Daniel Bigos
Tomasz Główka
Dariusz Kurek
Piotr Żelazko

TIN 2018L
11.06.2018

Dokumentacja projektu TIN 2018L

Treść zadania

Badanie wykorzystania opcji TCP w łączu WiFi pomiędzy smartfonem a laptopem.
Celem projektu jest stworzenie oprogramowania umożliwiającego przetestowanie, które opcje TCP można wykorzystać w komunikacji ze smartfonem po sieci WiFi, na które rozszerzenia pozwalają mechanizmy zaimplementowane w systemach mobilnych oraz kartach sieciowych testowanych urządzeń. Do zakresu projektu również należy przetestowanie ww. opcji.

Projekt oparty został na poniższych pracach inżynierskich:

- Kamil Skrobek: Testowanie opcji protokołu TCP;
- Paweł Kaźmierczak: Testy dla opcji protokołu TCP.

Nazwa własna

Projekt nosi nazwę *tcp-analyser.*

Wymagania

Funkcjonalne:

- Zbieranie danych z pomiarów w formie nadającej się do analizy
- Podsłuchiwanie ruchu pomiędzy dwoma urządzeniami używającymi naszego oprogramowania
- Symulowanie obciążenia łączy
- Każda aplikacja powinna pracować w trybie zarówno klienta jak i serwera
- W trybie klienta powinna istnieć możliwość wybrania IP serwera
- Testowanie pojedynczych rozszerzeń TCP - użytkownik powinien móc wyspecyfikować, które rozszerzenia chce testować

Niefunkcjonalne:

- Aplikacje mobilne powinny udostępniać prosty interfejs graficzny
- Aplikacja stacjonarna powinna pracować jedynie w trybie wsadowym
- Argumenty uruchomienia aplikacji stacjonarnej powinny być w notacji używanej w dokumentacji biblioteki Boost.Program_options

Podstawowe przypadki użycia

Uruchomienie testu

Aktorzy : użytkownik
Warunki początkowe : dwa urządzenia z zainstalowanymi aplikacjami oraz podłączone do tej samej sieci, znajomość IP obu urządzeń, urządzenie z zainstalowanym oprogramowaniem podsłuchującym ruch.
1. Użytkownik wybiera tryb pracy obu urządzeń (serwer/klient).
2. Użytkownik wybiera rozszerzenie TCP, które chce przetestować.
3. Użytkownik uruchamia test na serwerze.
4. Użytkownik na urządzeniu klienckim podaje adres IP serwera.
5. Użytkownik uruchamia test na kliencie.
6. Użytkownik zbiera logi.

Środowisko sprzętowo-programowe

Platformy:

- iOS (wersja 11.3);
- Android (wersja 6.0.1);
- Debian (wersja 9.1);
- Ubuntu (wersja 16.04) – platforma konfigurowania opcji TCP.

Środowisko sprzętowe:

- iPhone SE;
- iPhone 6s;
- Asus Zenfone;
- Router WiFi;
- Laptop z wbudowaną kartą sieciową WiFi.

Na środowisko programowe składa się:

- WireShark - podglądanie komunikacji miedzy urządzeniami, analiza pakietów;
- IDE - użyjemy narzędzi firmy Jetbrains (np. IntelliJ, Clion, AppCode)
- Biblioteki Boost, w szczególności Boost.Test.

Zmiany względem dokumentacji wstępnej:

Zrezygnowano z DummyNet i DBS, narzędzi symulujących obciążenia sieci, gdyż według wstępnych założeń jeżeli opcja zostaje zaobserwowana przy handshake'u to przyjmuje się, że jest ona zaimplementowana i nie ma potrzeby weryfikować jej symulacjami.

Architektura rozwiązania

Na rozwiązanie składa się kilka aplikacji (mobilnych oraz stacjonarnych) oraz konfiguracje narzędzi.

Aplikacje komunikują się między sobą przez pakiety TCP/IP. Każda z aplikacji może przyjąć rolę serwera albo klienta. W wyniku tego żadne z urządzeń nie jest traktowane na równi, nie ma znaczenia czy uruchomiamy serwer na iOS i klienta na Androidzie czy odwrotnie, wszystkie kombinacje są dozwolone. Dlatego też każda aplikacja poza interfejsem jest zbudowana bardzo podobnie. Każda aplikacja jest podzielona na następujące moduły (ze względu na to za co są odpowiedzialne):

- Interfejs użytkownika;
- Serwer;
- Klient;
- Zbieranie logów – dla aplikacji stacjonarnych.

Repozytorium GIT

Adres repozytorium git : https://github.com/pik694/tin-tcp-analyser.

