**פרויקט גמר – סדנה בתקשורת מחשבים**

**תומר ברנע – 316254713**

**סמסטר 2021ג**

**נושא הפרויקט - TCP over ICMP**

פרויקט זה הוא מימוש "Tunnel" להעברת תקשורת TCP ע"ג ICMP.

מטרת הפרויקט היא אפשור של תקשורת TCP מתוך HOST פנימי שתקשורת ה-TCP שלו מוגבלת או בתשלום בעזרתHOST חיצוני שתקשורת ה-TCP שלו אינה מוגבלת.

התקשורת בין ה-HOST הפנימי ל-HOST החיצוני יכולה להיות מעל פרוטוקולי תקשורת לגיטימיים שכן מאופשרים עבור ה-HOST הפנימי, ככה"נ DNS, ICMP, ייתכן פרוטוקולי IPv6 וכו'. אני בחרתי לממש את התקשורת מעל פרוטוקול ICMP.

הרכיב בצד המוגבל שמייצר תקשורת TCP שלא מאופשרת לצאת "ירכיב" אותה ע"ג פרוטוקול תקשורת מאופשר וישלח אותה לרכיב בצד שאינו מוגבל, שיבצע עבור הרכיב בצד המוגבל את תקשורת ה-TCP, ויחזיר את התשובות מעל אותו פרוטוקול תקשורת מאופשר לרכיב בצד המוגבל.

**רקע:**

במהלך הסדנה שמענו הרצאות ולמדנו רבות על תקשורת, בין היתר על הפרוטוקולים TCP ו-ICMP שרלוונטיים לפרויקט זה, וכן שמענו במס' הרצאות על Sockets, ובין היתר על Raw sockets.

לא קיים Socket שמאפשר Tunneling בין פרוטוקולים כפי שנדרש בפתרון, ולכן הפתרון שנבחר הוא לייצר Sockets לפי הצורך, תוך שימוש ב-Raw sockets.

**מבנה פקטת TCP** –

כידוע ה-Header וה-Data של הפרוטוקול "מורכבים" מעל שכבת ה-IP.

Table

Description automatically generated

**בחירת הפרוטוקול המאופשר –**

בחרתי לממש את הפתרון תוך שימוש בפרוטוקול ICMP, שככה"נ מאופשר ברשתות בהן תעבורת ה-TCP מוגבלת, אך אין זה פסול לממש אותו מעל כל פרוטוקול מאופשר אחר.

מבנה פקטת ICMP

Shape

Description automatically generated with medium confidence

גם ICMP "מורכב" מעל שכבת ה-IP.

Table

Description automatically generated

**איפה נרכיב את המידע שנרצה לשלוח –**

TCP ו-ICMP שניהם פרוטוקולים שמורכבים מעל שכבת ה-IP.

לפקטת ICMP יש שדה Content (או Data) שהוא בעצם ה-Payload של ה-ICMP. זהו בעצם הפתרון היחיד ההגיוני למיקום המידע שנרכיב ע"ג פקטות ה-ICMP.

Graphical user interface, text, application, Word

Description automatically generated

**כיצד להרכיב את הפקטה –**

לצורך מימוש הפתרון שלי בחרתי בשימוש בספריית Scapy ע"מ לייצר את הפקטות, משום שהיא מספקת ממשק נוח לקריאה, כתיבה ופרסור של פקטות.

זוהי ספרייה פשוטה יחסית לשימוש ומוכרת מאוד בקרב מפתחים בתחום רשתות התקשורת, ויש באינטרנט עשרות המלצות לשימוש בה.

לצורך שליחת הפקטות בחרתי בפתרון Raw sockets, משום שהן משמשות API נוח לשליחה וקבלה של פקטות.

ארחיב על כך בהמשך המסמך, אך קל יחסית אף לעבוד עם אובייקטי Sockets באופן סינכרוני באמצעות הספרייה שבחרתי לסינכרוניזציה - asyncio.

**איזה מידע ה-HOST החיצוני צריך –**

ה-HOST החיצוני יכול לקבל את המידע בלבד ולשלוח רק אותו אל ה-Service הרצוי, אך בחרתי בפתרון שלי להעביר מעל הפרוטוקול המאופשר את כל שכבת ה-TCP ואת כל שכבת ה-IP, כך שה-HOST החיצוני ישמש כסוג של Proxy בלבד, והתעבורה בעצם תיוצר ותתוחזק ע"י ה-HOST הפנימי.

לכן כאשר נג'נרט את פקטות ה-ICMP, בשדה ה-DATA יהיה בעצם כל המידע החל משכבת ה-IP והלאה.

**סביבת ההרצה – וירטואליזציה**

לצורכי קלות פיתוח, הרצה והדגמה בחרתי ליישם את הקוד שלי כך ששני ה-HOSTים הם בעצם המחשב עליו רץ ה-Tunneler. עשיתי זאת באמצעות יצירת רשתות וירטואליות במערכת הפעלה Linux (הפיתוח נעשה ב-Ubuntu מעל WSL2).

