Programowanie Systemowe

Sprawozdanie Projektowe

Temat Projektu:

Tunelowanie segmentów TCP

Autor:
Jan Wiśniewski
Nr. albumu 197662

Data złożenia sprawozdania:
29 stycznia 2026

Spis treści

TCP Header				
Bits	0-15			16-31
0	Source port			Destination port
32	Sequence number			
64	Acknowledgment number			
96	Offset	Reserved	Flags	Window size
128	Checksum			Urgent pointer
160	Options			

Rysunek 1: Struktura nagłówka TCP

1 Wstęp

Celem projektu była implementacja tunelu korzystajac z nieużywanego miejsca w nagłówku segmentu TCP oraz dodanie szyfrowania dla samych pakietów.

1.1 Układ Nagłówków

Nagłowek segmentu TCP zawiera 20 bajtów. Definicja *Options* jest listą składającą sie z elementów określających pojedyńczą opcje - rodzaju (kind, 1 byte), rozmiaru (length, 1 byte) i danych (data, zmienna długość). Liste musi kończyć zarezerwowana opcja o rodzaju 0 (End of options list) mająca tylko jeden bajt (kind).

Opcje mogą rozszeżyc nagłówek do łącznie 60 bajtów. Pomimo zarezerwowania określonej liczby opcji przez specyfikacje, w praktyce przynajmniej 20 bajtów jest zostawione niewykorzystane. Pozwala nam to przesłać nas własny payload i uniknąć skomplikowanego parsowania własnych nagłówków np. wynikającego z fragmentacji pakietów w protokole IPv4.

Na rysunku 1 został przedstawiony układ pól nagłówka. Options znajduje się miedzy stałym offsetem wynikającym z poprzednich pól - *hdrlen* oraz przesunieciem (pole Offset bajt 96), nazwanym *doff* - data offset.

Poprzez definicje własnej opcji z orginalnym adresem IP pakietów, możemy zmienić trasowanie[1] bez utraty informacji gdzie orginalnie pakiet miał być przesłany. Pozwala to stworzyć de facto automatyczny, dynamiczny NAT.

1.2 Trasowanie

Implementacja tunelu bedzie wymagała dwóch maszyn które przyjmą role przekażnika i klienta. Trasowanie rozpatrzymy z bioroąc pod uwage wysłąnie pakietów do internetu i otrzymanie ich z internetu.

Wysłanie

Klient będzie modyfikował destynacje wszystkich pakiety pochodzących

[1] w monenklaturze angielskiej również znane jako routing

od niego do przekaźnika oraz zapamieta orginalny cel podróży w opcjach TCP. Przekaźnik dla każdego otzymanego pakietu od klienta wydobedzie orginalny adres celu podróży oraz adres źródła ustawi na siebie poczym prześle pakiet dalej.

Otrzymanie
Przekaźnik dla każdego innego pakietu który nie należy do klienta ustawi adres celu podróży na klienta a źródła na siebie oraz zapamieta orginalne źródło w opcjach TCP. Kiedy klient otrzyma pakiet wydobedzie orginalny adres IP z opcji i ustawi go na pakiecie.

Otwarte gniazdo przez klienta nie ma wiedzy ze adres został podmieniony. Jeśli socket został utworzony do maszyny o adresie 10.0.0.2 to w momencie otrzymania odpowiedzi (która przechodzi przez przekaźnik) należy pamietać o zamienie adresu z przekaźnika na 10.0.0.2. Dzieki temu gniazdo rozpozna że pakiet jest z nim skojarzony.

2 Implementacja

Do implementacji zostanie wykorzystany modół kernela *netfilter* umożliwiający umieszczenie haków do poszczególnych etapów podróży pakietów w jądrze. Zostaną wykorzystane dwa punkty do filracji, `PREROUTING` i `POSTROUTING`. W `PREROUTING` zostanie podmieniony adres destynacji.

