סיכום מקוף

רשתות מחשבים

קובץ 2 מתוך 4 — פרקים 3–6

עמודים 81–160 מתוך הספר המקורי

תוכן קובץ זה:

- פרק 3 — Wireshark ומודל חמש השכבות
- פרק 4 — שכבת האפליקציה (HTTP, DNS, SMTP)
- פרק 5 — Scapy (כלי לניתוח ושליחת פקטות)
- פרק 6 — שכבת התעבורה (מבוא, פורטים, TCP/UDP)

הסיכום שומר על כל המושגים הטכניים, הגדרות, פקודות קוד ועקרונות מרכזיים

תוכן עניינים

פרק 3 Wireshark ומודל חמש השכבות 2

מסך ההסנפה הראשי — 4 חלונות

פילטרים חיוניים

Follow TCP/UDP Stream

סטטיסטיקות ו-Conversations

פרק 4 שכבת האפליקציה 4

מהו משאב רשת?

HTTP — מבנה בקשה ותגובה

Headers, סטטוס קודים, URL

GET עם פרמטרים, POST

Cookies ו-Cache

פרוטוקול DNS — שאילתה ותגובה

היררכיית שמות DNS

SMTP — דואר אלקטרוני

פרק 5 Scapy 9

הסנפה עם sniff()

יצירת פקטות ושכבות

שליחת פקטות, send()

שימוש מסקריפט Python

פרק 6 שכבת התעבורה 12

פורטים — מזהה האפליקציה

TCP ו-UDP

הכלי netstat

פרק 3

Wireshark ומודל חמש השכבות

כלי ניתוח תעבורת רשת — הסנפה, פילטרים, וניתוח פקטות

מסך ההסנפה הראשי — 4 חלונות

Wireshark הוא כלי ניתוח תעבורת רשת (Network Analyzer) שמאפשר להציג, לנתח ולסנן פקטות שעוברות בכרטיס הרשת. כדי להתחיל הסנפה: בחרו כרטיס רשת ← לחצו **Start**.

🖥 ארבעת החלונות המרכזיים

חלון	תפקיד
① **Display Filter** (אדום)	כתיבת ביטוי פילטר לסינון הפקטות המוצגות. לדוגמה: `http` , `dns` , `icmp`
② **רשימת הפקטות** (בחול)	כל פקטה שנקלטה. שדות: *No., Time, Source, Destination, Protocol, Length, Info*
③ **ניתוח הפקטה** (ירוק)	פירוט שכבתי של הפקטה הנבחרת — Ethernet, IP, TCP/UDP, שכבת האפליקציה
④ **תוכן הפקטה** (כתום)	ייצוג הקסדצימלי (hex) משמאל + ייצוג ASCII מימין

💡 טיפ חשוב: שדות בסוגריים מרובעים [ ]

שדה שמוצג בסוגריים מרובעים (כגון `[Response time: 155 ms]`) **אינו חלק מהפרוטוקול המקורי** — הוא מחושב על-ידי Wireshark עצמו, לנוחות המשתמש.

פילטרים חיוניים

פילטרים כתובים בשורת ה-Filter ומסננים אילו פקטות מוצגות. ניתן לבנות ביטויי פילטר בשני דרכים: (1) תפריט (Expression, (2 לחיצה ימנית על שדה בלשהו → *Selected* → *Apply as Filter*.

פילטר	משמעות
`http`	רק פקטות פרוטוקול HTTP
`dns`	רק פקטות DNS
`icmp`	בקשות והשובות ping
`tcp`	כל התעבורת TCP
`udp.port == 53`	תעבורת UDP על פורט 53 (DNS)
`ip.addr == 192.168.1.1`	פקטות מ/אל כתובת IP ספציפית
`ip.src == 192.168.1.1`	פקטות שמקורן ב-IP זה
`ip.dst == 8.8.8.8`	פקטות שיעדן ב-IP זה
`http.host == "www.wireshark.org"`	בקשות HTTP ל-Host ספציפי
`frame contains "Hello"`	מסגרות שמכילות הטקסט "Hello"
`frame.len < 100`	מסגרות באורך פחות מ-100 בתים
`frame.time >= "2011-01-16 07:30:00"`	מסגרות מזמן מסוים ואילך
`icmp.resptime > 100`	תשובות ICMP עם זמן תגובה > 100ms

`tcp.port == 80`	תעבורת TCP על פורט 80 (HTTP)

> **🔍 אובייקט frame**
>
> **frame** מסמן כל מסגרת שנקלטה, ללא תלות בפרוטוקול. שדות שימושיים: `frame.len` (אורך), `frame.time` (זמן), `frame contains "text"` (חיפוש תוכן חופשי).

אופרטורים לוגיים בפילטרים

שילוב פילטרים

```
http and ip.src == 192.168.1.1       (AND)
http or dns                              (OR)
not arp                                  (NOT)
tcp.port == 80 || tcp.port == 443        (OR עם ||)
ip.addr == 10.0.0.1 && http             (AND עם &&)
```

Follow TCP/UDP Stream

אחת הפיצ'רים החשובים ביותר ב-Wireshark. מפלטרת את כל הפקטות השייכות לאותו Socket ומציגה את התעבורה "מנקודת מבט האפליקציה".

> **📋 כיצד להשתמש ב-Follow TCP Stream**
>
> 1. סמנו פקטה כלשהי מהחיבור שמעניין אתכם
> 2. לחצו כפתור ימני ← **Follow TCP Stream**
> 3. יפתח חלון המציג את כל המידע שעבר ב-Socket
> 4. אדום = לקוח ← שרת | כחול = שרת ← לקוח
>
> **יתרון:** רואים רק Data של שכבת האפליקציה, ללא Headers של TCP/IP.

סטטיסטיקות

תחת תפריט **Statistics** ניתן למצוא:

כלי סטטיסטיקה	שימוש
Packet Lengths	התפלגות אורכי הפקטות בהסנפה
IO Graph	גרף כמות התעבורה בזמן אמת
Endpoints	כל ישויות הרשת שנקלטו (כתובות IP + מדדי תעבורה)
Conversations	כל "שיחות" דו-כיוון בין זוגות של ישויות רשת — IP, MAC, TCP, UDP

> **✅ סיכום פרק 3 — מה צריך לדעת**
>
> - 4 חלונות Wireshark: Filter, רשימת פקטות, ניתוח פקטה, hex dump
> - שדות בסוגריים מרובעים = חישוב של Wireshark, לא חלק מהפרוטוקול
> - פילטרים נפוצים: `http`, `dns`, `icmp`, `frame contains "..."`
> - Follow TCP Stream — לצפייה בתוכן ה-Socket בפורמט נוח
> - קליק על בית (byte) בחלון hex → מדגיש את השדה המתאים בחלון הניתוח
> - לחיצה ימנית → Apply as Filter → השתמשו כדי ללמוד שמות פילטרים חדשים

פרק 4

שכבת האפליקציה

הפרוטוקולים שרואפליקציות מדברות בהם — HTTP, DNS, SMTP

> 📌 **הגדרה — שכבת האפליקציה**
>
> **שכבת האפליקציה** היא אוסף הפרוטוקולים שבהם האפליקציות משתמשות באופן ישיר, והיא מספקת *הפשטה מעל תקשורת הנתונים* ברשת האינטרנט. היא מאפשרת לאפליקציות לשלוח ולקבל מידע מבלי לדאוג לפרטי ההעברה הנמוכים יותר.

🌐 פרוטוקול HTTP

HTTP (HyperText Transfer Protocol) הוא הפרוטוקול הנפוץ ביותר לגלישה באינטרנט. **HTTPS** הוא גרסתו המוצפנת (S = Secure). HTTP עובד על עיקרון **בקשה-תגובה**: הלקוח שולח בקשה, השרת מחזיר תגובה.

מבנה בקשת HTTP

מבנה בקשת HTTP — פורמט כללי

```
METHOD /path/to/resource HTTP/1.1\r\n
Header-Name: Header-Value\r\n
Another-Header: Value\r\n
\r\n
[Body - רק ב-POST]
```

דוגמה — בקשת GET לאתר CERN

```
GET /hypertext/WWW/TheProject.html HTTP/1.1\r\n
Host: info.cern.ch\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: he\r\n
User-Agent: Mozilla/5.0 (Windows NT 6.1)\r\n
Connection: Keep-Alive\r\n
\r\n
```

מבנה תגובת HTTP

מבנה תגובת HTTP

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52\r\n
Content-Length: 2652\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
<!DOCTYPE html>
<html>...תוכן הדף...</html>
```

> ⚠ **חשוב: תווי הפרדה**

- (CRLF) `\r\n` = סיום כל שורה (CR = carriage return, LF = line feed)
- `\r\n\r\n` = מסמן את **סיום ה-Headers** ותחילת ה-Body
- בשרטוט: **sp** = רווח, **lf** = \n, **cr** = \r

קודי מצב (Status Codes) HTTP

קוד	שם	משמעות
`200`	OK	הצלחה — המשאב הוחזר בראוי
`302`	Moved Temporarily	המשאב עבר מיקום זמנית — הלקוח צריך לפנות לכתובת חדשה
`404`	Not Found	המשאב המבוקש לא נמצא
`403`	Forbidden	אין הרשאת גישה למשאב
`500`	Internal Server Error	שגיאת שרת פנימית

Headers נפוצים

Header	מיקום	משמעות
`Host`	בקשה	שם הדומיין של השרת (חובה ב-HTTP/1.1)
`User-Agent`	בקשה	סוג הלקוח (דפדפן, גרסה, OS)
`Accept`	בקשה	סוגי תוכן שהלקוח מוכן לקבל
`Accept-Language`	בקשה	שפה מועדפת (he = עברית)
`Connection`	שניהם	Keep-Alive / Close — ניהול חיבור
`Content-Type`	תגובה	סוג התוכן שמוחזר (text/html, image/jpeg...)
`Content-Length`	שניהם	גודל ה-Body בבתים
`Cookie`	בקשה	שולח את ה-cookie השמור אל השרת
`Set-Cookie`	תגובה	השרת מגדיר cookie חדש אצל הלקוח
`Last-Modified`	תגובה	מועד שינוי המשאב — לצורך Caching
`If-Modified-Since`	בקשה	Conditional GET — שלח רק אם השתנה

מבנה URL

מבנה URL מלא

```
http://www.site.com/folder_a/folder_b/file_name.txt
└──┘   └──────┘└──────┘ └──────────────┘└──────────┘
מסלול          דומיין       סכמה (path)
```

כאשר הדפדפן פונה ל-`http://www.site.com/folder_b/file.txt`, השרת מחפש את הקובץ `file.txt` בתיקיית `folder_b` תחת ה-**root directory** שהמתכנת הגדיר (למשל `C:\wwwroot`).

HTTP GET עם פרמטרים

ניתן להעביר מידע לשרת כחלק מה-URL בבקשת GET באמצעות שימוש בתו `?` ואחריו זוגות `param=value`, מופרדים ב-`&`.

דוגמאות — GET עם פרמטרים

```
GET /search?q=application HTTP/1.1          (חיפוש בגוגל)
GET /calculate-next?num=16 HTTP/1.1         (17 ← מחזיר שרת מותאם אישית)
GET /calculate-area?height=3&width=4 HTTP/1.1  (6.0 ← שטח משולש)
GET /maps?saddr=Tel+Aviv&daddr=Jerusalem&dirflg=r HTTP/1.1
```

⚠ מגבלת GET

אורך URL מקסימלי בדפדפנים: ~2,000 תווים. לבן, להעברת קבצים, תמונות, טפסים ארוכים — יש להשתמש ב-**POST**.

HTTP POST

בקשת POST מאפשרת שליחת מידע ב-Body של הבקשה, ללא מגבלת אורך. שימושים עיקריים: העלאת קבצים, שליחת טפסים, פרסום תוכן.

📊 GET vs POST — מתי להשתמש?

מאפיין	GET	POST
מיקום הנתונים	ב-URL	ב-Body
מגבלת גודל	~2000 תווים	ללא מגבלה מעשית
שינוי בשרת?	לא (idempotent)	כן — משנה מצב
שימוש נפוץ	קריאת מידע, חיפוש	העלאת קבצים, טפסים, פרסום
חשיפה בכתובת?	כן (גלוי ב-URL)	לא (בגוף הבקשה)

Cache (מטמון)

מנגנון שמונע הורדה חוזרת של משאבים שלא השתנו, ומאיץ את חוויית הגלישה.

⚙ Conditional GET

1. בקשה ראשונה: הלקוח מוריד משאב (בגון לוגו). השרת מחזיר `Last-Modified: <תאריך>`
2. בקשה הבאה: הלקוח שולח `If-Modified-Since: <תאריך>`
3. אם לא השתנה → השרת מחזיר `304 Not Modified` (ללא body) → הדפדפן משתמש בCache
4. אם השתנה → השרת מחזיר `200 OK` עם המשאב החדש

בדפדפן: Ctrl+F5 = ריענון מאולץ (מתעלם מ-Cache)

Cookies ("עוגיות")

HTTP הוא פרוטוקול **stateless** (חסר-זיכרון) — כל בקשה עצמאית, השרת לא זוכר בקשות קודמות. Cookies פותרות בעיה זו על-ידי שמירת מזהה session.

🍪 מנגנון Cookies — זרימת התהליך

1. לקוח פונה לשרת בפעם הראשונה: `GET /`
2. השרת יוצר מזהה ייחודי ומחזיר: `200 OK`

`Set-Cookie: session_id=24601`

3. הלקוח שומר את ה-cookie על הדיסק
4. בכל בקשה הבאה הלקוח שולח: `GET /addItem cookie: session_id=24601`
5. השרת מזהה את הלקוח לפי ה-cookie ויכול לשמור מצב (סל קניות, התחברות, העדפות)

אורך חיים: לכל cookie נקבע תוקף. לאחר פקיעתו, השרת יוצר מזוהה חדש.

🧭 פרוטוקול DNS

📌 הגדרה — DNS

DNS (Domain Name System) — פרוטוקול שתפקידו העיקרי לתרגם שמות דומיינים (בגון *www.google.com*) לכתובות IP (בגון *173.194.112.148*). ניתן לחשוב עליו כ"ספר טלפונים" של האינטרנט.

שאילתה ותגובה — מבנה פקטת DNS

📩 שאילתת DNS (Query)

- **Transaction ID** — מזהה ייחודי לשאילתה
- **Flags** — 0x0100 = Standard Query
- **Questions: 1**
- **qname** — שם הדומיין הנשאל (www.google.com)
- **qtype = A** — סוג הרשומה המבוקשת
- **qclass = IN** — סוג הרשת

📨 תגובת DNS (Response)

- אותו Transaction ID ← מאפשר התאמה
- **Flags** = Standard Query Response
- **Answer RRs** — מספר רשומות תשובה
- **Name, Type, Class**
- **TTL** — זמן שמירה ב-Cache
- **Data** — כתובת ה-IP

סוגי רשומות DNS (Resource Records)

סוג	שם	תפקיד	דוגמה
A	Address	שם דומיין → כתובת IPv4	www.google.com → 173.194.112.240
PTR	Pointer	כתובת IP → שם דומיין (הפוך)	dns.google → 8.8.8.8
NS	Name Server	שרת DNS אחראי לאזור	google.com → ns1.google.com
MX	Mail Exchange	שרת דואר לדומיין	google.com → aspmx.l.google.com
CNAME	Canonical Name	כינוי (alias) לשם אחר	www.example.com → example.com

היררכיית DNS (Zones)

DNS משתמש במבנה היררכי. התו **.** (נקודה) יוצר את ההיררכיה.

היררכיית שמות דומיין — מעל לתחת

```
. (Root – שורש)
├── com
│   ├── google          → ns1.google.com אחראי
│   │   ├── www         → 173.194.112.240
│   │   └── maps        → 74.125.232.104
│   └── yahoo
├── org
│   └── cyber
│       └── il          → networks.cyber.org.il
└── il
    └── co
        └── google
```

⚡ TTL — Time To Live ברשומת DNS

כל רשומת DNS מכילה **TTL** — משך הזמן שהרשומה תישמר ב-Cache של הלקוח. לאחר שה-TTL פג, הלקוח חייב לשאול שוב את שרת ה-DNS. דוגמה: "TTL = "3 minutes, 30 seconds → הלקוח ישמור את הרשומה 3.5 דקות ורק אחר כך יתשאל מחדש.

הכלי nslookup — תשאול DNS

פקודות nslookup שימושיות

```
nslookup www.google.com           (שאילת רשומה A עבור google)
nslookup -type=A www.google.com   (תשאול מפורש לרשומת A)
nslookup -type=PTR 8.8.8.8        (של הכתובת 8.8.8.8-DNS מה שם ה?)
nslookup -type=NS google.com      (מי שרת ה-NS של google.com?)
nslookup -type=MX gmail.com       (שרת דואר של gmail)
```

תשאול רקורסיבי ידני — ממש ROOT עד לתשובה

```
nslookup maps.google.com <root-server-IP>   (שאלה לשרת ROOT)
→ מחזיר NS של .com

nslookup maps.google.com <com-NS-IP>         (שאלה ל-NS של .com)
→ מחזיר NS של google.com

nslookup maps.google.com <google-NS-IP>      (שאלה ל-NS של google.com)
→ מחזיר כתובת IP של maps.google.com ✓
```

📧 פרוטוקול SMTP (מחקר)

SMTP (Simple Mail Transfer Protocol) — פרוטוקול שימש לשליחת דואר אלקטרוני. בשונה מ-HTTP ו-DNS, ניתן ללמוד עליו בעצמאות על-ידי "אבחון" — ניתוח הסנפות מבלי לעיין בתיעוד.

עקרונות מרכזיים שנלמדו מהאבחון

- SMTP פועל ב-TCP, בדרך כלל על פורט 25 (או 587)
- זהו פרוטוקול **טקסטואלי** (כמו HTTP) — פקודות ותגובות הן מחרוזות
- מבנה: HELO → AUTH → MAIL FROM → RCPT TO → DATA → QUIT
- האימות (authentication) משתמש ב-**Base64 encoding**
- Python: `my_str.encode('base64')` / `.decode('base64')` ב-
- Wireshark-ב: סינון `smtp` ואז Follow TCP Stream לצפייה בפרוטוקול

✓ סיכום פרק 4 — מה צריך לדעת

- **HTTP:** בקשה-תגובה, GET (קריאה), POST (כתיבה/העלאה)
- **Headers:** Host, Content-Type, Content-Length, Cookie, Set-Cookie
- **Status codes:** 200 OK, 302 Redirect, 404 Not Found, 403 Forbidden, 500 Error
- **Cache:** Conditional GET עם If-Modified-Since / Last-Modified
- **Cookies:** Set-Cookie (שרת) → Cookie (לקוח) — ניהול sessions
- **DNS:** שאילתה A = שם→IP, PTR = IP→שם, TTL = תוקף, cache, NS = שרת אחראי
- **SMTP:** טקסטואלי, TCP, Base64, פורט 25/587

פרק 5

Scapy

כלי Python ליצירה, שליחה, הסנפה וניתוח פקטות רשת

📌 מהו Scapy?

Scapy היא ספריית Python חיצונית שמאפשרת **הסנפה, ניתוח, יצירה ושליחה** של פקטות רשת בכל שכבה. בניגוד ל-Sockets (שכבת אפליקציה בלבד), Scapy מאפשרת עבודה גם בשכבות נמוכות יותר (Ethernet, IP, TCP, UDP).

📡 הרצת Scapy

הפעלה + צבעים (Command Line)

