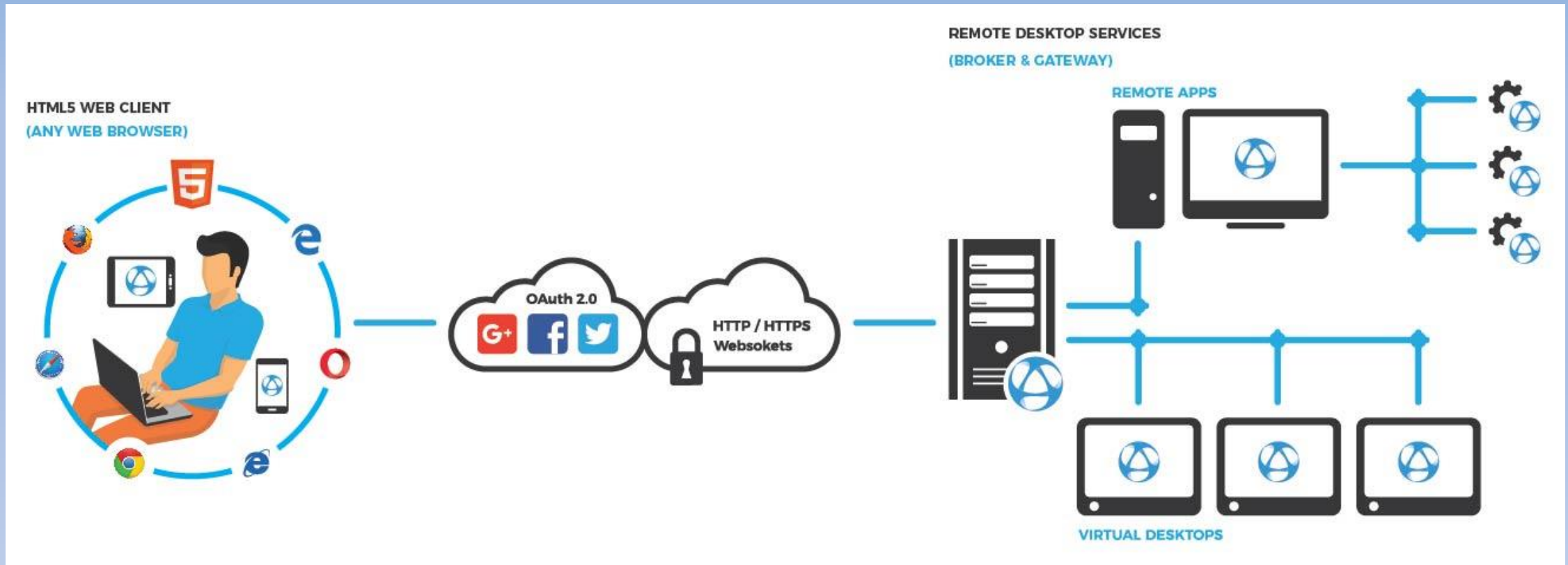


RDS SECURITY SYSTEM

Design by: Avi Feder & Daniel Yochanan



אז מהו RDS?

• RDS – (Remote Desktop Service)

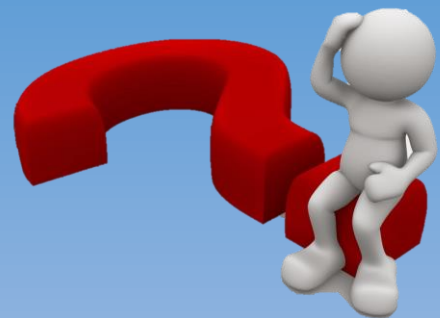
RDS, הוא אחד הרכיבים של Microsoft Windows המאפשרים למשתמש להשתלט על מחשב מרוחק או מכונה וירטואלית על גבי הרשת ולקבל ממשק משתמש מלא ואישי של Microsoft Windows.

לדוגמא, שירותי התחברות מרחוק של מכון לב הינו RDS, הנותן לכל סטודנט או מרצה במכון חשבון משתמש אישי משלו.