Jeśli adres zródła i destynacji jest rózny od localnej maszyny to w przekaźniku spowoduje to przekazanie pakietu do `ip_forward` (rysunek 2) i dalsze wysłanie pakietu korzystając ze scieżek ustawionych przez tablice trasowania[2]. W `POSTROUTING` dla przekaźnika zostanie zmieniony adres źródła.

Warto zaznaczyć że zmiane zródła dla przekaźnika trzeba to wykonać w `POSTROUTING` a nie `PREROUTING` gdyż inaczej jądro nie rozpozna że pakiet powinien być pszerzucony dalej, korzystając z wewnętrznej funkcji `ip_forward`.

Dla klienta zmiana źródła bedzie się odbywałą w `PREROUTING` gdyż zamierzamy otrzymać od razu pakiet na gniazdo a nie przesłać go dalej. W tym wypadku sciężka pakietów przejdzie przez `ip_local_delivery` kiedy otrzymuje pakiety oraz `ip_output` kiedy je emituje.

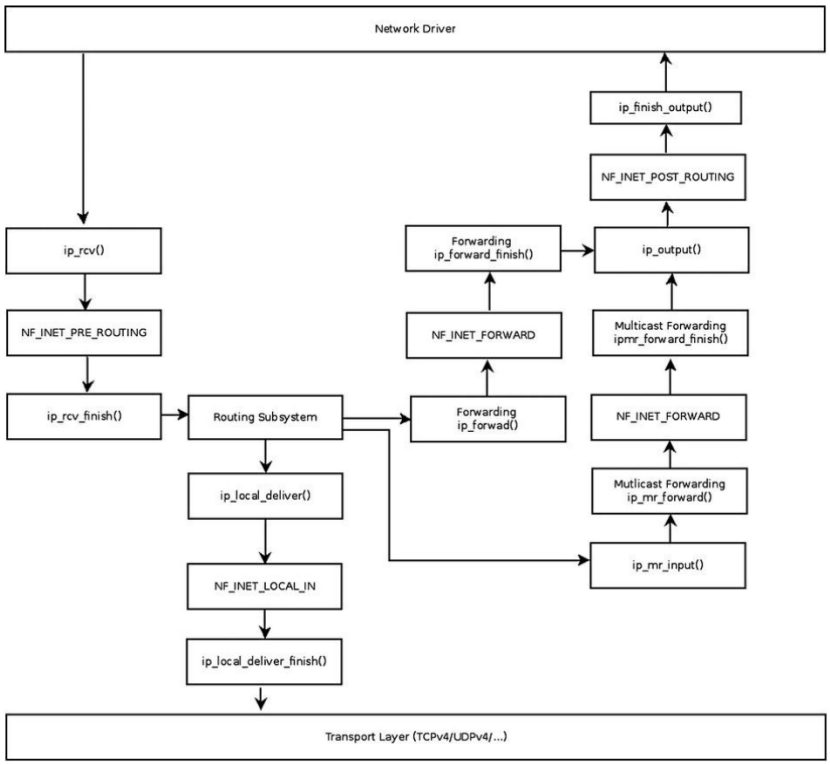
2.1 Szczegóły Implementacji

Kluczowym elementem projektu jest manipulacja strukturą `sk_buff` (socket buffer), która reprezentuje pakiet w jądrze. Poniżej przedstawiono najważniejsze aspekty techniczne rozwiązania.

