see [PF] 4.1, 4.2, 4.3

Lecture 6: Web Security

6.1 Background

6.2 Vulnerabilities in the “secure” channel

6.3 Mislead the user (address bar)

6.4 Cookies and same origin policy

6.5 Cross-site scripting (XSS)

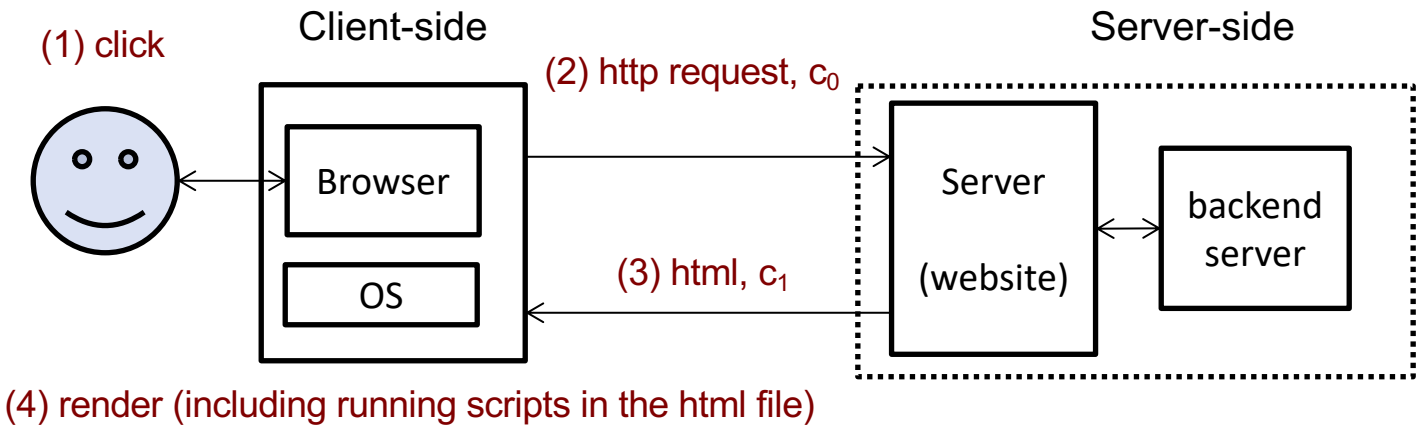
6.6 Cross-site request forgery

Summary & takeaway

- Web (HTTP) is in the “application” layer. It is a popular platform with many services built on top of it.
- HTTPS = HTTP on top of TLS. (inherit strength & weakness of TLS).
- The mechanism of cookies.
  - Cookies might contain authentication credential (authentication) and personal information (privacy).
  - Browser interacts with multiple sites, each site with its cookies. Need separation among different sites. Access control boundary between sites based on ***same-source-origin***. Unfortunately, turn out that the separation is weak (ambiguity, difficult for users to visually double-check, etc).
- XSS. XSRF.
- Script injection.
- Human-in-the-loop. (confusing URL and address bar).

6.1 Background

Overview

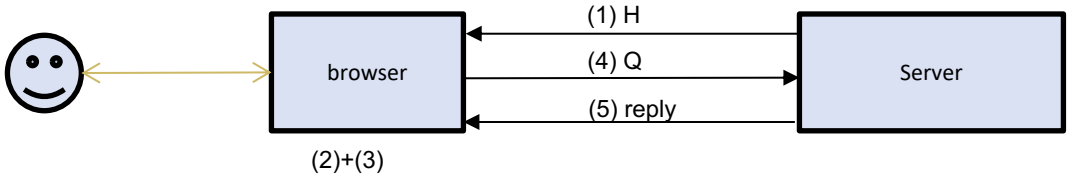


1. User clicks on a "link". e.g www.nus.edu.sg
2. A http request is sent to the server. (with cookie c_0 if any)
3. Server constructs and sends a "html" file to the browser, possibly with a cookie c_1 (the cookie can be viewed as some information the server wants the browser to keep, which to be passed to the server during the next visit. The cookie can be some secrets).
4. The browser renders (including running of script) the html file. The html file describes the layout to be rendered and presented to the user, which could include scripts. The html file also instruct the browser to update the cookies.

In Firefox, right-click, choose "view page source" to view the html file that is sent from the server to the browser.
(or Tool->Wed Developer->Toggle tool)
In Chrome, View->Developer->View Source (or View->Developer->Developer tool)

Closer look into an example.

1) Browser visits Google page. A html file H is sent to the browser. The browser renders H. (Try viewing the raw form of H. In chrome, view->developer->view source. Note that there are many occurrences of the keyword "<script>" which marks the beginning of a script.)



2) User enters the keywords "CS2017 NUS"

3) The browser, by running scripts in H, constructs a query Q.

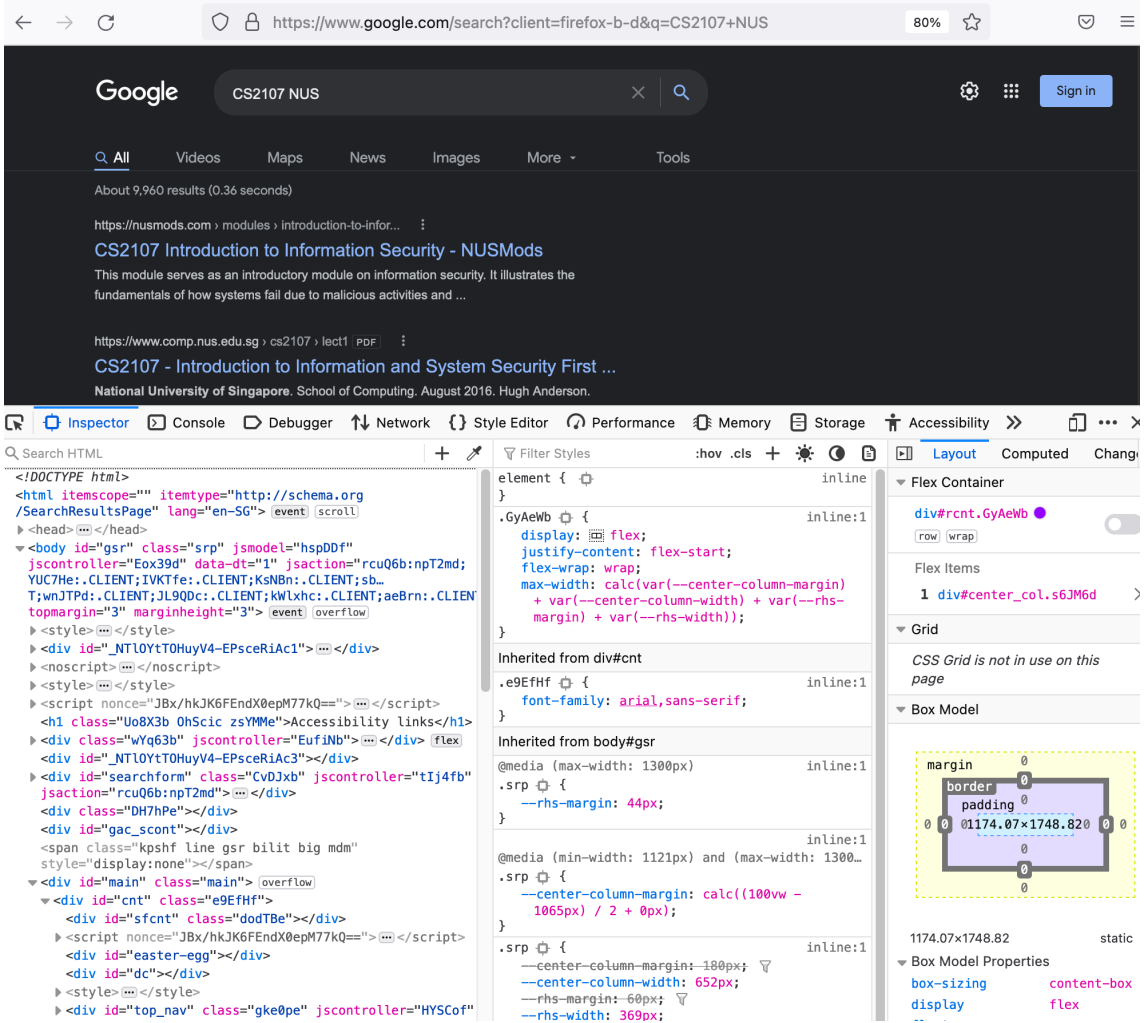
https://www.google.com.sg/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=CS2107+NUS

4) Browser sends query Q to Google.

- Note that additional information is added as parameters in Q. These info is useful for the server. The info could even be in the form of script.

5) The server constructs a reply. (in some cases, the reply contain substrings in (3)).

Right click,
Choose “inspect”

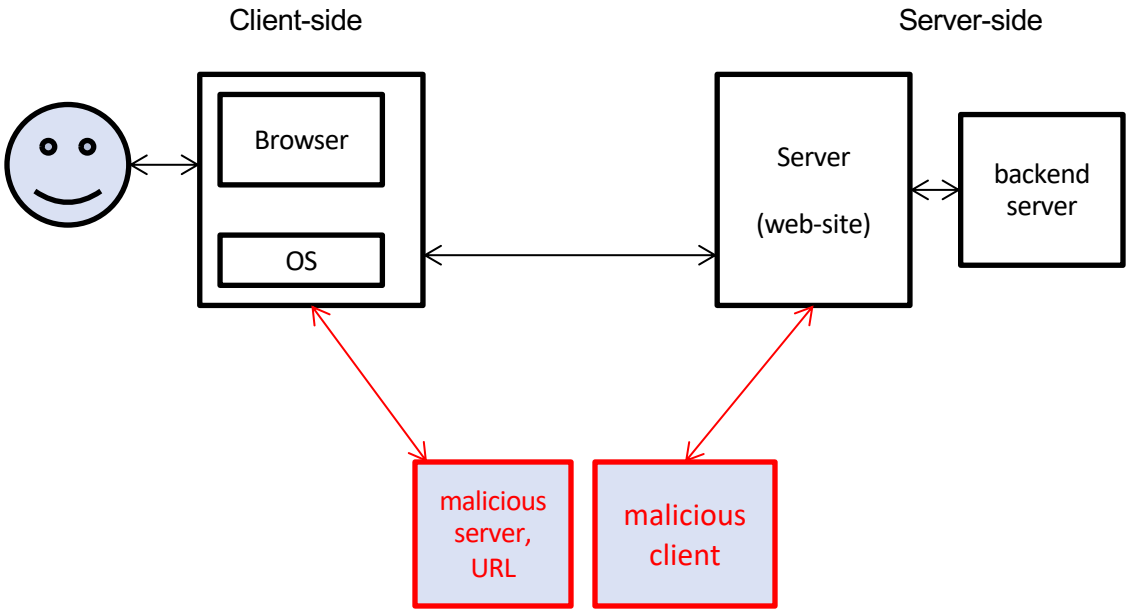


Complications of Web Security.

- In many OS'es, browsers run with the same privileges as the user. Hence, the browser can access the user's files. (note: this is not the case in Android).
- Browsers support a rich commands/controls set for content providers to render the content.
- Browsers manage sensitive user's info and secrets (stored in cookies).
- At any one time, there could be many servers providing the content in the browser (*e.g. different advertisements appear at the same time)*
- For enhanced functionality, many browsers support add-ons, plugins, extensions by third parties. *(definitions and differences of add-ons, plugins, extensions are not clear and depends on the browsers.)*
- Users upload info to the server (e.g. forum, names to be displayed).
- More and more sensitive data information is in the web/cloud.

