UAF Cyber Security Club

Practical Introduction to SSL and HTTPS

Arsh Chauhan

10/27/2017

Lecture

What's going on in the security world this week? Well, WPA2 was "broken", you may have read doomsday headlines about how this means all Wifi is insecure. Well they sorta lied to you! WPA2 has a flaw that allows attackers to MITM traffic between you and the AP but this doesn't necessarily mean they can see your passwords, credit-card information or kinky search terms. HTTPS to the rescue, we all (hopefully !) know how to verify an authentic https connection but let's see how to setup our website to use https. But first, some conceptual stuff.

Https = http + TLS (transport layer security)

Https is used for Secrecy and verification. Using https helps you verify that you're actually talking to google versus some version of "google" running on an attacker's machine. HTTPS can provide authentication for websites due to a system called the Certificate Authority (CA) system.

The CA system is based on trust. As a website owner, I request a CA to sign my certificate and they ask me to prove that I own the domain so I can get certs signed for iamaprettykitty.com and vc.csc.uaf.edu but not for uaf.edu or google.com. So how do they do this?

1) Email verification: The CA sends an email to a standard email address or list of email addresses, generally something like webmaster@example.com, administrator@example.com and have you enter a validation code. The idea is if the person requesting the certificate has the ability to receive email on these addresses then they control the domain
2) DNS record: The CA asks you to make a new DNS record and will run a query looking for that record as verification. Generally a TXT or CNAME record with a long string.
3) File: The CA will ask you to place a file with a particular name and content in the root of your website and will check for the file and verify its contents as verification.
4) Enterprise: I don't know what the official name for this is but this is how you get free SSL certs from UA. You ask OIT for access to a certificate manager site and they verify you're authorized to request certificates for that particular uaf subdomain and give you a pin code. You then send your requests to someone in OIT via the certificate manager and they approve or deny your request. In theory only for uaf subdomains but it's also used by the rave lab. This is how we get certs for CSC servers. So in theory, I should be able to request certs for csc.uaf.edu or any subdomain like vc.csc.uaf.edu but they should deny (and probably investigate) my request for a cert for uaf.edu

How can we trust the CA's?

Well, we just sorta trust them, browsers and operating systems have a list of trusted root CA's who can sign keys for other intermediate CA's. Your certs are generally not signed by a root CA but by intermediate CA's owned by the root CA's or other companies and browsers trust them since these intermediate CA's have certs signed by the root CA's. This is how Let's Encrypt a CA authority that issues free SSL certs is trusted by browsers and OS's.

Unfortunately sometimes this trust is broken. Chinese CA WoSign accidentally issued certs for base domains if you could prove control of a subdomain (https://thehackernews.com/2016/08/github-ssl-certificate.html) and sometimes it's more malicious like the Superfish malware on Lenovo computers (https://arstechnica.com/information-technology/2015/02/lenovo-pcs-ship-with-man-in-the-middle-adware-that-breaks-https-connections/), the sliver lining is browsers work real fast in de-trusting malicious, compromised or CA's with inadequate checks. Incidents like Superfish is why Firefox does not use the OS's trust store and relies only on it's internal trust store.

Can this be fixed?

Yes, let's just become better as a species and not break trust other's put in us. Ok that's not going to happen...so let's find some technical and policy techniques. I'm not going to go over these since they can each be multiple weekend lectures but there are 3 solutions i'm aware of

1) DNSSEC: Cryptographic signing of DNS replies (https://www.icann.org/resources/pages/dnssec-qaa-2014-01-29-en) (https://www.educause.edu/ir/library/pdf/EST1001.pdf)
2) HTTP Public Key Pinning (HPKP): Server tells the browser what public keys to trust (https://en.wikipedia.org/wiki/HTTP_Public_Key_Pinning) (https://tools.ietf.org/html/rfc7469#appendix-A) (https://www.owasp.org/index.php/Pinning_Cheat_Sheet)
3) DNS Certification Authority Authorization (CAA): new DNS record type that allows you to specify the CA(s) you allow to issue certs for your domains. Needs to be checked by the CA, so doesn't really help with non compliant CA's

Implementing SSL

Implementing SSL differs slightly between server softwares. We'll be using Nginx and OpenSSL, the OpenSSL steps are common for all servers and even cross-platform (assuming you have OpenSSL). We'll be using self-signed certificates since we cannot get rou certs signed by a CA because our VM's cannot be accessed from the internet. Self signed certificates can be generated with a single openssl command

```
openssl req -x509 -nodes -days 9999 -subj '/O=Kill All
Humans/OU=Cats/CN=catsrule.local/C=US/ST=Alaska/L=Fairbanks' -newkey rsa:2048 -keyout
mycert.key -out mycert.crt
```