2.1.1 Zarządzanie ciągłością pamięci i przesunięcia

Kluczowym wyzwaniem przy modyfikacji nagłówków jest fakt, że pakiety w jądrze Linux mogą być pofragmentowane (non-linear). Przed przystąpieniem do

[2]tablica jest widoczna korzystając z polecenia *ip route show*



Rysunek 2: Trasowanie pakietów przez jądro

modyfikacji, każda funkcja (np. `push_tcp_opt` czy `encrypt_skb_data`) wywołuje:

```
if (skb_is_nonlinear(skb))
    skb_linearize(skb);
```

Operacja ta gwarantuje, że cała zawartość pakietu znajduje się w ciągłym obszarze pamięci, co pozwala na bezpieczne użycie funkcji `memmove`. Proces wstrzykiwania opcji TCP o rozmiarze 8 bajtów (opcja + padding) przebiega według następującego schematu:

1. **Rozszerzenie nagłówka:** Wywołanie `skb_push(skb, 8)` przesuwa wskaźnik `data` bufora o 8 bajtów "w górę"(zmniejszając dostępny headroom).
2. **Relokacja nagłówków:** Za pomocą `memmove`, nagłówki IP i TCP są kopiowane o 8 bajtów wcześniej, zwalniając miejsce dokładnie tam, gdzie kończy się standardowy nagłówek TCP.
3. **Aktualizacja meta-danych:** Po fizycznym przesunięciu danych, konieczne jest wywołanie `skb_reset_network_header` oraz `skb_set_transport_header`, aby wskaźniki wewnętrzne struktury `skb` odnosiły się do nowych lokalizacji w pamięci.

2.2 Obliczanie sum kontrolnych i obsługa Offloadingu

Modyfikacja dowolnego pola w nagłówku IP (np. `daddr`) lub TCP (np. `options`, `payload`) powoduje natychmiastowe unieważnienie sum kontrolnych obliczonych przez kartę sieciową nadawcy.

W zaimplementowanej funkcji `check_ipv4` zastosowano mechanizm dwuetapowy:

- **Warstwa IP:** Wykorzystano funkcję `ip_fast_csum`, która oblicza sumę kontrolną nagłówka IPv4. Zgodnie ze specyfikacją, przed obliczeniem pole `iph->check` musi zostać wyzerowane.
- **Warstwa Transportowa (L4):** Dla protokołów TCP i UDP obliczenie sumy jest bardziej złożone, ponieważ wymaga uwzględnienia tzw. pseudonagłówka (zawierającego adresy IP źródła i celu oraz długość protokołu). Wykorzystano do tego funkcję:

```
*checkf = csum_tcpudp_magic(iph->saddr, iph->daddr, hlen,
                            iph->protocol, csum_partial(hptr, hlen, 0));
```

Warto zauważyć, że w kodzie przewidziano blok `#ifdef HW_OFFLOAD`. W nowoczesnych systemach obliczanie sum kontrolnych może zostać oddelegowane do sprzętu (NIC) poprzez ustawienie flagi `skb->ip_summed = CHECKSUM_PARTIAL`. W obecnej implementacji tunelu, ze względu na konieczność pełnej kontroli nad pakietem, domyślnie sumy przeliczane są programowo (`CHECKSUM_NONE`).

2.2.1 Manipulacja Opcjami TCP

Ponieważ standardowy nagłówek TCP ma stałą strukturę, dodanie własnej opcji wymaga przesunięcia danych w dół (jeśli w pakiecie znajduje się ładunek) oraz aktualizacji pola `doff` (data offset). Wykorzystano autorską funkcję `push_tcp_opt`, która rozszerza nagłówek o 4-bajtową wartość (oryginalny adres IP).

```
// Fragment dodawania opcji w module przekaźnika
if (iph->protocol == IPPROTO_TCP) {
    tcph = tcp_hdr(skb);
    __u32 addr_to_hide = iph->saddr;
    // Szyfrowanie adresu
    encrypt((unsigned char *)&addr_to_hide, 4, (char)2137);
    push_tcp_opt(skb, addr_to_hide);
    // Szyfrowanie payloadu
    encrypt_skb_data(skb);

    // Po przesunięciu wskaźników musimy ponownie pobrać nagłówek IP
    iph = ip_hdr(skb);
    iph->daddr = CLIENT_HOST;
}
```

2.2.2 Parsowanie opcji TCP

Podczas odbierania pakietu przez klienta lub przekaźnik, system musi odnaleźć ukryty adres IP w gąszczu innych opcji TCP (np. MSS, SackOK, Timestamps). Funkcja `read_tcp_opt` implementuje maszynę stanów operującą na formacie TLV (Type-Length-Value):

- **Kind 0:** End of List – przerwanie pętli.
- **Kind 1:** No-Operation – pominięcie bajtu.
- **Kind 255:** Nasza zdefiniowana opcja – odczyt 4 bajtów danych i ich deszyfrowanie.