```
scapy                          (הפעלה רגילה)
pip install ipython             (לממשק צבעוני)

# שינוי ערכות צבע בתוך Scapy:
conf.color_theme = DefaultTheme()
conf.color_theme = RastaTheme()
conf.color_theme = BrightTheme()
conf.color_theme = ColorOnBlackTheme()
```

Import לשימוש מתוך סקריפט Python

```
from scapy.all import *

# זמינות Scapy עכשיו כל פונקציות:
# sniff, send, IP, TCP, UDP, DNS, Ether, Raw ...
```

🔍 הסנפה — sniff()

⚙ פרמטרים של sniff()

פרמטר	תפקיד	דוגמה
`count`	מספר פקטות לקלוט	`count=10`
`lfilter`	פונקציית סינון — מקבלת פקטה, מחזירה True/False	`lfilter=filter_dns`
`prn`	פונקציה שתרוץ על כל פקטה שעוברת את הסינון	`prn=print_info`
`iface`	ממשק רשת ספציפי	`iface="eth0"`
`timeout`	זמן הסנפה בשניות	`timeout=30`

🔄 זרימת הסנפה עם lfilter ו-prn

פקטה מגיעה → **lfilter(פקטה)**: אם True → prn(פקטה) → נשמרת ב-packets → נזרקת, לא נשמרת כך עד שנאסף מספר ה-count המבוקש. אם False

דוגמה מלאה — הסנפת שאילתות DNS

סינון + הדפסת שאילתות DNS מסוג A

```
from scapy.all import *

# פונקציית סינון: רק שאילתות DNS מסוג A (opcode=0, qtype=1)
def filter_dns(packet):
    return (DNS in packet
            and packet[DNS].opcode == 0     # 0 = QUERY (לא Response)
            and DNSQR in packet
            and packet[DNSQR].qtype == 1)   # 1 = רשומת A

# פונקציית הדפסה: מדפיסה את שם הדומיין
def print_query_name(dns_packet):
    print(dns_packet[DNSQR].qname)

# הסנפה: 10 פקטות, הדפסת שם לכל אחת DNS
sniff(count=10, lfilter=filter_dns, prn=print_query_name)
```

גישה לשדות בפקטה

גישה לשדות — תחביר כללי

```
packet.show()              # הצגת כל הפקטה (כמו Wireshark)
packets.summary()          # סיכום קצר של רשימת פקטות

# גישה לשכבה ספציפית:
packet[DNS]                # שכבת DNS בלבד
packet[IP]                 # שכבת IP
packet[TCP]                # שכבת TCP

# גישה לשדה בתוך שכבה:
packet[DNS].opcode         # 0=QUERY, 1=IQUERY...
packet[DNS].qr             # 0=שאילתה, 1=תגובה
packet[DNSQR].qname        # שם הדומיין בשאילתה
packet[DNSQR].qtype        # 1) סוג הרשומה=A, 28=AAAA...)
packet[IP].src             # כתובת IP מקור
packet[IP].dst             # כתובת IP יעד
packet[TCP].sport          # פורט מקור
packet[TCP].dport          # פורט יעד
str(packet[Raw])           # תוכן גולמי כמחרוזת
```

סינון HTTP — פרוטוקול שScapy לא מכיר

סינון בקשות HTTP GET (Scapy מתייחס ל-HTTP כ-Raw)

```
def http_get_filter(packet):
    return (TCP in packet
            and Raw in packet
            and str(packet[Raw]).startswith('GET'))

# כל פקטת TCP שמתחילה ב-"GET" = בקשת HTTP GET
# ניתן לשנות ל-startswith('POST') לבקשות POST
```

🏗️ יצירת פקטות

יצירת פקטות ושכבות — כל הדוגמאות

```
# יצירת שכבת IP מחדל (ערכי ברירת מחדל בסיסית):
my_packet = IP()

# יצירה עם פרמטרים מפורשים:
my_packet = IP(dst='10.1.1.1', ttl=64)

# שינוי שדה לאחר יצירה:
my_packet.dst = '10.1.1.1'
my_packet.ttl = 5

# שכבות מוערמות עם אופרטור /:
my_packet = IP() / TCP()                        (IP + TCP)
my_packet = Ether() / IP() / TCP()              (Ethernet + IP + TCP)
my_packet = Ether() / IP() / TCP() / Raw("Hello")  (+ מידע גולמי)

