**מטלה 2 - IP and ICMP Attacks Lab**

גיא כץ - 322977265

אברהם דבורה – 315025718

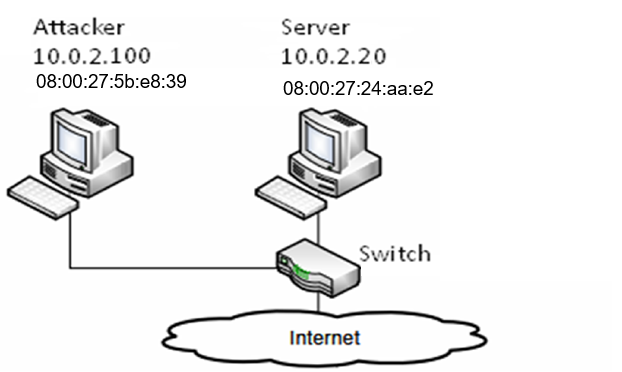
Task 1: IP Fragmentation

* 1. מבוא:

במטלה זו נרצה לבצע לנצל את העובדה שיש פיצול של Packet גדולה לכמה קטנות בתהליך שנקרא Fragmentation, ונראה איך אפשר לנצל אותו לתקיפה.

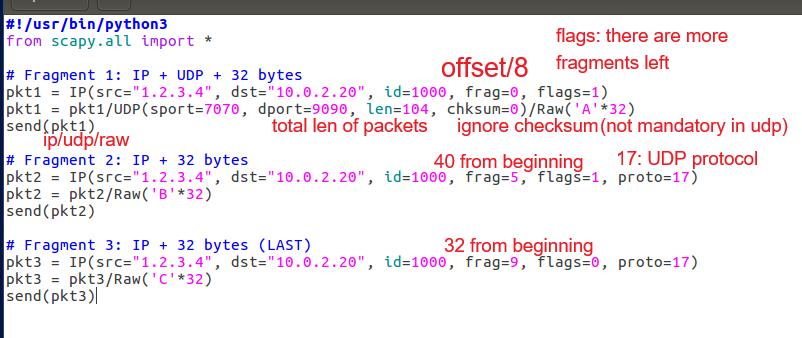
נעשה בהתחלה דוגמא לFragmentation, ולאחר מכן נכין בעצמנו התקפות Tear, Bonk, Ping of Death כמו שלמדנו בקורס הקודם, ונראה איך הן עובדות.

נצפה שPing of Death לא תעבוד, כי מערכות ההפעלה בימינו מכילות הגנות נגדו.

שרטוט הרשת:

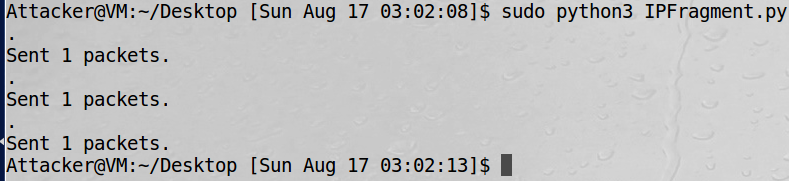
A1.2.1 ביצוע המשימה:

ניצור את הקוד של IP Fragmentation: אנחנו רוצים לשלוח שלוש Packets עם Data בגודל 32, רק לראשון צריך לתת עוד Header בגודל 8

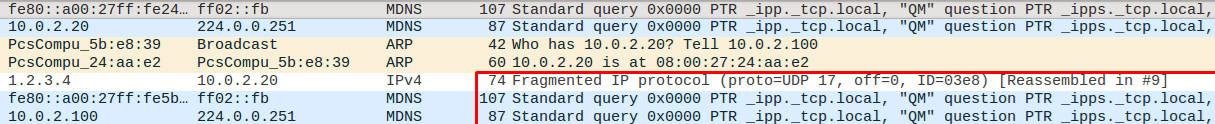


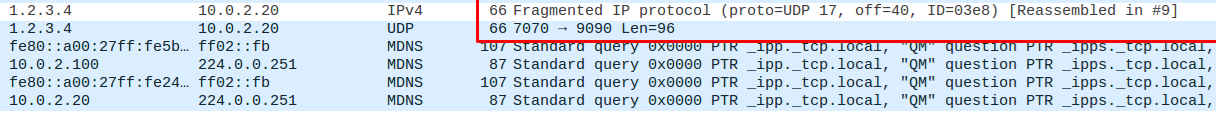
הכנסנו הערות, אנחנו יוצרים את החלק של הIP, UDP רק לראשון, וData בהתאם למספר הPacket.

