FILIPPO VALSORDA

16 Sep 2017

A SECURE CAPTIVE PORTAL BROWSER WITH AUTOMATIC DNS DETECTION

Captive portals are the worst.

Flaky detection. The OS and browser try to detect these annoying network features but fail quite often, leaving you with broken connections.

> DID YOU KNOW that probe-based captive portal detection really doesn't work very well, with ~30% FP *and* ~30% FN rate in Chrome?
>
> — Emily Stark (@estark37) 15 September 2017

Attack surface. Even if it worked reliably, you wouldn't want to use the OS automatic captive portal browser for security reasons. Since it can be triggered by a network attacker without user interaction, it's the perfect target. It had vulnerabilities before and there is no information about whether it's up to date and sandboxed. There's also no option to disable Javascript or install security extensions.

DNS. I want to pick my own DNS server, and more specifically run my own unbound, for a number of reasons: clean results, local zones, overrides, DNSSEC... However most captive portals will block UDP traffic to anything except their DNS resolver (or would be trivially bypassed). So every time getting past a captive portal involves opening Network Settings, removing the custom DNS, logging in, and hopefully (that is, rarely) remember to put the custom DNS server back in.

HTTP. Finally, since a captive portal literally relies on a MitM attack, it results inaccessible when using HTTPS Everywhere in "Block all unencrypted requests" mode.

To recap, logging in involves resetting the DNS server, opening an Incognito window, enabling Javascript, maybe fumbling with cookies, logging in, and reverting DNS settings.

A dedicated Chrome captive browser

To scratch this itch I decided to make my own captive portal browser based on Chrome, such that it can be secure and configured as I please.

The main challenge is reaching the DHCP-provided captive portal DNS resolver without changing system settings. Chrome lacks the ability to configure DNS upstreams, but supports SOCKS5 which proxies name resolution.

With 100 lines of Go I built a small SOCKS5 proxy based on `github.com/armon/go-socks5` that handles name resolution via a custom `net.Resolver` that always dials a fixed IP for the DNS server.

It automatically discovers the DHCP DNS server on macOS with this command:

```
ipconfig getoption en0 domain_name_server
```

Finally it starts a Chrome instance configured to use the SOCKS5 proxy with the following command and waits for it to quit:

```
open -n -W -a "Google Chrome" --args \
--user-data-dir="$HOME/Library/Application Support/Google/Captive" \
--proxy-server="socks5://$PROXY" \
--host-resolver-rules="MAP * ~NOTFOUND , EXCLUDE localhost" \
--no-first-run --new-window --incognito \
http://example.com
```

The separate `--user-data-dir` allows it to run alongside your normal Chrome instance, while still being (separately) configurable and `--incognito` ensures that no state is preserved across executions.

The commands can be configured with a TOML file to make the tool work on other operating systems.

Usage boils down to running `captive-browser` and logging into the captive portal from a secure, configurable, ephemeral environment. All without touching DNS settings.

Find the tool at github.com/FiloSottile/captive-browser and me on Twitter.