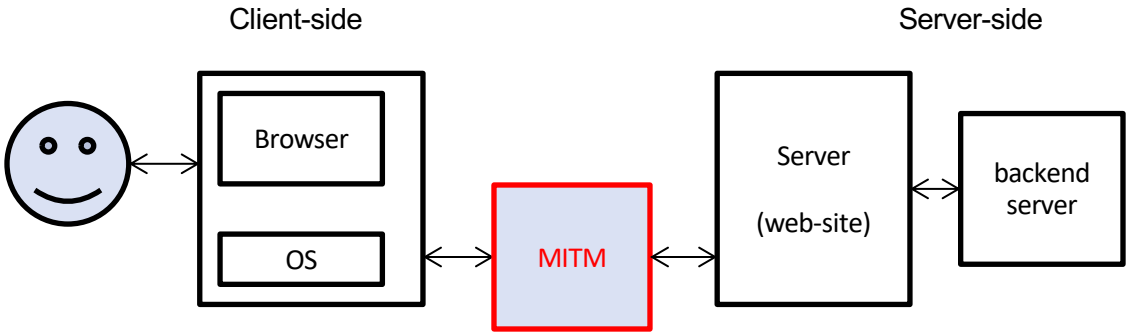
For desktop, browser is becoming the main application. (Situation slightly different in mobile os where users also interact with the web using "apps".)

Threat model 1: Attackers as another end systems.



Here, the attacker is just another end system. For e.g. a malicious web-server that the victim is lurked to visit, or attackers who has access to the targeted server.

Threat model 2: Attackers as MITM.



Here, the attacker is a Man-in-the-middle in the IP or lower layer. The attacker can gain MITM in a few ways, for e.g.,

- As the café-owner who provides the free wifi;
- Via DNS spoofing attack or ARP attack;
- As owner of the VPN server, last hop in the TOR network (not covered in this module).

We had covered this threat model in previous lectures.

Because the MITM is in the IP layer, the MITM can attempt to impersonate a server (e.g. phishing attack). So, this threat model also covers some settings of threat model 1 where the attacker is the server.

6.2 Vulnerability in the secure communication channel (SSL/TLS)

HTTPS = HTTP on top of SSL/TLS

Recap

- **Recap**: HTTPS considers threat model 2, i.e. a MITM in the IP layer. HTTPS is "secure" in this model. This is achieved by TLS (securing the channel using combination of crypto primitives and authentication protocol) and PKI (securing distribution of public key).

- **Recap**: From a user's perspective, the presence of padlock and correct domain name in the address bar assure that the browser is indeed interacting with the authentic server. (hence not under phishing attack).

- However, the above assumes that the system is designed and implemented correctly.

- A number of examples: Superfish (not covered in this year), Re-negotiation attack (tutorial), Freak attack (not covered in this year), BEAST attacks (similar to padding oracle) (not covered in this module).

6.3 Misleading user on domain name

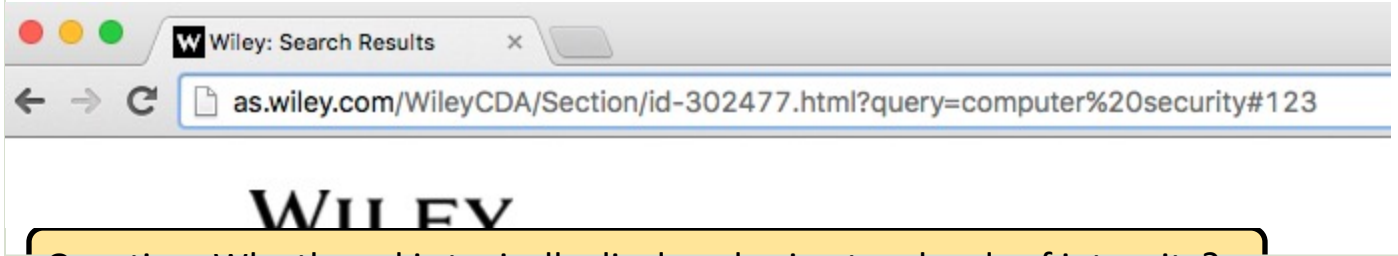
URL (Uniform Resource Locator)

A url consists of a few components. (see https://en.wikipedia.org/wiki/Uniform_Resource_Locator)

1. scheme,
2. authority (a.k.a the hostname) ,
3. path
4. query
5. fragment

e.g.

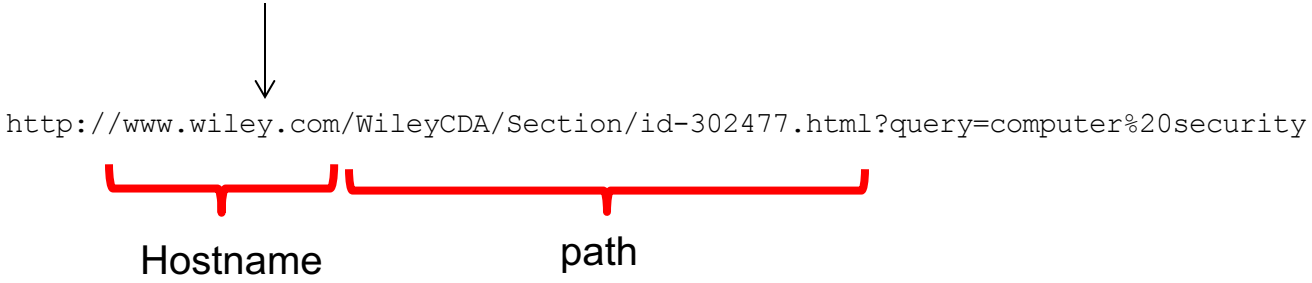
http://www.wiley.com/WileyCDA/Section/id-302477.html?query=computer%20security#12



Question: Why the url is typically displayed using two levels of intensity?

Source of confusion…

Visually, no clear distinction of “Hostname” and “path”.
The delimiter “/” can appear elsewhere, which confuses viewers.



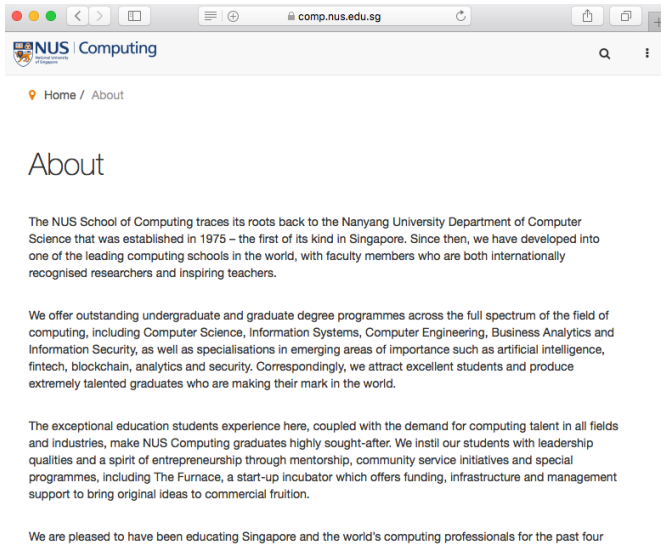
- A malicious website may register a hostname that contains the targeted hostname with a character that resembles the delimiter “/”

  www.wiley.com.66785689.in/Section/id-302477.html

The different intensities could help user to spot the attack.

e.g. from phishing email: nuslogin.789greeting.co.uk

- The browser Safari chooses to display the hostname *only* in the address bar.



Address bar spoofing

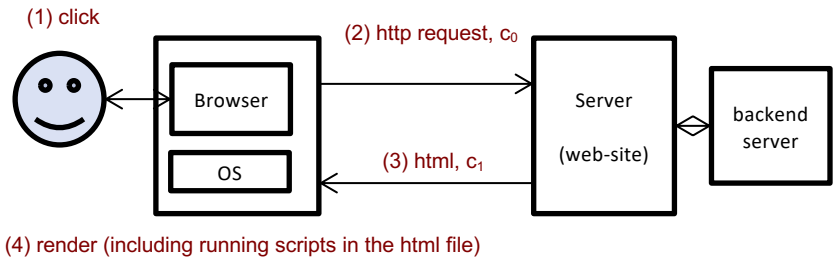
- Address bar is an important component to protect. The address bar is the only indicator of which url the page is rendering. If the address bar can be "modified" by the webpage, an attacker could trick the user to visit a malicious url **X**, while making the user to falsely believe that the url is **Y**.

- A poorly-designed browser may allow attacker to achieve that. E.g. in early design of some browsers, a web page could render objects/pop-up in arbitrary location, and thus allowing a malicious page to overlay spoofed address bar on top of the actual bar. Current version of popular browser have mechanisms to prevent that.

- See a more recent e.g.: **Android Browser All Versions - Address Bar Spoofing Vulnerability - CVE-2015-3830**

http://www.rafayhackingarticles.net/2015/05/android-browser-address-bar-spoofing-vulnerability.html

6.4 Cookie and Same-origin policy



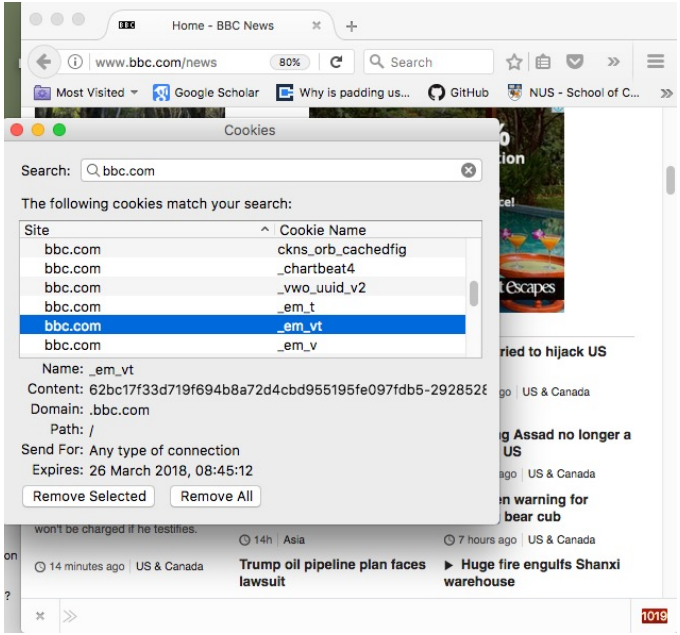
Remark: Many websites get users' consents to keep cookies. Why is this so?
EU's GDPR (General Data Protection Regulation).
https://www.cookiebot.com/en/gdpr-cookies/

Cookies

- A http cookie is a piece of data sent by the server in the response of a HTTP request. It is under the "Set-Cookie" header field. The browser permanently stores the cookie.
- Whenever a client revisits the website, the browser automatically sends the cookie back to the server.