בServer נרים שרת Netcat בפרוטוקול UDP:

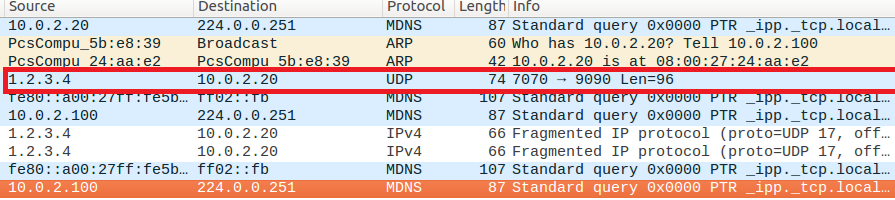
עכשיו כשהשרת מאזין, נריץ את הקוד מהAttacker:

רואים שהקוד רץ, ונשלחות שלוש Packets.

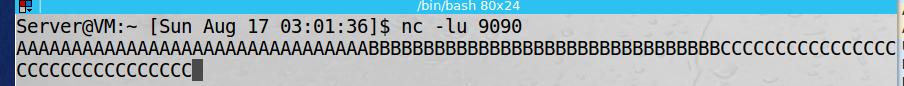
בwireshark של הAttacker:



רואים שנשלחו שלוש Packets. לא היינו בטוחים אם רלוונטי הARP, אם כן היינו שולחים את הMac בתוך התוכנית של הattacker כדי שהרשומה לא תתווסף.

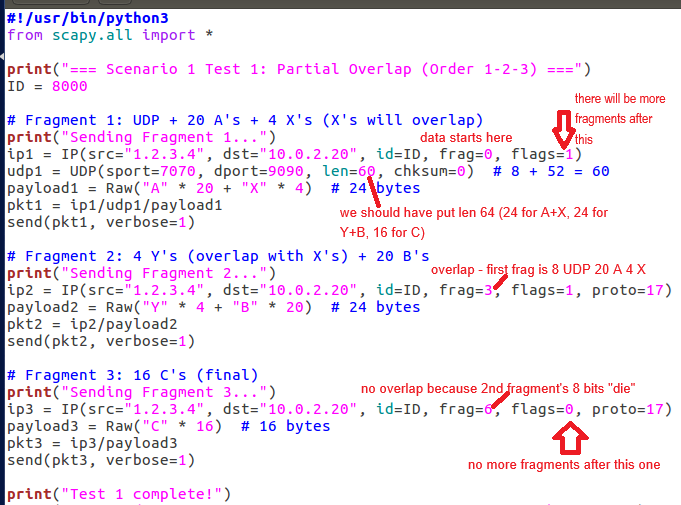
בצד השרת:

אפשר לראות שהPackets התקבלו בצד הServer, ושהוא חיבר אותן לPacket אחת בגודל 96.

נראה את הקבלה בNetcat של הServer:

אכן רואים שהPackets התקבלו, עברו Reassembly ומופיעות כהודעה תקנית בNetcat.

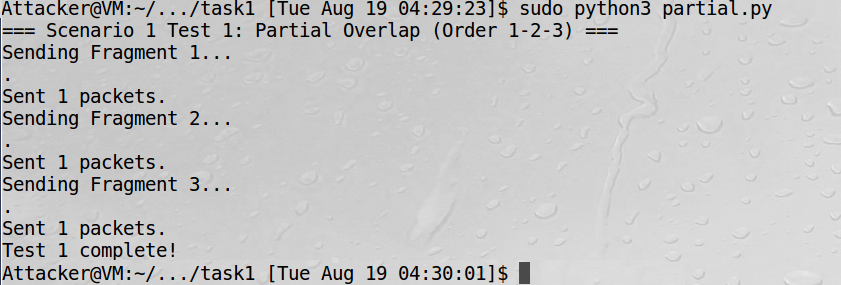
B1.2.1 – ביצוע המשימה

ניצור את הקוד שישלח שלושה Fragments, כאשר שניים מהם חופפים. (Teardrop)