יתרונות לשימוש בRDS

- אחד היתרונות הבולטים לשימוש בRDS הוא שניתן להתקין תוכנה לכל המשתמשים בבת אחת. ולתחזק את התוכנה אצל כולם בקלות. מאשר שכל משתמש יתקין את התוכנה אצלו.
- יתרון נוסף לשימוש בRDS הוא יתרון אבטחתי. כל המידע שעליו עובדים המשתמשים נשמר בשרתים המרוחקים ומאובטח בהם. במקום שכל משתמש ישמור את המידע אצלו והאבטחה תהיה תלויה במשתמש עצמו.



הבעיה

במהלך חיפושינו לרעיון לפרויקט בדקנו ושאלנו אנשים מהתעשייה על בעיות אבטחתיות שצצו להם. אחד מהם הציג בפנינו את הבעיה הבאה: בשירותי RDS, כל משתמש מקבל שם משתמש וסיסמא ויכול להתחבר לשרתים המרוחקים מכל מכשיר שהוא רוצה. עובדה זו היא בעיה אבטחתית מכיוון שברגע שתוקף משיג שם משתמש וסיסמא של משתמש כלשהו, יש לו גישה מלאה לשרתים המרוחקים!

למשל, האדם שהציג בפנינו את הבעיה דיבר על קופאיות בסופר שהמסוף שלהם מחובר לשרת RDS. ומשאירות את שם המשתמש והסיסמא שלהן על השולחן ליד הקופה וכל לקוח של הסופר יכול לקחת את הפרטים שלה ולהתחבר למערכת.

הפתרון הכללי לבעיה זו

נרצה להקים מערכת RDS שניתן להתחבר אליה רק דרך מחשבים ספציפיים שהוגדרו מראש כמחשבים מורשים.

ככה, גם במקרה ומישהו יגנוב את שם המשתמש והסיסמא, הוא לא יוכל להתחבר למערכת דרך מחשבים אחרים.



הפתרונות הקיימים בשוק כיום:

ישנם מספר פתרונות לבעיה זו אך לכל פתרון קיים חסרון מהותי שאנו באים לפתור.

• RD GATEWAY SERVER

RD GATEWAY זוהי מערכת נוספת שמקימים בצמוד לRDS.

על כל מחשב לקוח שרוצים שיוכל להתחבר למערכת מתקינים תעודת אבטחה. ברגע ההתחברות למערכת RDS, הRD GATEWAY מזהה את תעודת האבטחה ומאפשר למשתמש להתחבר למחשב המרוחק.

החיסרון בפתרון זה הוא שיש צורך בהקמת מערכת נוספת הכוללת שרת נוסף ותוכנות ייחודיות של Microsoft בעלות גבוהה.

פתרון נוסף הקיים בשוק כיום:

• SSH TUNNELING

ניתן למנוע חיבור ממחשבים לא מורשים ע"י כך שנדרוש אימות SSH מכל מחשב שמנסה להתחבר לRDS.

על כל מחשב לקוח שרוצים שיוכל להתחבר למערכת מתקינים תעודת אבטחה. על המשתמש שרוצה להתחבר לRDS להפעיל את תוכנת הSSH ולהתחבר גם אליה עם שם משתמש וסיסמא דרך הTerminal. וככה נוצר חיבור מאובטח עם הRDS.

החיסרון בפתרון זה הוא שיש צורך בידע טכני רב מצד הלקוח על מנת להפעיל את הSSH ולהתחבר אליו דרך הTerminal.

חיסרון נוסף הוא שיש צורך להחזיק שם משתמש וסיסמא נוספים לחיבור SSH.

הפתרון שלנו



הפתרון שלנו הוא פשוט ונוח שמאבטח את החיבור ללא צורך בהתערבות משתמש הקצה ובידע טכני נוסף מצידו.

הדרישה היחידה מצד הלקוח הוא שהסקריפט שכתבנו ירוץ ברקע תמיד, ללא התערבות מצידו.

הפעלת הסקריפט תתבצע בקינפוג הראשוני של מחשב הלקוח כך שיופעל אוטומטית עם הפעלת המחשב.

אז... איך הפתרון שלנו עובד?