Remarks:

- The cookies are sent back only to the "same origin", in the sense that, it is sent to the server which is the "origin" of the cookie. Viewing from access control perspective, the "same origin" set the boundary among different web sites. A script from a web site can only access cookies within its boundary. *(unfortunately, this is not done clearly, leading to many problems…)*
- There are also a few types of cookie, eg. session cookies (deleted after the session ends), secure cookies (can only be transmitted over https), etc. See https://en.wikipedia.org/wiki/HTTP_cookie#Session_cookie



Demo on Firefox

- right-click -> view page info -> security -> cookie
- Tools -> web developer -> Developer toolbar

Same-Origin Policy

A “script” which runs in the browser could access cookies. Which scripts can access the cookie? Due to security concern, browser employs this access control policy:

The script in a web page A (identified by URL) can access cookies stored by another web page B (identified by URL), only if both A and B have the same ***origin.***

Origin is defined as the combination of

protocol, hostname, and port number.

The above access policy is simple and thus seeming safe. However, there are several complications.

Definition of “same-origin”.

Example (from http://en.wikipedia.org/wiki/Same-origin_policy)

URL’s with the same origin as: http://www.example.com

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same protocol, host and port
http://www.example.com/dir2/other.html	Success	Same protocol, host and port
http://username:password@**www.example.com** /dir2/other.html	Success	Same protocol, host and port
http://www.example.com:**81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:**80**/dir/other.html	Depends	Port explicit. Depends on implementation in browser.

Limitations:

- *Confusing definition*: There are many exceptions, and exceptions of exceptions. Very confusing and thus prone to errors.
- Different browsers may adopt different definitions.

E.g. of Cookie application: Token-based authentication

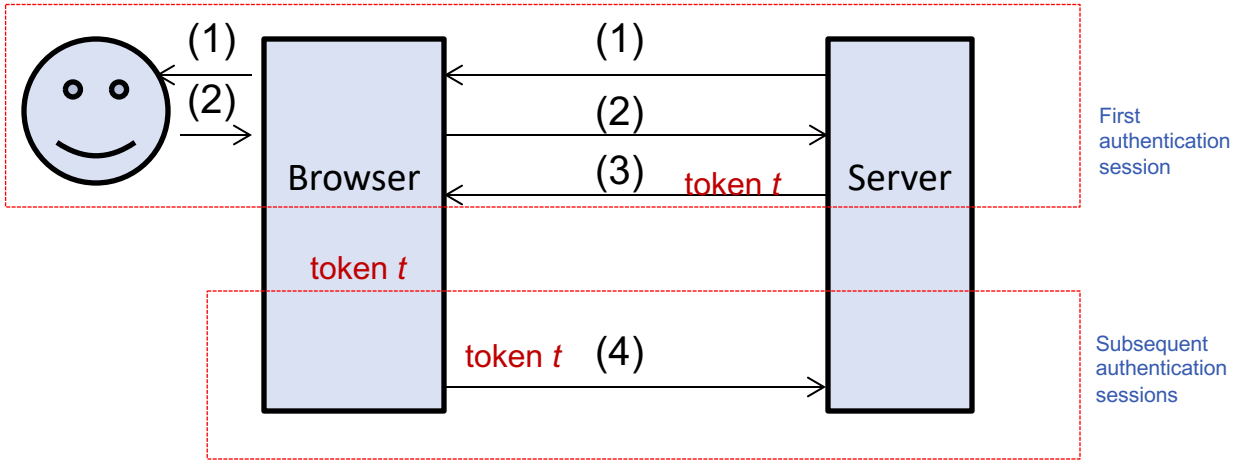
Cookie is useful. Here is one application for Single-Signed-On. *(This example appeared in the tutorial on renegotiation attack)*

To ease user tedious task of repeated login, many websites use "token-based" authentication. That is,

1. After a user ***A*** is being authenticated (for e.g. password verified), the server sends a value ***t***, know as token, to the user.
2. In subsequent connection, whoever presents the correct value is accepted as the authentic user ***A***. (note that user does not has to perform the tedious password authentication again).

Remarks

- Token typically has an expiry date.
- An identification of the session can be used as the token. So the token is also called SID.
- In web applications, the token is often stored in the cookie.



(1) Authentication challenge (e.g. asking for password).

(2) Authentication Response that involves the user.

(3) Server sends a token *t.* Browser keeps the token

(4) Browser presents the token *t.* Server verifies the token.

We assume that the communication channel is secure. (i.e. all communication over HTTPS (with server authenticated) and the HTTPS is free from vulnerabilities).

Choices of token

- Suppose *t* is a randomly chosen number, then the server needs to keep a table storing all the tokens it has issued. To avoid storing the table, one could use
  - (secure version) A message authentication code (MAC). The token ***t*** consists of two parts: a randomly chosen value, or meaningful information like the expiry date, concatenated with the mac computed using the server secret key.

    e.g ***t*** = "alicetan:16/04/2015:adc8213kjd891067ad9993a"
  - (insecure version) the cookie is some meaningful information concatenated with a sequence number that can be predicted.

    e.g t= "alicetan:16/04/2015:128829"

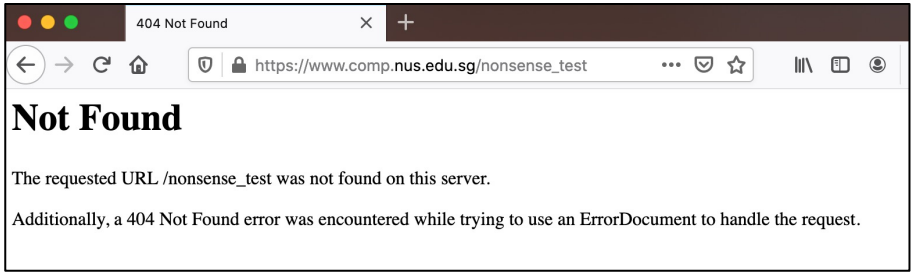
- For both methods, when the server finds out that the token is not in the correct format (or not the correct mac), the server rejects. The first method relies on the security of mac, while the second method relies on obscurity – attackers don't know the format.
- The second method is insecure because an attacker, who knows how the token is generated (for e.g. by observing its own token), can forge it. This further illustrates the weakness of security by obscurity.

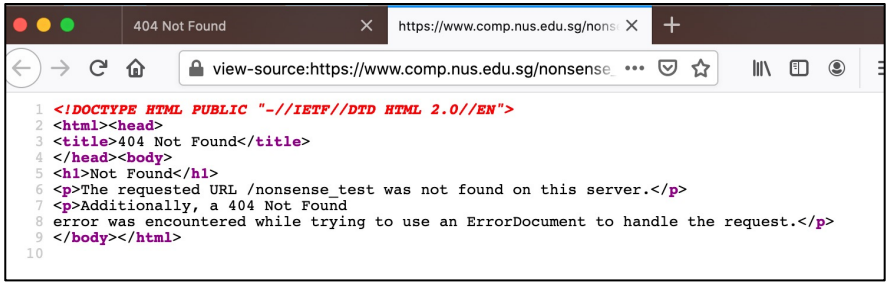
6.5 Cross Site Scripting (XSS) Attacks

Background

In some websites, if the browser sends a text that contains a substring ***s,*** the replying html sent by the server would also contain the same substring ***s.***

- Enter a wrong address.
  `http://www.comp.nus.edu.sg/nonsense_test`





- Search for a book in library `http://nus.preview.summon.serialssolutions.com/#!/search?q=heeheeheee`

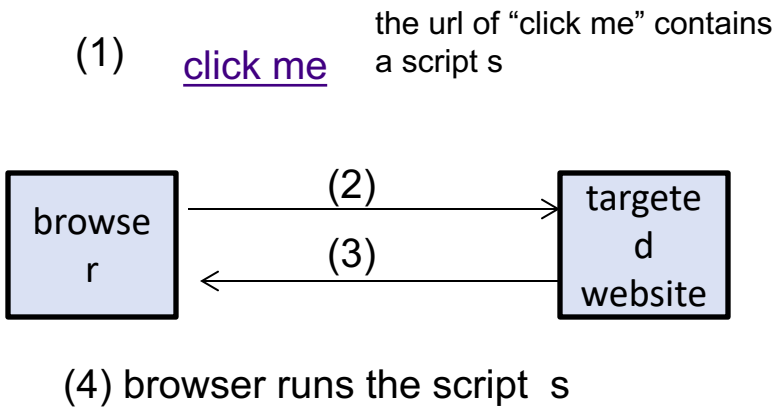
- What if the string ***s*** contains a script?

e.g. `http://www.comp.nus.edu.sg/<script>alert("heehee!");</script>`

(the above attack won't work as the server replaces the special character "<" by &lt)

Attack

1. Attacker tricks a user to click on an url, which contains the target website, and a malicious script ***s.*** e.g the link could be sent via email with “click me”, or a link in a malicious website.
2. The request is sent to the server.
3. The server constructs a response html. The response contains the script ***s***.
4. The browser renders the html page and runs the script ***s***



why this is an attack?

The malicious script could

1. deface the original webpage;
2. steal cookie.

- The control that protects access to the cookie is the same-origin policy.
- Now, since the malicious script is coming from the same-origin (the victim clicked on it..), it can now access cookie previously sent by the website.
- This is yet another example of *privilege escalation*. A malicious script has elevated privilege to read the cookie.

This attack exploits the client's trusts of the server.

types of XSS

- Reflection (aka non-persistent). The attack described before.

- Stored XSS (aka persistent). The script ***s*** is stored in the targeted website. For instance, in forum page.

Defense

- Most defense rely on input-validation carried out by the server. That is, the server filters and removes malicious script in the request while constructing the response page.

  However, this is not a fool proof method.

6.6 Cross Site Request Forgery (XSRF)

- Aka “sea surf”, cross-site reference forgery, session riding.
- This is the reverse of XSS. It exploits the server’s trust of the client. Previous attack of XSS exploits the client’s trust of the server.

Example:

- Suppose a client A is already authenticated by a targeted website S, say `www.bank.com` and the site keeps a cookie as “token”.
- The attacker B tricks A to click on a url of S. The url maliciously requests for a service, say transferring $1000 to the attacker Bob’s

  www.bank.com/transfer?account=Alice&amount=1000&to=Bob

- The cookie will also be automatically sent to S which is sufficient to convince S that A is already being authenticated. Hence the transaction will be carried out.

See https://en.wikipedia.org/wiki/Cross-site_request_forgery

Defense

Relatively easier to prevent compared to XSS.

Include authentication information in the request. For e.g.

www.bank.com/transfer?account=Alice&amount=1000&to=Bob&**Token=xxk34n890ad7casdf897e324**

6.7 Other Attacks

Misconfiguration, searching for filename…

Other attacks and terminologies

- Web tracking
  - Web bug (aka web beacon, tracking bug, tag, page tag).
  - Legitimate usage, PDPA. *(why websites keep getting consensus from user on cookies usage?)*
- Drive-by-Download.
- Pixel stealing. *(steal the pixel value on the display)*
- Clickjacking, User Interface redress attack *(trick the user to click on attacker overlayed GUI)*

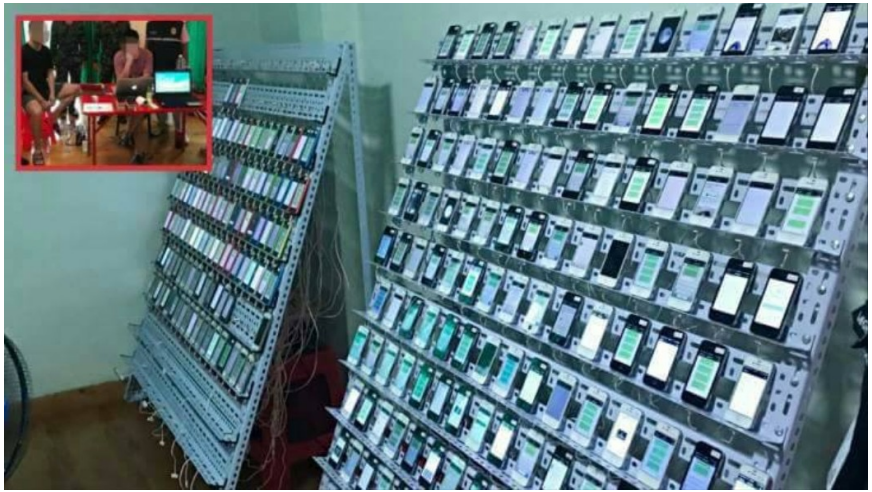
https://www.owasp.org/index.php/Clickjacking

- CAPTCHA
- Click fraud *(robots that click on advertisement)*

Question: Could merely visiting a malicious web-site (for e.g. wrongly clicked on a phishing email) subjected to attack?

