those wristbands they give you at the club to show you're of drinking age, except I'm not sure if they still do that because I haven't been to an Event since the release of Stardew Valley.

And if someone were to....*sideways look* *steal* those cookies?

For example, if you had Naruto's Facebook cookies, it doesn't matter whether he has a really good password, 2 Factor Authentication, or is particularly good mates with zucc and the boys.

You can just put Naruto's Facebook cookie in you own browser, and go to `facebook.com`, and you'll see *Naruto's* Facebook account logged in.

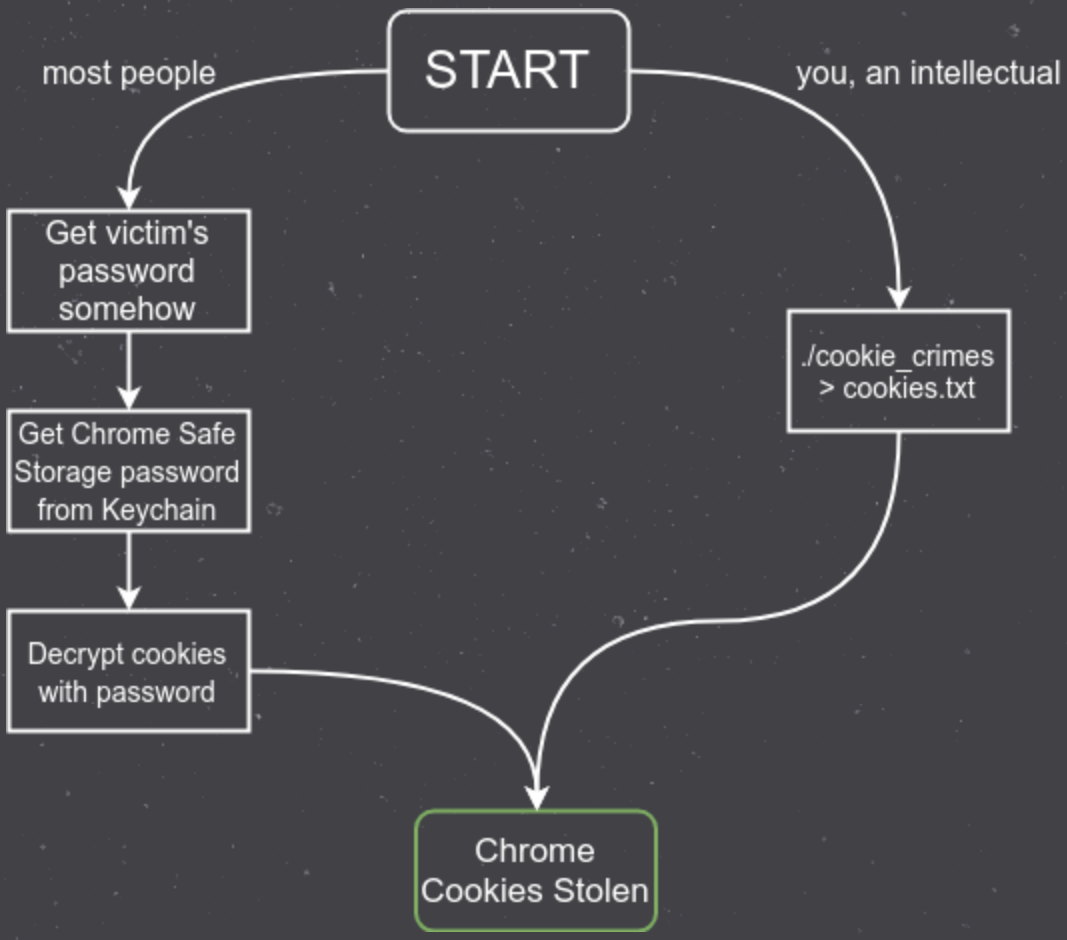
It's as easy as that.

Chrome's cookie security

Let's assume Naruto, the absolute chap, has OS X - but this works on Windows and Linux too.

Naruto's delicious, tasty Chrome cookies live in a file on his computer. Cleverly, Chrome encrypts his cookies with its own "Chrome Safe Storage" password. This password lives in Naruto's Login Keychain, so the only way to get the cookies is to unlock the keychain by typing in his password.

This is a perfectly legit way to get someone's cookies, but the reason we're here today is I'm pretty sure I found a way to skip all that.



Why should we have to decrypt the cookies? After all, Naruto can get his own cookies from Chrome without typing his password. Surely there's a way to just get Chrome to give us the cookies, if we ask nicely?

Step 1: Run Headless Chrome

Chrome can run in headless mode now. This means it can run without actually having a window displayed. Everything happens in the terminal. This is deeply powerful and probably going to all go wrong one day.