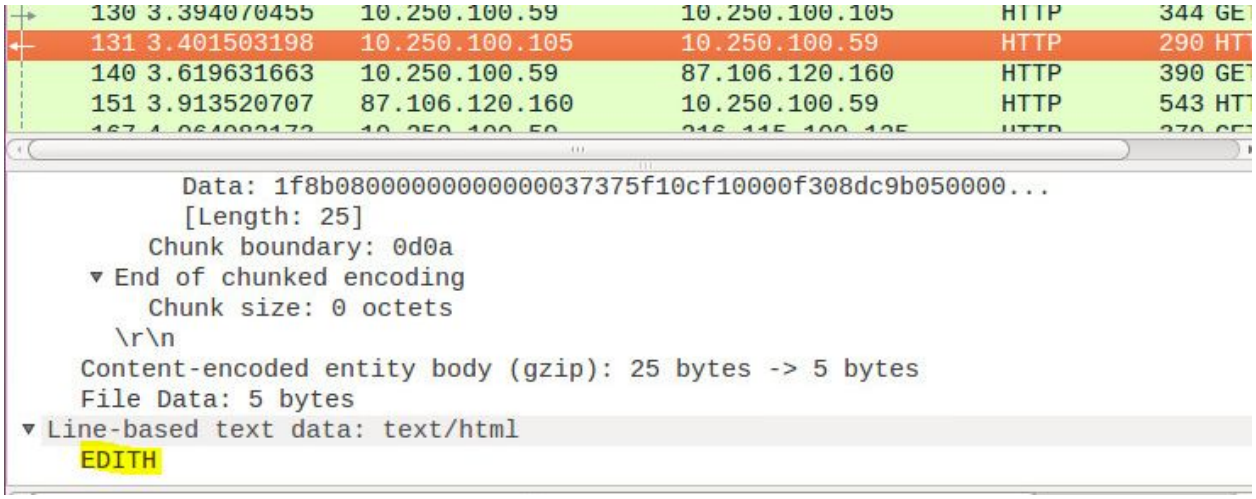
Let's break this command down

1) req -x509 : Request a self signed x509 certificate
2) nodes: Create an unencrypted private key
3) days: Days the cert is valid
4) subj: certificate subject fields (Formatting is important) (https://stackoverflow.com/questions/6464129/certificate-subject-x-509)
    a) O: Your organization
    b) OU: Your organizational unit within the organization
    c) CN: common name (domain name)
    d) Location (country,state, city)

5) newkey: Generates a new private key and CSR pair. Our command says we want a 2048 bit key
6) -keyout and -out are self explanatory

This command should produce 2 files: mycert.key and mycert.crt in the same directory. mycert.key is your private key and should NEVER leave your server. The crt file is served by your webserver to the client. Now let's move to the fun part...configuring Nginx to use SSL.

Your VM should already have a web server called NGINX running on it, the wiki claims NGINX powers Hulu, Netflix and even Cloudflare amongst other "high visibility" websites. So it's actually something people use in the real world. Other common web servers are Apache, Tomcat and IIS (We pretend IIS does not exist). Let's do something interesting, visit your webserver's IP in a browser and see your traffic in wireshark. So HTTP is bad since anyone can see your super secret cat name.



Remember, we generated those keys back in the beginning, well let's go back to them. Let's move these keys to somewhere that's not our home directory. Let's create a new folder for them in /etc/nginx named certs. Now move them to this folder.

```
csc@csc:/etc/nginx$ sudo mkdir certs
[sudo] password for csc:
csc@csc:/etc/nginx$ sudo mv ../../home/csc/catsrule.* certs/
csc@csc:/etc/nginx$ cd certs/
csc@csc:/etc/nginx/certs$ ls
catsrule.crt  catsrule.key
```

So now what...let's edit the nginx config to use SSL. Where are config files stored in Linux? Yes../etc, NGINX stores it's files in the aptly named /etc/nginx directory. Your individual site config files are in /etc/nginx/sites-available. You should only have one files named default in there. Open it and let's take a look. There's a lot of clutter since this is the default file (with some stuff I've added in), we'll go over this file in detail when we deal with configuring web servers. For now,we just need to add a few lines to enable and setup SSL

```
# SSL configuration
#
 listen 443 ssl default_server;
```

