Virtual Private Network

שרת חכם בשביל מערכת VPN

מגיש: נוראל גליק

פרטים מזהים

תאריך הגשה: 29/6/2022

שם העבודה: Virtual Private Network - שרת חכם בשביל מערכת VPN

חלופה: הגנת סייבר ומערכות הפעלה

פרטי התלמיד

שם התלמיד: נוראל גליק

תעודת זהות: 336266499

תאריך לידה: 22/05/2003

טלפון: 054-911-6102

כתובת מייל: norelglick21@gmail.com

פרטי בית ספר

שם בית הספר: הכפר הירוק ע"ש לוי אשכול

סמל מוסד: 580019

כתובת: הכפר הירוק 27800 רמת השרון

טלפון: 03-645-5621

פרטי המנחה

שם המנחה: יהודה אור

תעודת זהות: 023098007

טלפון: 050-734-4457

כתובת מייל: yoooda@gmail.com

תוכן עניינים

מבוא

בימינו יש הרבה סיבות לרצות להתחבר לרשת רחוקה. בשביל זה יצרו את מערכת הVPN. בעבודה שלי אסביר על המערכת הזו, עם פוקוס גדול מאוד על השרת.

מה זה VPN? תקציר

VPN, או שמו בעברית "רשת פרטית וירטואלית", הוא מערכת שרת-לקוח שבעזרתו אפשר להעביר מידע פרטי מרשת ארגונית למחשב שנמצא מחוץ לרשת הזו, והפוך. למה צריך את זה? קיים שני סיבות:

1) אם לארגון קיים מספר סניפים ברחבי העולם, ה-cost effectiveness של לסלול כבלי רשת פרטיים בין כל הסניפים היא לא הגיונית בכלל. לכן מעבירים את המידע בעזרת הVPN.
2) אם הארגון רוצה להעביר מידע מרשת הארגון לרשת ציבורית, לדוגמא, אם מישהו עובד מהבית ורוצה להתחבר לרשת הארגון, ה-cost effectiveness של לסלול כבלי רשת לכל בית עובד הוא לא הגיוני, ולכן אפשר להשתמש בVPN כדי לעשות את אותו עבודה.

בנוסף היום גם אנשים פרטיים (שלא קשורים לארגון) משתמשים בVPN ציבורי כדי להסוות את המידע שלהם או להתחבר לרשת במדינה אחרת.

הסבר קצר על העבודה

העבודה תהיה מחולקת לשני חלקים - שרת ולקוח. העבודה שלי מתמקדת ברשת, ואילו העבודה של טל ברילנט מתמקדת בלקוח. בעבודה שלי אני מתמקד בעיקר על השרת של מערכת הVPN, על אותנטיקציית RSA, ועל אלגוריתם החלפת מפתחות Diffie Hellman. בעבודה של טל הוא מתמקד על הצפנות ועל ממשק משתמש.

בנוסף, אסביר ואכתוב קוד על:

- איך הפקטות עוברות במערכת, מהמחשב, לשרת, ובחזרה.
- הפונקציות שהשתמשתי, ולמה הם חשובים.
- בעיות אבטחה וbottlenecks הקיימות במערכת ואיך אפשר לפתור אותם.

תהיה הרצה מודרכת למערכת ואראה בדיוק מה קורה בכל שלב. בנוסף יהיה מדריך למשתמש במקרה שאתם, הקוראים, ירצו להריץ את השרת.

תיאור מפורט של המערכת

המערכת בנוי ממספר שלבים:

1) הפעלה של השרת וקבלה של לקוח

צעד זה הוא יחסית פשוט. בהתחלה השרת עושה אינישליזציה למספר פרמטרים חשובים כגון הIP וPort שניתן להתחבר אליו, הערכים הפומביים של פונקציית Diffie Hellman ועוד. אחרי זה היא מחכה להתחברות של הלקוח.

2) קבלת סוג הצפנה

השרת מחקה שהלקוח ישלח לו את סוג ההצפנה. יש שני סוגים, אחד "חזק" ואחד "חלש".

3) אותנטיקציית RSA

השרת חייב להוכיח שהוא באמת השרת האמיתי ושהוא לא מתחזה. בשביל זה נשתמש באלגוריתם הנקרא RSA Authentication. בעזרתו, הלקוח יכול לאשר שהוא לא התחבר למישהו רשע שרוצה לגנוב ממנו מידע.
השרת מכין שני מפתחות שולח אחד מהם ללקוח, ואז מכין חתימה דיגיטלית ממשפט סודי שידועה רק לשרת וללקוח ולא עוברת ברשת, שולחת את החתימה הזאת ללקוח. השרת מחכה לאישור מהלקוח ועוברים לשלב הבא.

4) החלפת מפתחות Diffie Hellman

השרת והלקוח משתמשים באלגוריתם של Diffie Hellman כדי לקבל מפתח הצפנה שבעזרתו שניהם יצפינו את המידע שלהם. אלגוריתם זה חכם מאוד ובנוי כך שהשרת והלקוח הם היחידים שיודעים את המפתח הסודי להצפנה.
השרת עושה חישוב בעזרת הפרמטרים הפומביים (הpublic key וpublic modulus) ועז מחכה לקבל את החישוב של הלקוח. השרת שולח את החישוב שלו ללקוח ואז מחשב את המפתח המשותף שבעזרתו הוא יפענח ויצפין את המידע שמקבל ושולח ללקוח.

5) קבלת מידע מהלקוח ושליחה לעולם

השרת מחכה לקבלת מידע מהלקוח, מפענח את ההצפנה ומכין ממנו פקטה של scapy, משנה בו ערכים כמו הsource mac, destination mac וsource ip ושולח את הפקטה לראוטר שלו (ועז לעולם).

6) קבלת פקטה מהעולם ושליחה ללקוח

השרת מחכה ולקבל פקטה מהעולם, מצפין אותו ושולח אותו לשרת.

ניהול סיכונים

מהירות אינטרנט מוגבלת

מהירות האינטרנט של הלקוח היא פונקציה של מורכבות ההצפנה (המהירות שלה) כתלות בכמות התהליכונים (Threads) שרצים בשרת כדי לקבל ולשלוח מידע.

$$mb/s = thread\ amount * cryptographic\ function\ speed$$

הפתרון שלנו לבעיה הוא לתת ללקוח לבחור איזה סוג הצפנה הוא רוצה להשתמש: הצפנה חזקה אך איטית או הצפנה חלשה אך מהירה. בסוף לא השתמשנו בפתרון זה מכיוון שהצפנת הXOR שרצינו להכניס הוא לא בטוח בכלל.

תקיפת Man In The Middle

תקיפת MITM הוא סוג תקיפת סייבר בה התוקף גורם לשרת לחשוב שהוא הקליינט ולקליינט לחשוב שהוא השרת. כל מה שהוא עושה זה לקרוא ולהעביר את המידע שהוא מקבל לגוף השני וכך אין פרטיות ללקוח.

הפתרון שלנו לבעיה הוא לשלב בנוסף לאותנטיקציית RSA גם אלגוריתם של החלפת מפתחות Diffie Hellman. בגלל השילוב של האלגוריתמים האלו התוקף לא יכול להתערב באף שלב של הדיבור בין הלקוח לשרת ובסוף יהיה רק לשרת וללקוח את המפתח המשותף. בסוף השתמשנו בפתרון זה

תכנון לו"ז זמנים

- חקירה על מערכת הVPN
- תכנון השרת עם שיתוף טל ברילנט (אחראי הלקוח)
- כתיבת השרת
- כתיבת הספר

בדיקות

בתוכנה יהיה מספר בדיקות שירוצו במהלך הריצה:

- הדפסת הודעות לאחר שלבים ספציפיים כדי שיהיה אפשר להבין מתי התוכנית קרסה
- Exception handling כדי שהתוכנה לא תקרוס אחרי שהקליינט מפסיק את החיבור בפתאומיות
- Exception handling בשילוב עם זיהוי סוגי פקטות כדי לעשות אופרציות על הפקטות הנכונות כדי לא להרוס את כל הפקטות

יכולות של המערכת

השרת הVPN הוא מה שמחבר בין הלקוח לשאר האינטרנט והעולם. השרת אחראי על מספר דברים:

- להתחבר ולעשות אותנטיקציה עם הלקוח
- לקבל את הפקטות שמגיעות מהלקוח ולפענח אותן
- לשלוח את הפקטות לאינטרנט
- לקבל את הפקטות בחזרה מהאינטרנט
- להצפין אותן ולשלוח אותן ללקוח

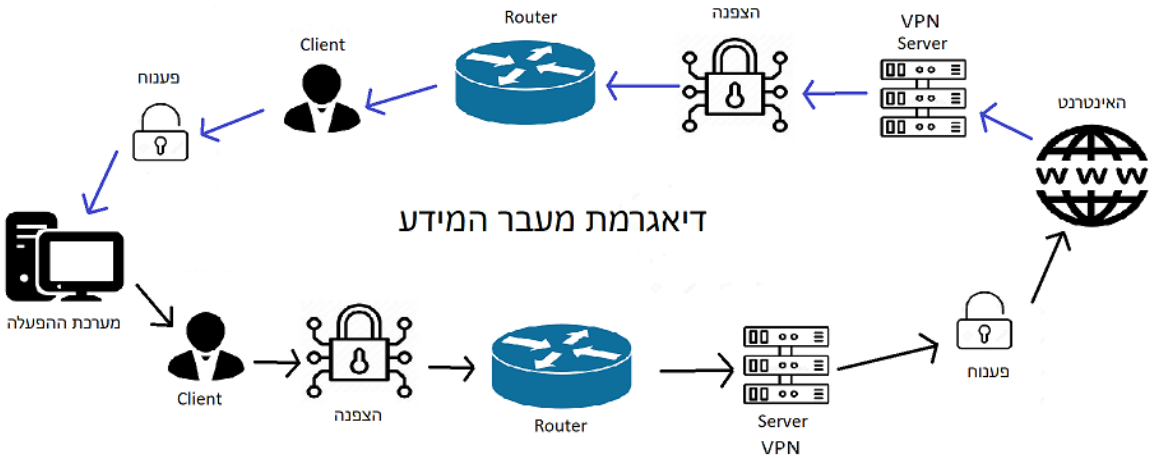
תחום הפרויקט וסוג הטכנולוגיה בו מתעסק

הפרויקט מוגדר תחת התחום "סייבר ומערכות הפעלה" ומתעסק בתחומי מחשב כמו רשתות, מערכת הפעלה, הצפנה, אבטחת מידע, ואותנטיקציה.

התכנולוגיות בו מוצגות בפרויקט הן חדשות יחסית אבל לא בלתי מוכרות.

חלק תיאורטי

מערכת VPN



בעזרת הממשקי רשת של מערכת ההפעלה אפשר לשלוח מידע על הכבל רשת המחובר לכרטיס רשת של המחשב. כרגע במחשב, טאבלט או טלפון שאתם משתמשים כדי לקרוא את זה, יש או כרטיס רשת או כרטיס ווי-פי (או שניהם) ובעזרתו אתם מחוברים לרשת הפרטית/ציבורית שלכם. לרשת שלכם יש נתב (router) ובעזרתו אתם יכולים לגלוש ברשת.

ההגיון מאחורי מערכת VPN יחסית פשוטה: כל פקטה שהמחשב רוצה לשלוח לרשת צריך לעבור הצפנה מלאה ואז אנקפסולציה (encapsulation) ואז להישלח לשרת VPN. השרת מפענח את ההצפנה הזאת ושולחת אותה לWWW (האינטרנט). השרת מקבל תשובה מהWWW, מצפין את התשובה הזאת ושולח אותה ללקוח. הלקוח מפענח את ההצפנה הזאת וקוראת את המידע.

בעצם מה שקורה כאן זה שהשרת VPN שולח בשביל הלקוח את כל הפקטות לעולם במקום שהלקוח ישלח בעצמו, וכך הוא "וורטואלית" מחובר לרשת שבו השרת ממוקמת.

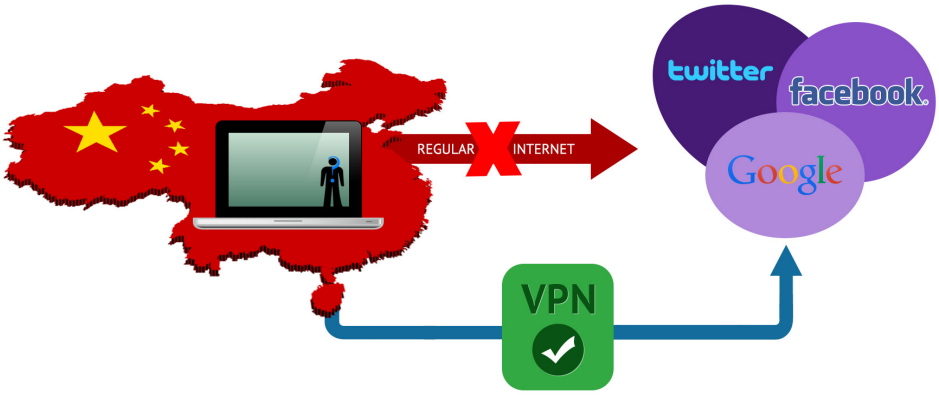
יתרונות של המערכת

1) התחברות לרשת שהיא לא של הלקוח

נניח שאני גר בישראל ואני עובד בחברה שהמשרדים שלה בארצות הברית. במשרדים שלהם יש את האתר אינטרנט שאני צריך לעבוד עליו, הוא מותקן על מחשב שאין לו גישה לWWW. הדרך היחידה שאני יכול לגשת למחשב הזה זה להיות ברשת של המשרד. בעזרת הVPN, כל פקטה שהמחשב שלי שולח בעצם נשלחת מהמשרדים אם הVPN הותקן שם. בגלל זה אני טכנית ברשת שלהם ולכן יש לי גישה לאתר האינטרנט.

2) הסבה של הכתובת רשת של הלקוח

מכיוון שהשרת שולח את כל הפקטות בשביל הלקוח, אם השרת מותקן ברשת אחרת, האתרים שאליהם הלקוח גולש לא יכולים לדעת את כתובת הרשת שלו. במקום, הם מקבלים את כתובת הרשת של הVPN. אם השרת מותקן במדינה אחרת, אפשר אפילו לעקוף חסימה של מדינות מאתרים! כלומר, אם הסדרה האהובה אליך בנטפליקס לא קיימת בנטפליקס ישראל אלה בסין, בעזרת הVPN (ואם הוא מותקן פיזית בסין) אתה כן יכול לראות את הסדרה הזאת.



3) פרטיות מהISP (ספק האינטרנט) של הלקוח

ישנם אנשים שלא רוצים שהISP שלהם, במקרא כלשהו, יראה את הפקטות שהמחשב שלהם שולח. אם הVPN מותקן במדינה אחרת, הISP של הלקוח רק יראה שהוא מדבר עם השרת VPN אבל לא יראה את הפקטות שבפנים (מכיוון שהם מוצפנות!).

אחרי שדיברנו על כל המערכת, נבין לעומק מה הצד של השרת עושה.

השרת

פתיחת השרת וקבלת הלקוח

הדבר הראשון שהשרת צריך לעשות הוא לפתוח חיבור מהעולם החיצוני לעצמו. זה נעשה בעזרת Sockets, שהם הדרך בה מערכות ההפעלה מתקשרות ברשת.
לאחר מכן השרת צריך לחכות שהלקוח יתחבר אליו ולקבל ממנו את סוג ההצפנה.

אותנטיקציית RSA

ישנה בעיה גדולה אם הלקוח מתחברת לשרת בלי לעשות אותנטיקציה: מישהו יכול להתחזות כהשרת ולגנוב את המידע שלך שעובר בין הלקוח לשרת (עוד מידע בנושא בפרק הסכנות). כדי שזה לא יקרה חייבים שהשרת יוכיח שהוא לא מתחזה. בשביל זה אפשר להשתמש באותנטיקציית RSA.

האלגוריתם הוא מאוד פשוט:

1) השרת והלקוח יודעים משפט סודי מראש - לדוגמה, שניהם יודעים שהמשפט הסודי הוא "נוראל גליק יצר אותי".
2) השרת מכין 2 מפתחות - אחד פרטי ואחד ציבורי. המפתחות האלה הם מיוחדות מכיוון שכל מה שהוצפן עם המפתח הסודי יכול להיפתח רק עם המפתח הציבורי וההפך.
3) השרת שולח ללקוח את המפתח הציבורי.
4) השרת עושה חתימה דיגיטלית בכך שהוא עושה Hash למשפט הסודי, מצפין את הHash ושולח ללקוח.
5) הלקוח משתמש במפתח הציבורי כדי לראות אם החתימה הדיגיטלית והמשפט הסודי שווים, ואם כן - זה הוכחה שזה השרת ולא מתחזה, מכיוון שרק הלקוח והשרת יודעים על המשפט הסודי (אלה אם כן מצליחים לעשות reverse engineer לקוד של התוכנית שלך ומוצאים את המשפט הסודי, נחזור לזה אחר כך)

אבל האלגוריתם הזה לבד הוא לא מספיק - חייבים לשלב אותו אם סוג הצפנה כדי שהמידע תהיה מוצפנת, ושמי שיושב בין הלקוח לשרת לא יוכל לדעת מה המידע שעובר. לכן נשלב את האלגוריתם של החלפת מפתחות Diffie Hellman עם הRSA.

