Social WiFi Authentication

Ross Nordstrom
University of Colorado - Colorado Springs
1420 Austin Bluffs Pkwy,
Colorado Springs, CO 80918
rnordstr@uccs.edu

ABSTRACT

Wireless network authentication techniques have been greatly studied and improved over the past two decades, but they all suffer from the human factor. A common problem in computer and network security is ensuring the people interacting with it do not introduce security holes. Residential and small business networks are an example of an area where security is very important but not well understood. The simple fact is that the algorithm keeping wireless networks is of no use if the network owner uses insecure passwords, or insecure practices in keeping those passwords a secret.

In this paper I investigate the use of OAuth for authenticating and managing authorization of wireless networks. By using OAuth, we can frame the authentication scheme to match users' mental model of access control. OAuth, through integration with Google+ or Facebook, let's network owners manage access to their network more intuitively by simplifying the concept to a matter of managing a list of people in a group. In addition to improving access control, authenticating with OAuth removes the need for passwords and the insecurities introduced from them in residential and small business wireless networks.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Authentication

General Terms

Security

Keywords

Security, Authentication, OAuth, Wifi, Wi-Fi, Cloud

1. INTRODUCTION

With the steady growth of wireless networks as the core component of our connectedness, the study and application of WiFi authentication is pervasive. Any improvements to the efficacy or ease of use of WiFi authentication can have an enormouse impact.

In this paper I present the existing methods of authentication for Wireless networks (specifically WLANs). In general, they fall into two categories targeting (1) Residential and Small Office networks and (2) Enterprise networks. While each suffers for different reasons, I propose both would benefit from integration with OAuth providers such as Google, Facebook, Github, Twitter, or a private OAuth server. Offloading authentication to these providers contributes to both small and enterprise networks in different ways, while improving the user experience in both cases.

1.1 Residential and Small Networks

The first type of network suffers heavily from lack of understanding of wireless networks and "the human factor," which introduces vulnerabilities not inherent to the authentication protocol. In these networks, there are likely to be relatively few users, each with relatively many devices. Most of these devices will already be authenticated to social OAuth providers, which would allow for wireless network authentication software to prompt the user to simply reuse that authentication for access to the WLAN.

These small networks are the primary target of this proposal, as they stand to gain the most through the ease of use introduced by socially-authenticated WLANs.

1.2 Enterprise Networks

The second type of network would benefit from integration with OAuth providers because it would offload maintenance of yet another server(s). Additionally, more and more enterprises are using OAuth for internal authentication, so this would allow them to reuse their main authentication mechanism [**?**].

2. MOTIVATION

Existing WiFi authentication mechanisms, while technically secure, tend to create a poor user experience and become vulnerable because of insecure behaviors of the users. Additionally, most methods of authentication make it difficult for an administrator to revoke access to the network, or even see who and what has access to begin with.

2.1 Authentication Types

There are several authentication techniques supported for communication over WiFi. They include the following types of authentication, roughly in ascending order of security and grouped by security: [2] [14]

2.1.1 Insecure Types

SSID Hiding: Wireless networks can be "hidden" by not broadcasting the SSID; however, it's extremely easy to detect hidden SSIDs. This is a "security through obscurity" technique, which does a poor job of providing protection.

Open WEP Authentication: Devices must have a Wired Equivalent Privacy (WEP) key matching the access point's WEP keys. These keys must be pre-shared, and any authenticated device can immediately authenticate with the access point. Use of WEP has been deprecated since 2004 and has been shown to be insecure.

Shared-Key WEP Authentication: Similarly, devices have a shared key, but now must authenticate in through a four-step handashake. The device sends a request to the access point, who responds with an unencrypted challenge text, which the device encrypts using the shared key and returns to the AP. Finally, the access point decrypts the device's response and returns it. This technique is actually less secure than Open WEP Authentication, but intruders can see both the clear and encrypted text, and therefore derive the shared key. Once derived, they could use the shared key for uninhibited access to the wireless network.

MAC Address Authentication: An access point can be configured with a white-list of allowed device MAC addresses. Any device with a MAC address on that list can communicate over the network. Authentication by MAC address is problematice for a couple reasons. First, it is onerous on the system administrator to have to manage the list of allowed MAC addresses, especially at scale. Secondly, it is actually pretty easy to spoof MAC addresses. An intruder simply needs to eavesdrop on a successful authentication, and then assume that device's MAC address.

