**Cyber Lab Ex1 – SYN Flood**

מגישים:

אוריאל שפירא – 314779745

גיא שמעון - 209306513

הסבר כללי:  
במטלה זו קיבלנו Docker שמכיל 3 מכונות – Server, Attacker, Monitor.

התבקשנו לבצע התקפת SYN Flood באמצעות מכונת ה-Attacker על ה-Server.

רעיון ההתקפה:

מכונה אחת (Attacker) שולח הודעות SYN רבות לשרת (Server).

השרת מחזיר הודעת SYN-ACK לכל הודעה כזו, ומקצה משאבים ע"מ להמשיך תקשורת עם המכונה.

מכיוון שכמות הודעות ה-SYN גדולה – השרת מקצה משאבים רבים לצורך התקשורת, מה שמוביל למצב בו לא קיימים לו משאבים מספיקים בשביל לתקשר עם מכונות אחרות. וכך בעצם אנו מבצעים Denial Of Service לשרת.

כמו כן, עבור כל פאקטת SYN, אנו מזייפים את כתובת ה-IP ממנה היא נשלחה ואת הפורט גם כן. זאת בשביל שהשרת לא ידע איזה IP/Port לחסום. וכך בעצם אנו מונעים מהשרת לעצור את ההתקפה.

מטרת העל של כל מכונה:

**Server** – שרת Apache2. משמש בתור מטרה לתוקף.

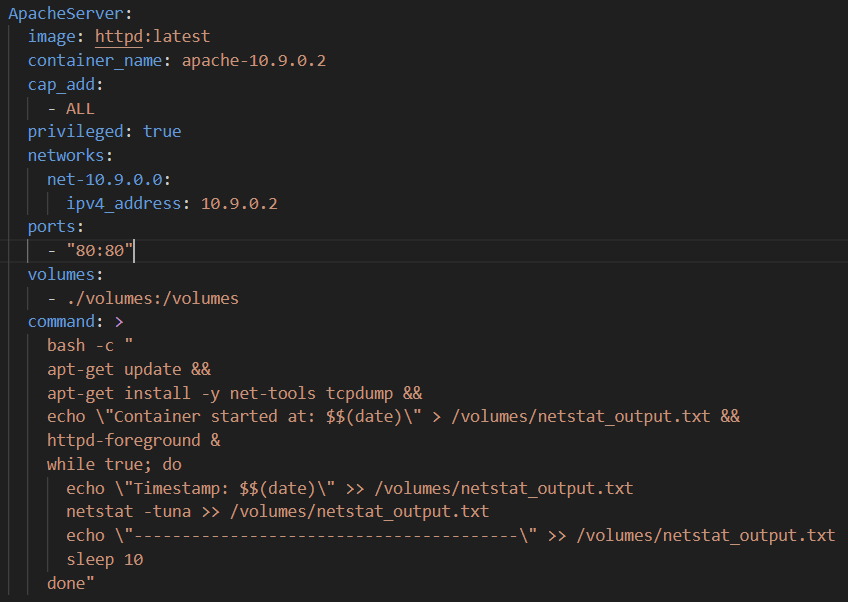
**Attacker** – מבצע שליחה של חבילות SYN מרובות על גבי Raw Socket בפרוטוקול TCP לפורט 80

ב-Server.

**Monitor** – שליחת PING לשרת ע"מ לנתר את העומס על השרת בזמן המתקפה. זאת ע"י בדיקת זמני RTT לפני ההתקפה, במהלך ההתקפה ואחריה.

הסבר על כל Image:

**Server**



שרת HTTP בכתובת 10.9.0.2 שמבצע חלק פסיבי בהתקפה בכך שהוא המטרה של התוקף.

כאשר ה-Image של השרת עולה. השרת מתקין כלי שנקרא tcpdump, מה שמאפשר לנו לנתר את כמות החיבורים במצב SYN RECV (חיבורים מהם השרת קיבל הודעת SYN וממתין להודעת ACK על הודעת ה-SYN ACK אותה שלח).