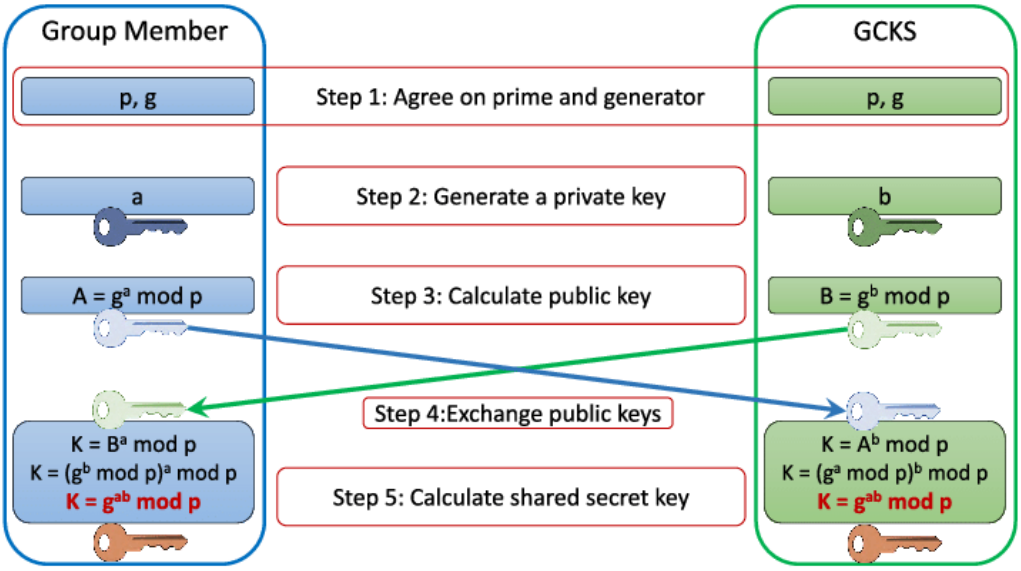
החלפת מפתחות Diffie Hellman בשילוב עם RSA

החלפת מפתחות הוא אלגוריתם שמטרתו הוא ששני צדדים שמדברים ברשת יסכימו על מפתח אחד שאיתו הם יכולים לעשות הצפנה סימטרית.

האלגוריתם עובד כך:

1) ישנם שני ערכים פומביים שהשרת והלקוח יודעים אליהם מראש: p, g.
2) הלקוח והשרת מגרילים מספר רנדומלי a, b.
3) הלקוח והשרת מחשבים מספר פומבי בעזרת האלגוריתם A = g^a mod p (אותו דבר אם B ו-b)
4) הלקוח והשרת מחליפים את המספרים הפומביים שלהם ומחשבים את המפתח המשותף בעזרת האלגוריתם K = B^a mod p (אותו דבר אם A ו-b)

האלגוריתם לבד הוא לא מספיק בטוח. מישהו יכול לשבת בין השרת ללקוח ולעשות אם כל אחד בנפרד את ההחלפת מפתחות, ובכל פעם שהוא מקבל מידע מהלקוח הוא מפענח, קורא, מצפין ושולח לשרת (נראה דוגמא עוד מעט). לכן נשלב את האלגוריתם הזה עם RSA כדי שמצב כזה יהיה בלתי אפשרי. כל מה שצריך לעשות זה להשתמש במפתח RSA הציבורי של השרת כדי להצפין את החישוב Diffie Hellman של הלקוח, ולשלוח אותו לשרת. ככה רק השרת (מי שיש לו את המפתח הפרטי) יכול לפענח את ההצפנה ולקבל את החישוב של הלקוח, וכך הפוך (השרת מצפין עם המפתח הפרטי ושולח ללקוח).



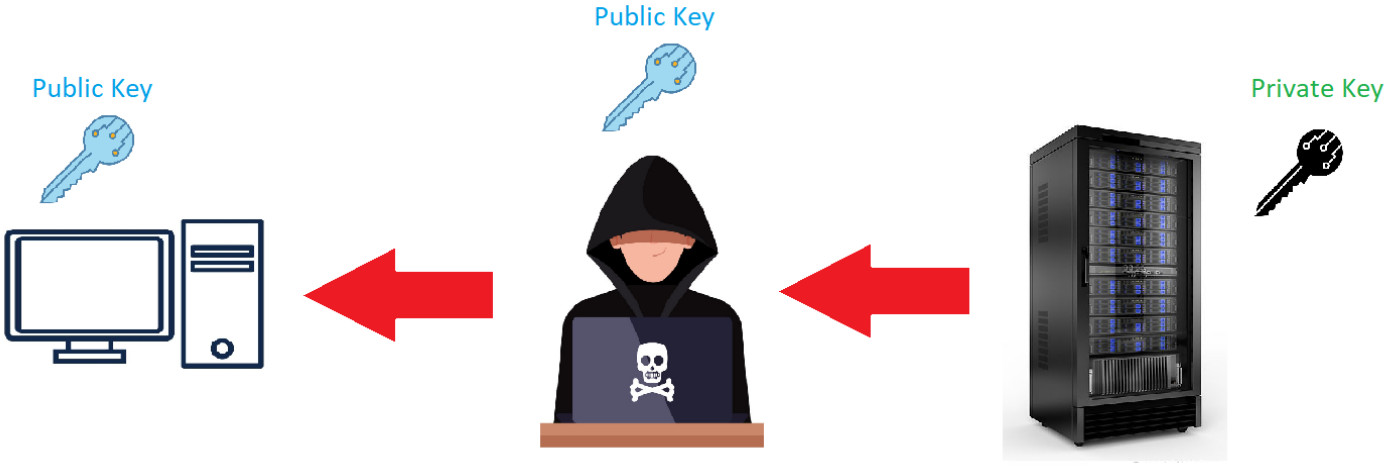
דוגמא לניסיון של Man In The Middle - גרסה ראשונה

כעת נראה דוגמא של תוקף שמנסה לקרוא את המידע שעובר בין השרת VPN ללקוח.

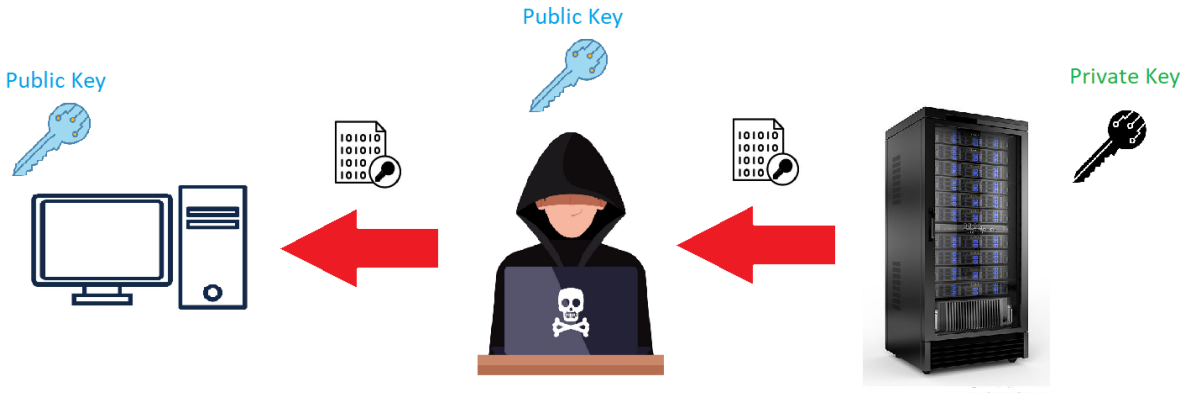
הנה תמונה של לקוח, תוקף, והשרת שלנו:

1) השרת מופעל והתוקף התחבר לשרת וגורם לו לחשוב שהוא הלקוח האמיתי.
2) הלקוח מתחבר לתוקף וחושב שהוא השרת האמיתי