# TCP עם פורטים מפורשים:
my_packet = Ether() / IP(ttl=4) / TCP(dport=80)

# HTTP מלא:
pkt = Ether()/IP()/TCP(dport=80)/Raw("GET / HTTP/1.0\r\n\r\n")

# Resolving אוטומטי (Scapy מבצע DNS):
my_packet = IP(dst="www.google.com")   # Scapy יתרגם את הכתובת

# Hexdump (ויצוג הקסדצימלי):
hexdump(my_packet)
```

🧅 עקרון ה-Encapsulation ב-Scapy

האופרטור `/` ממשש **כימוס** (**Encapsulation**):
`IP() / TCP()` → שכבת TCP מוכלת בתוך שכבת IP
כמו שראינו במודל חמש השכבות — כל שכבה עוטפת את השכבה שמעליה.
Scapy גם מציין זאת בסוגריים משולשים מקוננים: `<IP ... | <TCP ...>>`

שדות ש-Scapy ממלא אוטומטית

שדה	ערך אוטומטי	הסבר
`IP.version`	4	IPv4
`IP.ttl`	64	Time To Live ברירת מחדל
`IP.src`	127.0.0.1	כתובת IP מקומית
`IP.proto`	tcp/udp	לפי השכבה שמעל
`Ether.type`	0x0800	IPv4 over Ethernet
`TCP.dport`	http = 80	Scapy מתרגם שם לפורט

📤 שליחת פקטות

שליחת פקטות עם send()

```
my_packet = IP(dst="www.google.com") / Raw("Hello")
send(my_packet)       # 3 שליחה בשכבה (IP)
# פלט: . Sent 1 packets.

sendp(my_packet)      # 2 שליחה בשכבה (Ethernet) - נלמד בהמשך
```

> ⚠ **שתי דרכי שליחה**
>
> **send()** — שולח ברמת שכבה 3 (IP). השתמשו עם `IP()/...`
>
> **sendp()** — שולח ברמת שכבה 2 (Ethernet). השתמשו עם `Ether()/IP()/...`
>
> ב-Windows יש Bug בשליחה על 127.0.0.1 (loopback) — מומלץ להשתמש בשני מחשבים נפרדים.

🔍 Resolving — תרגום שמות

Resolving אוטומטי

```
my_packet = IP(dst = "www.google.com")
# Scapy מבצע DNS lookup ומציב כתובת IP בשדה dst
# בזמן show() — יראה את שם הדומיין
# האמיתית IP-בזמן שליחה — ישתמש בכתובת ה
```

✓ סיכום פרק 5 — API של Scapy

פונקציה / תחביר	שימוש
`sniff(count, lfilter, prn)`	הסנפה עם סינון ועיבוד
`send(pkt)`	שליחה בשכבה 3
`IP() / TCP() / Raw()`	בניית פקטה שכבתית
`pkt.show()`	הצגה מפורטת כמו Wireshark
`pkt[DNS].field`	גישה לשדה בשכבה
`DNS in pkt`	בדיקה האם שכבה קיימת
`hexdump(pkt)`	ייצוג הקסדצימלי
`str(pkt[Raw])`	המרת Raw למחרוזת
`from scapy.all import *`	Import לסקריפט

שכבת התעבורה

פורטים, TCP, UDP — מאפליקציה לאפליקציה

> 📌 **תפקיד שכבת התעבורה**
>
> שכבת התעבורה אחראית להעביר מידע **מתוכנית (תהליך) לתוכנית (תהליך) מרוחקת**. שתי מטרות עיקריות:
>
> 1. **ריבוב אפליקציות** — היכולת להתקשר עם כמה שירותים שונים על אותה כתובת IP בודדה (תמיד קיים)
> 2. **העברה אמינה** — הבטחה שהמידע יגיע שלם ובסדר הנכון (אופציונלי — TCP כן, UDP לא)

🚪 פורטים (Ports)

> 📌 **הגדרה — פורט**
>
> **פורט** הוא מזהה מספרי (0–65535) שמאפשר לשרת להבין לאיזו תוכנה מיועדת הבקשה. האנלוגיה: **אם כתובת IP = מספר הבניין, אז פורט = מספר הדירה.**
>
> - כתובת IP = מזהה הרכיב (מזהה הבניין)
> - פורט = מזהה התוכנה (מזהה הדירה)

פורטים ידועים (Well-Known Ports)

פורט	פרוטוקול	שירות
21 ,20	TCP	FTP (העברת קבצים)
22	TCP	SSH (גישה מרחוק מוצפנת)
25	TCP	SMTP (שליחת דואר)
53	UDP / TCP	DNS
80	TCP	HTTP
110	TCP	POP3 (קבלת דואר)
143	TCP	IMAP (קבלת דואר)
443	TCP	HTTPS (HTTP מוצפן)
8080	TCP	HTTP חלופי (שרתי פיתוח)

טווחי פורטים

טווח	שם	שימוש
0–1023	Well-Known Ports	שירותים סטנדרטיים (HTTP, DNS, FTP...)
1024–49151	Registered Ports	אפליקציות ידועות (MySQL=3306, PostgreSQL=5432)
49152–65535	Dynamic/Ephemeral	פורטי לקוח זמניים (מוקצים אוטומטית)

⚖️ TCP vs UDP

מאפיין	TCP	UDP
מהימנות	✓ מבטיח הגעה ושלמות	✗ אין הבטחת הגעה
סדר	✓ מבטיח סדר נכון	✗ ייתכן שינוי סדר
חיבור	✓ Connection-oriented (handshake)	✗ Connectionless
מהירות	◆ איטי יותר (overhead של אישורים)	✓ מהיר (ללא overhead)
שימושים	HTTP, HTTPS, FTP, SMTP, SSH	DNS, שידור חי, VoIP, משחקים

🔍 הכלי netstat

netstat הוא כלי שורת פקודה להצגת חיבורי רשת פעילים ופורטים שמאוזנים.

פקודות netstat

