Advanced Firewall Rules for Specific Threats

Table of Contents

Threat-Specific Rule Sets

Protection Against Known Attack Patterns

```bash
```

```
#!/bin/bash
# Advanced iptables rules for specific threats

# Create custom chains for different threat types
create_threat_chains() {
    # Chain for port scan detection
    iptables -N PORTSCAN_PROTECTION 2>/dev/null
    iptables -F PORTSCAN_PROTECTION

    # Chain for brute force protection
    iptables -N BRUTEFORCE_PROTECTION 2>/dev/null
    iptables -F BRUTEFORCE_PROTECTION

    # Chain for malware C&C blocking
    iptables -N MALWARE_BLOCK 2>/dev/null
    iptables -F MALWARE_BLOCK

    # Chain for exploit attempts
    iptables -N EXPLOIT_PROTECTION 2>/dev/null
    iptables -F EXPLOIT_PROTECTION
}

# Advanced Port Scan Detection
configure_portscan_detection() {
    # Detect NULL scan
    iptables -A PORTSCAN_PROTECTION -p tcp --tcp-flags ALL NONE -j LOG --log-prefix "NULL SCAN: "
    iptables -A PORTSCAN_PROTECTION -p tcp --tcp-flags ALL NONE -j DROP

    # Detect XMAS scan
    iptables -A PORTSCAN_PROTECTION -p tcp --tcp-flags ALL FIN,PSH,URG -j LOG --log-prefix "XMAS SCAN: "
    iptables -A PORTSCAN_PROTECTION -p tcp --tcp-flags ALL FIN,PSH,URG -j DROP

    # Detect FIN scan
    iptables -A PORTSCAN_PROTECTION -p tcp --tcp-flags ALL FIN -j LOG --log-prefix "FIN SCAN: "
    iptables -A PORTSCAN_PROTECTION -p tcp --tcp-flags ALL FIN -j DROP

    # Detect SYN-FIN scan
    iptables -A PORTSCAN_PROTECTION -p tcp --tcp-flags SYN,FIN SYN,FIN -j LOG --log-prefix "SYN-FIN SCAN: "
    iptables -A PORTSCAN_PROTECTION -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP

    # Detect SYN-RST scan
    iptables -A PORTSCAN_PROTECTION -p tcp --tcp-flags SYN,RST SYN,RST -j LOG --log-prefix "SYN-RST SCAN: "
    iptables -A PORTSCAN_PROTECTION -p tcp --tcp-flags SYN,RST SYN,RST -j DROP

    # Use recent module for sophisticated detection
    iptables -A PORTSCAN_PROTECTION -m recent --name portscan --set
    iptables -A PORTSCAN_PROTECTION -m recent --name portscan --update --seconds 60 --hitcount 20 -j LOG --log-p
    iptables -A PORTSCAN_PROTECTION -m recent --name portscan --update --seconds 60 --hitcount 20 -j DROP
}

# Brute Force Protection
configure_bruteforce_protection() {
    # SSH brute force protection
    iptables -A BRUTEFORCE_PROTECTION -p tcp --dport 22 -m state --state NEW -m recent --set --name SSH --rsource
    iptables -A BRUTEFORCE_PROTECTION -p tcp --dport 22 -m state --state NEW -m recent --update --seconds 60 --hi
    iptables -A BRUTEFORCE_PROTECTION -p tcp --dport 22 -m state --state NEW -m recent --update --seconds 60 --hi

    # HTTP/HTTPS brute force protection
    iptables -A BRUTEFORCE_PROTECTION -p tcp --dport 80 -m state --state NEW -m recent --set --name HTTP --rsourc
    iptables -A BRUTEFORCE_PROTECTION -p tcp --dport 80 -m state --state NEW -m recent --update --seconds 10 --hi
    iptables -A BRUTEFORCE_PROTECTION -p tcp --dport 80 -m state --state NEW -m recent --update --seconds 10 --hi

    # FTP brute force protection
    iptables -A BRUTEFORCE_PROTECTION -p tcp --dport 21 -m state --state NEW -m recent --set --name FTP --rsource
    iptables -A BRUTEFORCE_PROTECTION -p tcp --dport 21 -m state --state NEW -m recent --update --seconds 60 --hi

    # RDP brute force protection
    iptables -A BRUTEFORCE_PROTECTION -p tcp --dport 3389 -m state --state NEW -m recent --set --name RDP --rsou
    iptables -A BRUTEFORCE_PROTECTION -p tcp --dport 3389 -m state --state NEW -m recent --update --seconds 60 -