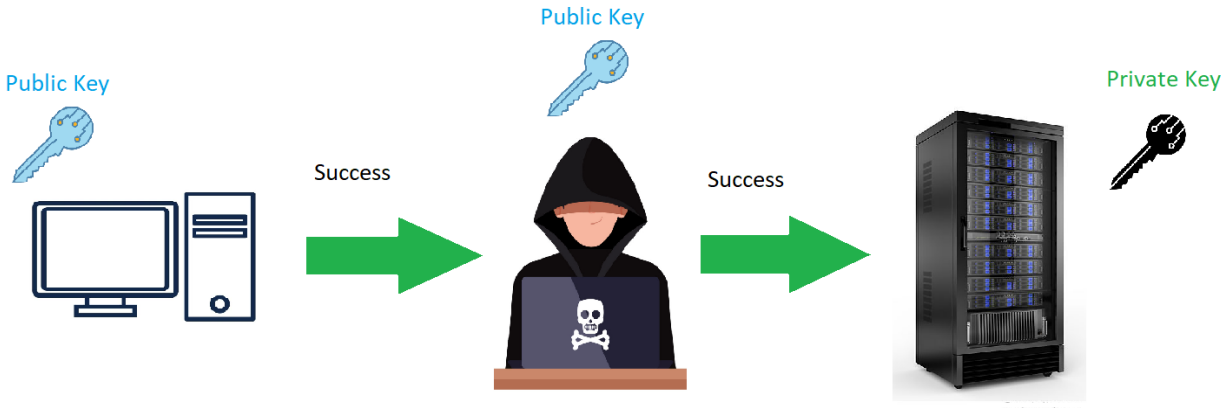
3) השרת שולח את המפתח הציבורי לתוקף ששולח ללקוח:



4) השרת עושה Hash ואז מצפין את המשפט הסודי שרק הוא והלקוח יודעים, ושולח לתוקף, והתוקף שולח ללקוח

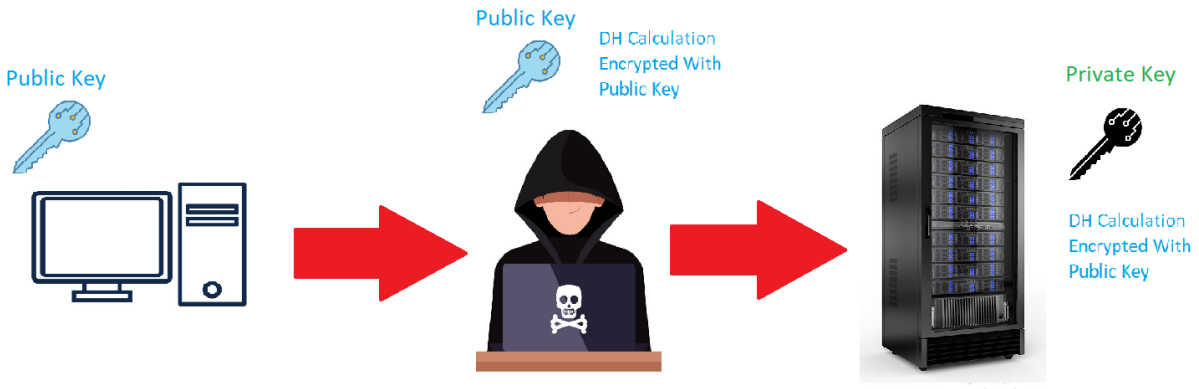


5) הלקוח מפענח את ההצפנה בעזרת המפתח הציבורי, משווה עם המשפט הסודי שמתוכנת אצלו בקוד ושולח "success" לתוקף ששלוח לשרת:



כעת הלקוח יודע שהוא בסופו של דבר מדבר עם השרת האמיתי ולא אם מתחזה. אבל, יכול להיות שיש מישהו בינו לבין השרת שרוצה לגנוב את המידע שעובר! לכן נכנס ההחלפת מפתחות של Diffie Hellman

6) הלקוח מחשב מספר בעזרת האלגוריתם של Diffie Hellman, ומצפין את המספר הזה עם המפתח הציבורי של השרת. הוא שולח את זה לתוקף ששולח את זה לשרת:



7) השרת מחשב את המספר שלו ושולח אותו לתוקף ששולח ללקוח.

כעת המצב נראה כך:

- ללקוח יש את החישוב הלא מוצפן של השרת ואת החישוב הלא מוצפן שלו, ולכן הוא יכול להכין מפתח משותף.
- לתוקף יש את החישוב הלא מוצפן של השרת, את החישוב המוצפן של הלקוח ואת המפתח הציבורי של השרת, ולכן לא יכול לחשב את המפתח המשותף.
- לשרת יש את החישוב המוצפן של הלקוח, את החישוב שלו ואת המפתח הפרטי שלו, ולכן יכול לחשב את המפתח המשותף.

עכשיו לשרת וללקוח יש את המפתח המשותף ולתוקף אין!

דוגמא לניסיון של Man In The Middle - גרסה שנייה

בדוגמה הזאת התוקף יעשה בעצמו חישוב Diffie Hellman נפרד לשרת והלקוח. כלומר המעבר מידע יראה כך:

1) הלקוח שולח את החישוב Diffie Hellman שלו לתוקף
2) התוקף שולח את החישוב Diffie Hellman הראשון שלו ללקוח
3) התוקף והלקוח מחשבים מפתח משותף
4) התוקף שולח את החישוב Diffie Hellman השני שלו לשרת
5) השרת שולח את החישוב Diffie Hellman שלו לתוקף
6) השרת והתוקף מחשבים מפתח משותף

ככה, כל פעם שהלקוח שולח לתוקף מידע מוצפנת התוקף מפענח עם המפתח הראשון, קורא את המידע, מצפין עם המפתח השני ושולח לשרת והפוך כאשר השרת שולח לתוקף.

נעבור שלב שלב עם המפתחות RSA ונראה למה התוקף לא יכול לעשות זו:

1) הלקוח שולח את החישוב Diffie Hellman שלו המוצפן על ידי המפתח הציבורי של השרת אל התוקף
2) התוקף שולח את החישוב Diffie Hellman הראשון שלו ללקוח
3) מכיוון שלתוקף אין את המפתח הפרטי הוא לא יכול לפענח את החישוב של הלקוח וכך לא יכול להקים מפתח משותף עם הלקוח.

זה עד כדי כך פשוט! התוקף והלקוח לא יכולים להחליט על מפתח משותף ופה התוקף מפסיד.

קליטת פקטות מהלקוח

כדי שמערכת VPN תעבוד, צריך שהשרת תקבל פקטות מהלקוח, תשנה אותם כדי שיהיהו מוכנים להישלח לעולם. אבל מה זה בדיוק אומר? ניקח דוגמא לפקטת ICMP שנשלחת מהOS של הלקוח לעולם לפני שהVPN מופעל:

```
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 192.168.1.39
  dst       = 8.8.8.8
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = None
     id        = 0x0
     seq       = 0x0
     unused    = ''
```

אפשר לראות פקטה רגילה. אחרי שתוכנת הVPN במחשב של הלקוח לוקחת את הפקטה הזאת, מצפינה את כולה ושולחת לנו (לשרת) אנחנו מפענחים את ההצפנה ומקבלים את הפקטה הזאת. עכשיו אי אפשר פשוט לשלוח את הפקטה ככה לעולם, כי העולם יחזיר את התשובה חזרה ללקוח שלנו (הsrc זה הIP שלו והMAC שלו). לכן השרת צריך לעשות כמה שינויים לפקטה, כך שהעולם ישלח לנו את התשובה במקום ללקוח. בנוסף מכיוון שאנחנו משנים את תוכן הפקטה הchecksum יהיה שונה ונצתרך לשנות גם אותה. אז, הנה מה שהשרת צריכה לעשות שהיא קולטת פונקציה מהלקוח:

1) לפענח אותה ולהכין מהמידע פקטה.
2) לחשב מחדש את הchecksum לאורך כל הפקטה.
3) להחליף את הSource IP והSource MAC לIP וMAC של השרת.
4) להחליף את הDestination MAC לMAC של הראוטר של השרת (לא רואים בתמונה מכיוון שזה נעשה בLayer נמוך יותר)
5) לשלוח לעולם

קליטת פקטות מהעולם

הצעד הזה הוא מאוד פשוט. השרת מקבל פקטה מהעולם הנראית כך (ממשיכים עם הדוגמה של ICMP):

```
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = 28
  id        = 0
  flags     =
  frag      = 0
  ttl       = 113
  proto     = icmp
  chksum    = 0x7802
  src       = 8.8.8.8
  dst       = 192.168.1.39
  \options   \
###[ ICMP ]###
     type      = echo-reply
     code      = 0
     chksum    = 0x0
     id        = 0x0
     seq       = 0x0
     unused    = ''
###[ Padding ]###
        load      = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

אנחנו רק מצפינים את הפקטה הזאת ושולחים ללקוח. הלקוח מתוכנת כך שהוא מתעסק עם שינוי המידע בפקטה, ואנחנו לא נוגעים בכלום (הלקוח נמצא בעבודה של טל ברילנט).

חלק מעשי

בפרק הזה אנחנו נכתוב את השרת של הVPN. השרת אחראי על:

- להתחבר ולעשות אותנטיקציה עם הלקוח
- להחליף מפתחות עם הלקוח
- לקבל ולפענח את הפקטות של הלקוח
- לשלוח את הפקטות לאינטרנט
- לקבל תשובה מהאינטרנט
- להצפין את הפקטות ולשלוח ללקוח

אתחול של פרמטרים

חייבים לעשות אתחול של מספר פרמטרים חיצוניים שישבו בקובץ בשם params_server.json. זה יהיה קובץ ג'ייסון ששומר את הערכים הבאים:

1) Server Port - The port used to connect to the server
2) Server IP - The IP used to connect to the server
3) Main Interface - The Server's OS Network Interface used to connect to the internet
4) Public Key Modulus - Public Diffie Hellman Variable
5) Public Key Base - Public Diffie Hellman Variable

בקוד של השרת, נשתמש בספרייה json כדי לפתוח את הקובץ ולהכניס את הערכים שבקובץ למשתנים:

```
# Init Parameters
with open('Server/params_server.json') as f:
    params = json.load(f)

port = params["port"]
server_ip = params["server_ip"]
interface = params["main_interface"]
public_key_modulus = params["public_key_modulus"]
public_key_base = params["public_key_base"]
```

בנוסף צריך להשיג את ה-Mac Address של השרת ושל הראוטר שלו. בשביל זה נשלח פקטה של ARP בעזרת Scapy וניגש לhwsrc של הפקטה שחזרה עלינו (שם שמור הMAC):

```
router_ip = conf.route.route("0.0.0.0")[2]
router_mac = srp1(Ether(dst="ff:ff:ff:ff:ff:ff") / ARP(pdst=router_ip))[ARP].hwsrc
```

אתחול של השרת

כעת אפשר להתחיל את השרת ולפתוח אותו לעולם בעזרת הSockets של פייתון. נכין סוקט חדש ונעשו לו bind לIP ולPort שהגדרנו בקובץ ג'ייסון:

```
# Open Socket
main_con = socket.socket()
main_con.bind((server_ip, port))
main_con.listen(5)
```

אותנטיקציית RSA

אחרי שיש חיבור ישיר בין השרת ללקוח, צריך להתחיל RSA Authentication. למדנו בחלק התיאורטי מה זה בדיוק וכעת ניישם אותו.

השרת מייצר שני מפתחות RSA, אחד פומבי ואחד פרטי, ושולח את הפומבי ללקוח:

```
(rsa_public_key, rsa_private_key) = rsa.newkeys(512)

# Send Public key to client
con.send(rsa_public_key.save_pkcs1(format='DER'))
print("RSA: Sent public key to client...")
```

בנוסף השרת צריך להכין Digital Certificate כדי להוכיח ללקוח שהוא אמיתי. בשביל זה שניהם יודעים על משפט מראש: "Tal and Norel secret message!".

נשתמש במפתח הפרטי וב-Hashing Algorithm SHA-1 כדי להכין את הCertificate, ונשלח ללקוח:

```
# Sign digital signature & send to client
message = "Tal and Norel secret message!".encode()
signature = rsa.sign(message, rsa_private_key, 'SHA-1')

con.send(signature)
print("RSA: Sent digital signature to client...")
```

כעת נחכה לתשובה מהלקוח שהוא אישר את האותנטיקציה בכך שנמתין למידע מהלקוח ואם קיבלנו מידע ריק אז נסגור את החיבור:

```
# Wait for answer
success = con.recv(1500).decode()
print("RSA: Received answer from client...")
if not success:
    print("RSA: Client closed connection")
    con.close()
    continue

print("RSA: Client Successfully Connected!")
# ----RSA Authentication End----
```

החלפת מפתחות Diffie Hellman

כפי שלמדנו בחלק התיאורטי, החלפת המפתחות עובדת בגלל אלגוריתם מתמטי. לכן רק צריך לתכנת את האלגוריתם הזה:

1) השרת מגריל מספר רנדומלי בין 100 ל-5000:

```
# ----Diffie-Hellman Key Exchange Start----
secret_number = random.randint(100, 5000)
```

2) השרת מקבל את החישוב של הלקוח ומפענח אותו בעזרת המפתח הפרטי:

```
client_calc = con.recv(1500)
client_calc = int(rsa.decrypt(client_calc, rsa_private_key).decode())
print("DH: Received Calculation from Client")
```

3) השרת מחשב את החישוב שלו ושולח לשרת:

```
con.send(str(pow(public_key_base, secret_number) % public_key_modulus).encode())
print("DH: Sent Calculation to Client")
```

4) השרת מחשב את המפתח המשותף:

```
key = pow(client_calc, secret_number) % public_key_modulus
dif_hel_key = key
```

קבלת הפקטות מהלקוח

בשרת מתחילים תהליך שיקבל מידע מהעולם ובתהליכון הראשי של התוכנה אנחנו מתחילים לולאה אינסופית שמקבלת פקטות מהלקוח, מפענחת אותם ומריצה את הפונקציה handle_connection:

```
# Start thread that will receive incoming packets from WWW
thread = threading.Thread(target=lambda: sniff(prn=recv_pkts, iface=interface))
thread.start()

# When getting packet from client, send it to WWW
while True:
    data = con.recv(1500)
    data = decrypt_packet(data, key, encryption_type)
    handle_connection()
```

הפונקציה handle_connection עושה את הפעולות הבאות:

1) מכין פקטה מסוג Ether חדשה מהמידע
2) מחשב מחדש את הchecksum לאורך כל הפקטה.
3) מחליף את הSource IP והSource MAC לIP וMAC של השרת.
4) מחליף את הDestination MAC לMAC של הראוטר של השרת
5) לשלוח לעולם

הפונקציה נראית ככה:

```python
def handle_connection():

    try:
        client_pkt = Ether(data)

        if Ether in client_pkt:
            client_pkt[Ether].src = get_if_hwaddr(interface)
            client_pkt[Ether].dst = router_mac
            client_pkt[Ether].chksum = None

        if IP in client_pkt:
            client_pkt[IP].chksum = None
            client_pkt[IP].src = get_if_addr(interface)

        if UDP in client_pkt:
            client_pkt[UDP].chksum = None

        if TCP in client_pkt:
            client_pkt[TCP].chksum = None

        #client_pkt.show()

        # Third stage: Send packet to WWW
        sendp(client_pkt, iface=interface, verbose=False)
    except Exception as e:
        print(f"[Client -> Server]: {e}")
```

הקליטה מהעולם

בתוך תהליכון נפרד בשרת, רצה פונקציה הנקראת recv_pkts שמקבלת פקטות מהעולם החיצוני, מצפינה אותם ושולחת ללקוח. בנוסף אנחנו מעיפים כל פקטה שאין לה את השכבת IP:

```
def recv_pkts(pkt):

