#!/usr/bin/env python3

"""

ARP IDS – Lightweight ARP Spoofing/Poisoning Detector

- Live sniffing on an interface OR offline analysis of a .pcap

- Detects IP->MAC changes, gateway MAC mismatch, and ARP reply storms

- Logs alerts to CSV and prints to console

Usage:

Live: sudo python arp\_ids.py --iface eth0 --gateway 192.168.1.1 --gateway-mac aa:bb:cc:dd:ee:ff

PCAP: python arp\_ids.py --pcap test\_arp\_spoof.pcap

"""

import argparse

import csv

import os

import time

from collections import defaultdict, deque

try:

from scapy.all import sniff, ARP, conf, rdpcap

except Exception as e:

raise SystemExit(

f"[!] Failed to import Scapy. Install it first:\n"

f" pip install scapy\n\nError: {e}"

)

# -------------------- Defaults --------------------

DEFAULT\_LOG = "arp\_ids\_events.csv"

DEFAULT\_STORM\_WINDOW = 5 # seconds

DEFAULT\_STORM\_THRESHOLD = 30 # >30 ARP replies in window -> alert

# -------------------- Helpers --------------------

def now\_str():

return time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())

def write\_csv(path, record):

file\_exists = os.path.exists(path)

header = list(record.keys())

with open(path, "a", newline="") as f:

w = csv.DictWriter(f, fieldnames=header)

if not file\_exists:

w.writeheader()

w.writerow(record)

def console\_alert(reason, pkt, extra=None):

ts = now\_str()

src\_ip = pkt[ARP].psrc

src\_mac = pkt[ARP].hwsrc

dst\_ip = pkt[ARP].pdst

dst\_mac = pkt[ARP].hwdst

print(f"[ALERT] {ts} | {reason} | {src\_ip} ({src\_mac}) -> {dst\_ip} ({dst\_mac}) | {extra or ''}")

def log\_event(log\_path, reason, pkt, extra=None):

ts = now\_str()

rec = {

"time": ts,

"reason": reason,

"src\_ip": pkt[ARP].psrc,

"src\_mac": pkt[ARP].hwsrc,

"dst\_ip": pkt[ARP].pdst,

"dst\_mac": pkt[ARP].hwdst,

}

if extra:

# flatten extras for readability

for k, v in extra.items():

rec[str(k)] = v

console\_alert(reason, pkt, extra)

write\_csv(log\_path, rec)

# -------------------- Detector --------------------

class ArpIDS:

def \_\_init\_\_(self, log\_path=DEFAULT\_LOG, gateway\_ip=None, gateway\_mac=None,

storm\_window=DEFAULT\_STORM\_WINDOW, storm\_threshold=DEFAULT\_STORM\_THRESHOLD):

self.log\_path = log\_path

self.gateway\_ip = gateway\_ip

self.gateway\_mac = gateway\_mac.lower() if gateway\_mac else None

self.storm\_window = storm\_window

self.storm\_threshold = storm\_threshold

self.ip\_to\_mac = {} # observed IP -> MAC

self.reply\_buckets = defaultdict(lambda: deque()) # talker key -> timestamps

def \_check\_binding(self, pkt):

src\_ip = pkt[ARP].psrc

src\_mac = pkt[ARP].hwsrc

old = self.ip\_to\_mac.get(src\_ip)

if old and old.lower() != src\_mac.lower():

self.\_alert("IP-to-MAC change (possible spoofing)", pkt,

{"old\_mac": old, "new\_mac": src\_mac})

self.ip\_to\_mac[src\_ip] = src\_mac

def \_check\_gateway(self, pkt):

if not self.gateway\_ip:

return

if pkt[ARP].psrc != self.gateway\_ip:

return

observed = pkt[ARP].hwsrc.lower()

if self.gateway\_mac and observed != self.gateway\_mac:

self.\_alert("Gateway MAC mismatch", pkt,

{"expected": self.gateway\_mac, "observed": observed})

def \_check\_storm(self, pkt):

key = (pkt[ARP].hwsrc or "") + "|" + (pkt[ARP].psrc or "")

q = self.reply\_buckets[key]

t = time.time()

q.append(t)

# drop timestamps outside window

while q and t - q[0] > self.storm\_window:

q.popleft()

if len(q) > self.storm\_threshold:

self.\_alert("ARP reply storm", pkt, {"count\_in\_window": len(q)})

def \_alert(self, reason, pkt, extra=None):

log\_event(self.log\_path, reason, pkt, extra)

def handle(self, pkt):

if ARP not in pkt:

return

if pkt[ARP].op != 2: # focus on ARP replies (op=2)

return

self.\_check\_binding(pkt)

self.\_check\_gateway(pkt)

self.\_check\_storm(pkt)

# -------------------- Runners --------------------

def run\_live(args):

iface = args.iface or conf.iface

print(f"[+] Starting ARP IDS on interface: {iface}")

if args.gateway:

print(f"[+] Gateway pin enabled for {args.gateway} "

f"(expected MAC: {args.gateway\_mac or 'discovering'})")

ids = ArpIDS(

log\_path=args.log,

gateway\_ip=args.gateway,

gateway\_mac=args.gateway\_mac,

storm\_window=args.storm\_window,

storm\_threshold=args.storm\_threshold

)

sniff(prn=ids.handle, filter="arp", store=False, iface=iface)

def run\_pcap(args):

print(f"[+] Reading PCAP: {args.pcap}")

packets = rdpcap(args.pcap)

ids = ArpIDS(

log\_path=args.log,

gateway\_ip=args.gateway,

gateway\_mac=args.gateway\_mac,

storm\_window=args.storm\_window,

storm\_threshold=args.storm\_threshold

)

count = 0

for p in packets:

try:

ids.handle(p)

count += 1

except Exception:

# skip malformed frames gracefully

pass

print(f"[+] Processed {count} packets from PCAP.")

# -------------------- CLI --------------------

def parse\_args():

ap = argparse.ArgumentParser(description="Lightweight ARP Spoofing Detector (Python + Scapy)")

src = ap.add\_mutually\_exclusive\_group(required=False)

src.add\_argument("--iface", help="Network interface for live sniffing (default: scapy's conf.iface)")

src.add\_argument("--pcap", help="Offline analysis from a .pcap file")

ap.add\_argument("--gateway", help="Gateway IP to pin (e.g., 192.168.1.1)")

ap.add\_argument("--gateway-mac", help="Expected gateway MAC to pin (aa:bb:cc:dd:ee:ff)")

ap.add\_argument("--log", default=DEFAULT\_LOG, help=f"CSV log path (default: {DEFAULT\_LOG})")

ap.add\_argument("--storm-window", type=int, default=DEFAULT\_STORM\_WINDOW,

help=f"Seconds for ARP storm window (default: {DEFAULT\_STORM\_WINDOW})")

ap.add\_argument("--storm-threshold", type=int, default=DEFAULT\_STORM\_THRESHOLD,

help=f"Max replies in window before alert (default: {DEFAULT\_STORM\_THRESHOLD})")

return ap.parse\_args()

def main():

args = parse\_args()

if args.pcap:

run\_pcap(args)

else:

run\_live(args)

if \_\_name\_\_ == "\_\_main\_\_":

main()