```
netstat        (חיבורים פעילים בלבד – ESTABLISHED)
netstat -a     (כל החיבורים, כולל LISTENING – ← פתוחים "פורטים")

# פורמט הפלט: Proto | Local Address | Foreign Address | State
# Local Address כ מוצגת: IP:Port
# דוגמה: TCP  0.0.0.0:8820  0.0.0.0:0  LISTENING
```

📊 מצבי חיבור ב-netstat

State	משמעות
`LISTENING`	השרת מחכה לחיבורים נכנסים (הפורט "פתוח")
`ESTABLISHED`	חיבור TCP פעיל ורץ
`TIME_WAIT`	חיבור שנסגר, בהמתנה לסגירה מלאה
`CLOSE_WAIT`	ממתין לסגירת הצד המקומי

שימוש מעשי — מציאת שרת Python שפתחתם

```
# 1. הפעילו את השרת שלכם (מאזין על פורט 8820):
#    python server.py

# 2. בחלון CMD חדש, הריצו:
netstat -a

# 3. חפשו שורה כמו:
#    TCP    0.0.0.0:8820    0.0.0.0:0    LISTENING
#    → השרת מאזין! הפורט פתוח.

# 4. לאחר שלקוח מתחבר:
#    TCP    127.0.0.1:8820  127.0.0.1:49234  ESTABLISHED
#    → חיבור פעיל!
```

💡 כתובת 0.0.0.0 בעמודת Local Address

כתובת `0.0.0.0` בעמודת Local Address משמעותה: השרת מאזין על **כל ממשקי הרשת** — כולל מהאינטרנט. כתובת `127.0.0.1` = מאזין רק לתעבורה מקומית (loopback) מאותו מחשב.

✓ סיכום פרק 6 — מה צריך לדעת

- **פורט** = מזהה תוכנה (1–65535). IP:Port = כתובת מלאה של שירות
- פורטים ידועים: HTTP=80, HTTPS=443, DNS=53, FTP=21, SMTP=25
- **TCP** = מהימן, עם חיבור, איטי יותר — HTTP, SMTP, SSH
- **UDP** = מהיר, ללא חיבור, ללא הבטחת הגעה — DNS, VoIP, שידור
- **netstat -a** = הצגת כל הפורטים הפתוחים (LISTENING) + חיבורים פעילים
- LISTENING = השרת מחכה; ESTABLISHED = חיבור פעיל; TIME_WAIT = בסגירה

נספח

קוד: שרת HTTP בסיסי ב-Python

ביצד למטש שרת HTTP שמגיש קבצים — מתרגיל 4.4

מה עושה שרת HTTP?

שרת HTTP מאזין על פורט מסוים (בדרך כלל 80), מקבל בקשות GET מלקוחות (דפדפנים), ומגיש להם קבצים מהתיקייה שהוגדרה כ-root directory.

שלד שרת HTTP — Python

server.py — שרת HTTP מלא

```
import socket
import os

# קבועים
SERVER_IP   = '0.0.0.0'     # 0.0.0.0 = מאזין לכל ממשק
SERVER_PORT = 80
SOCKET_TIMEOUT = 2
WEBROOT = 'C:\\webroot'     # תיקיית ה broot