2.1.2 Secure Types

WPA Authentication: WPA (Wi-Fi Protected Access) is a successor to WEP, addressing the vulnerabilities of WEP that introduced insecurities. Specifically, it uses TKIP (Temporal Key Integrity Protocol) to dynamically use a new key for each packet after initial authentication. Additionally it replaces WEP's use of CRC (Cyclic Redundancy Check) with Michael for better packet integrity verification. For all that, WPA suffers from two main vulnerabilities: (1) users can choose to use insecure secrets, and (2) can be bypassed through Wi-Fi Protected Setup.

WPA comes in two primary types: WPA-Personal (or WPA-PSK, pre-shared key) and WPA-Enterprise. They have different requirements and target audiences: [13]

- WPA-Personal: Intended for residential and small office networks. All devices authenticate with the same key. Also called "WPA-PSK."
- WPA-Enterprise: Intended for enterprise networks, required a RADIUS authentication server. It uses EAP for authentication. Also called "WPA-802.1X mode" or "WPA."

WPA2: Replaces WPA with the major contribution being CCMP, a stronger AES-based encryption. CCMP stands for "Counter Cipher Mode with block chaining message authentication code Protocol."

EAP Authentication: The Extensible Authentication Protocol (EAP) defines a set of interactions between the device and a RADIUS authentication server, with the access point relaying. Once authenticated, the device gets a unique key for continued authenticated access. It comes in a variety of types, with the most prominent being the PEAP (Protected EAP), and **insecure** LEAP (Lightweight EAP).

RF Shielding: Rather than worry about intruders gaining unauthorized access to the network, it can be simpler to coat the surrounding structure (e.g. room, office space, or building) with a material, film, or paint that prevents wireless signals from entering or escaping.

Token Identifiers: Smart cards and USB/Software Tokens can be used for one of the most secure authentication mechisms. The token provider (card, USB, or software) combines a stored token with a user-entered PIN in a time-sensitive algorithm to frequently produce new encryption codes. Authentication with the server requires the device to be synchronized with its clock.

2.2 Authentication Summary

Two summarize the types of authentication, let's boil them down into their significance to each of our two target WLAN categories: Residential/Small and Enterprise Networks.

2.2.1 *Residential and Small Networks*

Most of the authentication types targeting residential and small office rely on a shared key. Regardless of the security of the authentication technique, they all suffer vulnerabilities from the human element. From one angle, users could mismanage or misplace secrets, exposing the credentials to intruders. Another problem is using secure passwords that are still easy to remember is difficult to achieve. As hackers gain access to more and more leaked passwords, they are able to study human patterns and behaviors in password use, rendering techincally secure passwords insecure. However, if you use randomly generated passwords, the users will likely store them somewhere, exposing the system to attack.

2.2.2 *Enterprise Networks*

The authentication types targeting enterprise use typically rely on a RADIUS authentication server. RADIUS stands for "Remote Authentication Dial In User Service," and provides AAA (Authentication, Authorization, and Accounting) [12]. Authentication with a RADIUS server is usually with EAP or another authentication protocol. The user information can be stored locally in the RADIUS server or integrated with an external store, such as SQL, Kerberos, LDAP, or Active Directory.

2.3 OAuth

OAuth, or Open Authorization, is an open web standard for delegating access to a `third party` on behalf of a `resource owner` to a `resource provider`. It aims to make it easy for clients to share access to a single authentication provider without having to share sensitive credentials.

2.3.1 *OAuth Parties*

This open protocol benefits each party. Let's break down those benefits by the party.

Resource Owner (User): The main benefits to the user are improved identity security because they don't have to manage yet another set of credentials, and reduced work as third parties can access existing data about the user, such as a list of friends.

Third Party (Startup): The third party benefits in two main areas: (1) they have offloaded liability of storing and managing sensitive user credentials and data, and (2) they can reuse data already stored in the resource provider, such as a list of friends or the user's contact info. It's hard to implement a secure authentication service and keep it up to date. By letting the provider handle this and features like 2-factor authentication, the startup can focus on product development instead of authentication.

Resource Provider (Google/Facebook/etc): The resource provider benefits from the increased use of their service.

2.3.2 OAuth Access Control

The OAuth protocol allows the resource provider to control access levels (Read-only vs Read/Write) and access granularity (What data can the third party access: email? friends list?). Additionally, it improves the visibility the resource owner has into how their data will be used, improving trust in the system.