אנו מדפיסים לקובץ את תוצאת הרצת הפקודה "netstat -tuna" כל 10 שניות. כך אנו מתעדים את השינוי בחיבורים הפתוחים בשרת בין זמן ההתקפה לזמן שלפניה ואחריה.

קובץ זה נועד לתיעוד שלנו שהמתקפה אכן עשתה את שנדרשה ואינו חלק מהמטלה ולכן אינו מצורף.

דוגמה להדפסה לקובץ בעת המתקפה:

A screenshot of a computer

Description automatically generated

**Monitor**

A screen shot of a computer

Description automatically generated

מכונת ה-Monitor שולחת הודעת Ping יחידה לשרת ה-Apache כל 5 שניות.

תוצאת כל הודעה (הדפסת הסטטיסטיקות) נכתבת בקובץ ייעודי.

לכל שורה בקובץ זה ביצענו Parsing ויצרנו את הגרפים הבאים המתארים את כמות הודעות ה-PING שנשלחו עבור כל זמן RTT:

A graph of a bar graph

Description automatically generated with medium confidenceA graph of a number of blue bars

Description automatically generated

ע"פ הנחיות המטלה, נדרש לשלוח הודעת פינג יחידה כל 5 שניות. עם זאת, ע"מ לראות תוצאות יותר מגמתיות, ביצענו שליחה מאסיבית יותר של הודעות פינג ואלו התוצאות:

A graph of a function

Description automatically generatedA graph with blue and black bars

Description automatically generated

A screen shot of a computer

Description automatically generated**Attacker**

A screen shot of a computer program

Description automatically generated

ראשית, נשים לב שהתוקף רץ על תמונה מבוססת GCC/Python. זאת מכיוון שהתמונה של ubuntu לא מכילה את ה-Compiler/Interpreter ומתוך כך לא היה ניתן להריץ/לקמפל את קובץ ההרצה מבלי להוריד את ה-Compiler/Interpreter המתאימים.

השתמשנו בתמונות אלו כדי לייעל את תהליך ההתקפה (בפועל לא משפיע על זמן הריצה, שכן אנו ממתינים 20 שניות לפני ההתקפה).

ב-C: נקמפל את הקובץ המתאים שנמצא בתיקיית ה-volumes. נמתין 20 שניות ולאחר מכן נריץ אותו.

ב-Python: נוריד את התוספים שהקוד שכתבנו דורש. נמתין 20 שניות ולאחר מכן נריץ את קובץ ההתקפה.

התוקף מחכה 20 שניות בשביל שהשרת וה-Monitor יעלו באופן מלא, כלומר שיסיימו את ההתקנות הנדרשות עבורן ויפעלו כרגיל, כך נקבל סטטיסטיקות מלאות מהשרת וה-Monitor.

**קוד:**

הרעיון המרכזי שעומד בבסיס הקוד ב-C וב-Python, הוא יצירת RAW Socket, יצירת פאקטת SYN באופן ידני, ושליחה שלה בלולאה מס' רב של פעמים.

ב-C יכלנו לעבוד עם מצביעים ולשנות את רק הערכים הנדרשים בכל Header בכל שליחה של פאקטה (כתובת ופורט חדשים, Checksum חדש), כך ביצענו את המספר המצומצם יותר של שינויים בכל איטרציה. בעוד שב-Python היינו צריכים להגדיר את ה-Header בכל איטרציה מחדש.

בנוסף, ב-2 התוכניות אנו מבצעים IP & Port Spoofing כדי שהנתקף לא יוכל לחשוף את זהותנו ולחסום אותנו.

**קוד ב-C:**

A computer screen shot of a program code

Description automatically generatedנגדיר משתנים בהם נשתמש בהמשך הקוד ע"מ לתעד את הזמן שלקח לשלוח כל פאקטה בנפרד וביחד.

תיעוד זה יתבצע לתוך הקובץ syn\_flood\_log.txt.

A computer screen shot of a program code

Description automatically generated

נפתח Raw Socket ונגדיר אותו כך שיוכל לקבל Headers "מבחוץ".