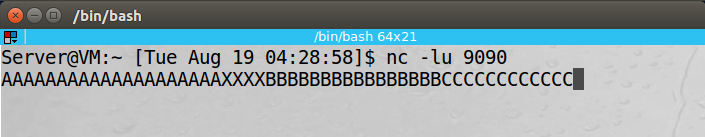
נ

רואים בקוד את שלושת הFragments שאנחנו בונים: השני עם frag נמוך מדי כך שנוצר overlap בין שמונת הביטים הראשונים שלו לבין שמונת הביטים האחרונים של הFragment הראשון.

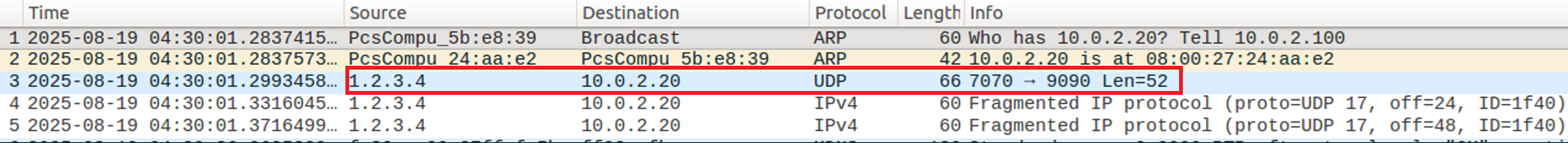
משתנה Flags מסמן אם יש עוד Fragment אחרי הנוכחי או לא, 1 מסמן שיש עוד.

נראה את ההרצה:

רואים ששולחים שלוש Packets, בסדר הנכון.

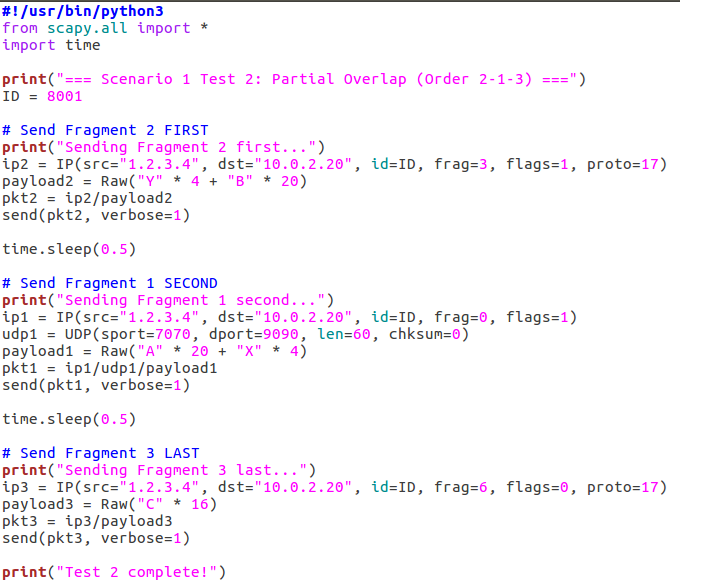
נראה את הצד המקבל: (הServer)

אפשר לראות ששמונה ביטים מהFragment השני "נבלעו", ושהCים נכנסו במקום שלהם בלי overlap עם Bים.

בwireshark:

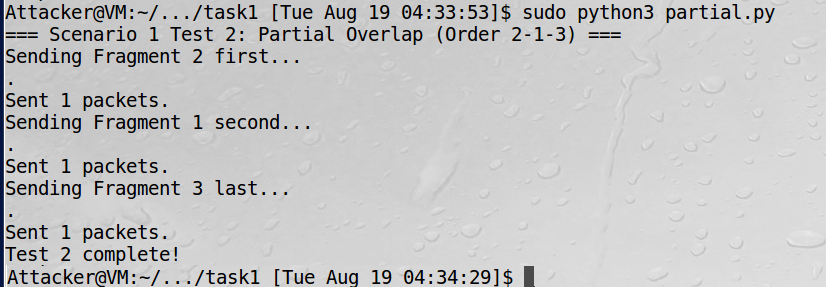
אפשר לראות שהPacket הורכב מחדש וגודלו 52, כי מתעלמים מה8 ביטים של הheader.

