**Mateusz Rus, Jakub Niezabitowski**

**Laboratorium 7**

**Zadanie 2**

Aby umożliwić atak DOS musimy dokonać następujących zmian:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Dane wysyłane są bez znaku końca linii. Dlatego w kodzie usuwamy dwa ostatnie bajty

Działanie serwera i klientów po umożliwieniu ataku:

Obraz zawierający tekst

Opis wygenerowany automatycznie

*Serwer*

Po ataku DOS możemy zauważyć że serwer zawiesił się

Obraz zawierający tekst

Opis wygenerowany automatycznie

Próbując połączenia nowym klientem nawiązanie połączenia nie jest możliwe.

Aby uodpornić nasz serwer na atak musimy ustawić funkcję SO\_RECVTIMEO na czas np. 4 sekund aby umożliwić w tym czasie zamknięcie gniazda i umożliwić komunikację z nowymi klientami:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Musimy również zadbać o odpowiednie zamknięcie deskryptora klienta:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Sprawdzenie działania serwera:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Sprawdzenie działania klienta:

Obraz zawierający tekst

Opis wygenerowany automatycznie

**Zadanie 3**

Wywołanie programu echo\_serv6\_ws\_pool.c



Otrzymaliśmy 1002 aktywne połączenia. 1000 z nich pochodzi z programu klienta a pozostałe 2 to uruchomione sesje ssh.

Uruchomienie programu klienta:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Otwarte gniazda:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Średnie czasy obsługi zgłoszeń w serwerze z mechanizmem poll:

Należy modyfikować parametr pushrate programu klienta na inne wartości oraz stałą MAXEVENTS w programie serwera. Następnie odczytujemy z kilku prób średni czas potrzebny na obsługę zdarzenia odczytując go z programu serwera

Dla MAXEVENTS = 1000000

|  |  |
| --- | --- |
| Pushrate | Średni czas obsługi [us] |
| 1000000 | 1172 |
| 100000 | 1205 |

Dla MAXEVENTS = 100000

|  |  |
| --- | --- |
| Pushrate | Średni czas obsługi [us] |
| 1000000 | 1118 |
| 100000 | 1184 |

Analizę przeprowadzamy również dla serwera z mechanizmem epoll:

Dla MAXEVENTS = 1000000

|  |  |
| --- | --- |
| Pushrate | Średni czas obsługi [us] |
| 1000000 | 1004 |
| 100000 | 1020 |

Dla MAXEVENTS = 100000

|  |  |
| --- | --- |
| Pushrate | Średni czas obsługi [us] |
| 1000000 | 1101 |
| 100000 | 1043 |

Możemy zaobserwować, że czasy serwerów POLL i EPOLL nieznacznie się od siebie różnią. Jednak

czasy serwera EPOLL są krótsze.

**Zadanie 4**

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Dla domyślnej wartości ulimit można otworzyć maksymalnie:

-1020 połączeń (serwer epoll i fork)

-1019 połączeń (serwer poll)

Sprawdzamy maksymalną liczbę otwartych plików w systemie:

Obraz zawierający tekst

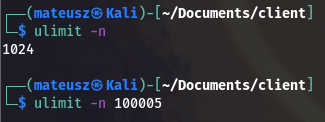
Opis wygenerowany automatycznie

Sprawdzamy maksymalny rozmiaru tablicy deskryptorów:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Zwiększamy liczbę plików otwartych w pojedynczym procesie:



Sprawdzenie liczby portów dostępnej dla klienta:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Po wykonanych zmianach program działa poprawnie:

Obraz zawierający tekst

Opis wygenerowany automatycznie

**Zadanie 5**

Działanie serwera:

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

Działanie klienta:

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

Bufor jest zbyt mały więc długie wiadomości przesyłane są w segmentach.

Musimy dokonać zmian w kodzie:

Zgodnie z instrukcją dodajemy flagę EPOLLET



Zastosowanie flagi SOCK\_NONBLOCK w funkcji accept4()

Obraz zawierający tekst

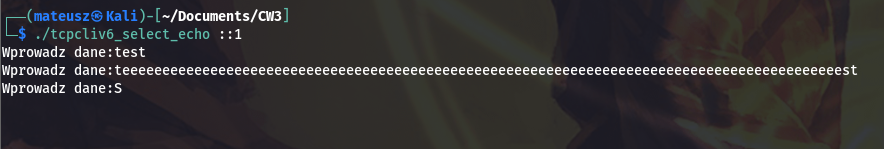
Opis wygenerowany automatycznie

Wykorzystanie pętli while():

Obraz zawierający tekst

Opis wygenerowany automatycznie

Działanie programu po wykonanych zmianach:



Komunikacja pomiędzy serwerem jest nawiązana, lecz nie dochodzi do wymiany danych, klient nie otrzymuje od serwera wiadomości zwrotnej.

Tcpdump:

