UNIWERSYTET KARDYNAŁA STEFANA WYSZYŃSKIEGO

W WARSZAWIE

WYDZIAŁ MATEMATYCZNO-PRZYRODNICZY

SZKOŁA NAUK ŚCISŁYCH

Kacper Sądzyński

Krzysztof Gorzkowski

Kamil Chibowski

Jan Brzeziński

Tobiasz Jeska

Skaner IPv4 dla wrażliwych protokołów (IPv4 scanner for vulnerable protocols)

WARSZAWA 2021

Spis treści

Wstęp 3

Wymagania systemowe programu 3
Minimalne: 3
Zalecane: 3

Docelowa funkcjonalność 3

Diagram Gantta 4

Kroki pośrednie 6

Diagram klas 6

Diagram sekwencji 7

Biblioteki dodatkowe 8
SNMP4J 8
Informacje ogólne 8
Wykorzystane zasoby 8

Wstęp

Program służy do poszukiwania adresów IPv4, które mogą posłużyć do ataków DDoS. Do napisania programu został wykorzystany język programowania Java. IDE wykorzystane do zbudowania tego projektu to Intellij IDEA ultimate 2021.1.

Wymagania systemowe programu

Minimalne:

- Wymaga 64-bitowego procesora
- Procesor:Intel(R) Core(TM) i7-8550U CPU @ 1.80 GHz, Rdzenie: 4, Procesory logiczne: 6
- Pamięć: 8 GB RAM
- Miejsce na dysku: 1 GB dostępnej przestrzeni

Zalecane:

- Wymaga 64-bitowego procesora
- Procesor:Intel(R) Core(TM) i5-9600KF CPU @ 3.70GHz, 3701 MHz, Rdzenie: 6, Procesory logiczne: 8
- Pamięć: 16 GB RAM
- Miejsce na dysku: 2 GB dostępnej przestrzeni

Docelowa funkcjonalność

Program docelowo ma służyć do poszukiwania adresów IPv4, które mogą posłużyć do ataków DDoS. Program będzie skanował sieć używając następujących protokołów:

- DNS
- SNMP
- NTP
- MemCached

Program będzie wywoływany z poziomu konsoli. W argumentach wywołania programu użytkownik może podać, jakich protokołów chce użyć oraz czy wynik chce zapisywać do pliku.
Za pomocą argumentu wywołania może również wyświetlić pomoc - listę dostępnych argumentów.

Diagram Gantta

	Krzysztof Gorzkowski	Tobiasz Jeska	Jan Brzeziński	Kacper Sądzyński	Kamil Chibowski	Czas łączny
tydzień 1	3 h					3 h
tydzień 2	9 h					9 h
tydzień 3	12 h					14 h
	2 h					
tydzień 4	4 h					43 h
	15 h		8 h	8 h	8 h	
tydzień 5	2,5 h					18 h
	4 h		2 h	3 h	3 h	
	2 h					
	1,5 h					
tydzień 6	0,75 h	1 h	2 h	1 h	2 h	17,5 h
	3,5 h					
	1,5 h	1 h	1,5 h	1 h	1 h	
			0,5 h	0,25 h	0,5 h	
tydzień 7	0,5 h					35,5 h
	2 h >>			<< 2 h		
	2,5 h					
	1,5 h					
	2,5 h	1 h	3 h	1,5 h	1 h	
		1 h				
	10 h				2 h	
	4 h					
	2 h				1 h	
tydzień 8	5,5 h		2,5 h			18,5 h
	2 h					
			3 h			
	1 h					
	1 h		1 h			
	2,5 h					
	1,5 h					

tydzień 9			1 h		1 h	12,5 h
		1,5 h				
	2 h >>		<< 2 h			
	2,5 h >>				<< 2,5 h	
	1 h			0,5 h		
	1 h					
	1 h				1 h	

LEGENDA
Czytanie o programowaniu sieciowym
Poznawanie platformy GitHub
Uczenie się Javy
Kodowanie
Dokumentacja
Testy

Kroki pośrednie

1. Utworzenie klasy Main oraz ArgAvailable, a także pliku args.txt, aby zdefiniować możliwe argumenty wywołania programu
2. Utworzenie generatora wszystkich adresów IPv4
3. Dodanie metody weryfikującej argumenty wywołania programu, przeniesienie generatora adresów do nowej klasy IPv4Addresses
4. Refaktoryzacja klasy DNSScanner - od teraz dziedziczy po klasie IPv4Addresses
5. Dodanie metody query w klasie DNSScanner umożliwiającej budowanie pakietu i wysyłanie zapytania
6. Dodano funkcjonalność pisania do pliku
7. Testowanie skanera DNS w celu znalezienia pakietów dających największą amplifikację - dodano funkcjonalność do testowania
8. Dodanie funkcjonalności do klasy SNMPScanner analogicznej do DNSScanner (konstrukcja pakietu, wysyłanie, słuchanie odpowiedzi)
9. Testowanie skanera SNMP
10. Refaktoryzacja DNSScanner - wielowątkowość+
11. Testowanie wielowątkowości
12. Refaktoryzacja DNSScanner - docelowy kształt