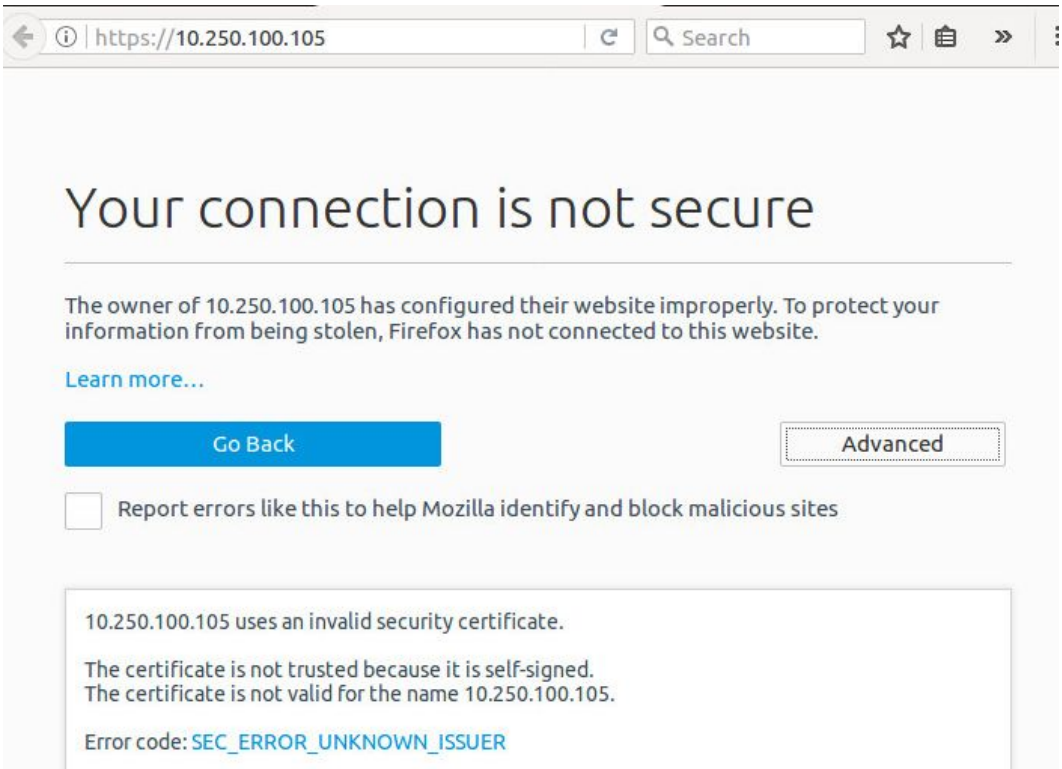
```
# SSL Configuration
ssl_certificate /etc/nginx/certs/catsrule.crt;
ssl_certificate_key /etc/nginx/certs/catsrule.key;

#Make it more secure
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_ciphers HIGH:!aNULL:!MD5;
```