Common simple implementation mistakes

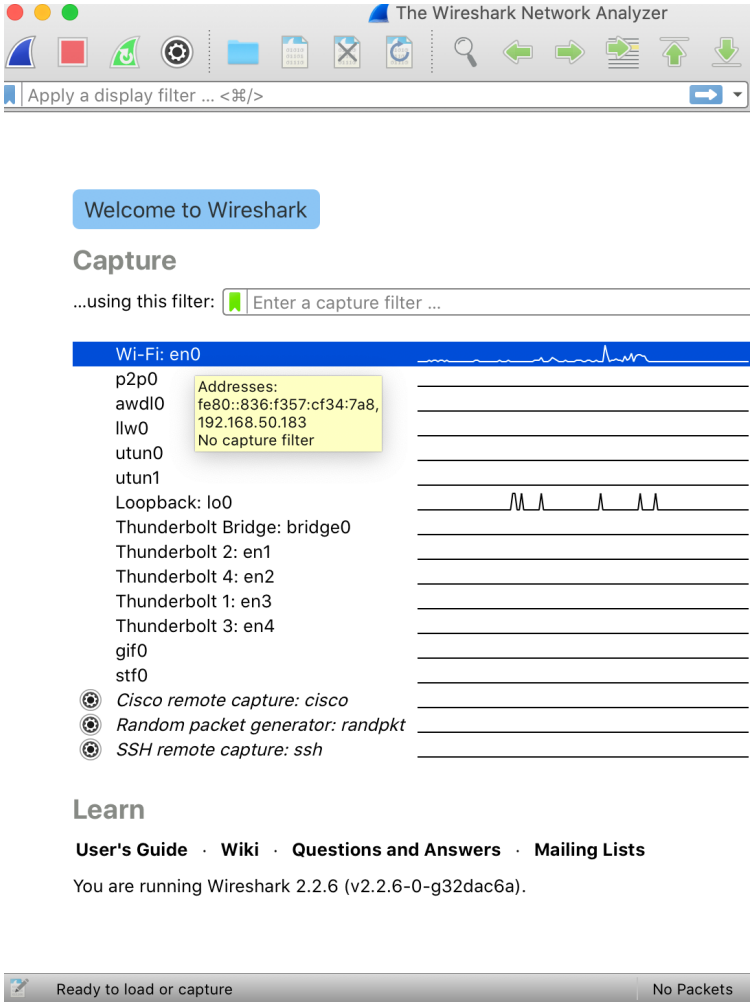
- Authentication/filtering at the client side.
- Security credential embedded in the public web pages.
- Server's secrets stored in cookies.
- Configuration errors.
- URL as secrets, e.g. in password reset link, or zoom link. This is acceptable and commonly used. However, some implementations do not have adequate protection or unknowingly leak info of the url.



https://motherboard.vice.com/en_us/article/43yqdd/look-at-this-massive-click-fraud-farm-that-was-just-busted-in-thailand

Wireshark Demo

(1) Choose the “network interface”



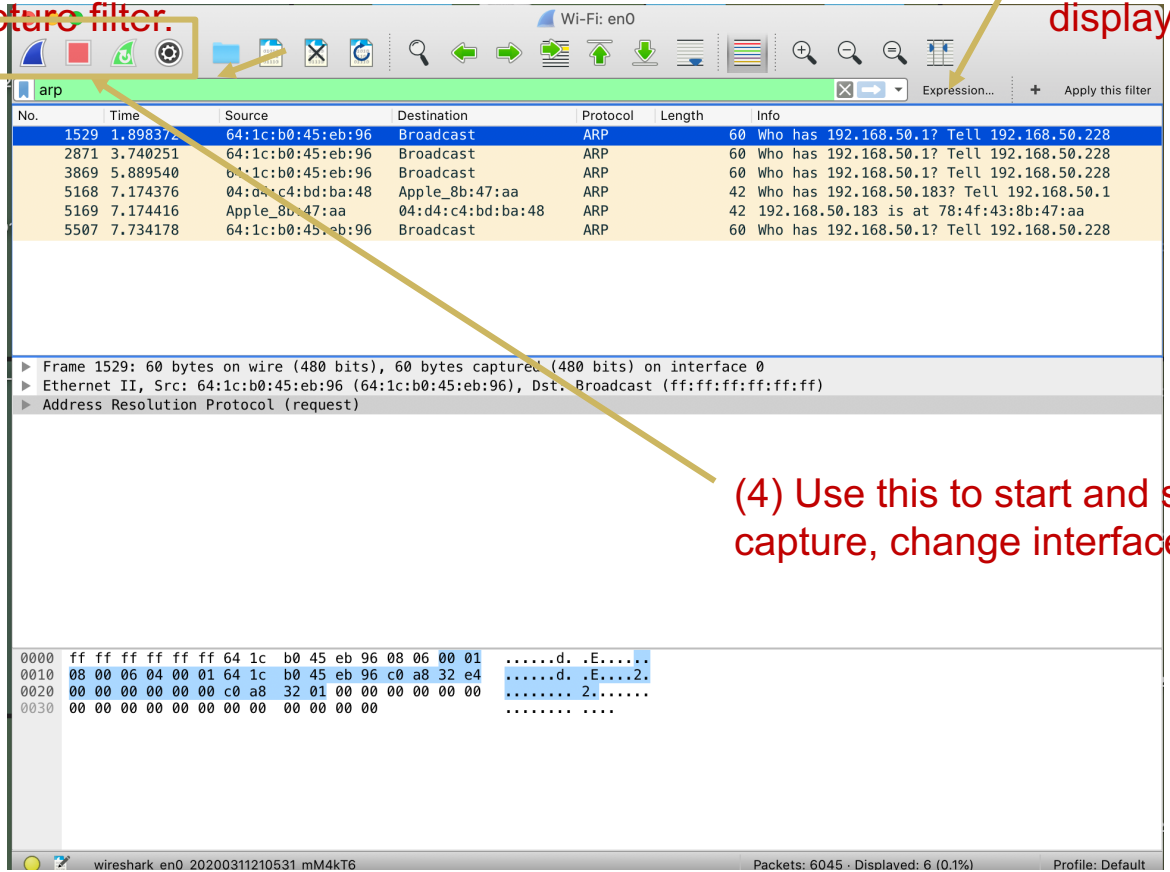
(2) Set a capture filter if required.
e.g simply leave it empty or “tcp”

This interface ip-address
Is 192.168.50.183.

Its mac address
(not shown in this screenshot)
Is 78:4f:43:8b:47:aa

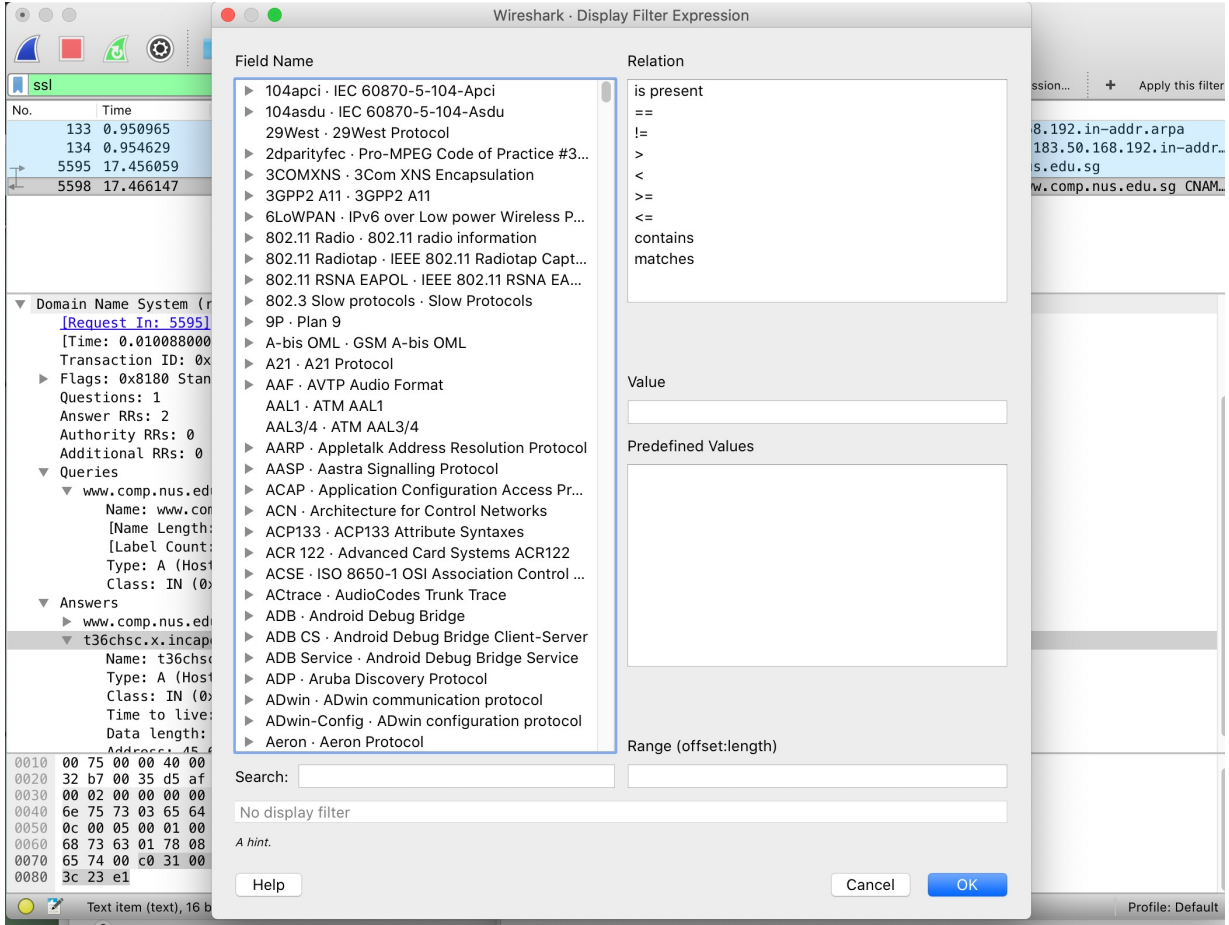
(3) Too many info. Use the display filter to select what to see. Note the difference between display and capture filter.