נעשה את אותו הדבר עם שינוי בסדר של הFragments, משתמשים כמעט באותו הקוד ומחליפים את סדר השליחה:



שינינו את הסדר של השליחה.

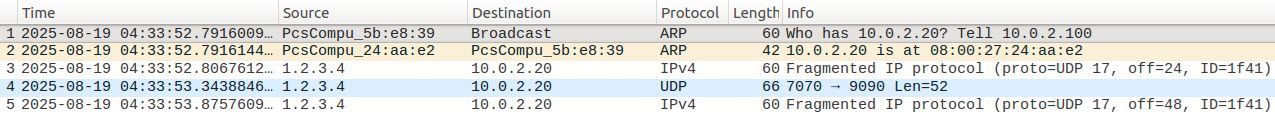
נראה הרצה של הקוד:



רואים שההרצה הצליחה ושנשלחו שלוש Fragments .

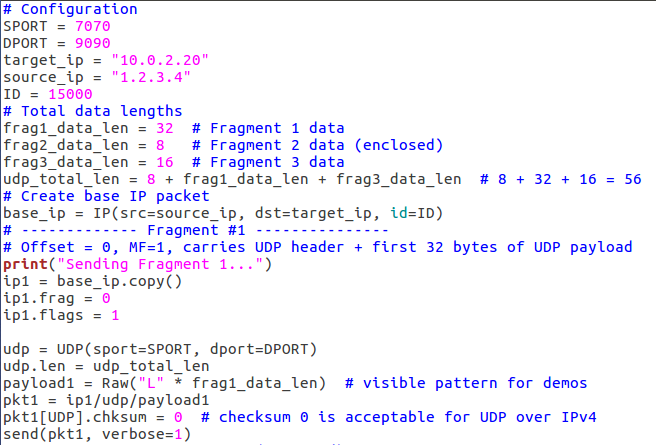
נראה את הצד המקבל: (הServer):

כמו מקודם, הY והB "נבלעים" אבל הServer מקבל את הFragments ומחבר אותם.

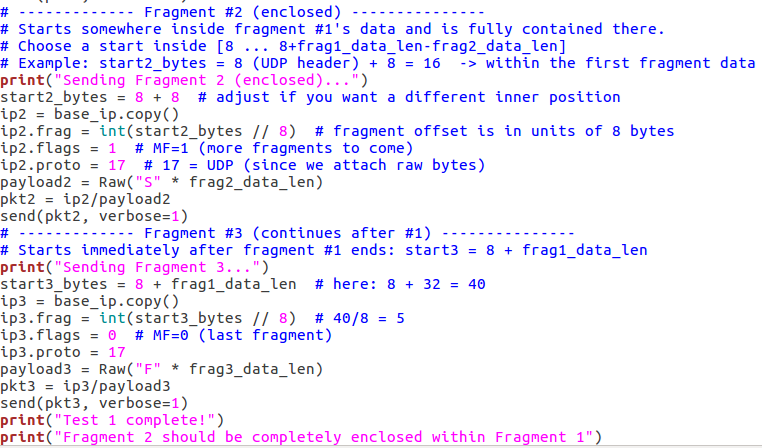
בwireshark:

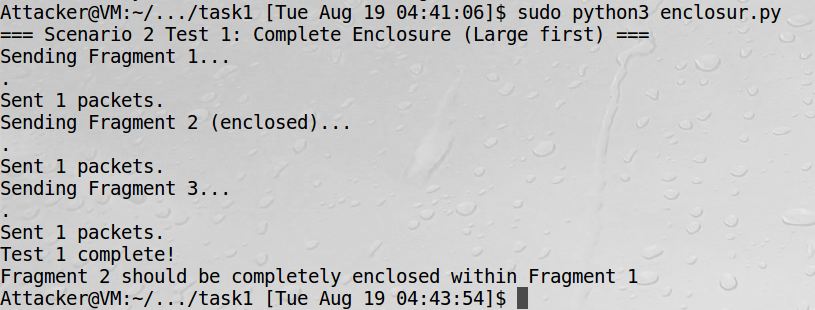
רואים שהPacket נבנה מחדש, פשוט סדר הקבלה שונה (ההרכבה זהה)

עכשיו אנחנו צריכיםFragment שני שכל גודלו נבלע בראשון, ולשלוח בשני סדרים שונים.

