Multidisciplinarita vo výučbe

V rámci tohto blogu bude poukázané na možnosti prepojenia vedomostí z rôznych oblastí informatiky a ich využitie pri tvorbe nástroja na penetračné testovanie. Vytvorený blog ale nebude obsahovať konkrétne technické detaily potrebné k implementácií. Jeho cieľom bude skôr abstraktne poukázať na súvislosti a vzájomnú nadväznosť. Uvedená myšlienka multidisciplinarity má motivovať študentov k tomu, aby sa nešpecializovali pri výučbe len na štúdium jedného predmetu, ale aby venovali pozornosť širšej škále oblastí a videli medzi nimi vzájomné prepojenie.

Tento blog poukáže na možnosť prepojenia vedomostí z oblasti počítačových sietí, kyberbezpečnosti, programovania, databázových systémov, analýzy dát strojovým učením a tvorby mobilných aplikácií. Ako vzorová úloha bude tvorba nástroja na penetračné testovanie (odhalenie zraniteľností v testovanom systéme), pričom pozornosť bude venovaná primárne na bezpečnosť z pohľadu počítačových sietí.

Vedomosti z oblasti počítačových sietí

Oblasť počítačových sietí predstavuje dôležitý základ pre pochopenie spôsobu komunikácie medzi zariadeniami v informačných systémoch. Práve táto komunikácia je často cieľom rôznych útokov (odchytenie, pozmenenie, oklamanie firewall-u, ...). Aby sa študenti dokázali orientovať v tejto oblasti, tak potrebujú rozumieť OSI modelu, hlavičkám jednotlivých dátových jednotiek (rámec, paket, segment) a mechanizmom, na základe ktorých sú založené sieťové protokoly (ARP, OSPF, DHCP, ...) a funkčnosť sieťových zariadení (prepínač, smerovač).

Vedomosti z oblasti kyberbezpečnosti

Penetračné testovanie je založené na realizácií útoku na vlastnú infraštruktúru s cieľom odhalenia zraniteľných miest. Sieťový administrátor a analytik sieťovej bezpečnosti musia mať teda hlboké vedomosti o rôznych typoch sieťových zraniteľností a útokoch. Ak tieto vedomosti majú, tak môžu následne navrhnúť nástroj na simuláciu rôznych útokov (STP root bridge manipulation, skenovanie aktívnych IP adries, ARP spoofing, VLAN hopping, odchytávanie a analýza prenášaných dát, ...).

Ako je možné vidieť, tak prvým dôležitým prepojením je využitie znalostí z oblasti počítačových sietí a ich využitie pre návrh nástroja pre simuláciu útoku. Napr. po štúdiu kurzu CCNA1/2 majú študenti znalosť o protokole ARP a vedia, že ARP tabuľka sa dokáže plniť aj po prijatí nevyžiadanej odpovede, čím môže útočník vykonať Man in the Middle útok. Aby mohol byť útok odsimulovaný, tak daný tester sieťovej bezpečnosti musí dokonale poznať správanie ARP protokolu, štruktúru jeho hlavičiek a spôsob komunikácie a prenášania správ (Unicast/Broadcast? ...).

Vedomosti z oblasti programovania

Vo chvíli, keď bola vykonaná analýza prostredia, protokolov a navrhnutý spôsob na vykonanie útoku, je možné začať pripravovať nástroj na vykonanie testu/útoku (angl. Exploit). Na internete existuje množstvo rôznych hotových nástrojov. Taktiež napr. OS Kali Linux obsahuje veľké množstvo predinštalovaných nástrojov, ktoré je možné rovno použiť na vykonanie rôznych penetračných testov.

Pre zaujatie študentov je vhodné využiť niektorý z existujúcich nástrojov. No z pohľadu vyučovania a pre lepšie pochopenie je vhodné vytvoriť si takýto nástroj vo vlastnej réžii. Tento prístup ale vyžaduje aspoň základné chápanie programovania. Simuláciu rôznych testov je ale možné značne zjednodušiť využívaním rôznych knižníc.