Diagram klas

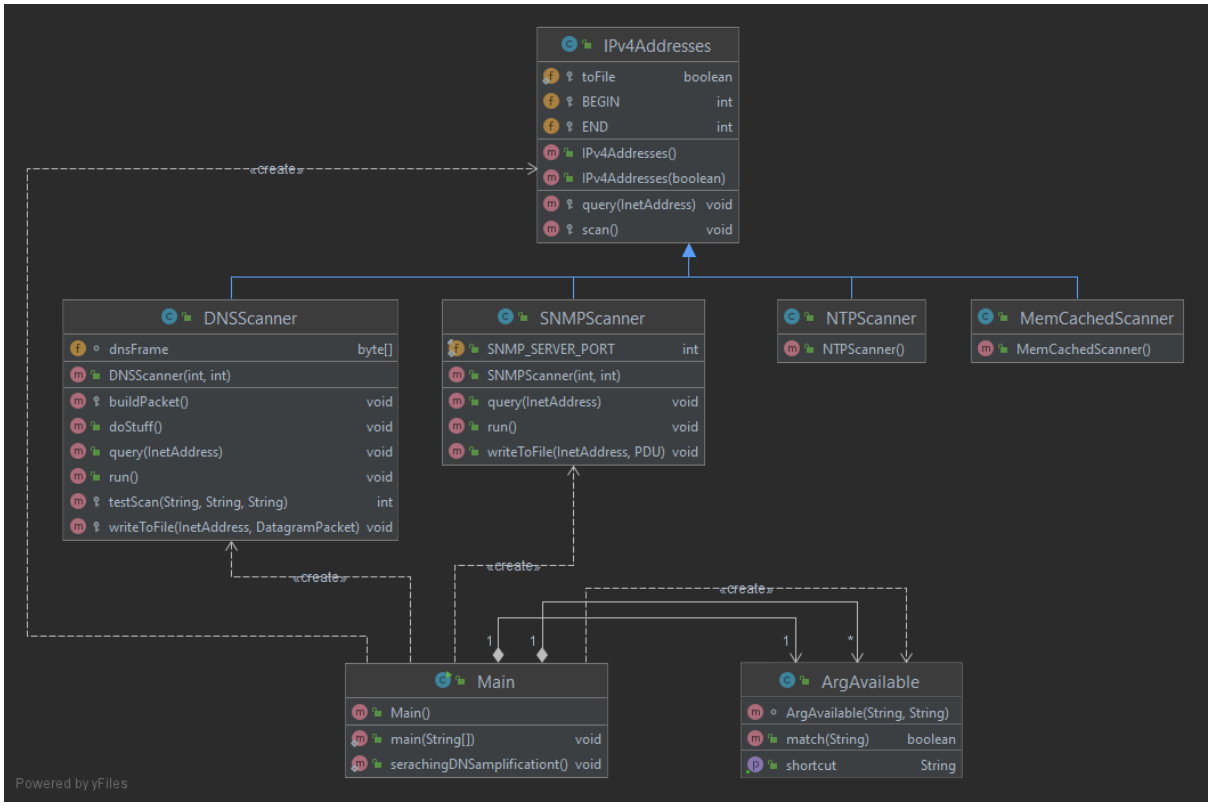
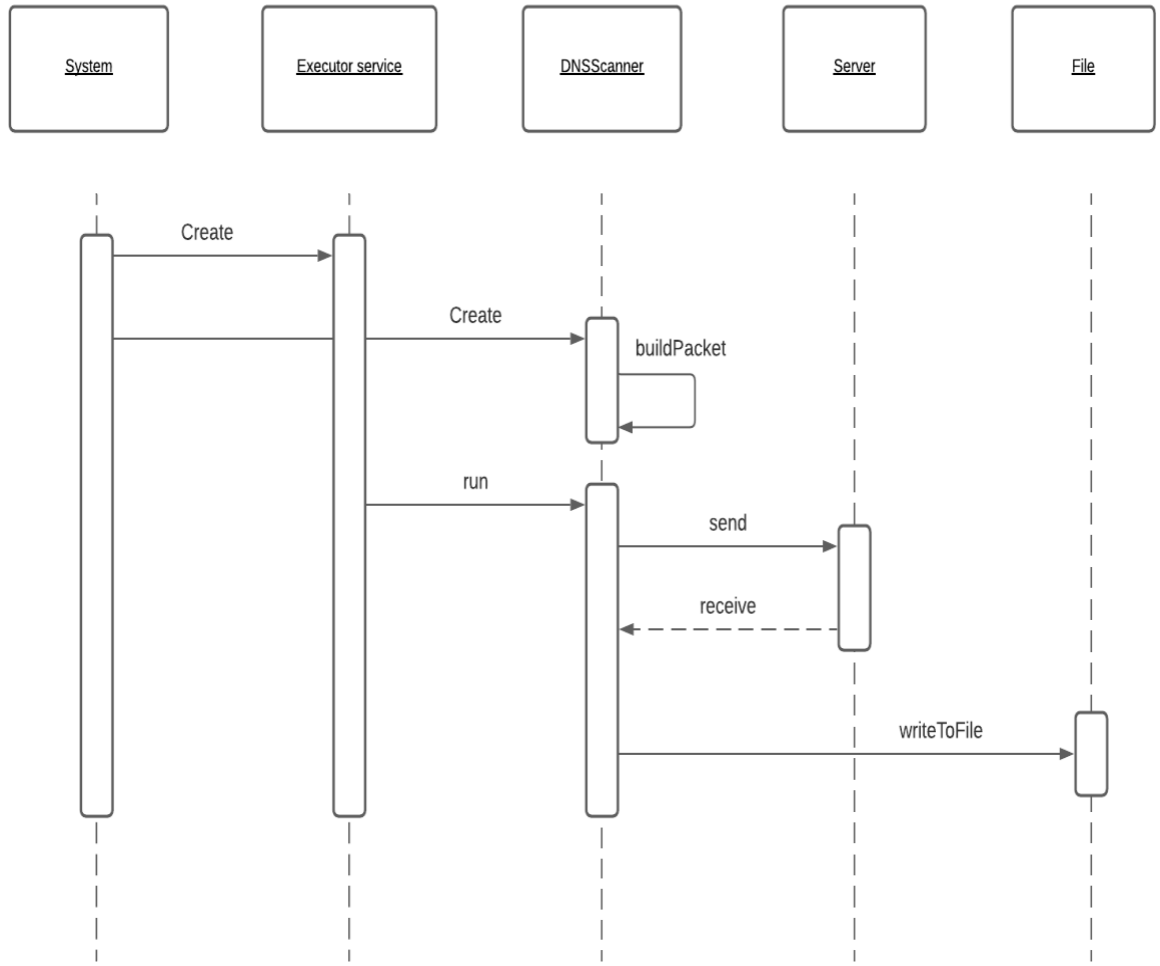