    if IP not in pkt:
        return

    if pkt[IP].src == get_if_addr(interface):
        return

    encrypted_packet = encrypt_packet(bytes(pkt))
    con.send(encrypted_packet)
```

הצפנה

בשביל ההצפנה השתמשנו בהצפנת Fernet. מכיוון שזה לא החלק שלי בעבודה תיאור מפורט על ההצפנה ויישומו בקוד נמצא עצל טל ברילנט.

מדריך למשתמש

שלום! זהו המדריך המלא להתקין את הVPN שלנו. אחרי שהתקנת את הVPN, אתה תקבל שירותים כמו:

- היכולת להתחבר לרשת אחרת דרך האינטרנט
- היכולת להחביא את המידע שנכנס ויוצא מהמחשב שלך מהספק או בעל הרשת שלך
- מהירות אינטרנט מאוד איטית!

התקנת הVPN לוקחת 4 שלבים:

- התקנת ספריות הפייטון הרלוונטיות
- הוספת אינטרפייס למחשב
- התקנת השרת ברשת אחרת
- הפעלת הVPN

בבקשה תבדקו שיש לכם pip וpython מותקנים על המחשב של הקליינט ושל השרת.

צעד ראשון: התקנת ספריות הפייטון הרלוונטיות

הספריות שיש להתקין כדי שהVPN יעבוד הם:

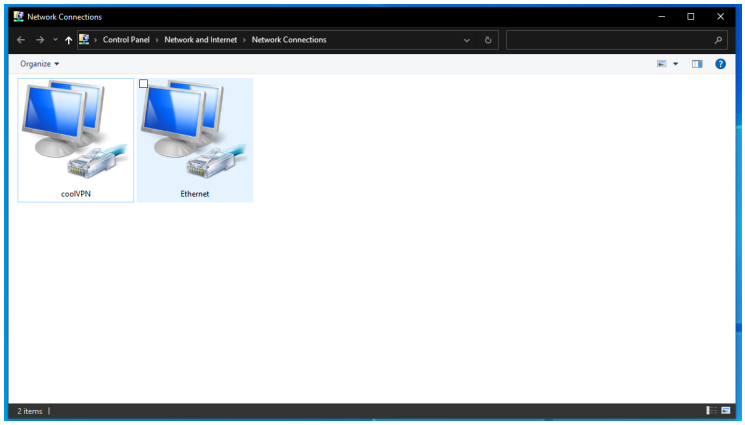
Threading, scapy, binascii, json, rsa, cryptography, base64

בבקשה פתחו את `cmd.exe` והקלידו pip install package והחליפו את package בכל אחד מהספריות מהרשימה.

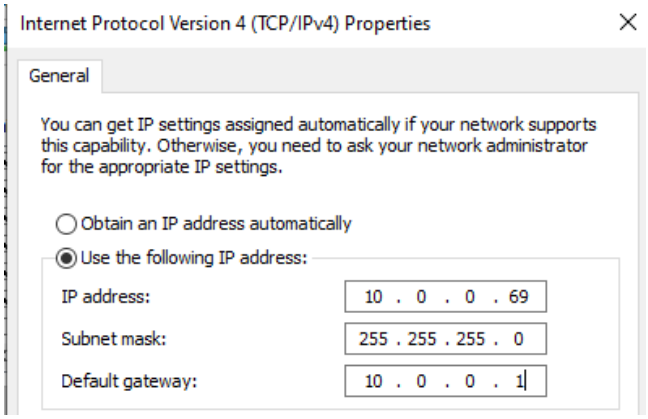
צעד שני: הוספת אינטרפייס למחשב

ניכנס לControl Panel, ונלחץ על "Network and Internet". משם נלך ל"Network and Sharing Center" ואז ל"Set up a new connection or network".

שם נכין Network חדש בשם "coolVPN" (השם לא חשוב, העיקר שזוכרים אותו).



ניכנס לProperties של coolVPN ונחפש את "Internet Protocol Version 4". נלחץ על "Use the following IP address:" ונכתוב "10.0.0.69" בשורה של IP Address ובשורה של Default gateway נכתוב 10.0.0.1:



כעת האינטרפייס מוכן.

בנוסף צריך להוסיף MAC לראוטר של האינרפייס. נפתח את cmd.exe ונקליד:

```
C:\Users\norel>netsh interface ipv4 add neighbors "Local Area Connection" 10.0.0.1 70-32-17-69-69-69
```

צעד שלישי: התקנת השרת ברשת אחרת

כעת רק נותר להתקית את הספריות מהצעד הראשון על המחשב של השרת ולהריץ את תוכנת השרת server.py.

צעד רביעי: הפעלת הVPN

כדי להפעיל את הVPN, צריך להריץ את הקובץ client.py

רפלקציה

העבודה שלי ושל טל על VPN הוא עבודה מאוד מעניינת בתחום הרשתות ומאוד הייתי ממליץ לכל מי שמעוניין בתחום ליישם תוכנה ושרת שבאמת עובד.

בתחילת העבודה נפגשנו כדי לחקור ולהבין אל הנושא. הדבר הראשון שחשבנו אליו הוא להכין VPN בעזרת השירות VPN שמובנית לתוך ווינדוס, אך לא הצלחנו ליישם תוכנה ושרת שמתחבר לזה. אחרי הרבה חקירה הגענו למסקנה שאפשר להכין רשת אינטרפייס חדש שבעזרתו נעביר את כל הפקטות של המחשב לתוכנה שלנו. בסוף זה עבד לנו וכך עובד הVPN כעת. אחרי שהקמנו את הVPN היינו צריכים לחשוב על סכנות הבטחה שיכולות להיות למי שמשתמש בתוכנה שלנו. למדנו על סוגי הצפנות ומה הדברים החיוביים ושליליים בלהשתמש בהם.

אני התמקדתי על האוטנתקציית RSA, על ההחלפת מפתחות Diffie Hellman ועל השרת. טל התמקד על הקליינט ועל ההצפנות.

אחד הדברים העיקריים שאני לוקח מהעבודה הזאת היא הידע שצברנו בנושא השרתות. לדוגמא, לא ידעתי בדיוק אם כל הפקטות יעברו בVPN שלנו ובפרוייקט גיליתי איזו פקטות כן עוברות ואיזו פקטות לא עוברות, ואם כך שיניתי את הקוד שלנו כדי שהפקטות האלו יעברו בסוף.

לדעתי אני וטל כתבנו את הVPN בצורה טובה מאוד, והדבר היחיד שהייתי משנה זה לעשות את הVPN יותר יעיל בכך שנוסף עוד תהליכונים לתוכנה. כך המהירות של הVPN יעלה. אך מכיוון שמטרת העבודה הוא לחקור וליישם VPN שעובד זה לא נחוץ כל כך.

אם היה לנו יותר משאבים, הייתי רוצה להשכיר שרת במדינה אחרת ולהריץ את שרת הVPN שלנו אליו כדי לבדוק אם התוכנה תעבוד ממדינה אחרת.

מאוד נהנתי לעשות את הפרויקט עם טל, ואני שמח שבחרתי בנושא הזה.

תודות

- תודות רבות לטל ברילנט שעשה איתי את הפרויקט המלא
- תודות רבות לאורי שמיר שעזר לי בנושא הרשתות ובנושא ההגנת סייבר ובלעדו התוכנה לא הייתה יוצאת כמו שהיא היום

ביבליוגרפיה

פונקציית XOR לבייטים - StackOverflow

מידע על ספריית הקריפטוגרפיה - Cryptography

יצירת מפתח להצפנה - StackOverflow

ציפוי מפתח בהצפנה - PKCS7

חלק בהצפנת פרנט - AES\PKCS7

חלק בהצפנת פרנט - CBC

הסבר על מודל ההצפנה - AES

נספחים

PreClient.py

```
from scapy.all import *
import os
import subprocess
from elevate import elevate


def set_route(route):
    # Return route to default
    os.system("route delete 0.0.0.0")
    os.system(f"route add 0.0.0.0 mask 0.0.0.0
{route}")


elevate(show_console=False)

router_ip = conf.route.route("0.0.0.0")[2]
time.sleep(4)
os.system("start /wait cmd /c py Client.py")
print("BALLS")
set_route(router_ip)
```

GUI.py

```
import PySimpleGUI as sg


def start_gui():
    # the color theme of the window
    sg.theme('DarkAmber')
    # the font of the title
    title_font = ("Ariel", 40)
    # the font f the normal text
    normal_font = ("Ariel", 20)
    # the font of the paragraphs
    long_font = ("Ariel", 15)

    # <<<<<< Stuff for the first window >>>>>> #

    # sg.text variable for the title
    vpn_name = sg.Text(
        "ברוכים הבאים לcoolVPN!",
        font=title_font,
        justification="right"
    )

    # sg.text variable for the made by
    made_by = sg.Text("נכתב על ידי נוראל גליק וטל ברילנט",
font=normal_font, justification="right")

    # sg.text variable for the first and second
paragraph
```

```
    explanation = sg.Text(
        "מטרתו היא לחבר משתמש מרשת אחת לרשת שניה. עם
אפשר גם להחביא אתn\השנים גילו שבעזרת הטכנולוגיה הזאת
הזהות "
"שלך ברשת, הוא עושה זאת בכמה דרכים. הדרך
הצפנה של כל המידע שיוצא מהמשתמש לשארn\הראשונה היא
העולם דרך "
הסוואה של המחשב שלרn\ "האינטרנט, והדרך השנייה היא
הדבר הכי טוב והעיקרי הוא שאף אחד חוץ מהשרת,n\n\באינטרנט
"
לראות את המידעn\ "אפילו ספק האינטרנט שלך, לא יכולים
שהמשתמש שולח או מקבל דרך האינטרנט כמו אתרים
שהמשתמשn\"
        "נכנס אליהם או דברים שהמשתמש מוריד",
        justification="right",
        font=long_font
    )

    explanation_2 = sg.Text(
        "שימוש בתוכנה הוא דבר שיתרום לכל אחד באופן כללי,
באבטחה גבוהה ברשת, בביטחון מידע n\ובעיקר לאנשים המעוניינים
"
מה n\ "ובפרטיות. אבטחה ברשת נשמע מאוד טוב, אבל
זה בעצם אומר? זה אומר שאי אפשר לגנוב מהמשתמש מידע, אי
אפשר לדעת את"
היסטוריית הגלישה של המשתמש ולכן אי אפשרn\"
ועוד גופים שאף אחד לא רוצהn\לדוגמה לשלוח אותה למפרסמים
שיחזיקו במידע "
"שלו. "
```