User data directories

Chrome stores your cookies, history, deepest secrets, etc. in a `user-data-dir`. By default (if you have no Chrome Profiles), this will be `$HOME/Library/Application Support/Google/Chrome/`.

Needless to say, this directory is The Good Stuff, and we want to be extremely up in it.

Enabling remote debugging

Because we live in a dystopian future, Chrome allows you to remotely control it for debugging and automation reasons.

We're going to set up a headless Chrome window, using the same `user-data-dir` as our victim.

This is apparently not possible according to Google.

> *"two running Chrome instances cannot share the same user data directory"*

What, you're not gonna try it just because a *webpage* told you not to? Of course we are.

Here's what to run:

```
/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome \
--headless \
--user-data-dir="$HOME/Library/Application Support/Google/Chrome/" \
https://gmail.com \
--remote-debugging-port=9222
```

This sets remote debugging listening on `localhost:9222` (the default port for Chrome Remote Debugging, now *that's* stealthy). It won't open a

browser window, either. We need to tell headless Chrome browse to something so we can open a tab and start debugging it. You can replace `gmail.com` with anything you like.

At this point, you can view any page as if Naruto opened it in a new tab (and so, see all his Facebook messages, in our example). You can do that by adding `--dump-dom` to the command above, which will print out the HTML of the page you ask for (`gmail.com` above). But that seems a bit tedious, not very stealthy, and we can do better than that if we just *believe*.

Normally, you're supposed to like, open a *second* Chrome, point it at the first Chrome, and use it to debug Chrome itself. Ya we're not gonna do that. We're gonna remotely debug Chrome with `curl`.

C'mon don't act like this isn't absolutely *wild*. There are some things humans were just never meant to do.

Step 2: Remotely debug Headless Chrome

Through furious and intense googling, I learned that it might be possible to get cookies out of Chrome by remotely debugging it. So, I set off to learn how Chrome's Remote Debugging works.

Basically, that remote debugging port (`localhost:9222`) is waiting for you to speak some janky websocket protocol, in which you can issue commands to Chrome. If you know how to say the command, Chrome will just.... do it.

Sidenote: The Chrome Dev Tools are absolutely Loose

When I was trying to figure out whether this was possible, I stumbled across this page: https://chromedevtools.github.io/devtools-protocol/

This is some documentation for the Protocol used by the Chrome Developer Tools (`Right Click > Inspect Element`, our trusted friend and ally). Presumably, on this page, some nice people at Google are graciously going to explain to me, an Outsider, how to use their Ancient and Powerful magic.

They say "hey, if you want to remotely debug Chrome, you need to speak the Remote Debugging Protocol." I'm ready for this page to teach me about how to remotely debug Chrome, but it says "we don't release that kind of information to punks like you".

They casually tell you to sniff and reverse-engineer the protocol if you want to know how it works, since the only client that speaks it is the Chrome Dev Tools themselves. The way they tell you to sniff it is using the Chrome Dev Tools to debug the *CHROME DEV TOOLS THEMSELVES*, and view the websocket data that the *first* set of Chrome Dev Tools is sending. The ease with which they suggest this solution is quite stressful.

Finally, they have a section called "How is the protocol defined?", which has two links to json files you're supposed to read, both of which 404.

How the heck does Remote Debugging work?

According to the Cursed Document linked above, there's a secret, barely documented `/json` endpoint on the remote debugging port. Let's view it.

In a new terminal window, run

```
$ curl -s localhost:9222/json

[ {
"description": "",
"devtoolsFrontendUrl": "/devtools/inspector.html?
ws=localhost:9222/devtools/page/7404BF41DC4E7512E0431577BABCE18A",
```

```
"id": "7404BF41DC4E7512E0431577BABCE18A",
"title": "about:blank",
"type": "page",
"url": "about:blank",
"webSocketDebuggerUrl":
"ws://localhost:9222/devtools/page/7404BF41DC4E7512E0431577BABCE18A"
} ]
```

See that `webSocketDebuggerUrl`? That's what we want.

Step 3: Issue the command over the websocket protocol

With the magic URL we got in the previous step, we can now speak the Ancient Language to control Chrome.

First we'll need something that can speak the websocket language. You can use anything, but I googled for some tool called `wsc`. In the proof-of-concept (which I made just for you) the whole thing is done in Python, so you don't need this.

Speak the Ancient Language

Ready? Let's do it

Using `wsc`:

```
wsc
ws://localhost:9222/devtools/page/7404BF41DC4E7512E0431577BABCE18A
Connected to
ws://localhost:9222/devtools/page/7404BF41DC4E7512E0431577BABCE18A
>
```

Oooh. Looks like a command prompt. If we knew what to type here, we could tell Chrome what to do.