}

# Malware Command & Control Blocking
configure_malware_blocking() {
```

```
    # Block known malware ports
    local MALWARE_PORTS=(
        "4444"  # Metasploit default
        "5555"  # Android ADB exploit
        "6666"  # Common backdoor
        "6667"  # IRC botnet
        "7777"  # Common trojan
        "8080"  # Common proxy/backdoor
        "9999"  # Common backdoor
        "12345" # NetBus
        "31337" # Back Orifice
        "65535" # RC/Backdoor
    )

    for port in "${MALWARE_PORTS[@]}"; do
        iptables -A MALWARE_BLOCK -p tcp --dport $port -j LOG --log-prefix "MALWARE PORT $port: "
        iptables -A MALWARE_BLOCK -p tcp --dport $port -j DROP
        iptables -A MALWARE_BLOCK -p udp --dport $port -j DROP
    done

    # Block suspicious outbound connections
    iptables -A MALWARE_BLOCK -p tcp --dport 25 -j LOG --log-prefix "SMTP BLOCK: "  # Prevent spam
    iptables -A MALWARE_BLOCK -p tcp --dport 25 -j DROP

    # Block TOR nodes (optional - uncomment if desired)
    # while read ip; do
    #     iptables -A MALWARE_BLOCK -s $ip -j DROP
    #     iptables -A MALWARE_BLOCK -d $ip -j DROP
    # done < /etc/tor-exit-nodes.txt
}

# Exploit Protection
configure_exploit_protection() {
    # Protect against fragment attacks
    iptables -A EXPLOIT_PROTECTION -f -j LOG --log-prefix "FRAGMENT ATTACK: "
    iptables -A EXPLOIT_PROTECTION -f -j DROP

    # Block invalid packets
    iptables -A EXPLOIT_PROTECTION -m state --state INVALID -j LOG --log-prefix "INVALID PACKET: "
    iptables -A EXPLOIT_PROTECTION -m state --state INVALID -j DROP

    # Protect against TCP flag anomalies
    iptables -A EXPLOIT_PROTECTION -p tcp --tcp-flags ALL ALL -j LOG --log-prefix "ALL FLAGS SET: "
    iptables -A EXPLOIT_PROTECTION -p tcp --tcp-flags ALL ALL -j DROP

    # Protect against new packets that aren't SYN
    iptables -A EXPLOIT_PROTECTION -p tcp ! --syn -m state --state NEW -j LOG --log-prefix "NEW NOT SYN: "
    iptables -A EXPLOIT_PROTECTION -p tcp ! --syn -m state --state NEW -j DROP

    # Protect against MSS manipulation
    iptables -A EXPLOIT_PROTECTION -p tcp -m tcpmss ! --mss 536:65535 -j LOG --log-prefix "INVALID MSS: "
    iptables -A EXPLOIT_PROTECTION -p tcp -m tcpmss ! --mss 536:65535 -j DROP
}

# Apply all chains to INPUT
apply_protection_chains() {
    iptables -A INPUT -j PORTSCAN_PROTECTION
    iptables -A INPUT -j BRUTEFORCE_PROTECTION
    iptables -A INPUT -j MALWARE_BLOCK
    iptables -A INPUT -j EXPLOIT_PROTECTION
}

# Main execution
create_threat_chains
configure_portscan_detection
configure_bruteforce_protection
configure_malware_blocking
configure_exploit_protection
apply_protection_chains
```

Application Layer Protection

Layer 7 Filtering with nftables

```bash
```

```
#!/usr/sbin/nft -f
# Advanced nftables configuration for application layer protection

flush ruleset

table inet filter {
  # Define sets for dynamic blocking
  set blacklist_v4 {
    type ipv4_addr
    flags timeout
    timeout 1h
  }

  set blacklist_v6 {
    type ipv6_addr
    flags timeout
    timeout 1h
  }

  # HTTP/HTTPS attack patterns
  set http_attack_patterns {
    type string
    elements = {
      "../../",          # Path traversal
      "<script",         # XSS attempt
      "';DROP TABLE",    # SQL injection
      "eval(",           # Code injection
      "base64_decode",   # Encoded payload
      "../etc/passwd",   # LFI attempt
      "cmd.exe",         # Command execution