(4) Click here to see various options for display filter.

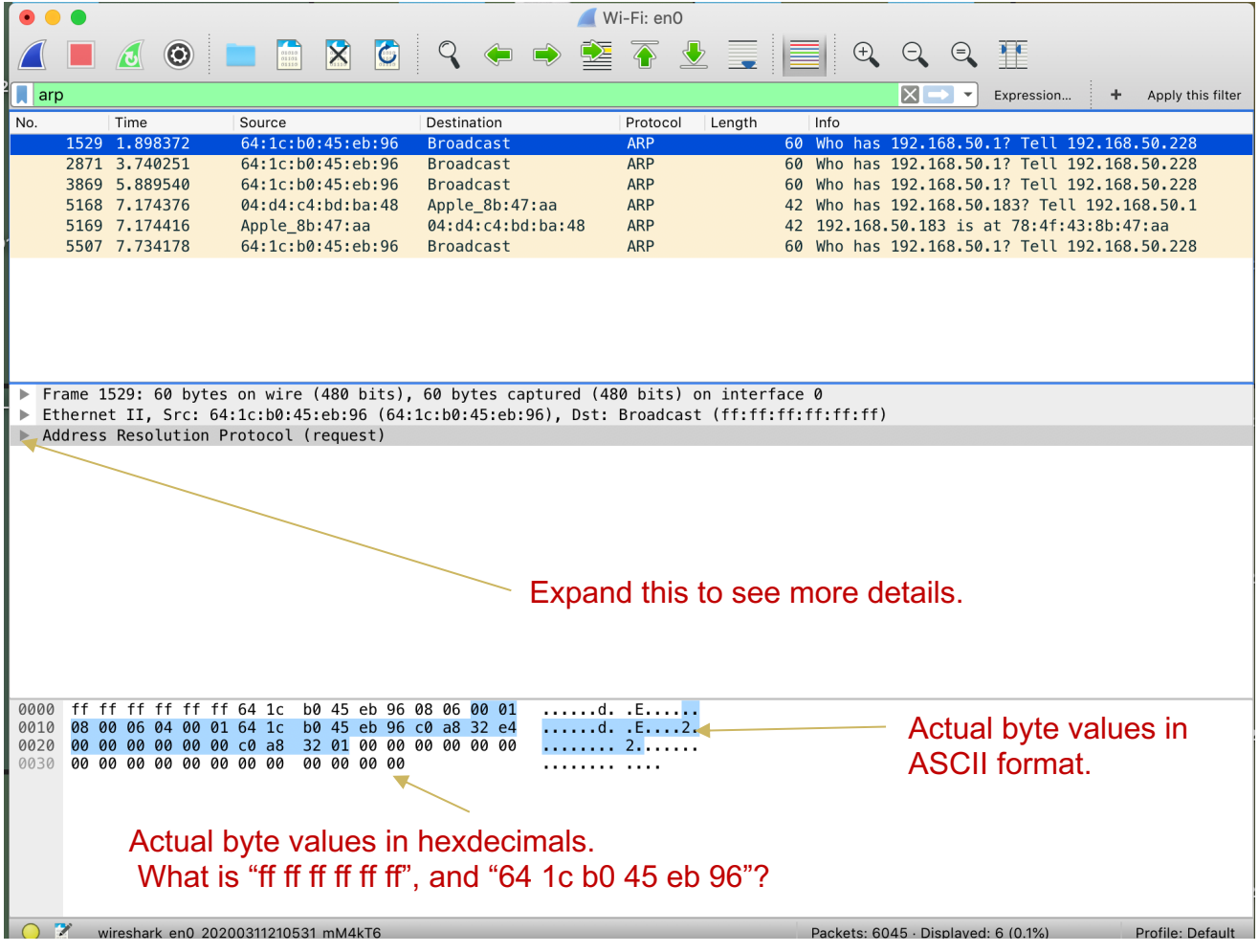


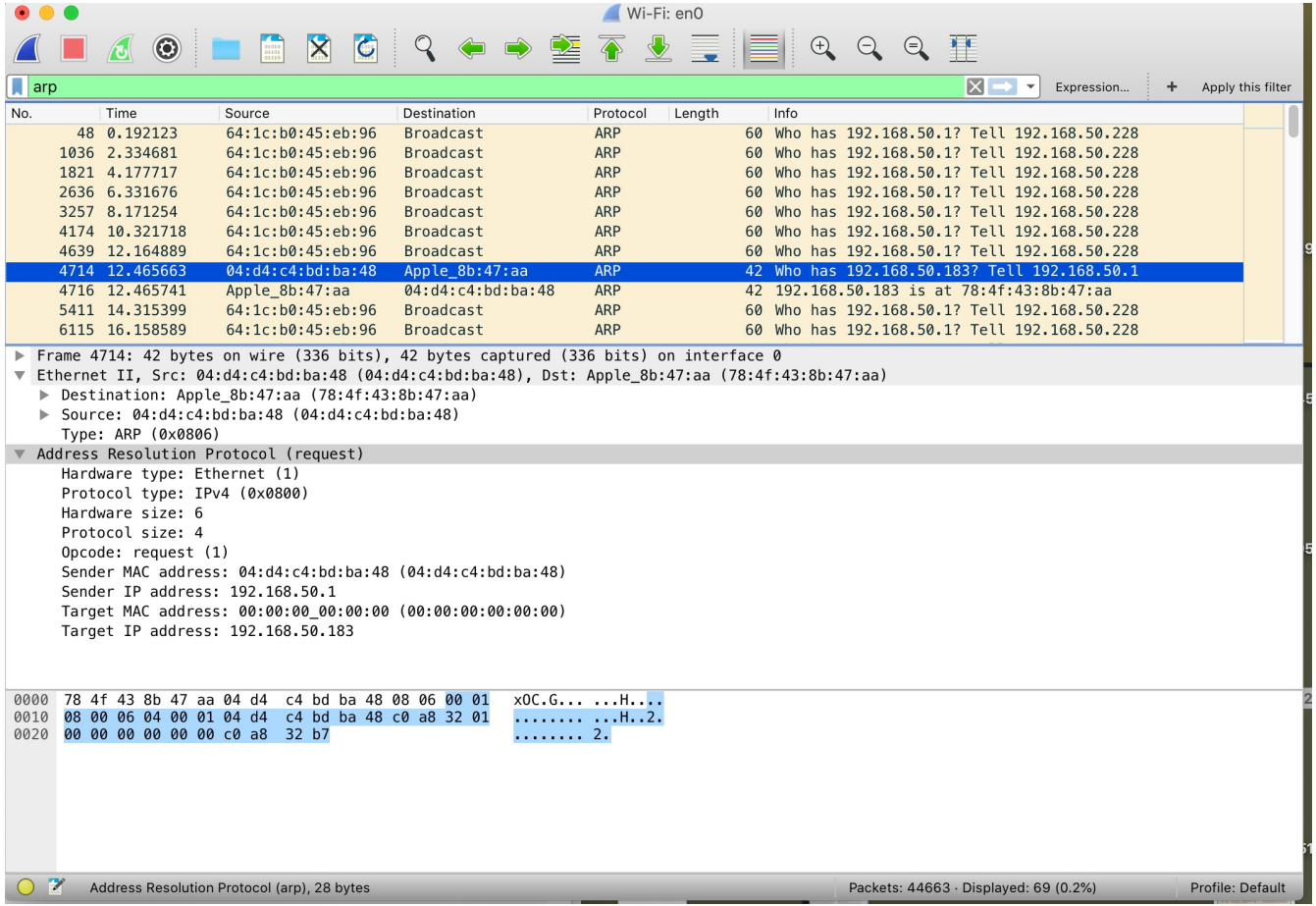
(4) Use this to start and stop capture, change interface.

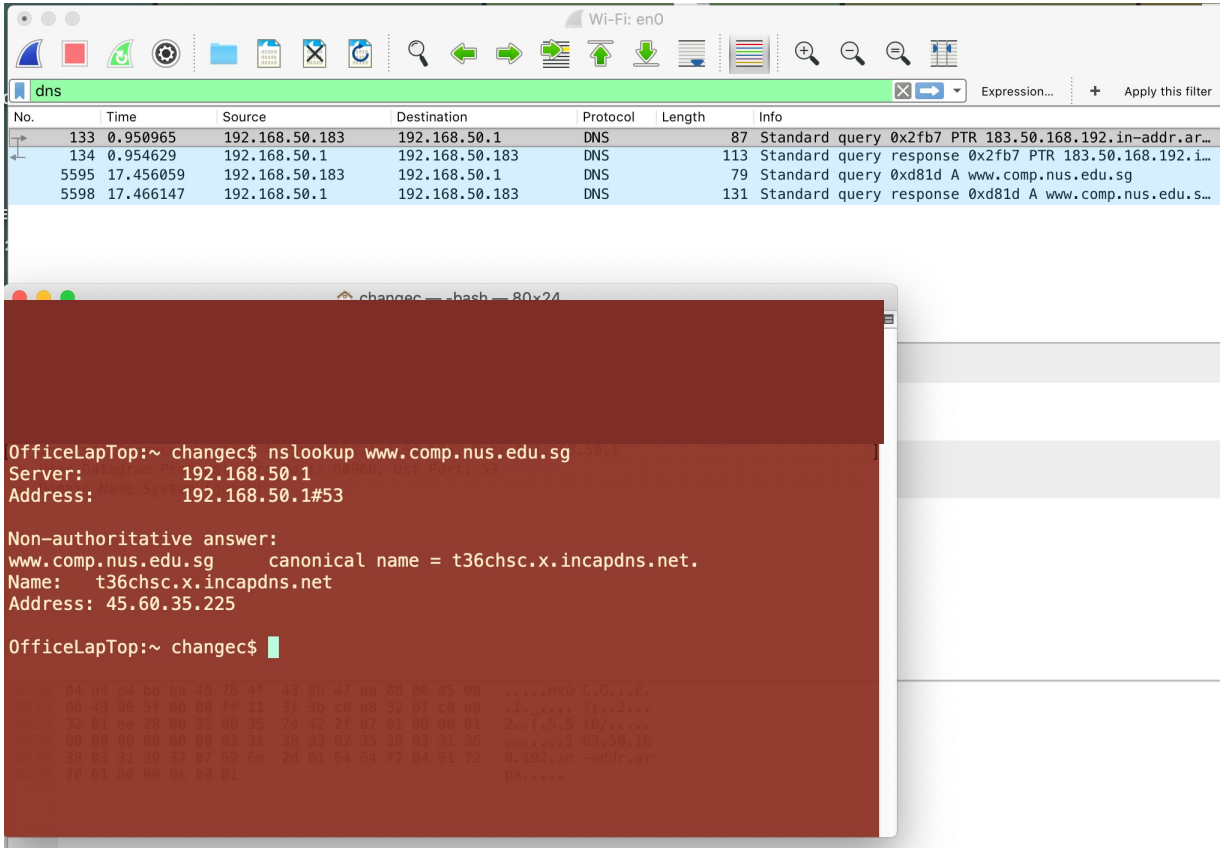
Display filter.