Pre začiatok je veľmi vhodné začať programovacím jazykom Python. Tento jazyk navyše poskytuje knižnicu **Scapy**, využitím ktorej je možné veľmi jednoducho vytvárať dátovú jednotku (rámec, paket, segment) a meniť jej atribúty podľa našich požiadaviek. Aké je to jednoduché ukáže nasledujúca vzorová úloha:

1. Vytvorte dátovú jednotku protokolu ICMP (echo request), ktorej nastavte zdrojovú IPv4 adresu na 192.168.8.13 a cieľovú IPv4 adresu na 10.20.22.7. Využitím vytvorenej dátovej jednotky následne vyneste citlivú správu (naplňte payload), ktorej znenie bude „Secret_MSG“.
   Pozn.: Využitím protokolu ICMP je možné vyniesť citlivé dáta zo siete, no zároveň je možné v prípade vygenerovania viacerých ICMP správ s meniacou sa cieľovou adresou vykonať skenovanie siete a zistiť dostupné IPv4 adresy v sieti, na ktoré je následne možné začať vykonávať útok.

   Riešenie:
   Nasledujúci zdrojový kód znázorňuje, ako je možné využitím knižnice Scapy vytvoriť dátovú jednotku a upraviť v nej akékoľvek parametre (obsiahnuté v hlavičkách).

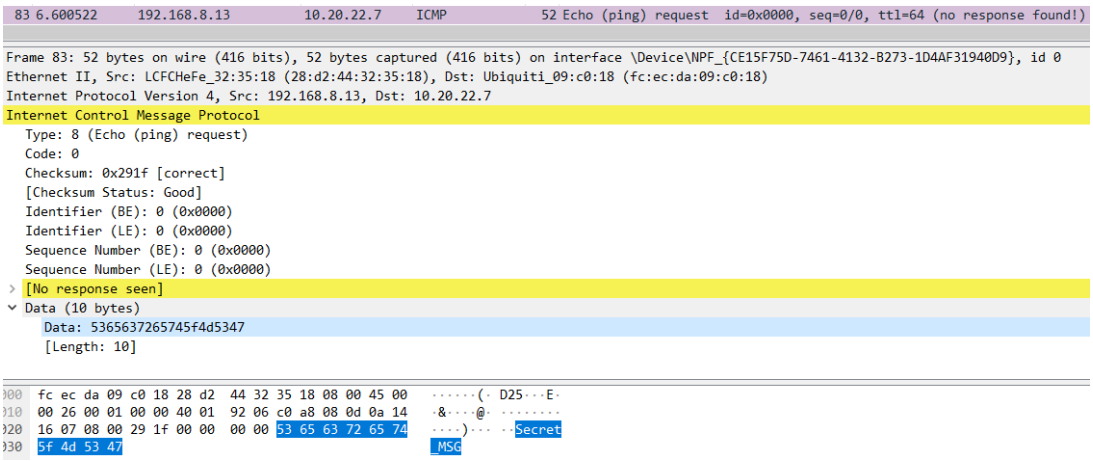
```
from scapy.layers.inet import ICMP, IP
from scapy.sendrecv import sr1

#vytvorenie dátovej jednotky = default IP  + default ICMP + payload
icmp = IP()/ICMP()/"Secret_MSG"

#úprava atribútov IP hlavičky
icmp[IP].src = "192.168.8.13"
icmp[IP].dst = "10.20.22.7"

#odoslanie požiadaviek
resp = sr1(icmp, timeout=2)
```

   Overenie úspešnosti odoslania dátovej jednotky s upravenými nastaveniami je možné štandardne využitím nástroja Wireshark. Nasledujúci obrázok znázorňuje očakávaný výstup:



Ako je možné vidieť, tak nami odoslaná dátová jednotka (echo request) nedostala odpoveď. Čo je ale v poriadku, keďže zdrojová a cieľová adresa boli vymyslené a teda nemal kto odpovedať a ani komu. V prípade použitia korektných adries by bolo možné vidieť aj úspešné prijatie odpovede.