דבר ראשון, הגישה היחידה לשרתי RDS תיהיה דרך סקריפט שיישב בנתב של RDS. סקריפט זה מעביר לRDS אך ורק הודעות שעוברות תהליך אישור. תהליך האישור עובד כך:

- עבור כל חבילה שיוצאת מהלקוח לשרת RDS נשלח באמצעות סקריפט הלקוח הודעה שמאשרת את החבילה.
- ברגע שהסקריפט שבנתב מזהה כמות מסויימת של חבילות שלא אושרו הוא חוסם את התקשורת עם משתמש קצה זה.
- כל עוד הסקריפט בנתב מקבל אישורים על הודעות שנשלחו יש גלישה מלאה לאותו משתמש קצה.

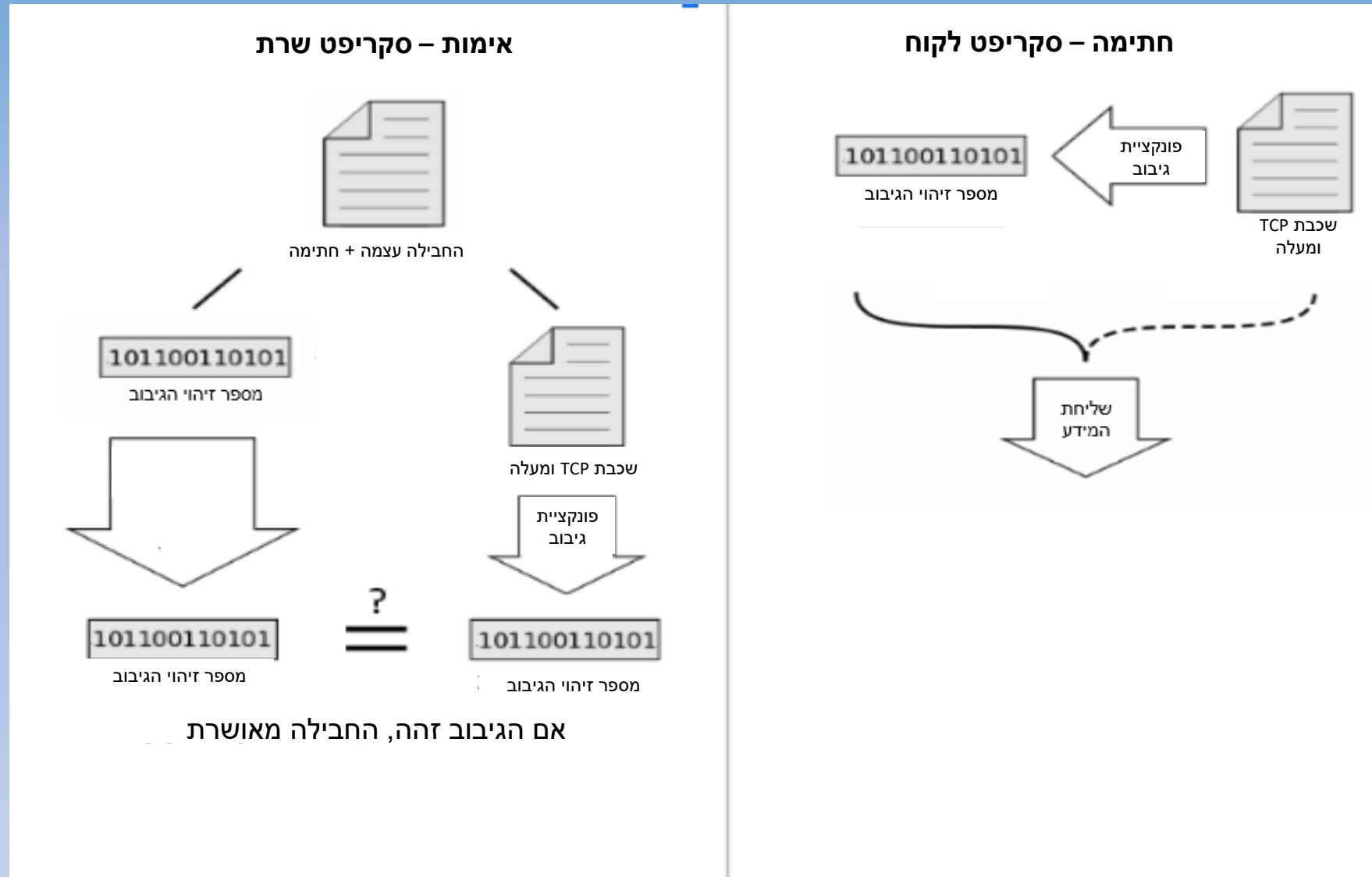
תהליך אישור החבילה