In addition to defining access to the resource, OAuth also makes it simple to manage and revoke access. Users can see a list of authorized third parties and revoke access at any time.

3. RELATED WORK

Secure wireless authentication is a critical problem to solve well. The global trend is for increased network usage and complexity, and within networks a growth in the use of wireless. With such importance there are a number of works exploring ways to improve wireless security, but very few focusing on how to improve the experience for residential and small networks. The most similar work I could find is by Coova [3].

3.0.3 CoovaAP

CoovaAP aims to solve similar problems to what I set out to accomplish. At its core, it has a lot of overlap with my goal; however, it is a relatively inactive project and I had difficulty setting it up to compare what it does to what I aim to implement.

> CoovaAP is an OpenWRT-based firmware designed especially for HotSpots. It comes with the CoovaChilli access controller built-in and makes it easily configurable. [3]

CoovaAP includes ChilliSpot, Coova's captive portal solution, and some built-in solutions for creating a HotSpot with authentication based on checking a terms of service link, Facebook integration.

4. PROPOSED DESIGN

I propose instrumenting a wireless network to enforce authorization through integration with an OAuth provider (Facebook). I will prototype a simple authorization scheme to allow the network owner's friends access to the network. To create the authentication and authorization flow, I will program a dual-band OpenWRT router to have an open SSID with a captive portal where the user will use OAuth with Facebook to prove to the router who they are. At setup time, the network owner will integrate the router with their list of Facebook friends. As users present themselves to the router, if they are found to be in the owner's list of friends, they will be presented temporary credentials to the second, WPA-encrypted SSID. Rather than take on the challenge of creating per-user keys on the encrypted SSID, I will simply automate the resetting of the key to some frequent interval, such as 1-2 times a day.

While my initial prototype will be fairly manual, it is meant to be a proof of concept. Should this idea prove itself, my proposal is to publish the implementation as open-source router firmware and open-source client apps/services to automate the handshaking. By open-sourcing the implementation, router manufacturers could include the software in their products and device OS's could be modified to implement the client-side automation.

4.1 Handshaking

Here, I show 1 and then describe the steps and actions taken by the network owner, user(s), and instrumented router to achieve OAuth-secured wireless network.

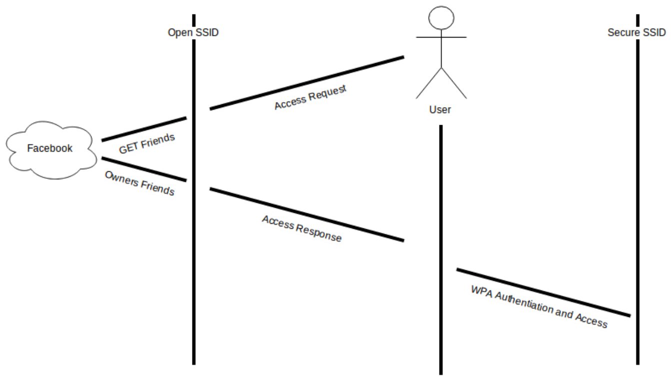


Figure 1: Handshake process shown visually

Setup: First, the network owner will authenticate the router on her behalf with Facebook, granting it access to her list of friends. The router will store the resulting server access token so it can periodically query Facebook in order to update the list of friends. This list will include an identifier for each friend.

The owner will then setup two SSIDs: an open SSID with a captive portal requesting users to authenticate with Facebook, and a WPA-encrypted SSID with full network access.

Access Request: Users will connect to the open SSID and complete an OAuth authentication with Facebook, granting the captive portal server some basic information about them, including an identifier. If the user declines or fails to complete the OAuth request, they will be denied.

Access Authorization: The captive portal server will search for the user's identifier in the network owner's list of friends. If found, the captive portal will continue with an Access Response.

Access Response: The captive portal will then response with the credentials needed for the encrypted SSID.

Switch to Secure SSID: Initially the user will need to manually switch over to the encrypted SSID and enter the credentials provided in the `Access Response` step. Eventually this flow would be automated by instrumenting the client, such as an Android service.

5. IMPLEMENTATION

Given the scope of this project, I aimed to achieve a minimum viable prototype, leaving improvements to get it actually adopted for Future Work6.