```
הרבה יותר מזה, אבטחה\n        "אבל אבטחה ברשת אומרת
של משתמש יהיה \nברשת אומרת שלהאקר שמנסה לפרוץ למכשיר
הרבה יותר קשה עד "
אם המשתמש מחזיק \n        "לכמעט בלתי אפשרי להצליח
בתוכנה שלנו",
        justification="right",
        font=long_font
    )

    # <<<<<< Stuff for the second window >>>>>> #

    user_guide = sg.Text("מדריך למשתמש",
justification="right", font=title_font)
    first_step = sg.Text(
        "לפני שנשתמש בתוכנה ישנם כמה דברים שנצטרך לעשות"
הרלוונטיות" \nקודם לכן. קודם כל, יש להתקין את הספריות
",        " דרך פיפ של שפת התכנות פייטון. הספריות הן
        justification="right",
        font=long_font
    )
    libraries = sg.Text("Threading, scapy, binascii,
json, rsa, cryptography, base64, pysimplegui",
                        justification="center",
                        font=long_font
                        )
    second_step = sg.Text("לאחר מכן, יש להוסיף ממשק
רשת חדש למערכת ההפעלה", justification="right",
                          font=long_font)
    third_step = sg.Text("עכשיו יש לוודא שהשרת רץ",
```

```
justification="right",
                                font=long_font)
   forth_step = sg.Text("וכעת כל מה שנשאר זה להריץ את
תוכנת הלקוח", justification="right",
                                font=long_font)
   more_info = sg.Text("בשביל מידע נוסף ומפורט לגבי
הרצת התוכנה יש לגשת לפרק המדריך למשתמש בספר הפרויקט")

   # <<<<<< Stuff for the third window >>>>>> #

   thank_you = sg.Text("תודה רבה לכם שבחרתם להשתמש
בתוכנה שלנו, אנחנו בטוחים שלא תתחרטו",
justification="right",
                              font=normal_font)

   # First layout.
   layout_column1 = [[vpn_name],
                     [made_by],
                     [explanation],
                     [explanation_2]]

   layout1 = sg.Column(layout_column1, key='-COL1-',
element_justification='right')

   # Second layout
   layout_column2 = [[user_guide],
                     [first_step],
                     [libraries],
                     [second_step],
```

```
                        [third_step],
                        [forth_step],
                        [more_info]]

   layout2 = sg.Column(layout_column2,
visible=False, key='-COL2-',
element_justification='right')

   # Third layout
   layout_column3 = [[thank_you]]

   layout3 = sg.Column(layout_column3,
visible=False, key='-COL3-',
element_justification='right')

   # the layout that has the first 3 layouts
   main_layout = [[layout1, layout2, layout3],
[sg.Button('Next'), sg.Button('Exit')]]

   # Create the Window
   window = sg.Window('coolVPN', main_layout,
size=(880, 800))
   # Event Loop to process "events" and get the
"values" of the inputs

   layout = 1  # The currently visible layout

   while True:
       event, values = window.read()
```

```
        print(event, values)
        # if the Exit button has been clicked
        if event in (None, 'Exit'):
            break
        # if the Next buttone has been clicked
        if event == 'Next':
            if layout != 3:

window[f'-COL{layout}-'].update(visible=False)
                layout = layout + 1 if layout < 3
else 1

window[f'-COL{layout}-'].update(visible=True)
            else:
                break
   window.close()
```

Client.py

```
# import all the relevant packages
import threading
from scapy.all import *
import json
import rsa
import os
from elevate import elevate
from cryptography.fernet import Fernet
import base64
import GUI
import atexit


def recv_packets():
   # a function that receives the packets from the
server and sends them back to the interface
   while True:
       # receive the packet
       data = main_con.recv(4092)  # receive the
packet from the server
```

```
        # decrypt the packet
        try:
            data = decrypt_packet(data)  # decrypt
the received packet
        except Exception:
            continue

        # change packet variables so it sends it to
the interface from the computer
        try:
            pkt = Ether(data)

            if Ether in pkt:
                pkt[Ether].src = vpn_router_mac
                pkt[Ether].dst = vpn_mac
                pkt[Ether].chksum = None

            if IP in pkt:
                pkt[IP].chksum = None
                pkt[IP].dst = "10.0.0.69"

            if UDP in pkt:
                pkt[UDP].chksum = None

            if TCP in pkt:
                pkt[TCP].chksum = None

            # send the packet to the interface
            sendp(pkt, iface=vpn_interface,
```

```
verbose=False)
        # if there is an error
        except Exception as e:
            # print the exception
            print(f"[Server -> Client] {e}")
            # return to the start of the loop
            continue


def on_packet_sniff(pkt):
    # a function that sends the server the packets
that the client sniffs from the interface
    # sent encrypted packet
    #    if IP not in pkt:
    #        return

    #    if pkt[IP].dst == "10.0.0.69":
    #        return

    try:
        main_con.send(encrypt_packet(bytes(pkt)))
    except Exception:
        print("[CLIENT] Error ENCRYPTING packet
headed to Server")


def encrypt_packet(pkt):
    # Encrypting a packet using the
cryptography.fernet library
```

```
    # Bibliography:
    # Cryptography library information -
https://cryptography.io/en/latest/
    # Creating your own fernet key -
https://stackoverflow.com/questions/44432945/generat
ing-own-key-with-python-fernet

    global fernet_obj
    # Fernet encryption
    return fernet_obj.encrypt(pkt)


def decrypt_packet(enc_pkt):
    # a function for decrypting a packet
    # Cryptography library information -
https://cryptography.io/en/latest/
    # Creating your own fernet key -
https://stackoverflow.com/questions/44432945/generat
ing-own-key-with-python-fernet

    global fernet_obj
    # fernet encryption
    return fernet_obj.decrypt(enc_pkt)


def on_connect(server_ip, server_port):
    global dif_hel_key
    global fernet_obj
```

```
    # a function responsible for the connection with
the server
    # Connect to server
    try:
        # connect to the vpn server
        main_con.connect((server_ip, server_port))
    # if there is an error
    except:
        print("[Client] Failed to connect to the
given IP or Port. Restarting Client...\n\n")
        return False

    print("[Client] Connected Successfully to the
Server.")

    # ----RSA Authentication Start----
    # Receive public key
    rsa_public_key =
rsa.key.PublicKey.load_pkcs1(main_con.recv(1500),
format='DER')
    print("[Client] RSA: Received public key from
server...")

    # Receive signature
    signature = main_con.recv(1500)
    print("[Client] RSA: Received digital signature
from server...")

    # Verify Signature
```

```
    try:
        message = "AMONGUS".encode()
        rsa.verify(message, signature,
rsa_public_key)
    except:
        print("[Client] RSA: Verification Failed.
Restarting Client...")
        return False

    print("[Client] RSA: Verification Successful.
Beginning DH Key Exchange")

    # If success, send to server
    main_con.send("success".encode())
    # ---- RSA Authentication End----

    # ----Diffie-Hellman Key Exchange Start----
    secret_number = random.randint(100, 5000)

    # Calculate DH Client Number & encode it using
UTF8
    client_calc = str(pow(public_key_base,
secret_number) % public_key_modulus)
    client_calc =
rsa.encrypt(client_calc.encode("utf-8"),
rsa_public_key)
    main_con.send(client_calc)
    print("[Client] DH: Sent Calculation to Server")
```

```
    server_calc = int(main_con.recv(1500).decode())
    print("[Client] DH: Received Calculation from
Server")

    key = pow(server_calc, secret_number) %
public_key_modulus
    dif_hel_key = key  # assign the Diffie Hellman
public key
    print(key)

    # ----Diffie-Hellman Key Exchange End----

    print("[Client] DH Key Exchange Successful.
Starting VPN Tunnel...")