I'm going to save you the 50 open Chrome tabs, reading of Ruby from 2012 on GitHub, and discovery of the very bug report which created this feature and just show you the command.

```
>{"id": 1, "method": "Network.getAllCookies"}
[{
"domain": "mail.google.com",
"expires": -1,
"httpOnly": false,
"name": "GMAIL_AT",
"path": "/mail/u/0",
"secure": true,
"session": true,
"size": 42,
"value": <unencrypted cookie value appears here>
},
...<the rest of the cookies>
```

Because we asked nicely, Chrome just *gives* us the cookies. This bypasses the whole Chrome Safe Storage password thing because Chrome itself decrypts the cookies.

And we did it all without needing to become the root user, or otherwise know Naruto's password.

You can plug these cookies into a Chrome Extension (for example, EditThisCookie), and you'll be logged in to Naruto's gmail if you just go to `mail.google.com` in your browser.

Aaaaaand that's it. Crimes successful. Directed by Quentin Tarantino.

If you want to try this at home, here's the code to just straight up do it for you: https://github.com/defaultnamehere/cookie_crimes. If you want to try using it on someone else, maybe this is the website for you.

Prevention

If you would not like this attack being used against you, I can wholeheartedly recommend not letting someone else execute code on your computer.

The industry-standard best practice is to rapidly acquire blessed amulets, mysterious crystals of unknown but surely spicy origin, and/or pay-as-you-go racketeering schemes to prevent the aforementioned execution of code.

Failing that, I guess there's also:

Channel-binding cookies

The good folks online have come up with a way to make stealing cookies harder. It's called Channel-Bound Cookies. This lets websites ask you to prove that you have a special Token Binding Key before you can use cookies on that website. This means that if you want to use cookies you stole from someone's machine, you also have to steal their Token Binding Key, and use it to impersonate their browser.

This feature is in Google Chrome, but disabled by default, and being removed if I have understood the heated discussion in this thread. The feature is also in Edge (which is what Internet Explorer is called now, post Identity Crisis).

Even if Chrome used it, most websites don't use Channel-binding. This is because, well, it's not much harm to the website if Naruto's cookies get used by someone else. That's more Naruto's problem, in their eyes.

Detection

In theory, people can detect the theft of cookies. Google, for instance, knows that they gave the Gmail cookie above to Naruto. They can also know, that you, with a different browser, OS, and IP address, might not be Naruto. They can also detect your channel-binding errors. But hey, I haven't seen this technique fail because of this so far.

FAQ

Wait so you have to already be running code on someone's computer for this to work? That's not a big deal at all!

I mean yeah pretty much you're right. It's not that big a deal. Nobody panic. Everybody stay cool. The kinds of people who are stealing people's browser cookies are just gonna have an easier time since there's no more decrypting.

Why is this even good then?

- You don't need to know someone's password to do it (unlike other methods)
- It's simple (one command to run)

I'm pretty sure this is the best way of getting Chrome cookies once you're in someone's computer. I sure wouldn't bother with any other method.

Are you going to tell Google about this critical security flaw?

Nah, they know about it. It's a feature of Chrome, after all. I even saw them deciding to add it.

UPDATE: I told 'em about it juuuuuust in case.

> *I have to say this is working as intended. The remote debugging protocol is meant to provide full access, including cookies, and running Chrome with a flag makes it work.*
>
> *I am surprised cookies can be read from a headful Chrome profile by the headless Chrome. We have plans to make profiles inter-operable, but that didn't happen yet. Maybe cookies are supported though, I didn't look too close.*

Thanks Google!

Isn't it kinda irresponsible to publish this outta nowhere?

see you in hell i guess

My head hurts a little and I feel tired.

Maybe you're dehydrated. Try drinking some crisp cool water.

Does this work on Firefox or *shifty eyes* Internet Explorer/Edge?

Dunno, I haven't tried it and I have no idea how these browsers work. *You* could try finding out.

UPDATE: Ya boi wunderwuzzi23 did this for Firefox and Microsoft Edge.

Hey so how did you end up finding this? Like why did you need to get someone's Chrome cookies in the first place?

Thanks for taking the time to read this blog post.

I wrote this because they were out of pearls at the bubble tea place. You can talk to me about it on Twitter if you want: @mangopdf

---

Thanks to @hackgnar, who wondered aloud "there must be *some* way to get the cookies without root, since the user can get their own cookies".

Thanks also to @noncetonic and wardolphin, for their guidance in pointing out that headless Chrome and remote debugging is a scary combination.