      "/bin/bash",       # Shell access
      "wget http",       # Remote file download
      "curl http"        # Remote file download
    }
  }

  # DNS filtering
  set blocked_domains {
    type string
    elements = {
      "malware.com",
      "phishing.net",
      "tracker.org"
    }
  }

  chain input {
    type filter hook input priority 0; policy drop;

    # Connection tracking
    ct state established,related accept
    ct state invalid drop

    # Check blacklists
    ip saddr @blacklist_v4 drop
    ip6 saddr @blacklist_v6 drop

    # Loopback
    iif lo accept

    # ICMP rate limiting
    ip protocol icmp limit rate 10/second accept
    ip6 nexthdr icmpv6 limit rate 10/second accept

    # HTTP/HTTPS content filtering
    tcp dport { 80, 443 } ct state new {
      # Limit connection rate
      limit rate over 30/second add @blacklist_v4 { ip saddr timeout 1h }
      limit rate 30/second accept
    }

    # SSH with port knocking
    tcp dport 22222 ct state new limit rate 3/minute accept comment "Hidden SSH"
```

```
    # Log and reject
    limit rate 5/minute log prefix "nftables drop: "
    reject with icmpx type port-unreachable
  }

  chain forward {
    type filter hook forward priority 0; policy drop;
  }

  chain output {
    type filter hook output priority 0; policy accept;

    # Block suspicious outbound
    tcp dport { 25, 465, 587 } reject comment "Block SMTP"
    tcp dport { 6660-6669, 7000 } reject comment "Block IRC"

    # DNS filtering
    udp dport 53 @th,96,160 @blocked_domains reject
  }
}

# NAT table for port knocking
table ip nat {
  chain prerouting {
    type nat hook prerouting priority -100;

    # Port knocking sequence: 7000, 8000, 9000 opens SSH
    tcp dport 7000 ct mark set 0x1
    tcp dport 8000 ct mark 0x1 ct mark set 0x2
    tcp dport 9000 ct mark 0x2 ct mark set 0x3
    tcp dport 22 ct mark 0x3 dnat to :22222
  }
}
```

Geographic Blocking

Country-Based Filtering

```
python
```

```python
#!/usr/bin/env python3
# geo_firewall.py - Geographic IP blocking

import ipaddress
import requests
import subprocess
import json
from typing import Set, List

class GeoFirewall:
    def __init__(self):
        self.ipset_name = "geo_block"
        self.blocked_countries = []
        self.allowed_countries = ["US", "CA", "GB"]  # Whitelist approach

    def download_ip_ranges(self, country_code: str) -> Set[str]:
        """Download IP ranges for a country"""
        url = f"https://www.ipdeny.com/ipblocks/data/countries/{country_code.lower()}.zone"
        try:
            response = requests.get(url, timeout=10)
            if response.status_code == 200:
                return set(response.text.strip().split('\n'))
        except:
            pass
        return set()

    def create_ipset(self):
        """Create ipset for geographic blocking"""
        subprocess.run(["ipset", "create", self.ipset_name, "hash:net", "hashsize", "4096"],
                       stderr=subprocess.DEVNULL)
        subprocess.run(["ipset", "flush", self.ipset_name], stderr=subprocess.DEVNULL)

    def add_country_to_ipset(self, country_code: str, action: str = "block"):
        """Add country IP ranges to ipset"""
        ip_ranges = self.download_ip_ranges(country_code)

        for ip_range in ip_ranges:
            if ip_range and not ip_range.startswith('#'):
                try:
                    # Validate IP range
                    ipaddress.ip_network(ip_range)
                    subprocess.run(["ipset", "add", self.ipset_name, ip_range],