הסקריפט של הנתב לוקח מכל חבילה שמיועדת לשרת את שכבת ה-TCP שלה ומעלה (כולל ה-load) ומעביר מידע זה בפונקציית HASH256.

במקביל, הסקריפט בצד הלקוח גם עושה את אותו התהליך, כלומר, לוקח מכל חבילה את שכבת ה-TCP שלה ומעלה (כולל ה-load) ומעביר מידע זה בפונקציית HASH256 ושולח זאת לסקריפט של הנתב בפורט מסויים.

הסקריפט של הנתב מקבל חבילות אישור אלו ומשווה את ה-hash שהוא מקבל ל-hash שהוא חישב, בדומה לחתימה דיגיטלית. ברגע שיש יותר מרף מסוים של כמות הודעות שלא עוברות אישור זה, הנתב חוסם את התקשורת עבור לקוח זה.

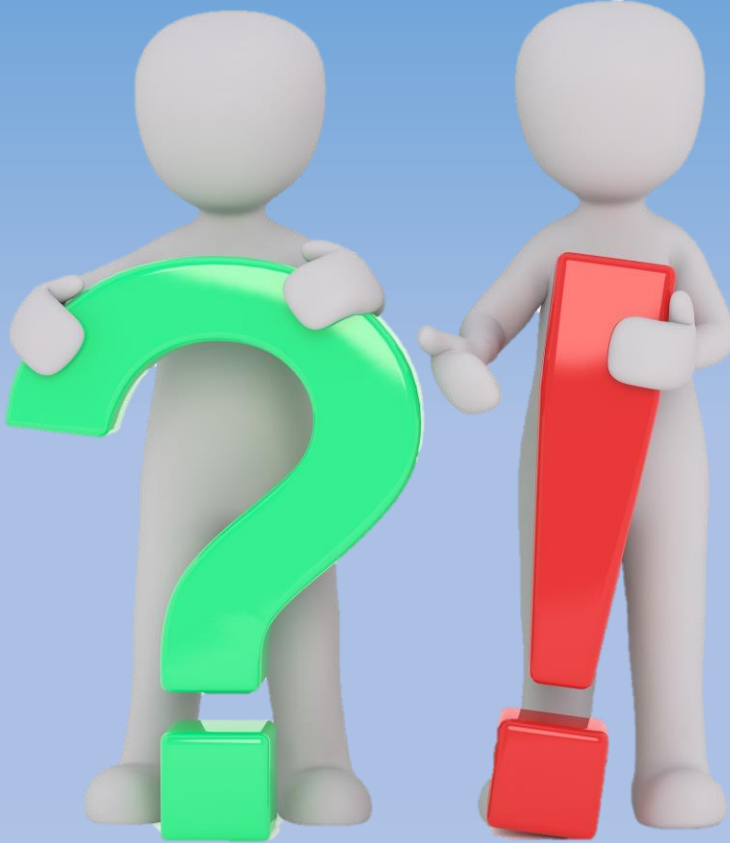
תהליך האישור יראה ככה:



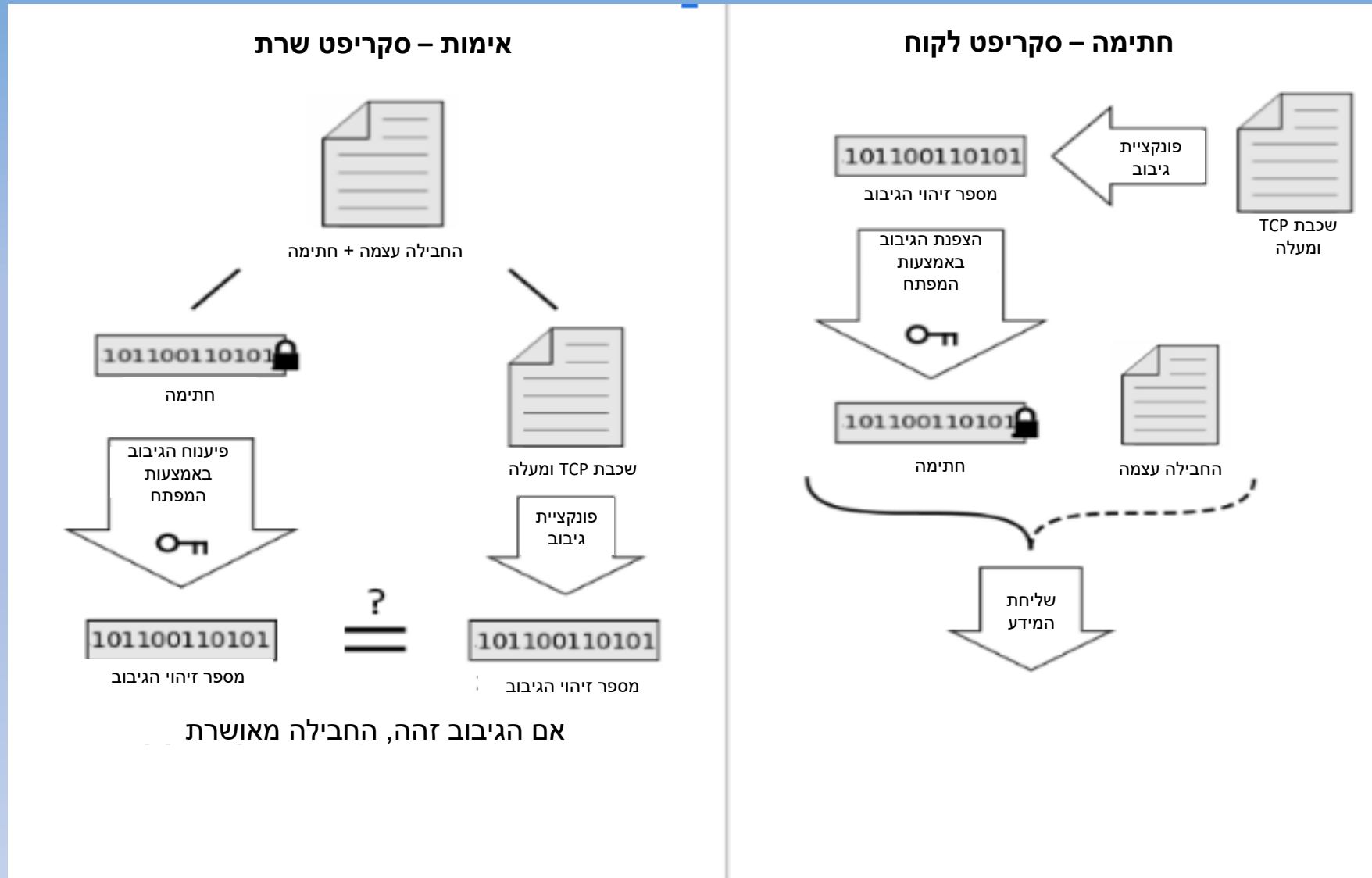
בעיית אבטחה ופתרונה

שאלה מתבקשת היא, איך נמנע מתוקף שמכיר את האלגוריתם שלנו מלשלוח גם כן לסקריפט של הנתב את חבילות האישור (hash על שכבת ה-TCP ומעלה) ובכך הנתב עלול לאשר אותו ולתת לו גישה מלאה לשרת?