    # Create fernet object

    conv_dh = str(dif_hel_key).encode()
    conv_dh_padded = conv_dh + bytes(32 -
len(conv_dh))
    f_key = base64.urlsafe_b64encode(conv_dh_padded)

    # Converting the key into a cryptography.fernet
object
    fernet_obj = Fernet(f_key)

    # Change default packet routing to the VPN custom
interface
    set_route("10.0.0.1")
```

```
    print("VPN Turned on. This only works if ran in
administrative mode")

    # Start receiving packets from server
    thread_recv =
threading.Thread(target=recv_packets)
    thread_recv.daemon = True
    threads["recv"] = thread_recv
    thread_recv.start()

    thread_sniff = threading.Thread(target=lambda:
sniff(prn=on_packet_sniff, iface=vpn_interface))
    thread_sniff.daemon = True
    threads["sniff"] = thread_sniff
    thread_sniff.start()

    print("[Client] VPN Tunnel Successfully online.")
    return True


def start_cli():
    print("\n\n[CLI] Please enter a command or type
'help' for a list of commands")
    while True:
        # ask for input
        command = input(">")
        # if the command is not valid
        if command not in commands:
            print("[CLI] Command not found. Listing
```

```
all commands:")
            commands["help"]()
            continue

        # Run Command
        commands[command]()


def start_client():

    # start the gui
    GUI.start_gui()
    # after the user closes the gui, start CLI and
questions
    # clear screen
    os.system("cls")

    # cli loop
    while True:
        print("[CLI] Hello, welcome to Tal & Norel's
VPN")
        print("[CLI] Please choose a mode:")
        print("  (1) Public VPN (Connect to
Internet)")
        print("  (2) Custom VPN (Connect to Custom
Server)")

        # user has to choose between connecting to a
custom vpn or to a public vpn
```

```
        while True:
            vpn_type = input(">")
            if vpn_type == "1" or vpn_type == "2":
                break
            # if the user did not choose a valid
answer
            else:
                print("[CLI] Please Choose a
Displayed Answer:")

        # If Custom, ask for IP & Port
        if vpn_type == "2":
            # IP
            print("\n[CLI] Please enter the IP
Address of the server:")
            vpn_ip = input(">")

            # Port
            print("\n[CLI] Please enter the Port of
the server:")
            vpn_port = input(">")

        # If Public, use the default
        else:
            vpn_ip = server_ip
            vpn_port = server_port

        success = on_connect(server_ip=vpn_ip,
server_port=vpn_port)
```

```
        # If failed to connect, retry client
        if not success:
            continue

        # If succeeded, exit loop & start CLI
        break

    start_cli()


# Command Functions

def command_exit(desc=False):
    if desc:
        return """
            Command: Exit Client
            Usage: exit
            Description: Closes connection to the VPN
Server & exits the client.
            """

    # ------------------------ Begin Command
-----------------------

    print("Exiting Client...")

    # Close the connection to the server
    main_con.close()
```

```
    exit()


def set_route(route):
    # Return route to default
    os.system("route delete 0.0.0.0")
    os.system(f"route add 0.0.0.0 mask 0.0.0.0
{route}")


def command_help(desc=False):
    if desc:
        return """
            Command: Help
            Usage: help
            Description: Shows all commands, their
usage, and gives a description of what each command
does.
            """

    # ------------------------ Begin Command
------------------------
    print("[CLI] Listing all commands:")
    for command, func in commands.items():
        print("\n\n------------------------------")
        print(func(desc=True))
        print("------------------------------\n\n")
```

```
# Ask for Administrative
# elevate()


# Init Parameters
with open('Client/params_client.json') as f:
   params = json.load(f)

# extract to variables from the params.json
server_port = params["port"]
server_ip = params["server_ip"]
vpn_interface = params["vpn_interface"]
main_interface = params["main_interface"]
public_key_modulus = params["public_key_modulus"]
public_key_base = params["public_key_base"]

# router addresses
router_ip = conf.route.route("0.0.0.0")[2]
router_mac = srp1(Ether(dst="ff:ff:ff:ff:ff:ff") /
ARP(pdst=router_ip))[ARP].hwsrc
vpn_router_mac = "70:32:17:69:69:69"
vpn_mac = get_if_hwaddr(vpn_interface)

# encryption variables
dif_hel_key = 0  # public variable for the diffie
hellman key
fernet_obj = None

# the server-client socket
```

```
main_con = socket.socket()
# threads dictionary
threads = {"recv": None, "sniff": None}  #
respective keyword for each thread
# cli commands dictionary
commands = {"exit": command_exit, "help":
command_help}

# Start client
start_client()
```

Server.py

```
import threading
from scapy.all import *
import binascii
import json
import rsa
from cryptography.fernet import Fernet
import base64


# Receive from client
def handle_connection():
   try:
       client_pkt = Ether(data)

       if Ether in client_pkt:
           client_pkt[Ether].src =
get_if_hwaddr(interface)
           client_pkt[Ether].dst = router_mac
           client_pkt[Ether].chksum = None
```

```
        if IP in client_pkt:
            client_pkt[IP].chksum = None
            client_pkt[IP].src =
get_if_addr(interface)

        if UDP in client_pkt:
            client_pkt[UDP].chksum = None

        if TCP in client_pkt:
            client_pkt[TCP].chksum = None

        # client_pkt.show()

        # Third stage: Send packet to WWW
        sendp(client_pkt, iface=interface,
verbose=False)
    except Exception as e:
        print(f"[Client -> Server]: {e}")


# Function that receives all incoming packets from
WWW
def recv_pkts(pkt):
    if IP not in pkt:
        return

    if pkt[IP].src == get_if_addr(interface):
        return
```

```
    try:
        encrypted_packet = encrypt_packet(bytes(pkt))
        con.send(encrypted_packet)
    except:
        print("[SERVER] Error ENCRYPTING packet
headed to Client")
        return


def encrypt_packet(pkt):
    # Encrypting a packet using the
cryptography.fernet library or a Xor function
    # Bibliography:
    # Cryptography library information -
https://cryptography.io/en/latest/
    # Creating your own fernet key -
https://stackoverflow.com/questions/44432945/generat
ing-own-key-with-python-fernet

    global fernet_obj

    # Fernet encryption
    return fernet_obj.encrypt(pkt)


def decrypt_packet(enc_pkt):
    # a function for decrypting a packet
```

```
    global fernet_obj
    # fernet decryption
    return fernet_obj.decrypt(enc_pkt)


print("Running server...")

# Init Parameters
with open('Server/params_server.json') as f:
    params = json.load(f)

port = params["port"]
server_ip = params["server_ip"]
interface = params["main_interface"]
public_key_modulus = params["public_key_modulus"]
public_key_base = params["public_key_base"]

# encryption variables
fernet_obj = None

router_ip = conf.route.route("0.0.0.0")[2]
router_mac = srp1(Ether(dst="ff:ff:ff:ff:ff:ff") /
ARP(pdst=router_ip))[ARP].hwsrc

# Open Socket
main_con = socket.socket()
main_con.bind((server_ip, port))
main_con.listen(5)
```

```
print("Server has been started!")

# Start receiving connection & dealing with them
while True:

    # Wait for a connection
    con, addr = main_con.accept()

    # ----RSA Authentication Start----
    (rsa_public_key, rsa_private_key) =
rsa.newkeys(512)
    print(rsa_public_key)

    # Send Public key to client
    con.send(rsa_public_key.save_pkcs1(format='DER'))
    print("RSA: Sent public key to client...")

    # Sign digital signature & send to client
    message = "AMONGUS".encode()
    signature = rsa.sign(message, rsa_private_key,
'SHA-1')

    con.send(signature)
    print("RSA: Sent digital signature to client...")

    # Wait for answer
    success = con.recv(1500).decode()
    print("RSA: Received answer from client...")
    if not success:
```

```
        print("RSA: Client closed connection")
        con.close()
        continue

    print("RSA: Client Successfully Connected!")
    # ----RSA Authentication End----

    # ----Diffie-Hellman Key Exchange Start----
    secret_number = random.randint(100, 5000)

    # Receive DH Client Calculation
    client_calc = con.recv(4092)
    try:
        client_calc = rsa.decrypt(client_calc,
rsa_private_key)
    except Exception:
        print("Error Decrypting DH Client Calculation
using RSA Priv Key. Connection Closed")
        con.close()
        continue

    client_calc = int(client_calc.decode("utf-8"))
    print("DH: Received Calculation from Client")

    con.send(str(pow(public_key_base, secret_number)
% public_key_modulus).encode())
    print("DH: Sent Calculation to Client")

    key = pow(client_calc, secret_number) %
```

```
public_key_modulus
   dif_hel_key = key
   print(key)

   # ---- Diffie-Hellman Key Exchange End----

   # Set Fernet Object

   # Convert diffie-hellman key into a valid fernet
key
   conv_dh = str(dif_hel_key).encode()
   conv_dh_padded = conv_dh + bytes(32 -
len(conv_dh))
   f_key = base64.urlsafe_b64encode(conv_dh_padded)

   # Converting the key into a cryptography.fernet
object
   fernet_obj = Fernet(f_key)

   # Start thread that will receive incoming packets
from WWW
   thread = threading.Thread(target=lambda:
sniff(prn=recv_pkts, iface=interface))
   thread.daemon = True
   thread.start()

   # When getting packet from client, send it to WWW
   while True:
```

```
        data = con.recv(4096)

        if not data:
            con.close()
            thread.join()
            break

        try:
            data = decrypt_packet(data)
        except Exception as e:
            print("[SERVER] Error DECRYPTING packet
coming from Client")
            print(len(data))
            continue

        handle_connection()
```