                                   stderr=subprocess.DEVNULL)
                except:
                    continue

        print(f"Added {len(ip_ranges)} IP ranges for {country_code}")

    def apply_iptables_rules(self):
        """Apply iptables rules for geographic blocking"""
        # Remove existing rules
        subprocess.run(["iptables", "-D", "INPUT", "-m", "set", "--match-set",
                        self.ipset_name, "src", "-j", "DROP"], stderr=subprocess.DEVNULL)

        # Add new rule
        subprocess.run(["iptables", "-I", "INPUT", "1", "-m", "set", "--match-set",
                        self.ipset_name, "src", "-j", "DROP"])

        # Log blocked attempts
        subprocess.run(["iptables", "-I", "INPUT", "1", "-m", "set", "--match-set",
                        self.ipset_name, "src", "-j", "LOG", "--log-prefix", "GEO-BLOCKED: "])

    def setup_whitelist_mode(self):
        """Setup firewall in whitelist mode - only allow specific countries"""
        self.create_ipset()

        # Create a whitelist ipset
        subprocess.run(["ipset", "create", "geo_allow", "hash:net", "hashsize", "4096"],
                       stderr=subprocess.DEVNULL)

        # Add allowed countries
        for country in self.allowed_countries:
            ip_ranges = self.download_ip_ranges(country)
```

```
            for ip_range in ip_ranges:
                if ip_range and not ip_range.startswith('#'):
                    try:
                        subprocess.run(["ipset", "add", "geo_allow", ip_range],
                                       stderr=subprocess.DEVNULL)
                    except:
                        continue

        # Apply whitelist rules
        subprocess.run(["iptables", "-I", "INPUT", "1", "-m", "set", "!", "--match-set",
                        "geo_allow", "src", "-j", "DROP"])

    def block_high_risk_countries(self):
        """Block countries known for high cyber threat activity"""
        high_risk = ["CN", "RU", "KP", "IR", "SY"]  # Example list

        self.create_ipset()
        for country in high_risk:
            self.add_country_to_ipset(country)

        self.apply_iptables_rules()
        print(f"Blocked high-risk countries: {', '.join(high_risk)}")

# Usage
if __name__ == "__main__":
    geo_fw = GeoFirewall()
    geo_fw.block_high_risk_countries()
```

Rate Limiting & DDoS Protection

Advanced Rate Limiting

bash

```
#!/bin/bash
# Advanced DDoS protection with rate limiting

# Connection limit per IP
configure_connection_limits() {
    # Limit total connections per IP
    iptables -A INPUT -p tcp -m connlimit --connlimit-above 100 --connlimit-mask 32 -j REJECT --reject-with tcp-reset

    # Limit new connections per second per IP
    iptables -A INPUT -p tcp -m state --state NEW -m hashlimit \
        --hashlimit-name conn_rate \
        --hashlimit-mode srcip \
        --hashlimit-above 10/sec \
        --hashlimit-burst 20 \
        --hashlimit-htable-expire 10000 \
        -j DROP

    # SYN flood protection with adaptive rate limiting
    iptables -N SYN_FLOOD
    iptables -A INPUT -p tcp --syn -j SYN_FLOOD
    iptables -A SYN_FLOOD -m hashlimit \
        --hashlimit-name syn_flood \
        --hashlimit-mode srcip \
        --hashlimit-above 5/sec \
        --hashlimit-burst 10 \
        --hashlimit-htable-expire 10000 \
        -j LOG --log-prefix "SYN FLOOD: "
    iptables -A SYN_FLOOD -m hashlimit \
        --hashlimit-name syn_flood \