Podział prac

Wykaz osób odpowiedzialnych za poszczególne części projektu:

1. Konfiguracja narzędzi - wszyscy (każdy z członków zespołu skonfiguruje przydzielone mu narzędzia);
2. Aplikacja stacjonarna - wszyscy
3. Aplikacja na iOS - Piotr Żelazko i Daniel Bigos;
4. Aplikacja na Androida - Tomasz Główka oraz Dariusz Kurek;
5. Analiza wyników - Dariusz Kurek i Tomasz Główka
6. Koordynacja prac - Piotr Żelazko

Założenia projektu

Badanie opcji TCP odbywało się w oparciu o założenie, że jeżeli strony serwera i klienta nawiązują połączenie z zadaną opcją TCP to dana opcja jest uznana za działającą/zaimplementowaną po obydwu stronach i nie ma potrzeby jej dokładniej analizować.

W związku z tym założeniem zrezygnowano z tworzenia modułu do WireShark oraz korzystania z innych programów wymienionych w dokumentacji wstępnej.

Badania również nie podejmowały zagadnienia czy na podstawie ruchu pakietów widać, że dana opcja jest stosowana, jak i też nie badano zakresu i funkcjonalności danych opcji.

Wytworzone oprogramowanie

W wyniku pracy nad projektem wytworzono następujące aplikacje:

- Aplikacja Android, w trybach pracy klient, serwer
- Aplikacja iOS, w trybach pracy klient, serwer
- Aplikacja Unix, w trybach pracy klient, serwer

Celem działania aplikacji jest nawiązanie połączenia TCP, przesłanie danych tekstowych w ramach komunikacji i zakończenie połączenia.
Wszystkie aplikację działają zarówno w trybie klienta jak i serwera (do wyboru przez użytkownika) i pobierają adres IP i port połączenia.
Aplikacja Unix umożliwia ustawienie opcji TCP, jako że API iOS nie pozwala na to.

Wstępne rozpoznanie

Rozpoznanie opcji TCP

Zapoznanie się z listą opcji TCP[1] ujawniło, że 10 opcji dla tego protokołu jest nieaktualna (‘obsolete’) ze względu na zastąpienie ich funkcjonalności innymi opcjami TCP, np. opcja ‘Timestamps’ (nr. 8) zdezaktualizowała opcje ‘Echo’ (nr. 6) i ‘Echo Reply’ (nr. 7).
Dodatkowo na liście figuruje wiele pozycji opcji które nie zostały opisane w dokumentach RFC, np. opcje 16 i 17 („Skeeter” i „Bubba”) stanowią pozostałości porzuconego projektu. [2]
Poza tym znajdują się tam numery opcji które zostały zarezerwowane pod przyszłe funkcjonalności.

Rozpoznanie implementacji na systemach Linux

Dla systemów Linux włączenie bądź wyłączenie danych opcji TCP wiąże się z modyfikacją pliku/ów w folderze /proc/sys/net/ipv4/. [3] Weryfikacja wymienionej funkcjonalności wykazała jednak, że wiele z aktualnych opcji TCP nie jest zaimplementowana w jądrze systemu Linux.

Rozpoznanie implementacji na systemach mobilnych

Na podstawie poprzedniego punktu przyjęto założenie, że dla systemu Android (który bazuje na jądrze Linuxa) opcje niezaimplementowane dla jądra Linux nie są również zaimplementowane w systemach operacyjnych Android.

Rozpoznanie funkcjonalności wykrywania zerwania połączenia TCP dla urządzeń mobilnych

Wykryto, że dla obydwu systemów mobilnych Android[4] i iOS[5] zaimplementowane są mechanizmy których celem jest wykrycie czy połączenie TCP zostało zerwane.

Wyniki testów i analiza

W wyniku uruchomienia wcześniej wymienionych aplikacji i monitorowania ruchu przez program Wireshark uzyskano logi (formatu .pcapng), które zostały załączone do niniejszej dokumentacji. W wyniku analizy ich treści zanotowano następujące wystąpienia opcji:

Komunikacja (serwer - klient)	**Występujące opcje TCP:**
Android - Unix: bez wymuszenia opcji	Timestamps
Android - Unix: wszystkie opcje	Timestamps
iPhone - Unix: bez wymuszenia opcji	Timestamps
iPhone - Unix: wszystkie opcje	Timestamps, SACK
Unix – Android: bez wymuszenia opcji	Timestamps
Unix – Android: wszystkie opcje	Timestamps, SACK
Unix – iPhone: bez wymuszenia opcji	Timestamps
Unix – iPhone: wszystkie opcje	Timestamps, SACK

Z powyższych wyników widać, że jedyną domyślnie włączoną opcją TCP, która występuje dla wszystkich zbadanych komunikacji jest opcja ‘Timestamps’.

Jedyną opcją którą udało się uzyskać przy wymuszeniu wszystkich możliwych opcji okazało się ‘SACK’.