2.2.3 Szyfrowanie i Bezpieczeństwo

Zgodnie z założeniami projektu, zaimplementowano prosty mechanizm szyfrowania XOR dla przesyłanych adresów oraz danych pakietu. Klucz szyfrujący został ustalony na wartość `2137`. Szyfrowanie odbywa się bezpośrednio na buforze pakietu przed obliczeniem sum kontrolnych.

```
void encrypt(unsigned char *data, int len, char key) {
    for (int i = 0; i < len; i++) {
        data[i] ^= key;
    }
}
```

Funkcja `encrypt_skb_data` iteruje po bajtach wewnątrz `skb->data`, co pozwala na ukrycie zawartości przesyłanego tunelu przed prostymi narzędziami do sniffingu (np. Wireshark bez odpowiedniego parsera).

2.3 Problemy Napotkane i Wnioski

Podczas prac nad projektem najwięcej trudności sprawiło poprawne zarządzanie rozmiarem pakietu w jądrze oraz trasowanie pakietów w ramach struktury `netfilter`.

Użycie funkcji takich jak `skb_push` i `skb_put` jest niezbędne, aby stos sieciowy jądra nie uznało zmodyfikowanego pakietu za uszkodzony (tzw. *malformed packet*). Próba operowania na wskaźnikach bez aktualizacji metadanych bufora `sk_buff` prowadziła do natychmiastowego odrzucenia pakietu przez kolejne warstwy stosu.

Prawidłowa zamiana adresów pakietów, wymuszająca trasowanie przez odpowiednie ścieżki jądra, wymagała precyzyjnego rozpoznania różnic między poszczególnymi hakami oraz analizy kodu źródłowego jądra. Autor musiał zrozumieć, które funkcje wewnętrzne odpowiadają za przerzucanie pakietów i pod jakimi warunkami to robią. Szczególną trudność sprawiła poprawna implementacja mechanizmu odpowiadającego za `ip_forward`, wokół którego w internecie krąży wiele sprzecznych i nieścisłych informacji[3]. Ostatecznie kluczem okazało się uzycie `POSTROUTING` zamiast `PREROUTING` do zamiany źródłowego adresu dzieki czemu potrzeba trasowania przez `ip_forward` zostałą wykryta przez jądro

$$\text{local host} \neq \text{source adress} \rightarrow \text{ip\_forward}$$

- **Transparentność:** Rozwiązanie działa bez modyfikacji aplikacji użytkownika (gniazda są nieświadome tunelowania).
- **Wydajność:** Dzięki implementacji bezpośrednio w `netfilter`, narzut czasowy na przetwarzanie pakietu jest minimalny w porównaniu do rozwiązań typu *userspace proxy*.
- **Szyfrowanie:** Mimo prostoty algorytmu XOR, projekt udowadnia możliwość implementacji mechanizmów *Deep Packet Inspection* (DPI) i modyfikacji danych w locie.

3 Podsumowanie

Projekt zakończył się sukcesem, tworząc działający tunel TCP wykorzystujący opcje nagłówka do przesyłania metadanych trasowania. System poprawnie radzi sobie z translacją adresów w obu kierunkach, de facto emulując działanie dynamicznego NAT-u przy jednoczesnym ukryciu rzeczywistych adresów docelowych poprzez szyfrowanie.

[3] `https://stackoverflow.com/questions/22020573/how-to-move-packet-from-nf-inet-pre-routing-to-nf-inet-post-rou`