        --hashlimit-mode srcip \
        --hashlimit-above 5/sec \
        --hashlimit-burst 10 \
        --hashlimit-htable-expire 10000 \
        -j DROP
    iptables -A SYN_FLOOD -j RETURN

    # UDP flood protection
    iptables -A INPUT -p udp -m hashlimit \
        --hashlimit-name udp_flood \
        --hashlimit-mode srcip \
        --hashlimit-above 50/sec \
        --hashlimit-burst 100 \
        --hashlimit-htable-expire 10000 \
        -j DROP

    # ICMP flood protection
    iptables -A INPUT -p icmp --icmp-type echo-request -m hashlimit \
        --hashlimit-name icmp_flood \
        --hashlimit-mode srcip \
        --hashlimit-above 2/sec \
        --hashlimit-burst 5 \
        --hashlimit-htable-expire 10000 \
        -j DROP
}

# HTTP/HTTPS specific rate limiting
configure_http_rate_limiting() {
    # Limit HTTP requests per IP
    iptables -A INPUT -p tcp --dport 80 -m state --state NEW -m hashlimit \
        --hashlimit-name http_rate \
        --hashlimit-mode srcip \
        --hashlimit-above 30/sec \
        --hashlimit-burst 50 \
        --hashlimit-htable-expire 10000 \
        -j DROP

    # Limit HTTPS requests per IP
    iptables -A INPUT -p tcp --dport 443 -m state --state NEW -m hashlimit \
        --hashlimit-name https_rate \
        --hashlimit-mode srcip \
        --hashlimit-above 30/sec \
        --hashlimit-burst 50 \
        --hashlimit-htable-expire 10000 \
```

```
        -j DROP

    # Slowloris attack protection
    iptables -A INPUT -p tcp --dport 80 -m conntrack --ctstate NEW -m hashlimit \
        --hashlimit-name slowloris \
        --hashlimit-mode srcip \
        --hashlimit-above 10/min \
        --hashlimit-burst 15 \
        --hashlimit-htable-expire 60000 \
        -j DROP
}

# DNS amplification protection
configure_dns_protection() {
    # Rate limit DNS responses
    iptables -A INPUT -p udp --sport 53 -m hashlimit \
        --hashlimit-name dns_amplification \
        --hashlimit-mode srcip \
        --hashlimit-above 10/sec \
        --hashlimit-burst 20 \
        --hashlimit-htable-expire 10000 \
        -j DROP

    # Block DNS ANY queries
    iptables -A INPUT -p udp --dport 53 -m string --string "0000ff0001" --algo bm --from 40 -j DROP
}

configure_connection_limits
configure_http_rate_limiting
configure_dns_protection
```

Deep Packet Inspection Rules

Content-Based Filtering

```
python
```

```python
#!/usr/bin/env python3
# dpi_firewall.py - Deep packet inspection firewall

import pyshark
import ipaddress
import re
import subprocess
from typing import Dict, List, Tuple
import asyncio

class DPIFirewall:
    def __init__(self, interface: str = "eth0"):
        self.interface = interface
        self.suspicious_patterns = {
            'sql_injection': [
                r"(\bunion\b.*\bselect\b)",
                r"(\bselect\b.*\bfrom\b.*\bwhere\b)",
                r"(\bdrop\b.*\btable\b)",
                r"(\binsert\b.*\binto\b)",
                r"(\bupdate\b.*\bset\b)",
                r"(1\s*=\s*1)",
                r"(1\s*'\s*or\s*'1'\s*=\s*'1)",
            ],
            'xss': [
                r"<script[^>]*>.*?</script>",
                r"javascript:",
                r"on\w+\s*=",
                r"<iframe[^>]*>",
                r"eval\s*\(",
            ],
            'command_injection': [
                r";\s*cat\s+/etc/passwd",
                r";\s*ls\s+-la",
                r";\s*wget\s+",
                r";\s*curl\s+",
                r"\|\s*nc\s+",
                r"`.*`",