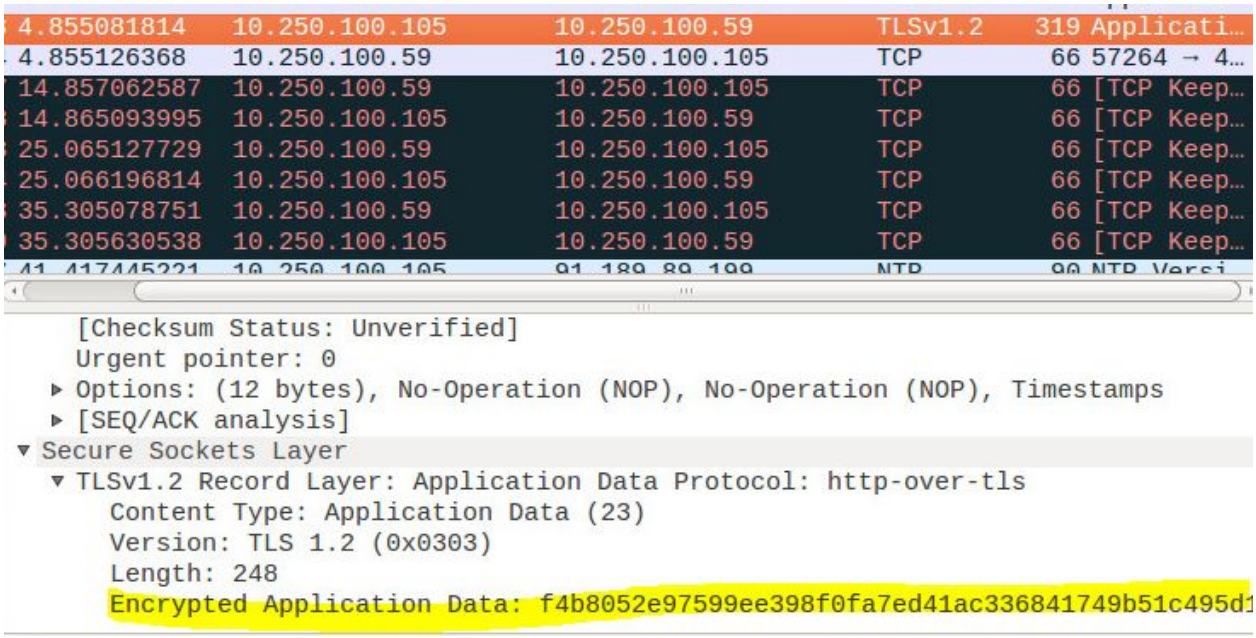
The ssl_certificate setting expects the crt file we made using openssl. In our case, this just contains one certificate but on a real server this file would have all the certificates in the chain. The ssl_certificate_key expects the path to the key corresponding to the given certificate (This is the key we generated with openssl). The key and certificate file have to be generated as a pair since the key is unique to the certificate. Again, this key NEVER leaves your machine. We'll test our config and restart the NGINX server with

```
sudo nginx -t
sudo service nginx restart
```

Let's test our SSL config



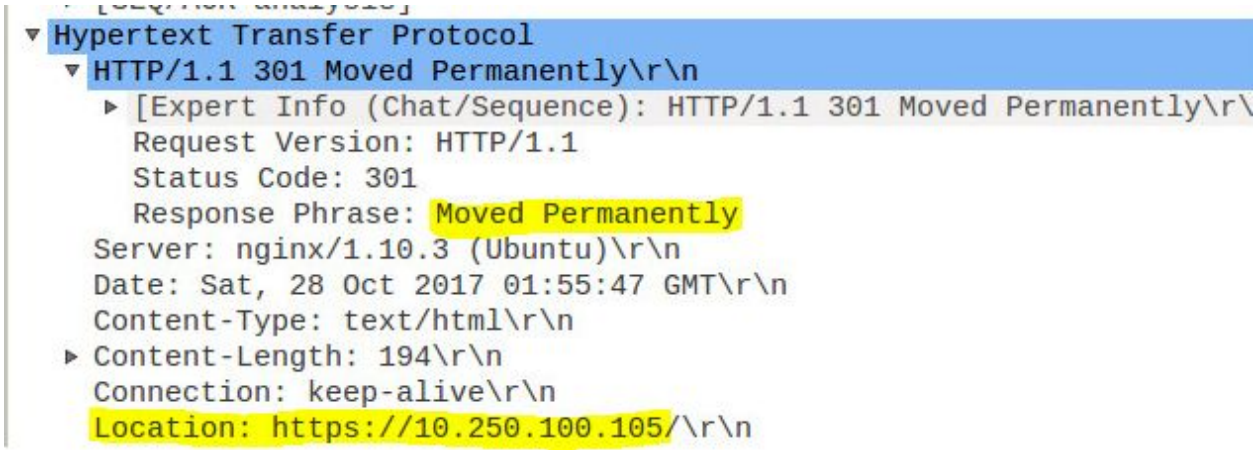
Ok,so we get an error which basically says the browser does not trust the cert issuer which is ok for our case since we're using a self signed cert (You should not overlook this for any real site). Let's go back to wireshark



All we see is encrypted data, our super secret cat name is now safe. But we aren't really secure yet since by default, we still serve data over HTTP so let's make some small changes to always force clients to use HTTPS.

```
# Force HTTPS
if ($scheme != "https")
{
        return 301 https://$host$request_uri;
}
```

And voila, all connections now go via https.



Lab

Requirements

1) Linux VM (tested on ubuntu 16.04) with NGINX
2) VM with Browser and Wireshark

Scenario

1) Give them a webserver with my random cat name generator server.
2) Show them that I can see their super secret cat name in wireshark
3) Switch to HTTPS and voila !!!