לצורך פישוט יצירת הסביבה כתבתי bash script שמייצר את הסביבה הרצויה בקלות.

ה-script מייצר רשתות וירטואליות וכרטיסי רשת וירטואליים בעזרת פקודות בסיסיות במערכת הפעלה Linux ושימוש בקונספט ה-Network namespace.

בעצם באמצעות הפרדת ה-Network namespaces יצרתי באופן וירטואלי רשתות כך שרק לאחת מהן יש גישה "החוצה" אל האינטרנט, וכרטיסי הרשת הוירטואליים מקושרים זה לזה ובכך מגשרים מעל ה-Network namespaces השונים.

להלן איור שמתאר את החלוקה לרשתות וכרטיסי רשת וירטואליים:

Text

Description automatically generated

כאשר כל חץ מתאר veth peer (בסה"כ שלושה זוגות, שמקשרים בין 4 Network namespaces)

**סינכרוניזציה –**

ברור מאליו שיש צורך לסנכרנן בין שלבי ה-Tunneler (פירוט בהמשך המסמך).

לצורך סינכרוניזציה בחרתי להשתמש בספריית asyncio הפייתונית. ספרייה זו מאפשרת סינכרוניזציה בלי צורך בחלוקה ל-Threads או ל-Processes נפרדים, והיא יחסית פשוטה לשימוש.

**תיאור הפתרון –**

הפתרון מחולק לשני תהליכים (אחד בכל צד, או במקרה שלנו – אחד בכלNetwork namespace מקשר)

בצד המוגבל (Local) ירוץ ה- LocalClient (שממומש בקובץ local.py), ובצד החופשי ירוץ ה-RemoteServer (שממומש בקובץ remote.py).

כל אחד מהצדדים מייצר 3 אובייקטי Socket, שיושבים בהתאמה ב-Local-Remote או ב-Remote-Local.

בצד ה-Local:

1. Socket להסנפת תעבורת TCP יוצאת
2. Socket לשליחה וקבלה של תעבורת ICMP
3. Socket לשליחת תעבורת TCP לאחר שהתקבלה מה-Remote

בצד ה-Remote:

1. Socket להסנפת תעבורת ICMP (ושליחת תעבורת ICMP חוזרת)
2. Socket לשליחת תעבורת TCP ל-Service
3. Socket לקבלת תעבורת TCP חוזרת מה-Service

קיימת הפרדה בין ה-Sockets השונים בגלל השימוש ב- Raw sockets והפרדה ל-Network namespaces, וכן לצורך נוחות.

ה-Local וה-Remote רצים שניהם בלולאה אינסופית בה מתקבלת תעבורה בצד אחד, מתבצע עיבוד, ולאחר מכן נשלחת תעבורה לצד השני.

ה-Local מקשר בין Socket 1 ל-2 לתעבורה יוצאת ואז בין 2 ל-3 לתעבורה נכנסת

ה-Remote מקשר בין Socket 1 ל-2 לתעבורה נכנסת ואז בין 3 ל-1 לתעבורה יוצאת.

בנוסף, בקובץ utils.py קיימות פונקציות עזר שמשמשות את שני האובייקטים ובקובץ consts.py קיימים קבועים שנמצאים בשימוש בקוד.

פונקציות עזר רלוונטיות:

הפונק' serialize\_data, deserialize\_data – משתמשות בפונק' מתוך הספרייה Scapy כדי לעבד את המידע ולהרכיב מחדש את הפקטות.

הפונק' set\_bpf שמשתמשת בפונק' העזר \_generate\_filter\_with\_tcpdump שבעזרת tcpdump מייצרת bpf filter ובעזרת ה-syscall setsockopt מוסיפה אותו ל-socket.

**התקנה ושימוש –**

את הקוד ניתן להריץ גם ב-HOSTים נפרדים כמובן, אך דבר זה ידרוש שינויים קטנים כמו שמות ה-interfaceים עליהם עושים bind וכתובות ה-IP.

ע"מ להריץ את ה-Tunneler בתצורתו הוירטואלית יש להריץ את run.sh עם הרשאות root על מ"ה Linux.

לאחר מכן יהיה ניתן לקשר בין כרטיס הרשת שנמצא -network namespace בשם local לכרטיס רשת שנמצא ב-network namespace בשם remote.

לצורך הדגמה ניתן להריץ את הבינארי netcat (nc) בשני הצדדים בעזרת הפקודות הבאות:

sudo ip netns exec remote nc -s 3.3.3.2 -lp 55555

sudo ip netns exec local nc 3.3.3.2 55555

שתי פקודות אלה בעצם ייצרו שני חלונות, וכאשר כותבים מידע בצד אחד – הוא עובר באופן שקוף לצד השני באמצעות ה-Tunneler.