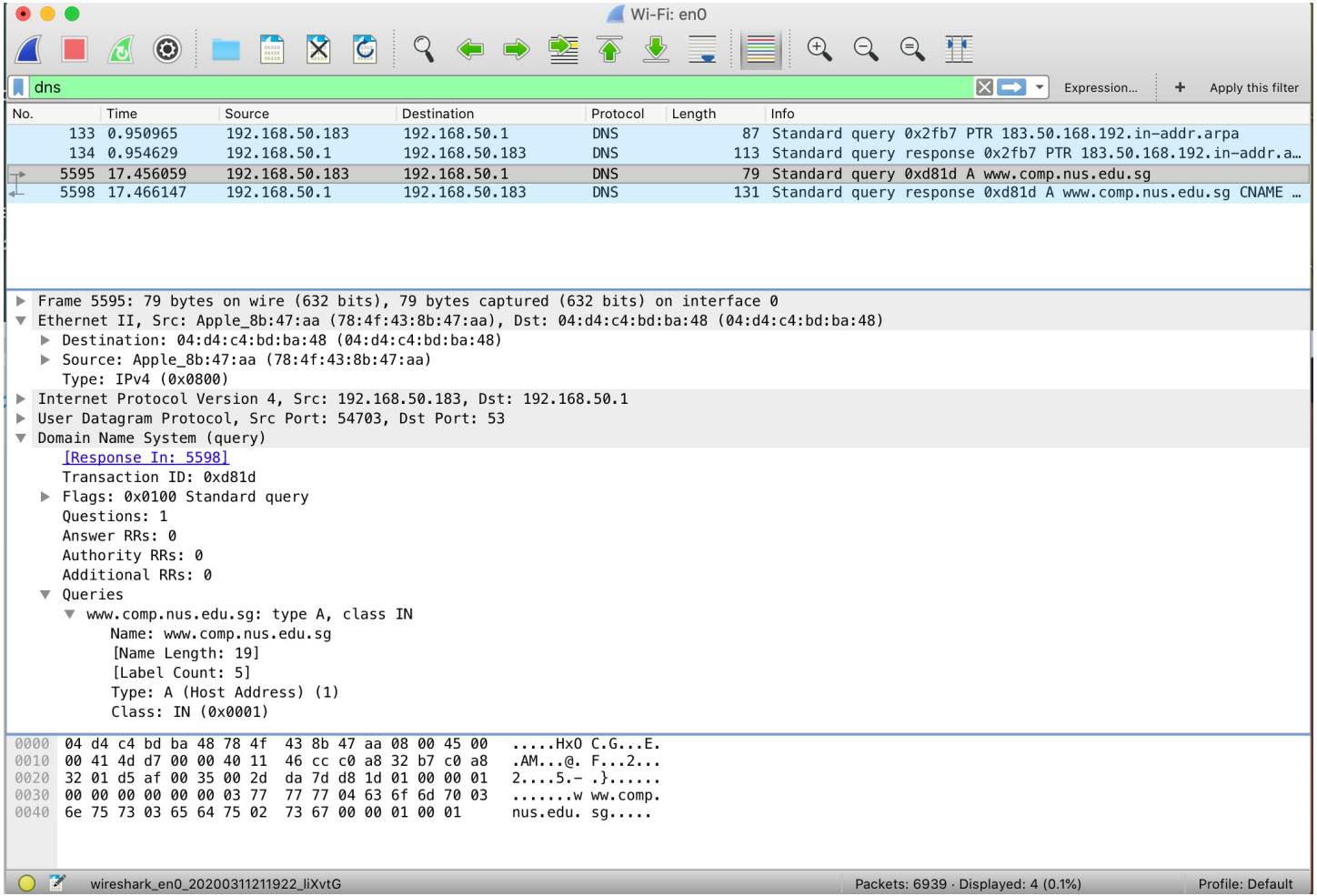


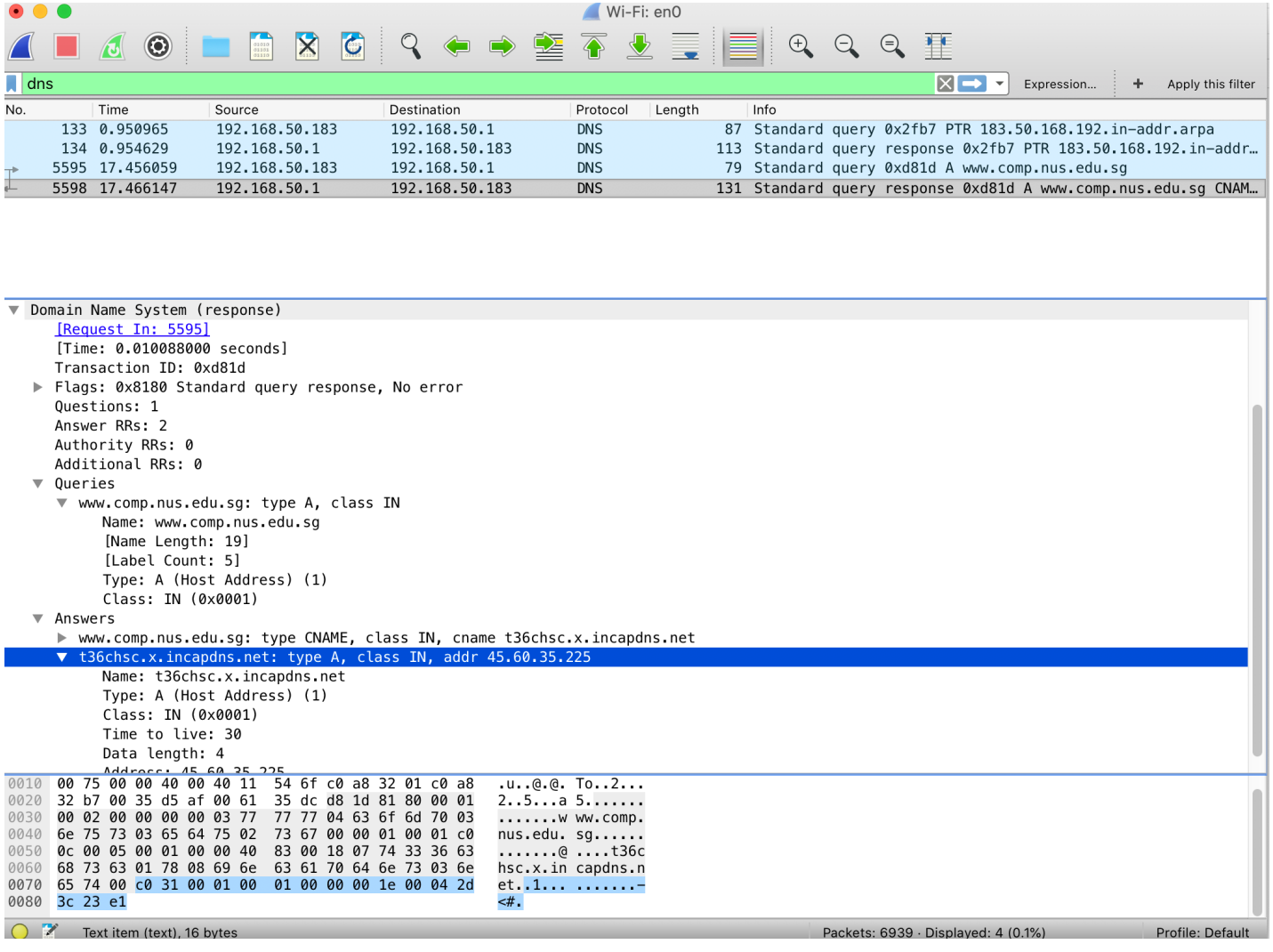
ARP: What is in the frame 5168 and 5169?




Wi-Fi: en0
arp
Expression...
Apply this filter
No. Time Source Destination Protocol Length Info
48 0.192123 64:1c:b0:45:eb:96 Broadcast ARP 60 Who has 192.168.50.1? Tell 192.168.50.228
1036 2.334681 64:1c:b0:45:eb:96 Broadcast ARP 60 Who has 192.168.50.1? Tell 192.168.50.228
1821 4.177717 64:1c:b0:45:eb:96 Broadcast ARP 60 Who has 192.168.50.1? Tell 192.168.50.228
2636 6.331676 64:1c:b0:45:eb:96 Broadcast ARP 60 Who has 192.168.50.1? Tell 192.168.50.228
3257 8.171254 64:1c:b0:45:eb:96 Broadcast ARP 60 Who has 192.168.50.1? Tell 192.168.50.228
4174 10.321718 64:1c:b0:45:eb:96 Broadcast ARP 60 Who has 192.168.50.1? Tell 192.168.50.228
4639 12.164889 64:1c:b0:45:eb:96 Broadcast ARP 60 Who has 192.168.50.1? Tell 192.168.50.228
4714 12.465663 04:d4:c4:bd:ba:48 Apple_8b:47:aa ARP 42 Who has 192.168.50.183? Tell 192.168.50.1
4716 12.465741 Apple_8b:47:aa 04:d4:c4:bd:ba:48 ARP 42 192.168.50.183 is at 78:4f:43:8b:47:aa
5411 14.315399 64:1c:b0:45:eb:96 Broadcast ARP 60 Who has 192.168.50.1? Tell 192.168.50.228
6115 16.158589 64:1c:b0:45:eb:96 Broadcast ARP 60 Who has 192.168.50.1? Tell 192.168.50.228
Frame 4714: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
Ethernet II, Src: 04:d4:c4:bd:ba:48 (04:d4:c4:bd:ba:48), Dst: Apple_8b:47:aa (78:4f:43:8b:47:aa)
Destination: Apple_8b:47:aa (78:4f:43:8b:47:aa)
Source: 04:d4:c4:bd:ba:48 (04:d4:c4:bd:ba:48)
Type: ARP (0x0806)
Address Resolution Protocol (request)
Hardware type: Ethernet (1)
Protocol type: IPv4 (0x0800)
Hardware size: 6
Protocol size: 4
Opcode: request (1)
Sender MAC address: 04:d4:c4:bd:ba:48 (04:d4:c4:bd:ba:48)
Sender IP address: 192.168.50.1
Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
Target IP address: 192.168.50.183
0000 78 4f 43 8b 47 aa 04 d4 c4 bd ba 48 08 06 00 01 xOC.G... ...H....
0010 08 00 06 04 00 01 04 d4 c4 bd ba 48 c0 a8 32 01 ....H..2.
0020 00 00 00 00 00 00 c0 a8 32 b7 .... 2.
Address Resolution Protocol (arp), 28 bytes
Packets: 44663 · Displayed: 69 (0.2%)
Profile: Default