                r"\$\(.*\)",
            ],
            'path_traversal': [
                r"\.\./\.\./",
                r"\.\.\\\.\.\\",
                r"/etc/passwd",
                r"/etc/shadow",
                r"c:\\windows\\system32",
            ],
            'suspicious_user_agents': [
                r"nikto",
                r"sqlmap",
                r"metasploit",
                r"nmap",
                r"masscan",
                r"python-requests",
                r"curl/",
                r"wget/",
            ]
        }

        self.blocked_ips = set()
        self.packet_stats = {}

    async def analyze_packet(self, packet):
        """Analyze packet for suspicious content"""
        try:
            # Extract packet data
            if hasattr(packet, 'ip'):
                src_ip = packet.ip.src
                dst_ip = packet.ip.dst

                # Check HTTP/HTTPS traffic
                if hasattr(packet, 'http'):
                    await self.check_http_packet(packet, src_ip)
```

```
            # Check DNS traffic
            if hasattr(packet, 'dns'):
                await self.check_dns_packet(packet, src_ip)

            # Check for suspicious payloads
            if hasattr(packet, 'data'):
                await self.check_payload(packet.data.data, src_ip)

        except AttributeError:
            pass

    async def check_http_packet(self, packet, src_ip: str):
        """Check HTTP packet for attacks"""
        suspicious = False

        # Check URI
        if hasattr(packet.http, 'request_uri'):
            uri = packet.http.request_uri
            for pattern_list in [self.suspicious_patterns['sql_injection'],
                                 self.suspicious_patterns['xss'],
                                 self.suspicious_patterns['path_traversal']]:
                for pattern in pattern_list:
                    if re.search(pattern, uri, re.IGNORECASE):
                        print(f"[DPI] Suspicious URI from {src_ip}: {uri[:50]}...")
                        suspicious = True
                        break

        # Check User-Agent
        if hasattr(packet.http, 'user_agent'):
            user_agent = packet.http.user_agent
            for pattern in self.suspicious_patterns['suspicious_user_agents']:
                if re.search(pattern, user_agent, re.IGNORECASE):
                    print(f"[DPI] Suspicious User-Agent from {src_ip}: {user_agent}")
                    suspicious = True
                    break

        # Check POST data
        if hasattr(packet.http, 'file_data'):
            post_data = packet.http.file_data
            for pattern_type, patterns in self.suspicious_patterns.items():
                if pattern_type != 'suspicious_user_agents':
                    for pattern in patterns:
                        if re.search(pattern, post_data, re.IGNORECASE):
                            print(f"[DPI] Suspicious POST data from {src_ip}")
                            suspicious = True
                            break

        if suspicious:
            await self.block_ip(src_ip)

    async def check_dns_packet(self, packet, src_ip: str):
        """Check DNS packet for suspicious queries"""
        if hasattr(packet.dns, 'qry_name'):
            domain = packet.dns.qry_name

            # Check for DNS tunneling (long domain names)
            if len(domain) > 100:
                print(f"[DPI] Possible DNS tunneling from {src_ip}: {domain[:50]}...")