נבנה את הקוד, (קרדיט ליאנה וג'ון), הקוד דומה למה שעשינו מקודם:

המשך הקוד בעמוד הבא:

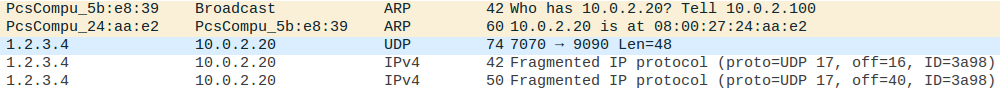


נראה הרצה:

שליחה דומה למקודם.

נראה את הצד המקבל: (הServer):

רואים שכל הSים "נבלעו"

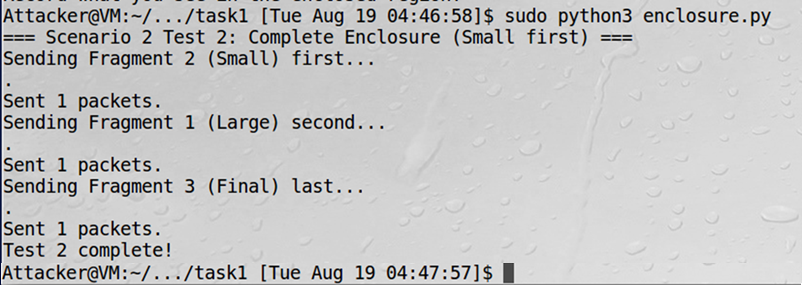
בwireshark:

רואים שהServer חיבר את הPackets חזרה, כמו מקודם יש לנו שלוש רשומות.

עכשיו נראה את אותו דבר עם שינוי סדר: (קוד של יאנה וג'ון בעמוד הבא)



שינינו בקוד את הסדר, אבל עיקר הדברים זהה.

נראה הרצה:

רואים שנשלחו שלוש Fragments בהצלחה.

נראה את הצד המקבל:

רואים שהSים נבלעו שוב.

בWireshark:

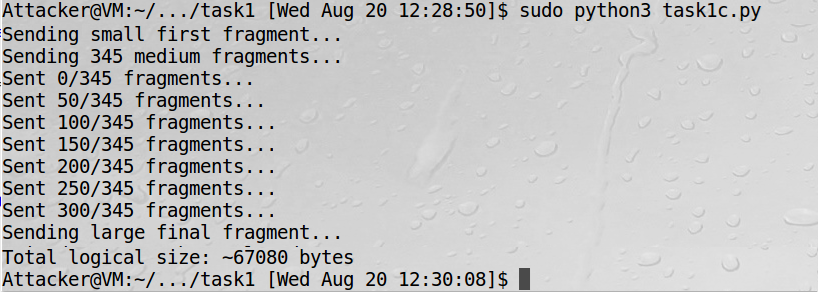
רואים שהServer בונה מחדש את הPacket, ובגלל הReassemble התוצאה זהה.

C1.2.1 ביצוע המשימה

נרצה לשלוח Packet עם offset שיגרום לחריגה מהגודל התקין של UDP: (Bonk)

ניצור את הקוד: (השראה קלה מיאנה)

אנחנו רוצים לשלוח הרבה packets קטנות כדי לעבור את הרף של 65535 ביטים, אז אנחנו שולחים 345 Fragments בגודל 192. אח"כ אנחנו שולחים Fragment עם offset גדול מדי כדי לראות מה קורה.

נריץ את הקוד:

אפשר לראות שאנחנו שולחים את כל הFragments בצורה כזאת שאנחנו חורגים מהגודל המותר.