Diagram sekwencji



Biblioteki dodatkowe

❖ SNMP4J

➢ Informacje ogólne

Wersja: org.snmp4j:snmp4j:3.4.4
Biblioteka służy do wygodnej obsługi pakietu SNMP w języku Java.

➢ Wykorzystane zasoby

- Address - interfejs, który przechowuje adres IP i port serwera, do którego będziemy wysyłać zapytanie
- GenericAddress - klasa służąca do implementacji obiektów innych typów
  - parse(String) - zwraca obiekt typu Address z przypisanym adresem IP i numerem portu oraz typem protokołu (np. udp) podanymi w argumencie wywołania - konstrukcja argumentu (typu String): protokół: x.x.x.x/port
- CommunityTarget - klasa, która reprezentuje właściwości pakietu i cel jego wysłania
  - setCommunity(OctetString) - ustawia parametr community, zgodny z podanym w argumencie wywołania (np. “public”)
  - setAddress(Address) - ustawia adres docelowego serwera jako podany w argumencie wywołania
  - setVersion(int) - ustawia wersję pakietu (np. 2c)
  - setTimeout(long timeout) - ustawia domyślny czas oczekiwania na odpowiedź od serwera na podany w argumencie wywołania (w milisekundach); jeżeli odpowiedź nie nadejdzie w tym czasie, będzie zapisana jako null
  - setRetries(int) - ustawia liczbę dodatkowych prób wysłania pakietu w razie nieotrzymania odpowiedzi od serwera
- OctetString - klasa, która reprezentuje tekst w formie zrozumiałej dla innych klas biblioteki
  - konstruktor (String) - konwertuje obiekt typu String podany w argumencie wywołania na OctetString
- SnmpConstants - klasa, która zawiera wszystkie stałe potrzebne do działania paczki (użyto stałej version2c)
- Snmp - klasa służąca do tworzenia socketa, wysyłania i odbierania pakietów SNMP
  - konstruktor (DefaultUdpTransportMapping) - przydziela mapowanie transportu podane w argumencie wywołania
  - listen() - ustawia mapowanie transportu na tryb słuchania
  - send(PDU, CommunityTarget) - wysyła zapytanie zawarte w pierwszym argumencie wywołania zgodnie z parametrami określonymi przez drugi argument
  - close() - zamyka sesję
- PDU - klasa, która reprezentuje jednostkę danych protokołu

        - add(VariableBinding) - dodaje zmienną OID do tego PDU
        - setType(int) - ustawia typ PDU na zgodny z podanym w argumencie wywołania
        - getBERLength() - zwraca ilość bajtów tworzących PDU (rozmiar zapytania/odpowiedzi) zakodowanego zgodnie z Basic Encoding Rules
- VariableBinding - klasa łącząca OID ze zmienną stanowiącą parametr potrzebny do działania paczki
    - konstruktor (OID) - łączy OID podane w argumencie wywołania ze zmienną o wartości Null
- DefaultUdpTransportMapping - klasa implementuje mapowanie transportu UDP oparte na standardzie IO Java i używa wewnętrznego wątku do nasłuchiwania w gnieździe wejściowym
- ResponseEvent - klasa odbierająca odpowiedź i łącząca PDU zapytania z odpowiednią odpowiedzią
    - getResponse() - zwraca PDU odpowiedzi