2. Ďalšou jednoduchou úlohou by bolo vytvorenie vlastného nástroja na odchytávanie komunikácie. Znova je za týmto účelom možné využiť knižnicu Scapy.
Riešenie:

```
from scapy.layers.inet import IP
from scapy.packet import Raw
from scapy.sendrecv import sniff

# odchytenie 5 ICMP správ a vytvorenie poľa pre ukladanie kľúčových charakteristík
packets = sniff(filter="icmp", count= 5)
rcv_data = []

#cyklus, ktorý prejde každou odchytenou ICMP správou
for packet in range(0, 5):
    #do JSON podoby uloží hľadané charakteristiky (zdrojová/cieľová adresa a payload)
    packet_info = {
        "src_address": packets[packet][IP].src,
        "dst_address": packets[packet][IP].dst,
        "payload": packets[packet][Raw].load.decode()
    }
    #uloženie dát do vytvoreného poľa
    rcv_data.append(packet_info)
#výpis obsahu odchytených dát
print(rcv_data)
```

Očakávaný výstup programu je nasledovný:
[{'src_address': '10.10.10.19', 'dst_address': '142.251.36.110', 'payload': 'abcdefghijklmnopqrstuvwabcdefghi'}]

Ako je možné vidieť, tak na niekoľkých riadkoch kódu je možné odchytiť ľubovoľnú komunikáciu (definovaná filtrom) a následne z nej získať požadované charakteristiky, uložiť ich napr. do JSON štruktúry a následne je s nimi možné pracovať ďalej podľa požiadaviek.

Problém ale nastáva vtedy, ak by sme odchytávali veľké množstvo dát, prípadne by sme sa potrebovali dostať spätne k historickým dátam. V tomto prípade je vhodné dané dáta nemať uložené len v pamäti spustenej aplikácie, ale ukladať ich niekam do databázy, v ktorej je následne možné vykonávať efektívnejšie vyhľadávanie, archiváciu a pod.

Vedomosti z oblasti databázových systémov

Už keď vieme, že práca s databázou sa hodí pri spracovávaní veľkého množstva dát, tak pre jej plnohodnotné využitie je potrebné vedieť využívať aspoň jej základné funkcionality, medzi ktoré patrí vytvorenie tabuľky (CREATE TABLE), vloženie dát (INSERT) a získavanie dát (SELECT). Dotazy do databázy sa štandardne píšu využitím SQL jazyka. Ako je možné vidieť, tak ďalšou kľúčovou oblasťou, ktorá má priame prepojenie či už na oblasť kyberbezpečnosti alebo programovania, je oblasť databázových systémov. Vzorové ukážky práce s databázou sú uvedené v nasledujúcich výpisoch:

1. Pripojenie sa na databázu:

```
#import knižnice pre prácu s PostgreSQL databázou
import psycopg2
#pripojenie sa na databázu
connection = psycopg2.connect(user="postgres",
                              password="DBpass123!",
                              host="127.0.0.1",
                              port="5432",
                              database="test_DB")
#vytvorenie objektu pre prácu s databázou
cursor = connection.cursor()
```

2. Vytvorenie tabuľky:

```
#vytvorenie SQL dotazu
create_table_query = """CREATE TABLE new_table (
    msg_id SERIAL PRIMARY KEY,
    src_ip VARCHAR(15) NOT NULL,
    dst_ip VARCHAR(15) NOT NULL,
    payload VARCHAR(32) NOT NULL
)"""
#vykonanie dotazu
cursor.execute(create_table_query)
#uloženie vykonaných zmien
connection.commit()
```

3. Vloženie dát:

```
#vytvorenie SQL dotazu
insert_query = """INSERT INTO new_table
(src_ip, dst_ip, payload)
VALUES (%s, %s, %s)
"""
```

```
#špecifikácia vstupných dát, ktoré budú do databázy vložené
value_tuple = ("10.10.10.19", "10.20.30.40", "Secret_MSG")
#vykonanie požiadavky s vložením vstupných dát
cursor.execute(insert_query, value_tuple)
#uloženie vykonaných zmien
connection.commit()
```

4. Získavanie dát:

```
#odoslanie dotazu pre získanie dát
cursor.execute("SELECT * from new_table")
#uloženie prijatých dát
record = cursor.fetchall()
```

S databázou sa dá samozrejme pracovať aj cez grafické nástroje, ktoré sú súčasťou inštalačného balíčka väčšiny databázových systémov. Práca s GUI je vhodná pre jednoduchšiu vizuálnu kontrolu.