אין עם זה שום בעיה אצלנו! מכיוון שכל התעבורה בין הסקריפטים שלנו מתבצעת בצורה מאובטחת מקצה לקצה. בסקריפט של הנתב ישנו מפתח ציבורי המתאים למפתח פרטי (הצפנה אסימטרית) שיושב אצל כל הלקוחות. באמצעות צמד מפתחות אלו אנו מעבירים לכל לקוח מפתח סימטרי. וכך אין לתוקף אפשרות לשלוח הודעות אישור לסקריפט של הנתב ולעבור לשרתי ה-RDS שלנו מכיוון שאין לו איך לקבל מפתח סימטרי מהנתב ללא המפתח הפרטי של הלקוחות.



כעת תהליך האישור יראה ככה:



פונקציות עיקריות בקוד שלנו



- Key Exchange •
- Insert Data •
- Send Hash •
- Remove Data •
- Black List •

Key Exchange - צד שרת

באמצעות סוקטים, השרת מאזין לפורט מסויים באופן קבוע. כל לקוח שרוצה לנהל שיחה עם שרת הRDS, פונה לסקריפט לקבל מפתח עם הודעת "Hello". ברגע זה השרת יוצר ללקוח מפתח סימטרי, מצפין אותו באמצעות המפתח הא-סימטרי ושולח אותו ללקוח. השרת שומר את המפתח הסימטרי שיצר בסמיכות לIP, PORT של אותו הלקוח. בנוסף, הלקוח מחדש את המפתח שלו בכל x זמן שהוגדר מראש על מנת למקסם את האבטחה.

```
with socket.socket() as sock:
    try:
        # if the port number already taken, the following line will not work
        sock.bind((GW_ADDRESS, KEY_EXCHANGE_PORT))
        print("success in binding")
    except:
        print("error in binding")
        sys.exit()
sock.listen(0)
while True:
    client_socket, client_address = sock.accept()
    valid, data = get_msg(client_socket)
    if valid:
        data = data.split(" ")
        if data[0] == HELLO_MASSEGE:
            if (client_address[0], int(data[1])) not in key_dic:
                key = int.from_bytes(Fernet.generate_key(), "big")
            else:
                key = key_dic[(client_address[0], int(data[1]))]
            client_socket.send(create_msg(get_key(key)).encode())
            key_dic[(client_address[0], int(data[1]))] = key
            print(key_dic)
```

Insert Data - צד שרת

פונקציה זו מקבלת כל חבילה שנשלחת לשרת הRDS, לוקחת ממנה את שכבת הTCP ומעלה, מעבירה אותה בפונקציית HASH256 ומכניסה את הפלט למאגר הנתונים השייך ללקוח זה (לפי IP, PORT). כל זאת לצורך אימות ההודעות כפי שהסברנו לעיל. פונקציה זאת עובדת במקביל בעוד כמה תהליכים ולכן היה חשוב השימוש בMutex (מנעול לקטע קריטי). לולא המנעול היה איבוד ודריסה של מידע במאגר.

```
try:
    if IP in p and p[IP].dst == RD_ADDRESS and p[Ether].src != GW_MAC_ADDRESS and p[
        Ether].dst == GW_MAC_ADDRESS and TCP in p:
        hash = hashlib.sha256(bytes(p[TCP])).hexdigest()
        key = (p[IP].src, p[TCP].sport)
        mutex.acquire()
        if key in checkIP_dict and hash not in checkIP_dict[key]:
            checkIP_dict[key] = checkIP_dict[key] + [hash]
        elif key not in checkIP_dict:
            checkIP_dict[key] = [hash]
        mutex.release()
        print(len(checkIP_dict[key]), key, "insert")
except Exception as e:
    print(e, "error in insertData")
finally:
    if mutex.locked():
        mutex.release()
    sys.exit()
```