                await self.block_ip(src_ip)

            # Check for DGA domains (high entropy)
            if self.calculate_entropy(domain) > 4.0:
                print(f"[DPI] Possible DGA domain from {src_ip}: {domain}")
                await self.block_ip(src_ip)

            # Check for suspicious TLDs
            suspicious_tlds = ['.tk', '.ml', '.ga', '.cf', '.click', '.download']
            if any(domain.endswith(tld) for tld in suspicious_tlds):
                print(f"[DPI] Suspicious TLD from {src_ip}: {domain}")

    async def check_payload(self, payload: str, src_ip: str):
        """Check raw payload for suspicious content"""
        if not payload:
```

```
            return

        # Check for binary exploits
        if b'\x90\x90\x90\x90' in payload:  # NOP sled
            print(f"[DPI] Possible buffer overflow attempt from {src_ip}")
            await self.block_ip(src_ip)

        # Check for shell commands
        shell_commands = [b'/bin/sh', b'/bin/bash', b'cmd.exe', b'powershell.exe']
        for cmd in shell_commands:
            if cmd in payload:
                print(f"[DPI] Shell command detected from {src_ip}")
                await self.block_ip(src_ip)

    def calculate_entropy(self, string: str) -> float:
        """Calculate Shannon entropy of a string"""
        import math
        from collections import Counter

        if not string:
            return 0

        entropy = 0
        counter = Counter(string)
        total = len(string)

        for count in counter.values():
            probability = count / total
            if probability > 0:
                entropy -= probability * math.log2(probability)

        return entropy

    async def block_ip(self, ip: str):
        """Block IP address using iptables"""
        if ip not in self.blocked_ips:
            self.blocked_ips.add(ip)
            subprocess.run(['iptables', '-A', 'INPUT', '-s', ip, '-j', 'DROP'])
            print(f"[DPI] Blocked IP: {ip}")

    async def start_capture(self):
        """Start packet capture and analysis"""
        print(f"[DPI] Starting deep packet inspection on {self.interface}")

        capture = pyshark.LiveCapture(interface=self.interface)

        async for packet in capture.sniff_continuously():
            await self.analyze_packet(packet)

# Usage
if __name__ == "__main__":
    dpi = DPIFirewall(interface="eth0")
    asyncio.run(dpi.start_capture())
```

Automated Rule Generation

Machine Learning-Based Rule Generation

```
python
```

```python
#!/usr/bin/env python3
# ml_firewall_rules.py - ML-based firewall rule generation

import numpy as np
import pandas as pd
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
import joblib
import subprocess
from datetime import datetime, timedelta
from typing import List, Dict, Tuple

class MLFirewallRuleGenerator:
    def __init__(self):
        self.model = IsolationForest(contamination=0.1, random_state=42)
        self.scaler = StandardScaler()
        self.rules_generated = []
        self.traffic_data = []

    def collect_traffic_data(self, duration_minutes: int = 60) -> pd.DataFrame:
        """Collect network traffic data for analysis"""
        print(f"Collecting traffic data for {duration_minutes} minutes...")

        # Parse netstat output
        data = []
        end_time = datetime.now() + timedelta(minutes=duration_minutes)

        while datetime.now() < end_time:
            output = subprocess.check_output(['ss', '-tunap'], text=True)

            for line in output.split('\n')[1:]:
                if line and not line.startswith('State'):
                    parts = line.split()
                    if len(parts) >= 5:
                        try:
                            proto = parts[0]
                            local = parts[3]
                            remote = parts[4]

                            # Extract port and IP
                            if ':' in local and ':' in remote:
                                local_port = int(local.split(':')[-1])
                                remote_ip = remote.rsplit(':', 1)[0]
                                remote_port = int(remote.split(':')[-1])

                                data.append({
                                    'timestamp': datetime.now(),
                                    'protocol': proto,