Vo chvíli, keď máme dáta uložené v databáze, tak ich je možné začať analyzovať, vyhľadávať v nich špecifické vzory, anomálie a pod. Analýza môže byť štatistická (hľadanie maximálnych, minimálnych, často sa objavujúcich hodnôt a pod.) alebo pokročilejšia využitím metód strojového učenia.

Vedomosti z oblasti analýzy dát strojovým učením

Analýza dát je komplexný proces. Aktuálne jedným z najpoužívanejších spôsobov je využívanie strojového učenia na klasifikáciu/regresiu rôznych situácií. Modely strojového učenia na svojom pozadí pracujú s pokročilou matematikou, avšak pre prvý kontakt, so strojovým učením, nepotrebujeme hlboké matematické znalosti. Pri základnej práci so strojovým učením stačí chápať princípu, ktorý je možné vyjadriť v zdrojovom kóde na zopár riadkoch.

Princíp práce so strojovým učením je nasledovný (venovať sa budeme prístupu „Učenie s učiteľom“). Tento prístup je založený na tom, že najskôr zozbierame dáta a vytvoríme z nich dátovú sadu (štandardná tabuľka v Exceli), kde stĺpce predstavujú sledované charakteristiky a riadky predstavujú samotné záznamy (napr. o dátových tokoch a pod.). Následne pridáme jeden nový stĺpec, ktorý manuálne vyplníme značkou (špecifikujeme hľadanú vlastnosť – napr. či daná množina charakteristík danej komunikácie predstavuje útok alebo nie). Dátová sada teda obsahuje X maticu (záznamy) a Y vektor (značka, či sa jedná o útok alebo nie). Túto dvojicu parametrov následne vložíme ako argumenty pre zvolený model (v našom prípade klasifikátor) strojového učenia, ktorý sa na daných dátach natrénuje. Naučí sa aká kombinácia v X matici predstavuje akú hodnotu v Y vektore. Následne tento natrénovaný model použijeme na predikovanie značky Y pre novo odchytené / analyzované dáta. Z pohľadu zdrojového kódu by to mohlo vyzerať nasledovne:

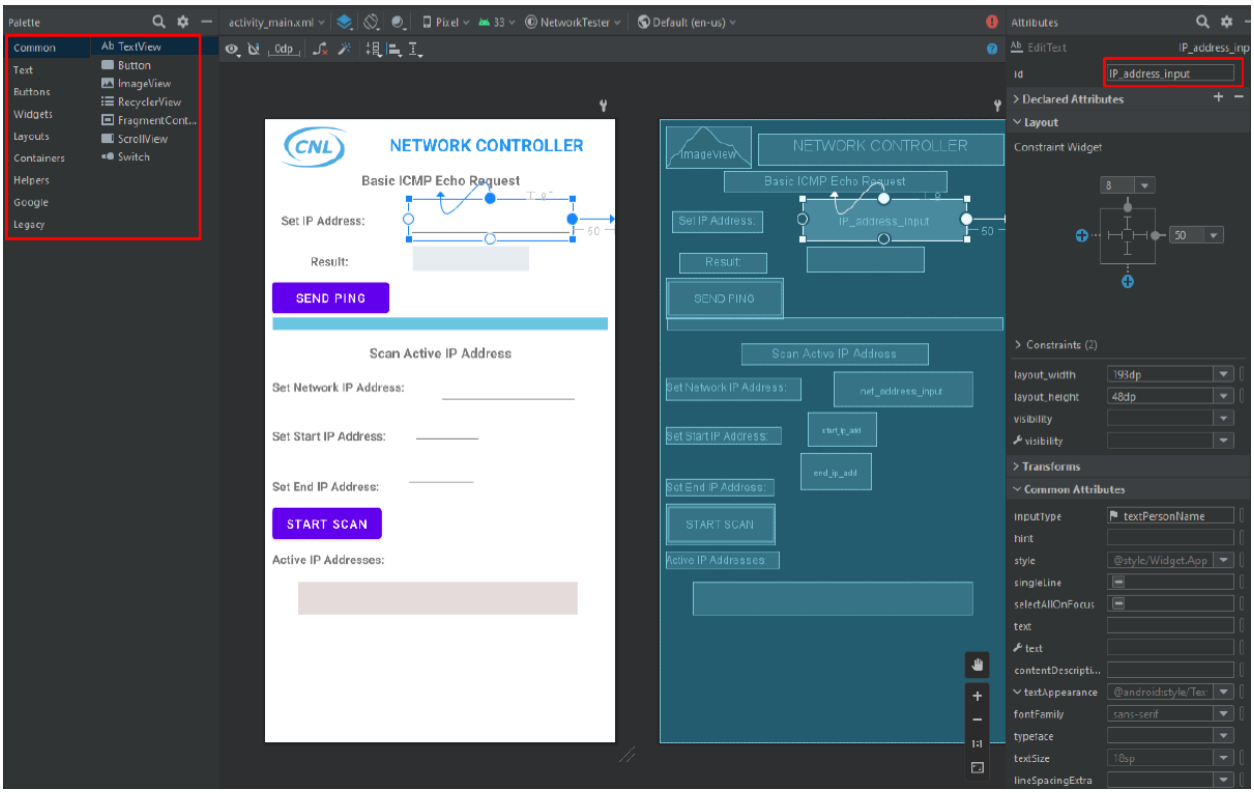
```
#importovanie potrebnej knižnice
from sklearn.ensemble import RandomForestClassifier
#vytvorenie objektu reprezentujúceho zvolený klasifikátor – v tomto prípade
random forest
rf = RandomForestClassifier(criterion="entropy", max_depth=5, n_estimators=5)
#trénovanie modelu
rf.fit(X_train, y_train)
#predikcia na nových dátach, ktoré model ešte nevidel
final_predict = rf.predict(X_test)
```

UPOZORNENIE: Ako je možné vidieť z predchádzajúcich častí tohto dokumentu, tak rôzne oblasti informatiky majú vzájomné prepojenie, ktoré umožňuje tvorbu rôznych sofistikovaných riešení v oblasti informačných systémov, ich zabezpečenia a pod. Vzhľadom k tomu, je pri štúdiu potrebné venovať pozornosť rôznym oblastiam a predmetom štúdia!!!

BONUS: Vedomosti z oblasti tvorby mobilných aplikácií

Motiváciu študentov je možné zvýšiť napr. tým, že si vyskúšajú vytvoriť aplikáciu využitím iného typu používateľského rozhrania a to napr. tvorbou mobilnej aplikácie. Vyššie uvedené princípy aj v tomto prípade ostávajú rovnaké, no zvýši sa rozsah aplikačnej domény a jednoduchosť prístupu k dátam a vytváranému riešeniu.

Vývoj mobilných aplikácií je možné vykonávať napr. využitím vývojového prostredia Android Studio, kde je možné jednoduchým ťahaním elementov vytvoriť používateľské rozhranie, čo znázorňuje nasledujúci obrázok:



Dané elementy rozhrania sú identifikovateľné svojím ID, cez ktoré je k nim možné pristupovať z BackEnd časti aplikácie, kde po stlačení istého tlačidla môže dôjsť k volaniu obslužnej funkcie, ktorá už je napísaná v bežnom programovacom jazyku ako Kotlin, Java a pod. a využíva princípy uvedené v prechádzajúcich sekciách tohto dokumentu.