5.1 Proof of Concept Goals

Here I break down the work I did to create an initial proof of concept. Note that all captive portal communication would eventually be over SSL/TLS to ensure the credentials eventually passed to the user cannot be read by eavesdroppers.

5.1.1 Captive Portal

The first thing I need to do is implement a captive portal on an OpenWRT router, where I can redirect all users of the open SSID to an authentication page.

5.1.2 Router-Facebook Integration

The next thing I need to do is integrate the router with Facebook and authenticate it on behalf of the owner in order to get a list of their friends. Additionally I need to securely store the OAuth token allowing the router to authenticate for the owner to Facebook.

5.1.3 User-Facebook Authentication

The next step is to have users authenticate with Facebook so the captive portal server can learn their identity and see if they are one of the owner's friends. If they are found to be in that list, the server will present the user with the credentials to the secure SSID.

5.1.4 Programmatic SSID Re-keying

The final step in the proof of concept is to show that I can program the router to periodically change the access credentials for the encrypted SSID.

5.2 Proof of Concept Implementation

It turns out there's an open source tool for instrumenting an OpenWRT router with a captive portal and RADIUS. The product is CoovaAP [3], and it can be deployed as a simple firmware upgrade to OpenWRT-enabled routers. Specifically I worked with a Linksys WRT54GL [8].

An interesting and extremely relevant feature of CoovaAP is support for "Facebook HotSpot Captive Portal." Unfortunately, it has not been maintained and has not worked in several years [7]. In lieu of this, I decided to try out integrating with Google+ OAuth as a proof of concepts. Luckily, CoovaAp support programming a HotSpot server, so I tried instrumenting this. I was able to setup a Google Project with which to authenticate, but could not get the Hotspot server to integrate with it in the time I had. Details on the code I wrote and setup I did can be found in the appendix 9. Many of the challenges I faced had to do with the documentation for using CoovaAp. In many cases, the documentation was outdated, in other cases there simply wasn't the level of detail I could have used to improve my setup time.

6. FUTURE WORK

Future work on my idea will mainly take more time in order to complete a proof of concept demonstrating OAuth-based access to a wifi network. Since it's already been done with CoovaAp, the major focus should be on ease of use, supporting several providers, and marketing the idea rather than ability to do the authentication.

6.1 Ease of Use

Just as the original goal of my research was to improve security of WiFi networks by making it easy for non-tech-savvy users to maintain secure networks, future work should respect this primary goal. Ease of use of the system falls into two categories: (1) ease of setting up and maintaining the networks, and (2) ease of authenticating to the network with a minimum of work

With the prevalence social authentication on users' devices, it should be feasible to reuse existing authenticated sessions with identity providers (Facebook, Google, Twitter, etc...) so that users can authenticate to the wireless network with minimal user input. It should be easy to streamline the ease of use since Wireless Network authentication uses a browser page, which should automatically reuse the device's authentication session, since that's core to the design of OAuth.

A more problematic space will be authenticating devices to the network. Luckily, Google has some good documentation on how to handle device authentication with their implementation of OAuth 2 [5]. However, research will have to be done on how to make it easy to replicate device authentication with other services, like Facebook and Twitter.

6.2 Support Major Providers

Another challenge to deal with in future work is ensuring the system can integrate with most providers users might want to work with. On paper, OAuth2 claims to standardize authentication workflows regardless of provider; however, in practice each provider's integration looks a little different. Future work will need to identify the most important authentication providers and ensure support for each of those. One additional requirement should include making the system easily extensible so enterprises could integrate with their own proprietary OAuth system.

Currently, the top candidates to target would be Facebook, Google+, Twitter, and Github. Twitter and Github have the least-obvious architectures to enforce authorization of "who can access my wireless network," unlike Facebook and Google+ which support user-managed "groups" of friends.

6.3 Marketing

Finally, no system is of any use if it's unknown to the audience that might use it. Most of the marketing work to ensure a system like this gains popularity will be from partnership with one or more of the major router manufacturers like Linksys, D-Link, and Cisco. Additionally, the system will be best-served by being made available as an Open Source project so that developers can easily collaborate to extend the OAuth providers it integrates with, and ensure the system is secure. As we know from the historical implementations of security systems, the best way to ensure their success is by making them public so that as many people as possible can try to find holes in the system.