                                    'local_port': local_port,
                                    'remote_ip': remote_ip,
                                    'remote_port': remote_port
                                })
                        except (ValueError, IndexError):
                            continue

            time.sleep(5)  # Collect every 5 seconds

        return pd.DataFrame(data)

    def extract_features(self, df: pd.DataFrame) -> np.ndarray:
        """Extract features from traffic data"""
        features = []

        # Group by remote IP
        for ip in df['remote_ip'].unique():
            ip_data = df[df['remote_ip'] == ip]

            features.append([
                len(ip_data),  # Connection count
                ip_data['local_port'].nunique(),  # Unique local ports
                ip_data['remote_port'].nunique(),  # Unique remote ports
                (ip_data['timestamp'].max() - ip_data['timestamp'].min()).seconds,  # Duration
                len(ip_data[ip_data['protocol'] == 'tcp']) / max(len(ip_data), 1),  # TCP ratio
```

```
            len(ip_data[ip_data['remote_port'] < 1024]) / max(len(ip_data), 1),  # Privileged port ratio
        ])

    return np.array(features)

def train_model(self, normal_traffic_df: pd.DataFrame):
    """Train anomaly detection model on normal traffic"""
    features = self.extract_features(normal_traffic_df)

    # Scale features
    features_scaled = self.scaler.fit_transform(features)

    # Train model
    self.model.fit(features_scaled)

    # Save model
    joblib.dump(self.model, 'firewall_ml_model.pkl')
    joblib.dump(self.scaler, 'firewall_scaler.pkl')

    print("Model trained on normal traffic patterns")

def detect_anomalies(self, traffic_df: pd.DataFrame) -> List[str]:
    """Detect anomalous IPs in traffic"""
    features = self.extract_features(traffic_df)
    features_scaled = self.scaler.transform(features)

    # Predict anomalies
    predictions = self.model.predict(features_scaled)

    # Get anomalous IPs
    anomalous_ips = []
    unique_ips = traffic_df['remote_ip'].unique()

    for i, pred in enumerate(predictions):
        if pred == -1:  # Anomaly
            anomalous_ips.append(unique_ips[i])

    return anomalous_ips

def generate_rules(self, anomalous_ips: List[str]) -> List[str]:
    """Generate firewall rules for anomalous IPs"""
    rules = []

    for ip in anomalous_ips:
        # Generate iptables rule
        rule = f"iptables -A INPUT -s {ip} -j DROP"
        rules.append(rule)

        # Generate nftables rule
        nft_rule = f"nft add rule inet filter input ip saddr {ip} drop"
        rules.append(nft_rule)

        # Generate fail2ban filter
        f2b_rule = f"[ml-anomaly-{ip.replace('.', '-')}]\nenabled = true\nfilter = anomaly\naction = iptables[name=ML, p
        rules.append(f2b_rule)

    self.rules_generated.extend(rules)
    return rules

def apply_rules(self, rules: List[str], dry_run: bool = True):
    """Apply generated firewall rules"""
    if dry_run:
        print("DRY RUN - Rules to be applied:")
        for rule in rules:
            print(f"  {rule}")
    else:
        for rule in rules:
            if rule.startswith('iptables') or rule.startswith('nft'):
                subprocess.run(rule.split(), check=False)
                print(f"Applied: {rule}")

def adaptive_learning(self):
    """Continuously learn and adapt rules"""
```

```
        while True:
            # Collect recent traffic
            recent_traffic = self.collect_traffic_data(duration_minutes=10)

            # Detect anomalies
            anomalous_ips = self.detect_anomalies(recent_traffic)

            if anomalous_ips:
                print(f"Detected {len(anomalous_ips)} anomalous IPs")

                # Generate rules
                rules = self.generate_rules(anomalous_ips)

                # Apply rules (dry run by default)
                self.apply_rules(rules, dry_run=True)

            # Wait before next iteration
            time.sleep(600)  # 10 minutes

# Usage
if __name__ == "__main__":
    ml_firewall = MLFirewallRuleGenerator()

    # Collect normal traffic for training
    normal_traffic = ml_firewall.collect_traffic_data(duration_minutes=60)

    # Train model
    ml_firewall.train_model(normal_traffic)

    # Start adaptive learning
    ml_firewall.adaptive_learning()
```