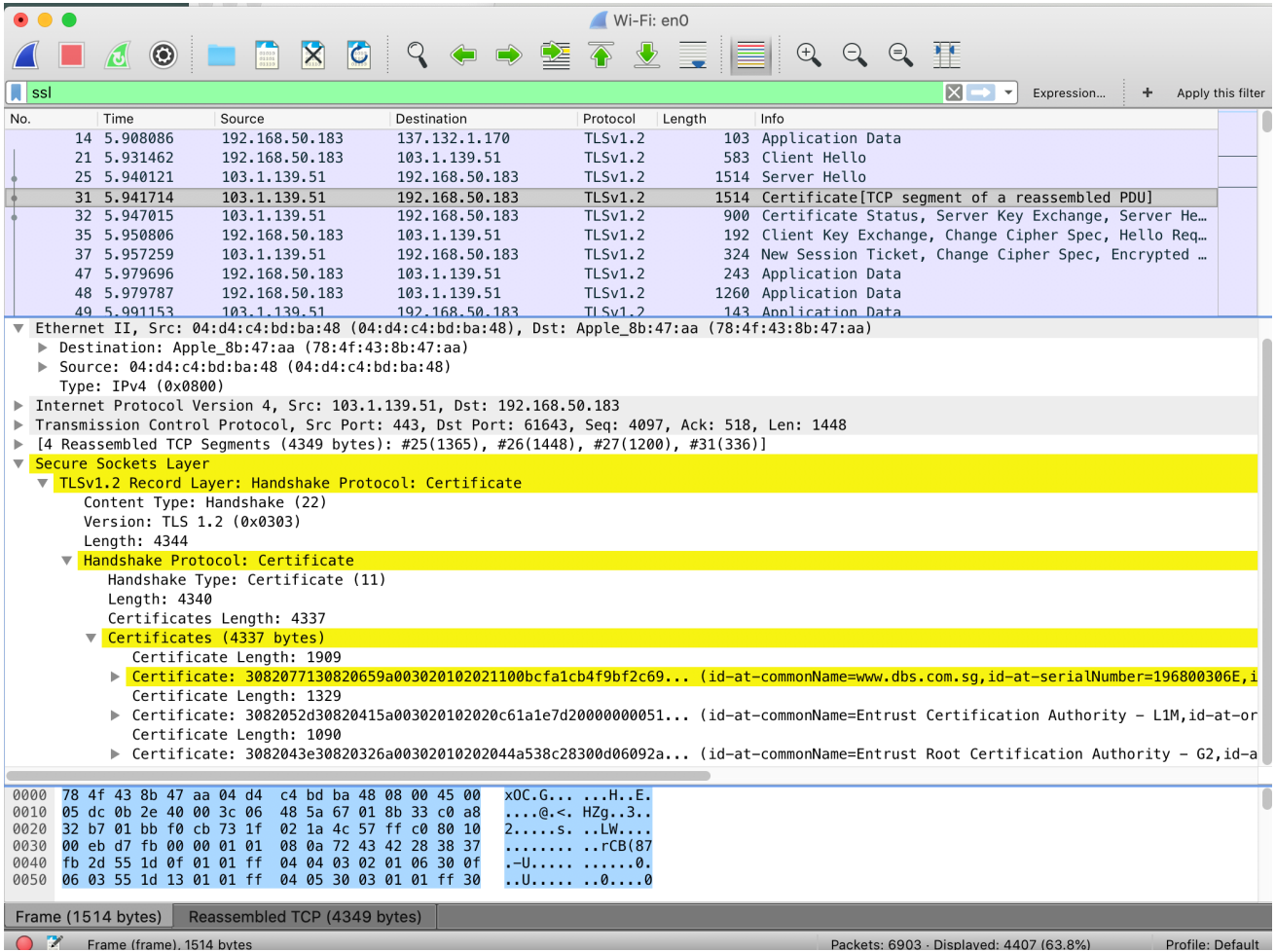

Wi-Fi: en0
dns
Expression...
Apply this filter
No. Time Source Destination Protocol Length Info
133 0.950965 192.168.50.183 192.168.50.1 DNS 87 Standard query 0x2fb7 PTR 183.50.168.192.in-addr.ar…
134 0.954629 192.168.50.1 192.168.50.183 DNS 113 Standard query response 0x2fb7 PTR 183.50.168.192.i…
5595 17.456059 192.168.50.183 192.168.50.1 DNS 79 Standard query 0xd81d A www.comp.nus.edu.sg
5598 17.466147 192.168.50.1 192.168.50.183 DNS 131 Standard query response 0xd81d A www.comp.nus.edu.s…
changec — -bash — 80×24
OfficeLapTop:~ changec$ nslookup www.comp.nus.edu.sg
Server: 192.168.50.1
Address: 192.168.50.1#53
Non-authoritative answer:
www.comp.nus.edu.sg canonical name = t36chsc.x.incapdns.net.
Name: t36chsc.x.incapdns.net
Address: 45.60.35.225
OfficeLapTop:~ changec$

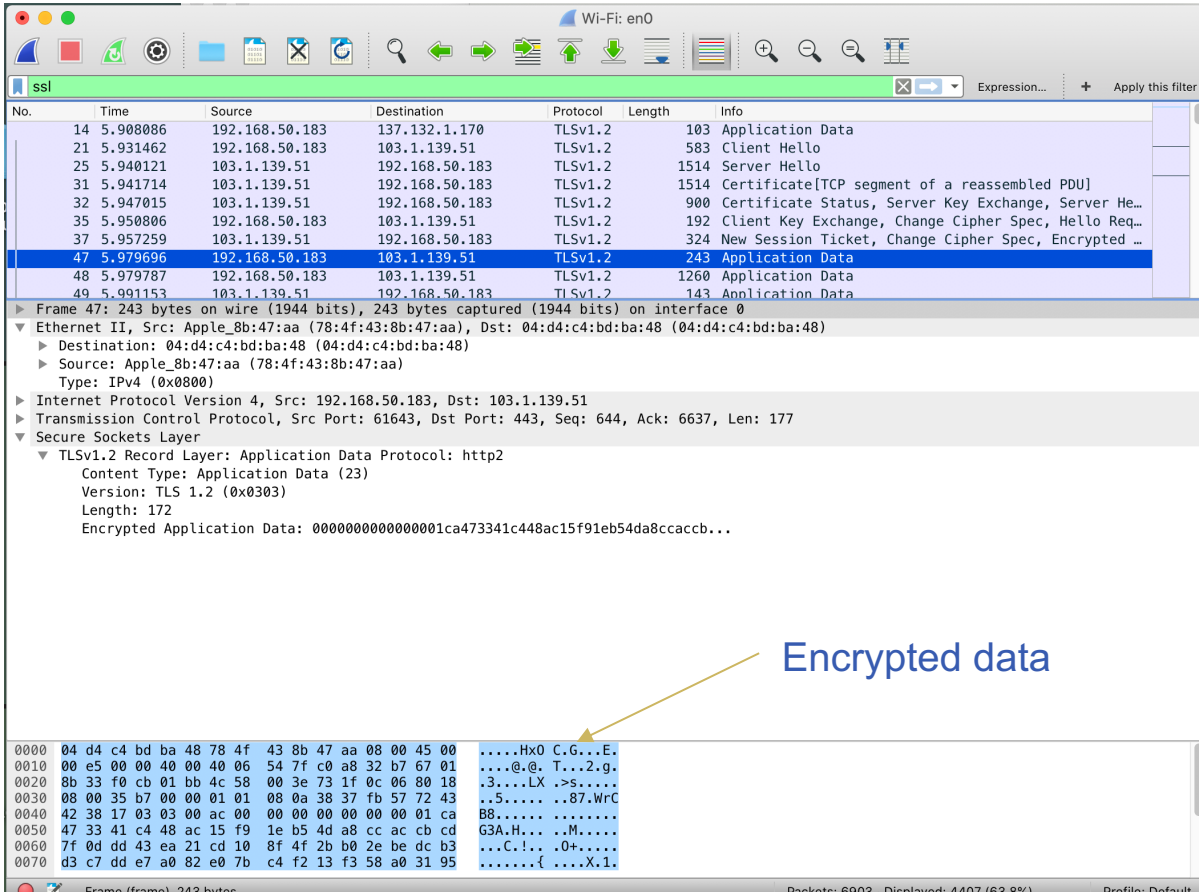

Wi-Fi: en0
dns
Expression...
Apply this filter
No. Time Source Destination Protocol Length Info
133 0.950965 192.168.50.183 192.168.50.1 DNS 87 Standard query 0x2fb7 PTR 183.50.168.192.in-addr.arpa
134 0.954629 192.168.50.1 192.168.50.183 DNS 113 Standard query response 0x2fb7 PTR 183.50.168.192.in-addr.a…
5595 17.456059 192.168.50.183 192.168.50.1 DNS 79 Standard query 0xd81d A www.comp.nus.edu.sg
5598 17.466147 192.168.50.1 192.168.50.183 DNS 131 Standard query response 0xd81d A www.comp.nus.edu.sg CNAME …
Frame 5595: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface 0
Ethernet II, Src: Apple_8b:47:aa (78:4f:43:8b:47:aa), Dst: 04:d4:c4:bd:ba:48 (04:d4:c4:bd:ba:48)
Destination: 04:d4:c4:bd:ba:48 (04:d4:c4:bd:ba:48)
Source: Apple_8b:47:aa (78:4f:43:8b:47:aa)
Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.50.183, Dst: 192.168.50.1
User Datagram Protocol, Src Port: 54703, Dst Port: 53
Domain Name System (query)
[Response In: 5598]
Transaction ID: 0xd81d
Flags: 0x0100 Standard query
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0
Queries
www.comp.nus.edu.sg: type A, class IN
Name: www.comp.nus.edu.sg
[Name Length: 19]
[Label Count: 5]
Type: A (Host Address) (1)
Class: IN (0x0001)
0000 04 d4 c4 bd ba 48 78 4f 43 8b 47 aa 08 00 45 00 .....HxO C.G...E.
0010 00 41 4d d7 00 00 40 11 46 cc c0 a8 32 b7 c0 a8 .AM...@. F...2...
0020 32 01 d5 af 00 35 00 2d da 7d d8 1d 01 00 00 01 2....5.- .}......
0030 00 00 00 00 00 00 03 77 77 77 04 63 6f 6d 70 03 ....w ww.comp.
0040 6e 75 73 03 65 64 75 02 73 67 00 00 01 00 01 nus.edu. sg.....
wireshark_en0_20200311211922_liXvtG
Packets: 6939 · Displayed: 4 (0.1%)
Profile: Default


Wi-Fi: en0
dns
Expression...
Apply this filter
No. Time Source Destination Protocol Length Info
133 0.950965 192.168.50.183 192.168.50.1 DNS 87 Standard query 0x2fb7 PTR 183.50.168.192.in-addr.arpa
134 0.954629 192.168.50.1 192.168.50.183 DNS 113 Standard query response 0x2fb7 PTR 183.50.168.192.in-addr…
5595 17.456059 192.168.50.183 192.168.50.1 DNS 79 Standard query 0xd81d A www.comp.nus.edu.sg
5598 17.466147 192.168.50.1 192.168.50.183 DNS 131 Standard query response 0xd81d A www.comp.nus.edu.sg CNAM…
Domain Name System (response)
[Request In: 5595]
[Time: 0.010088000 seconds]
Transaction ID: 0xd81d
Flags: 0x8180 Standard query response, No error
Questions: 1
Answer RRs: 2
Authority RRs: 0
Additional RRs: 0
Queries
www.comp.nus.edu.sg: type A, class IN
Name: www.comp.nus.edu.sg
[Name Length: 19]
[Label Count: 5]
Type: A (Host Address) (1)
Class: IN (0x0001)
Answers
www.comp.nus.edu.sg: type CNAME, class IN, cname t36chsc.x.incapdns.net
t36chsc.x.incapdns.net: type A, class IN, addr 45.60.35.225
Name: t36chsc.x.incapdns.net
Type: A (Host Address) (1)
Class: IN (0x0001)
Time to live: 30
Data length: 4
0010 00 75 00 00 40 00 40 11 54 6f c0 a8 32 01 c0 a8 .u..@.@. To..2...
0020 32 b7 00 35 d5 af 00 61 35 dc d8 1d 81 80 00 01 2..5...a 5.....
0030 00 02 00 00 00 00 03 77 77 77 04 63 6f 6d 70 03 .....w ww.comp.
0040 6e 75 73 03 65 64 75 02 73 67 00 00 01 00 01 c0 nus.edu. sg.....
0050 0c 00 05 00 01 00 00 40 83 00 18 07 74 33 36 63 .....@ t36c
0060 68 73 63 01 78 08 69 6e 63 61 70 64 6e 73 03 6e hsc.x.in capdns.n
0070 65 74 00 c0 31 00 01 00 01 00 00 00 1e 00 04 2d et..1... .....-
0080 3c 23 e1 <#.
Text item (text), 16 bytes
Packets: 6939 · Displayed: 4 (0.1%)
Profile: Default