7. CONCLUSION

I set out to investigate how to make wireless networks more secure by taking an improved ease-of-use approach by integrating network authentication with OAuth. What I found was a free-to-use captive portal implementation called CoovaAP. CoovaAP had an existing Facebook integration, but it was not maintained and no longer works with modern Facebook. What it did offer me was a relatively easy way to install firmware with an instrumented captive portal. In all, I learned about current and historic wireless network authentication and security protocols as well gaining a deeper knowledge of OAuth.

After conducting the investigation I did, I think there is value in integrating wireless network authentication with a popular, open, and extensible protocol like OAuth. As discussed in the Future Work section 6, the focus of any research furthering this area will need to focus ease of use, supporting multiple providers, and marketing awareness of the concept.

8. REFERENCES

[1] David Bird. Coovaap and facebook. [Online; accessed 23-November-2014].

[2] Cisco. Authentication types for wireless devices. Technical report, 2009.

[3] coova.org. Coovaap firmware, 2011. [Online; accessed 23-November-2014].

[4] coova.org. openwrt-wrt54g-squashfs.bin, 2011. [Online; accessed 23-November-2014].

[5] Google. [Online; accessed 14-December-2014].

[6] Google. [Online; accessed 24-November-2014].

[7] LinkedIn. Coovaap and facebook. [Online; access 14-December-2014.

[8] Linksys. Wireless-g broadband router (wrt54gl). [Online; accessed 23-November-2014].

[9] Ross Nordstrom. [Code; created 24-Novemenber-2014].

[10] Ross Nordstrom. [Code; created 24-Novemenber-2014].

[11] vmixus. Walkthrough: How to setup a public hotspot with coovaap. [Online; accessed 24-November-2014].

[12] Wikipedia. Radius — Wikipedia, the free encyclopedia, 2014. [Online; accessed 23-November-2014].

[13] Wikipedia. Wi-fi protected access — Wikipedia, the free encyclopedia, 2014. [Online; accessed 22-November-2014].

[14] Wikipedia. Wireless security — Wikipedia, the free encyclopedia, 2014. [Online; accessed 22-November-2014].

9. APPENDIX

9.1 CoovaAP

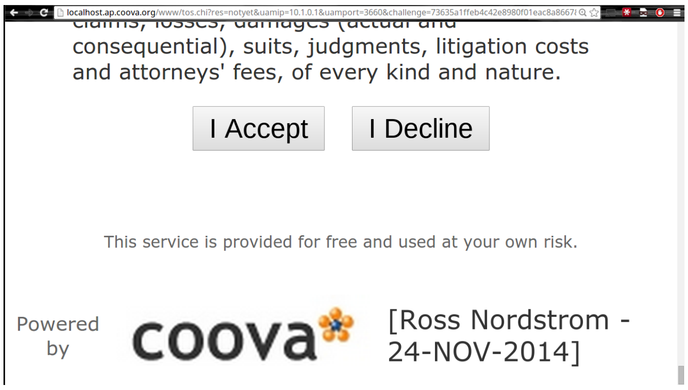
The CoovaAP setup was fairly straight-forward, if flaky. As mentioned in the proposal, I used a Linksys WRT54GL [8] with the corresponding CoovaAP binary, available on their public website [4]. After the firmware was installed (by using the router's existing Firmware Update page), I had access to CoovaAP. I found that the Admin page allowing me to configure the AP was very flaky in Chrome (version TODO) and Firefox (version TODO), but consistent in Internet Explorer (version 11.0.9600).

Having learned that the Facebook integration is old and no longer works [1], I decided to continue with my original plan of implementing my own captive portal server.

9.1.1 Captive Portal Setup

I managed to find a good tutorial [11] on setting up a custom captive portal page and followed it for the following main steps:

1. Clear RAM: To ensure a fresh configuration, SSH into the router with `ssh root@192.168.1.1` (password "root"), then run `mtd erase nvram && reboot` .
2. Setup Hotspot: Back in the Admin page (browse to http://192.168.1.1 in IE), setup a `HotSpot Type = Internal Hotspot` with `Registration Mode = ToS Acceptance` and Save.
3. Fix Broken Portal: In the SSH session, open (with ViM for example) `/et c/chilli/www/tos.chi`, then change **line 37**:
   (a) from: `if [ "$tos" = "1" ]; then`
   (b) to: `if [ "$HS_REG_MODE" = "tos" ]; then`
4. Confirm it worked: Now try to access the internet via the router (SSID: `Coova`), you should see the Terms of Service page with Accept and Decline buttons at the bottom.