Send Hash - צד לקוח

פונקציה זו מבצעת HASH על שכבת ה-TCP ומעלה של הודעות הנשלחות לשרת, מצפינה את הפלט באמצעות המפתח הסימטרי שקיבלה מהשרת (במידה ואין לה מפתח סימטרי היא מבצעת key Exchange עם השרת) ושולחת הודעת אישור זו לסקריפט של הנתב לפורט אליו הוא מאזין – כפי שנראה בשקופית הבאה.

```
try:
    if IP in p and p[IP].dst == RD_IP and p[IP].src == MY_IP and TCP in p and p[Ether].src == MY_MAC:
        if p[TCP].sport not in key_dict.keys():
            mutex.acquire()
            if p[TCP].sport not in key_dict.keys():
                keyExchangeClient(p[TCP].sport)
            if mutex.locked():
                mutex.release()
        f = Fernet(key_dict[p[TCP].sport].to_bytes(64, byteorder="big"))
        hash = str(hashlib.sha256(bytes(p[TCP])).hexdigest())
        encrypted_message = f.encrypt(hash.encode())
        pToSend = Ether(dst=DEFAULT_GW_MAC) / IP(version=4, src=MY_IP, dst=GW_IP) \
            / UDP(sport=p[TCP].sport, dport=CLIENT_PORT) / Raw(load=encrypted_message)
        pToSend.show2()
        sleep(0.2)
        sendp(pToSend, iface=IFACE, verbose=0)
except Exception as e:
    print(e, "error in sendHash")
finally:
    if mutex.locked():
        mutex.release()
    sys.exit()
```

Remove Data - צד שרת

הפונקציה הראשית מסניפה את חבילות האישור (חבילות הHASH של הלקוח – כפי שראינו בשקופית הקודמת) בפורט מסויים, מפענחת את ההצפנה שלהן באמצעות המפתח הסימטרי שהוצמד לאותו הלקוח ומעבירה את התוצאה לפונקציה Remove Data. פונקציה זו מסירה את הHASH ממאגר הנתונים של אותו הלקוח. כמובן שגם פה נשתמש בMutex על מנת למנוע דריסה או איבוד של נתונים.

```
try:
    if IP in p and p[IP].dst == GW_ADDRESS and UDP in p and p[
        UDP].dport == CLIENT_PORT:
        key = (p[IP].src, p[UDP].sport)
        hash = p.load
        f = Fernet(key_dic[key].to_bytes(64, byteorder="big"))
        decrypted_message = f.decrypt(hash).decode()
        removeData(decrypted_message, key)
        removeCounter(key)
except:
    print("error in decrypt")
finally:
    sys.exit()
```

```
def removeData(hash, key):
    try:
        mutex.acquire()
        if key in checkIP_dict and hash in checkIP_dict[key]:
            checkIP_dict[key].remove(hash)
        mutex.release()
        print(len(checkIP_dict[key]), key, "remove")
    except Exception as e:
        print(e, "error in removeData")
    finally:
        if mutex.locked():
            mutex.release()
```

18

Black List - צד שרת

פונקציה זו רצה באופן קבוע, ובודקת האם ישנו לקוח אשר לא קיבל אישור על מכסה מסויימת של חבילות. ברגע שהיא מוצאת לקוח כזה היא מכניסה אותו לרשימה השחורה ובעצם מונעת ממנו לתקשר עם שרת הRDS, פונקציה זו מוסיפה בצמוד ללקוח ברשימה השחורה גם את תאריך החסימה שלו על מנת לשחרר אותו כעבור זמן מסויים שנקבע מראש.

```
try:
    while True:
        sleep(1)
        mutex.acquire()
        checkIP_dict_copy = dict(checkIP_dict)
        mutex.release()
        for ipList in checkIP_dict_copy:
            if len(checkIP_dict_copy[ipList]) > PACKETS_TO_BLOCK:
                black_list_mutex.acquire()
                if ipList[0] not in black_list:
                    mutex.acquire()
                    del checkIP_dict[ipList]
                    mutex.release()
                    black_list[ipList[0]] = datetime.now()
                black_list_mutex.release()
        print(black_list)
except Exception as e:
    print(e, "error in black list ip")
```

לסיום

כעת נריץ את הקוד.

לשם ההדגמה נשתמש בשני מחשבים שונים, אחד המריץ את סקריפט הלקוח ולכן יצליח לגשת לRDS. לעומת זאת מחשב שני לא יריץ את סקריפט הלקוח ולכן לא יצליח להתחבר לRDS.

על מנת לדמות את הנתב בגישה לRDS של מכון לב, הקמנו מכונה וירטואלית נוספת שתשמש כביכול כנתב בכניסה למערכת של מכון לב עבור אותם מחשבים.