E.g. on SSL/TLS

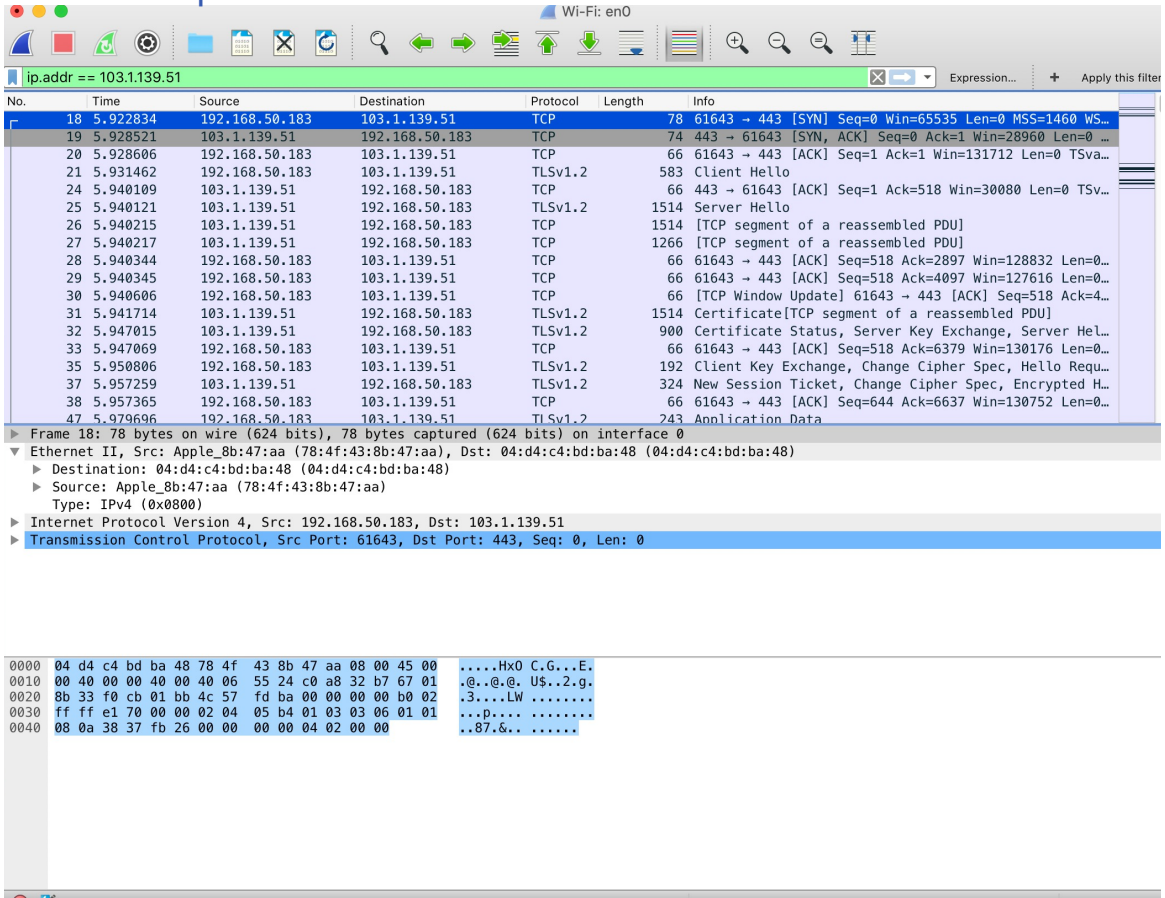
Set display filter to “ssl”. Find the TLS handshake (authenticated key exchange).



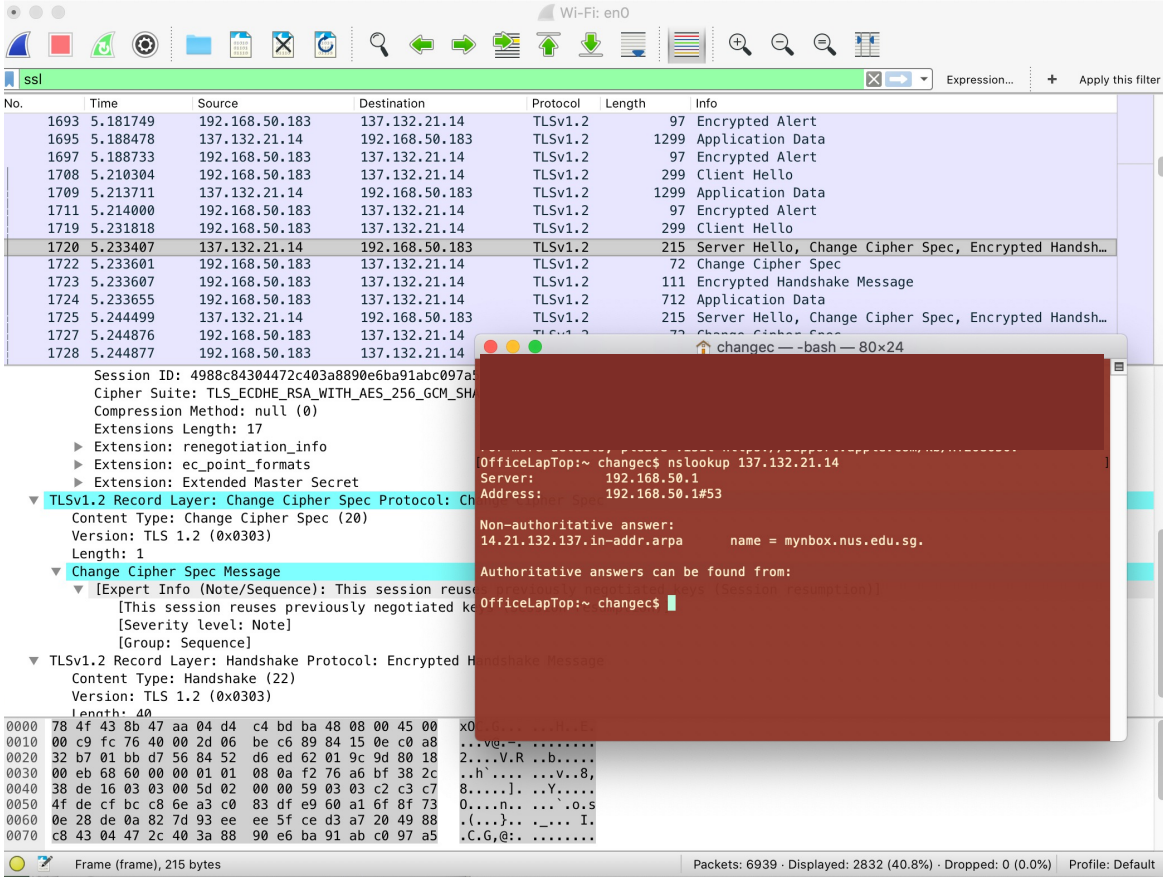

Wi-Fi: en0
ssl
Expression...
Apply this filter
No. Time Source Destination Protocol Length Info
14 5.908086 192.168.50.183 137.132.1.170 TLSv1.2 103 Application Data
21 5.931462 192.168.50.183 103.1.139.51 TLSv1.2 583 Client Hello
25 5.940121 103.1.139.51 192.168.50.183 TLSv1.2 1514 Server Hello
31 5.941714 103.1.139.51 192.168.50.183 TLSv1.2 1514 Certificate[TCP segment of a reassembled PDU]
32 5.947015 103.1.139.51 192.168.50.183 TLSv1.2 900 Certificate Status, Server Key Exchange, Server He…
35 5.950806 192.168.50.183 103.1.139.51 TLSv1.2 192 Client Key Exchange, Change Cipher Spec, Hello Req…
37 5.957259 103.1.139.51 192.168.50.183 TLSv1.2 324 New Session Ticket, Change Cipher Spec, Encrypted …
47 5.979696 192.168.50.183 103.1.139.51 TLSv1.2 243 Application Data
48 5.979787 192.168.50.183 103.1.139.51 TLSv1.2 1260 Application Data
49 5.991153 103.1.139.51 192.168.50.183 TLSv1.2 143 Application Data
Frame 47: 243 bytes on wire (1944 bits), 243 bytes captured (1944 bits) on interface 0
Ethernet II, Src: Apple_8b:47:aa (78:4f:43:8b:47:aa), Dst: 04:d4:c4:bd:ba:48 (04:d4:c4:bd:ba:48)
Destination: 04:d4:c4:bd:ba:48 (04:d4:c4:bd:ba:48)
Source: Apple_8b:47:aa (78:4f:43:8b:47:aa)
Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.50.183, Dst: 103.1.139.51
Transmission Control Protocol, Src Port: 61643, Dst Port: 443, Seq: 644, Ack: 6637, Len: 177
Secure Sockets Layer
TLSv1.2 Record Layer: Application Data Protocol: http2
Content Type: Application Data (23)
Version: TLS 1.2 (0x0303)
Length: 172
Encrypted Application Data: 0000000000000001ca473341c448ac15f91eb54da8ccaccb...
Encrypted data
0000 04 d4 c4 bd ba 48 78 4f 43 8b 47 aa 08 00 45 00 .....HxO C.G...E.
0010 00 e5 00 00 40 00 40 06 54 7f c0 a8 32 b7 67 01 ....@.@. T...2.g.
0020 8b 33 f0 cb 01 bb 4c 58 00 3e 73 1f 0c 06 80 18 .3....LX .>s.....
0030 08 00 35 b7 00 00 01 01 08 0a 38 37 fb 57 72 43 ..5..... ..87.WrC
0040 42 38 17 03 03 00 ac 00 00 00 00 00 00 00 01 ca B8...... ....
0050 47 33 41 c4 48 ac 15 f9 1e b5 4d a8 cc ac cb cd G3A.H... ..M.....
0060 7f 0d dd 43 ea 21 cd 10 8f 4f 2b b0 2e be dc b3 ...C.!.. .O+.....
0070 d3 c7 dd e7 a0 82 e0 7b c4 f2 13 f3 58 a0 31 95 ....{ ....X.1.

Looking at all IP packets from the ip.adder 103.1.139.51

Note that at TCP layer, there are many other more interactions (3 way-handshake, the “ACK” etc). Details not required in this module.



We can use nslookup to find out more about an ipaddress.
(known as reverse DNS lookup. It finds the domain name from ip-address.) We can also find the geolocation by using some services from the web. Does not always work.



There are many more tools and features. (e.g. the “follow”, “Analyze”, “export” etc)
Explore to find out more.