localhost.ap.coova.org/www/tos.chi?res=notyet&uamip=10.1.0.1&uamport=3660&challenge=73635a1ffeb4c42e8980f01eac8a8667

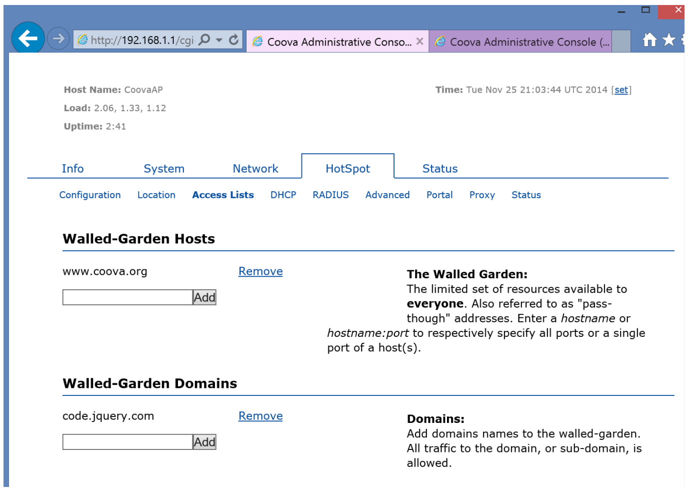
claims, losses, damages (actual and consequential), suits, judgments, litigation costs and attorneys' fees, of every kind and nature.

This service is provided for free and used at your own risk.

Powered by

[Ross Nordstrom - 24-NOV-2014]

Figure 2: Modified the Captive Portal!


http://192.168.1.1/cgi

Coova Administrative Conso...

Coova Administrative Console (...

Host Name: CoovaAP
Load: 2.06, 1.33, 1.12
Uptime: 2:41
Time: Tue Nov 25 21:03:44 UTC 2014 [set]

Walled-Garden Hosts

www.coova.org

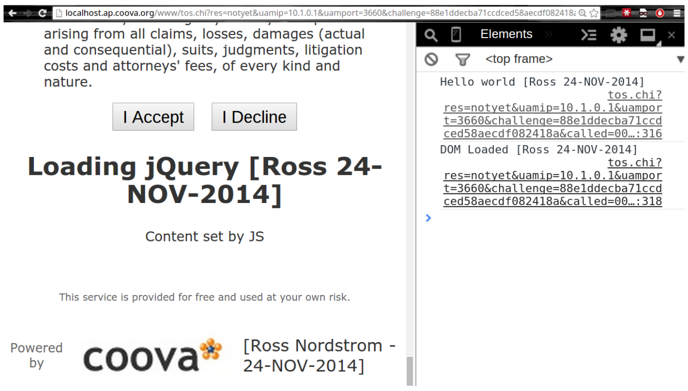
The Walled Garden:
The limited set of resources available to **everyone**. Also referred to as "pass-though" addresses. Enter a *hostname* or *hostname:port* to respectively specify all ports or a single port of a host(s).

Walled-Garden Domains

code.jquery.com

Domains:
Add domains names to the walled-garden. All traffic to the domain, or sub-domain, is allowed.

Figure 3: Allow jQuery within the Walled Garden


localhost.ap.coova.org/www/tos.chi?res=notyet&uamip=10.1.0.1&uamport=3660&challenge=88e1ddecba71ccdced58aecdf082418

arising from all claims, losses, damages (actual and consequential), suits, judgments, litigation costs and attorneys' fees, of every kind and nature.

Loading jQuery [Ross 24-NOV-2014]

Content set by JS

This service is provided for free and used at your own risk.