Na podstawie powyższych rezultatów można wyciągnąć wniosek, że zbadane systemy mobilne nie implementują w pełni standardu TCP, jedynie podstawę niezbędną do prostej komunikacji.

Opcje TCP dla Windows OS

W związku z koncentracją pracy nad urządzeniami mobilnymi oraz niemożnością przetestowania urządzeń mobilnych z Windows OS, nie przetestowano opcji TCP dla systemów Windows. W związku z tym przeszukano dokumentacje i serwisy pomocy dotyczące Windows OS na podstawie czego stwierdzono:

- Zarządzanie opcjami TCP dla systemów Windows (Desktop i Server) jest rozproszone pomiędzy wpisy rejestru i konfigurację netsh. Niektóre opcje zostały wyszczególnione dla systemów Windows Server – co umożliwia też odnalezienie ich w innych systemach. Niemniej nie da się wykluczyć czy potwierdzić, że w innych systemach one działają lub nie.
- Dojście do konfiguracji odbyło się poprzez wyszukiwanie każdej opcji z osobna (symbol „?" oznacza brak rezultatów, także i potwierdzeń nie istnienia takiej opcji).
- Niektóre opcje występują pod inną nazwą („Window Scale" występuje pod nazwą „Autotuning") lub stanowią część innej opcji („Maximum Segment Size" stanowi część „Maximum Transfer Unit").

Poniżej tabela rezultatów dla wyszukiwania konfiguracji wybranych opcji TCP dla Windows:

Nr. opcji	Nazwa Opcji	Lokalizacja	Uwagi
2	Maximum Segment Size[6]	Terminal: "netsh"	Składowa MTU: MTU=MSS + TCP & IP headers
3	Window Scale[7]	Terminal: "netsh"	Domyślnie włączone; Figuruje jako Autotuning: "netsh interface tcp set global autotuninglevel=disabled"
4	SACK Permitted[8]	Regedit (rejestr): SackOpts	HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters; Może nie działać w niektórych dystrybucjach
8	Timestamps[9]	Rejestr; standard Winsock	IPPROTO_TCP > TCP_TIMESTAMPS;
34	TCP Fast Open Cookie[10]	Rejestr; standard Winsock	IPPROTO_TCP > TCP_FASTOPEN

Podsumowanie

Refleksje z realizacji projektu

Znacząca część projektu wiązała się z wyszukaniem informacji w Internecie. Wyszukiwanie informacji dotyczących implementacji opcji TCP jako całości, a nie opcji składowych, było szczególnie trudne, gdyż to zagadnienie nie jest nigdzie opisane wprost. Można wyszukiwać implementacji pojedynczych opcji, ale w przypadku nie znalezienia informacji nie można wykluczyć, że ta opcja nie jest implementowana. Może to wiązać się z tym, że opcja jest zrealizowana jako część innej funkcjonalności.

Kolejna refleksja zakłada, że bardzo mało ludzi schodzi poniżej poziomu sockets co mogłoby też tłumaczyć brak materiałów wprowadzających dla osób nieobeznanych z tym. Z tego powodu jest mało źródeł związanych z opcjami, jak i też nie są one zebrane w jednym miejscu.

Zadziwiające jest to, że pomimo rezerwacji opcji TCP, wiele z nich nie jest wykorzystywanych (zaimplementowanych).

Oszacowanie czasu poświęconego na projekt

Przeszukiwanie dokumentacji i źródeł informacji	130h
Pisanie aplikacji	80h
Testy i opracowanie wyników	50h
Razem	**260h**

Źródła:

1. Specyfikacja opcji TCP:
   https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml
2. Opcje :
   https://www.ietf.org/mail-archive/web/tcpm/current/msg05424.html
3. TCP dla OS Linux:
   https://www.unix.com/man-page/all/7/tcp/
4. Heartbeat dla Android:
   https://productforums.google.com/forum/#!msg/nexus/fslYqYrULto/lU2D3Qe1mugJ
5. Keepalive dla iOS:
   https://stackoverflow.com/questions/25665837/is-it-possible-to-activate-tcp-keepalive-on-apple-ios-devices
6. Maximum Segment Size/MTU:
   https://serenity-networks.com/how-to-change-the-tcpip-mtu-on-windows-server-2016/
7. Window Scale:
   https://technet.microsoft.com/en-us/library/cc162519.aspx
   http://www.thewindowsclub.com/window-auto-tuning-in-windows-10
8. Sack Permitted:
   https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc938200(v=technet.10)
9. Timestamps:
   https://support.microsoft.com/en-us/help/224829/description-of-windows-2000-and-windows-server-2003-tcp-features
10. TCP Fast Open Cookie:
    https://msdn.microsoft.com/en-us/library/windows/desktop/ms738596(v=vs.85).aspx