A screen shot of a computer

Description automatically generated

נגדיר פאקטה ונאפס אותה.

נגדיר מצביעים למיקומים המתאימים ל-TCP Header ו-IP Header בפאקטה (כך ששינוי שלהם ישנה ערכים בפאקטה).

נגדיר pseudo header שישמש אותנו בהמשך לחישוב ה-checksum ב-TCP Header.

לאחר מכן נקרא לפונקציה ששמה ערכים ב- Headers(נעבור עליה בהמשך).

A computer screen shot of a program

Description automatically generatedכעת נעבור להתקפה – ישנם ערכים ב-Headers שמשתנים בכל פאקטה,

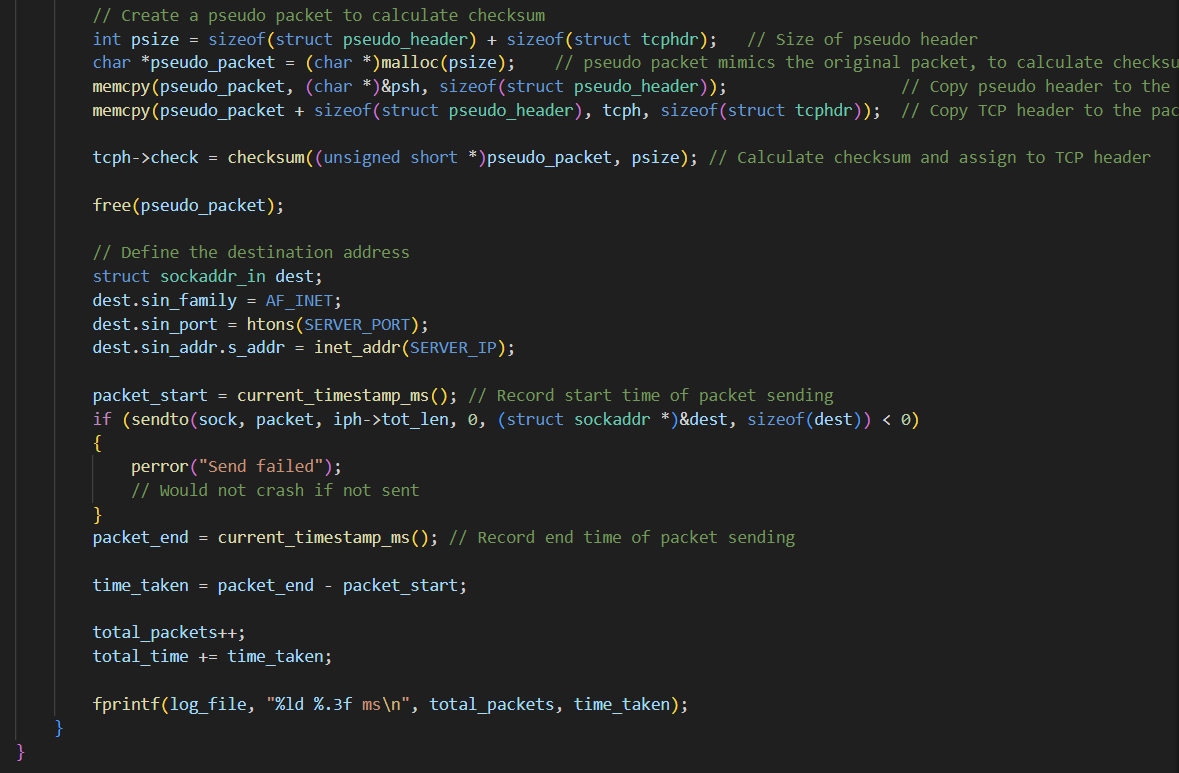
שהם ה-Source Address, Source Port, Checksum.

נשים לב שהקבוע MASK מוגדר להיות 256. כך שכל הערכים יהיו בין 0 ל-255, כלומר שהכתובת אכן חוקית.