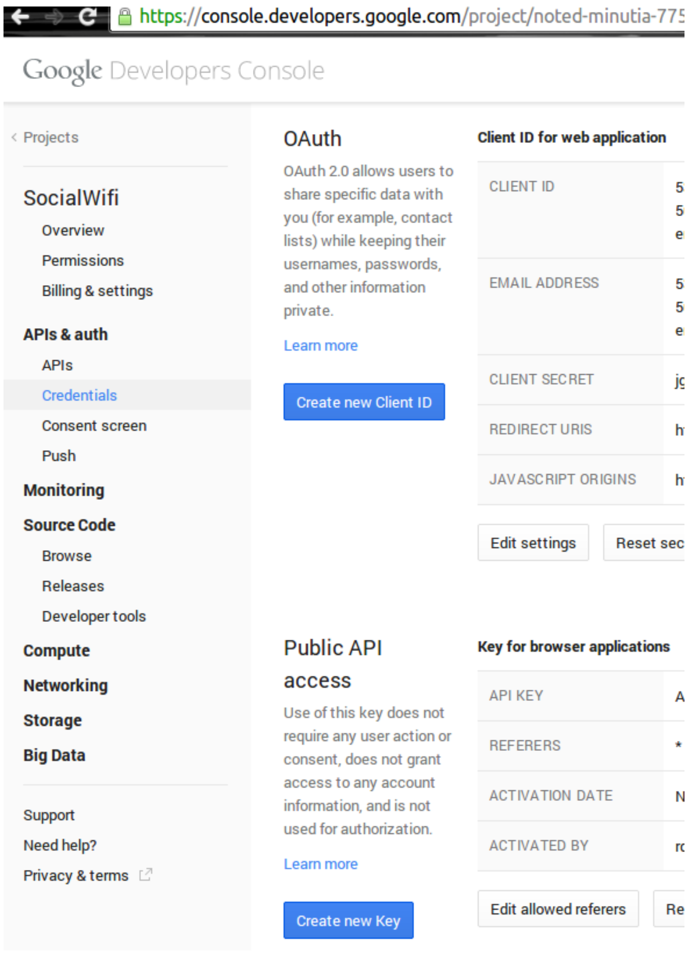
Powered by

[Ross Nordstrom - 24-NOV-2014]

```
Hello world [Ross 24-NOV-2014]
                                   tos.chi?
 res=notyet&uamip=10.1.0.1&uampor
 t=3660&challenge=88e1ddecba71ccd
 ced58aecdf082418a&called=00…:316
DOM Loaded [Ross 24-NOV-2014]
                                   tos.chi?
 res=notyet&uamip=10.1.0.1&uampor
 t=3660&challenge=88e1ddecba71ccd
 ced58aecdf082418a&called=00…:318
```

Figure 4: and then use jQuery

5. Explore: Next I tried to understand how the ToS page was served. Frighteningly, it seems to be a served up by a Bash script. I managed to find the code driving the Footer and modify it to include my name 2, by modifying a section of `/etc/chilli/www/config-local.sh`.
6. Run jQuery: Knowing I could modify code, I next tried to run some custom JavaScript, anticipating that I could use a JavaScript library to implement an OAuth client. To do this, I added a `<script>` tag including _http code.jquery.comjquery-1.11.1.js. This of course didn't work initially because of the captive portal. To fix this, I added "code.jquery.com" to my allowed Domains in the Admin portal's Access List 3. I was successfully able to use jQuery within the page then to overwrite a DOM element after page load. The result is a console message and modified text 4, having added some JavaScript to the served page [9].
7. Demo OAuth with Google: Now that I know I can run arbitrary JavaScript, I wanted to try running the demonstration Google OAuth code [6] from the router [10]. To accomplish this, I created a Google project 5 and configured a Client ID with which to authenticate.


https://console.developers.google.com/project/noted-minutia-775

Google Developers Console

< Projects

SocialWifi
- Overview
- Permissions
- Billing & settings

APIs & auth
- APIs
- Credentials
- Consent screen
- Push

Monitoring

Source Code
- Browse
- Releases
- Developer tools

Compute

Networking

Storage

Big Data

Support

Need help?

Privacy & terms

OAuth

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private.

Learn more

Create new Client ID

Client ID for web application

CLIENT ID	5 5 e
EMAIL ADDRESS	5 5 e
CLIENT SECRET	jg
REDIRECT URIS	h
JAVASCRIPT ORIGINS	h

Edit settings | Reset sec

Public API access

Use of this key does not require any user action or consent, does not grant access to any account information, and is not used for authorization.

Learn more

Create new Key

Key for browser applications

API KEY	A
REFERERS	*
ACTIVATION DATE	N
ACTIVATED BY	rc

Edit allowed referers | Re

Figure 5: Google Project to authenticate against

An important step was to add "localhost.ap.coova.org" to the allowed referers and JavaScript Origins fields.