# מיפוי סיומות לסוגי MIME
CONTENT_TYPES = {
    '.html': 'text/html; charset=utf-8',
    '.txt':  'text/html; charset=utf-8',
    '.jpg':  'image/jpeg',
    '.css':  'text/css',
    '.js':   'text/javascript; charset=UTF-8',
    '.ico':  'image/x-icon',
}

def get_file_content(file_path):
    """קורא קובץ ומחזיר תוכן בינארי"""
    with open(file_path, 'rb') as f:
        return f.read()

def get_content_type(file_path):
    """מחזיר Content-Type לפי סיומת קובץ"""
    ext = os.path.splitext(file_path)[1].lower()
    return CONTENT_TYPES.get(ext, 'application/octet-stream')

def build_http_response(status_code, body=b'', content_type='text/html'):
    """תקינה בונה תגובת HTTP"""
    status_messages = {
        200: 'OK', 302: 'Moved Temporarily',
        403: 'Forbidden', 404: 'Not Found', 500: 'Internal Server Error'
    }
    status_text = status_messages.get(status_code, 'Unknown')

    # שורת תגובה + Headers
    response  = f'HTTP/1.0 {status_code} {status_text}\r\n'
    response += f'Content-Length: {len(body)}\r\n'
    response += f'Content-Type: {content_type}\r\n'
    response += '\r\n'  # סוף = שורה ריקה Headers
```

```
    return response.encode() + body  # Headers + Body

def handle_client(client_socket):
    """מטפל בחיבור לקוח בודד"""
    client_socket.settimeout(SOCKET_TIMEOUT)
    try:
        # קבלת הבקשה מהלקוח
        request = client_socket.recv(4096).decode()

        # פרסור שורת הבקשה: GET /path HTTP/1.1
        first_line = request.split('\r\n')[0]
        parts = first_line.split(' ')
        if len(parts) < 2 or parts[0] != 'GET':
            client_socket.close()
            return

        url = parts[1]

        # / → index.html
        if url == '/':
            url = '/index.html'

        # הסרת פרמטרים אם יש: /page?param=val → /page
        if '?' in url:
            url = url.split('?')[0]

        # בניית נתיב קובץ מלא
        file_path = WEBROOT + url.replace('/', '\\')

        if not os.path.isfile(file_path):
            # 404 → קובץ לא קיים
            response = build_http_response(404, b'File Not Found')
        else:
            # קריאת הקובץ ושליחה
            body = get_file_content(file_path)
            ctype = get_content_type(file_path)
            response = build_http_response(200, body, ctype)

        client_socket.send(response)

    except Exception as e:
        print(f"Error: {e}")
    finally:
        client_socket.close()

def main():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind((SERVER_IP, SERVER_PORT))
    server_socket.listen(5)
    print(f'Server listening on {SERVER_IP}:{SERVER_PORT}')

    while True:
        client_socket, client_address = server_socket.accept()
        print(f'New connection from {client_address}')
        handle_client(client_socket)

if __name__ == '__main__':
    main()
```

הדפדפן ככלי ניפוי (Debug)

בדפדפן Chrome, לחצו **F12** כדי לראות: **Network** ←

- כל הבקשות שנשלחו ואת ה-Headers שלהן
- קוד התגובה (Status Code)
- Content-Type שהתקבל
- גודל הקובץ וזמן הטעינה

קוד: הסנפת DNS עם Scapy

dns_sniffer.py — שומר כל שאילתות DNS לקובץ

```
from scapy.all import *

dns_log = []

def filter_dns(packet):
    """מסנן שאילתות DNS מסוג A בלבד"""
    return (DNS in packet
            and packet[DNS].opcode == 0
            and DNSQR in packet
            and packet[DNSQR].qtype == 1)

def log_dns_query(packet):
    """שומר את שם הדומיין לרשימה ומדפיס"""
    domain = packet[DNSQR].qname
    src_ip = packet[IP].src if IP in packet else 'unknown'
    print(f'[DNS Query] {src_ip} → {domain}')
    dns_log.append(f'{src_ip},{domain}')

print('Starting DNS sniffer... Press Ctrl+C to stop')
try:
    sniff(lfilter=filter_dns, prn=log_dns_query)
except KeyboardInterrupt:
    print(f'\nCaptured {len(dns_log)} DNS queries')
    with open('dns_log.csv', 'w') as f:
        f.write('Source IP,Domain\n')
        for entry in dns_log:
            f.write(entry + '\n')
    print('Saved to dns_log.csv')
```

🧿 סיכום כולל — קובץ 2 מתוך 4

פרק	נושאים מרכזיים
פרק 3 Wireshark	4 חלונות, פילטרים (http/dns/icmp/frame), Follow TCP Stream, Statistics
פרק 4 שכבת אפליקציה	HTTP (GET/POST/Headers/Status), URL, Cache, Cookies, DNS (A/PTR/NS/TTL), SMTP
פרק 5 Scapy	sniff(count/lfilter/prn), IP()/TCP()/Ether()/Raw(), send(), show(), גישה לשדות
פרק 6 שכבת התעבורה	פורטים (IP:Port), Well-Known Ports, TCP vs UDP, netstat -a