כדי לחשב את הפורט, ניקח מספר רנדומלי ונחשב את תוצאת המודולו שלו עם המספר המקסימלי לפורט, פחות 1024 (כמות הפורטים ששמורים לשימוש ע"י המערכת). לבסוף נוסיף 1024 כדי לדאוג שהפורט שבחרנו אינו בטווח של הפורטים ששמורים לשימוש המערכת.

נגדיר את הערכים החדשים הנדרשים ונבצע השמה במקומות המתאימים ב-Headers.

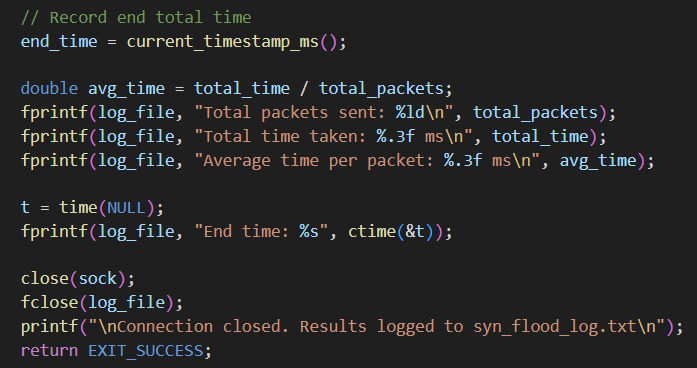
נשים לב בלולאת ה-For, ש-NUM\_OF\_TRIES מוגדר ל-10,000 ו-NUM\_OF\_ITERATIONS מוגדר ל-100 כנדרש.



נגדיר pseudo packet שנועדה לצורך חישוב ה-Checksum של ה-TCP Header.

לאחר מכן נגדיר לאן הפאקטה תישלח (sendto מקבל כתובת בתצורת struct sockaddr. לכן ההשמות שביצענו ל-Headers אינן רלוונטיות כאן).

נשלח את הפאקטה ונחשב את הזמנים המתאימים.

לבסוף, נתעד את כלל הסטטיסטיקות בקובץ ה-log ונסיים את התוכנית.

תיעוד פונקציית handle\_packet:

A computer screen shot of a program

Description automatically generatedמטרת פונקציה זו היא לבצע את ההשמות הנדרשות ב-Headers.

נשים לב שה-Headers מצביעים לכתובות בתוך הפאקטה. לכן העברה שלהם לפונקציה בתור מצביעים – משנה את ערכי הפאקטה במקומות המתאימים ב-Header המתאים.

A screen shot of a computer program

Description automatically generated**קוד ב-Python:**

נגדיר את הקבועים בהם נשתמש בריצת התוכנית.

נפתח את קובץ התיעוד במצב w כדי לאפס אותו (מהריצה הקודמת).

נפתח RAW Socket שמשתמש בפרוטוקול TCP. באמצעות setsockopt נגדיר שה-socket מקבל IP Header "מבחוץ" (כלומר שאנחנו יוצרים אותו).

לאחר מכן נבצע את ההתקפה. נריץ את הפונקציה syn\_flood בלולאה 100 פעמים.

A computer screen with text

Description automatically generatedלבסוף נתעד את הסטטיסטיקות בקובץ ייעודי.

A screen shot of a computer code

Description automatically generatedתיעוד פונקציית syn\_flood:

ראשית, נפתח את קובץ התיעוד במצב append (מכיוון שאנחנו מפעילים את הפונקציה הזו מס' פעמים, נרצה שכל הרצה לא תמחק את התוצאות הקודמות).

לאחר מכן, בלולאה – נגדיר Source IP ו-Source Port רנדומליים. מהם תישלח הפאקטה הנוכחית.

נייצר IP & TCP Headers באמצעות פונקציות עליהם נעבור בהמשך.

נרכיב את הפאקטה מה-Headers ונשלח אותה למטרה.

לבסוף נתעד את הזמן שלקח לשלוח את הפאקטה בקובץ התיעוד.

תיעוד פונקציית create\_ip\_header:

A screenshot of a computer program

Description automatically generated

פונקציה זו נותנת ערכים לתאים שונים ב-IP Header, ולבסוף מבצעת לכל הערכים pack ל-Header יחיד לפי המקרא הבא: B = 1 Byte (unsigned char), H = 2 Byte (unsigned short), 4s = 4 Byte String

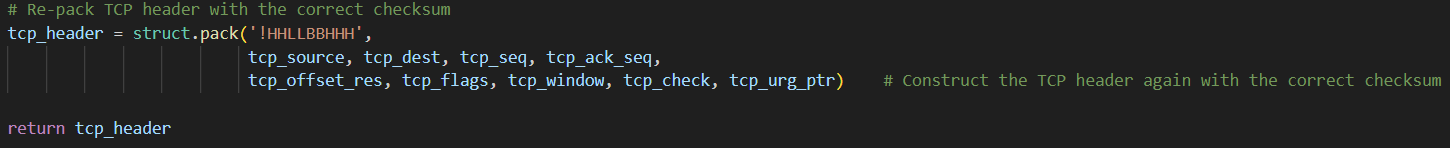
והסימן קריאה בהתחלה מסמן שהמידע צריך "להידחס" בצורת big endian.

נשים לב שהפונקציה מבצעת pack פעמיים. הפעם הראשונה כדי שיהיה ניתן לחשב את ה-Checksum (נדרש header שלם לצורך כך). והפעם השנייה בשביל להכניס את שדה ה-Checksum לתוך הפאקטה.

תיעוד פונקציית create\_tcp\_header:

A screen shot of a computer program

Description automatically generated



פונקציה זו נותנת ערכים למשתנים שמהווים את הבסיס ל-TCP Header.

לבסוף, בדומה ל-IP Header, הפונקציה מבצעת Pack ל-Header יחיד, מחשבת Checksum (כאן אנו משתמשים ב-pseudo header בדומה לקוד ב-C) ולבסוף מבצעת Pack סופי ל-Header אותו מחזירה.

פרטים נוספים:

בזמן ההתקפה, ניתן להיכנס ל-Wireshark ולראות שאכן התוקף שולח הודעות SYN מכתובות IP רנדומליות מפורטים רנדומליים:

A screenshot of a computer

Description automatically generated

נשים לב שהערת ה-TCP Out-Of-Order מופיעה בגלל שלא הגדרנו את שדה ה-SEQ ב-TCP Header כך שיתקדם עם כמות ההודעות שנשלחו.

כמו כן, בזמן ההתקפה כתבנו לקובץ כל שליחה של פאקטה ותיעדנו את הזמן שלקח לכל פאקטה להישלח.

A graph of a number of blue bars

Description automatically generatedמהנתונים האלו הכנו את הגרפים הבאים עבור C ו-Python:

A graph of a number of blue and white bars

Description automatically generated

**מסקנות:**

בסוף הקבצים מהם הכנו את הגרפים, הוספנו את הסטטיסטיקות הבאות:

**C**

A black background with white text

Description automatically generated

**Python**

A black background with white text

Description automatically generated

ניתן לראות ש-syn\_flood.c שלח פאקטות בקצב מהיר יותר מ-syn\_flood.py.

מתוך כך, אנחנו מצפים שהקוד ב-C יגרום להאטה משמעותית יותר בתפקוד השרת אותו אנו תוקפים, שכן יש לשרת פחות זמן בין כל SYN ולכן מקצה משאבים מהר יותר מאשר המשאבים שמקצה עבור הקוד ב-Python – כלומר, אי הזמינות של השרת קורית מהר יותר.

עם זאת, מהגרפים שראינו על ההתפלגות של ה-RTT של הודעות ה-Ping שנשלחו עבור הקוד ב-Python וב-C אנו רואים האטה משמעותית יותר בקוד ב-Python.

משמע ש-syn\_flood.py יצר האטה משמעותית יותר בתפקוד השרת.

ולכן ההנחה שלנו שגויה.