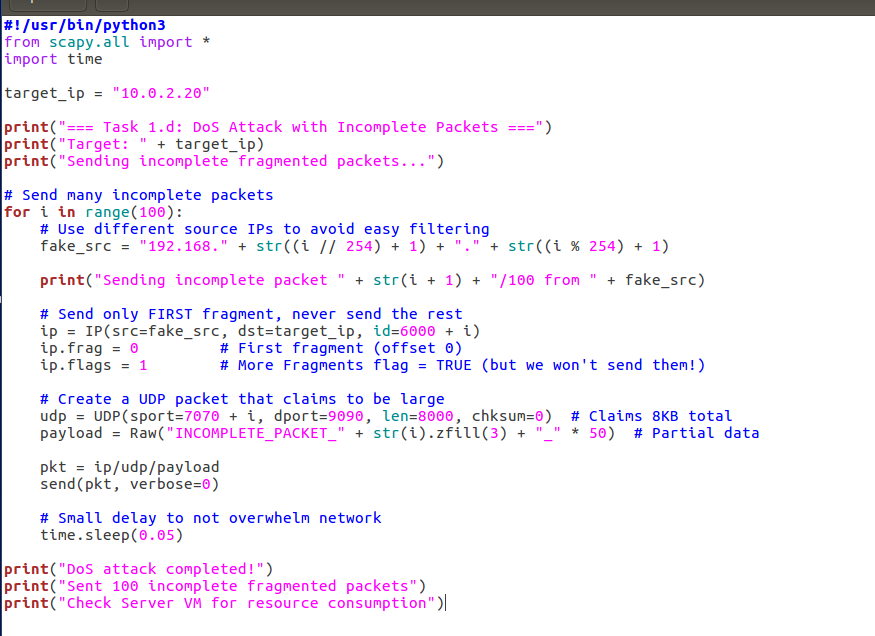
נראה את זה בwireshark: (לא רואים בצד השרת שום פלט)

רואים שמתקבלת שגיאה של BAD UDP LENGTH, כלומר המערכת מזהה שמדובר בPacket גדול מדי ולא מתרגשת.

לאחר מכן אנחנו מקבלים TTL Exceeded שאומר שהיא לא הצליחה לחבר את הPacket.

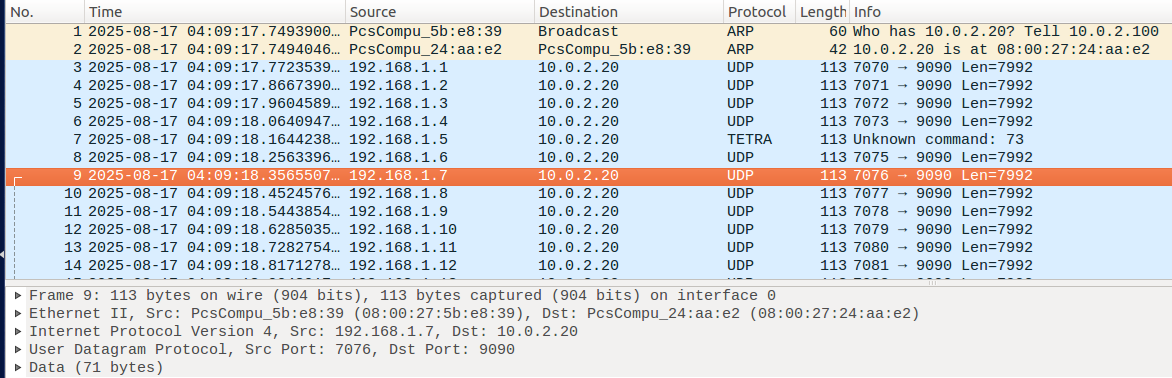
D1.2.1 ביצוע המשימה:

נרצה ליצור Packets שבנויים בצורה לא טובה ולראות אם מערכת ההפעלה תקרוס בעקבותיהם (Ping of Death)

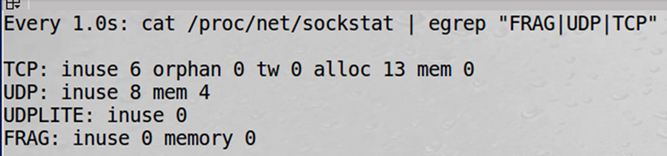
ניצור את הקוד:

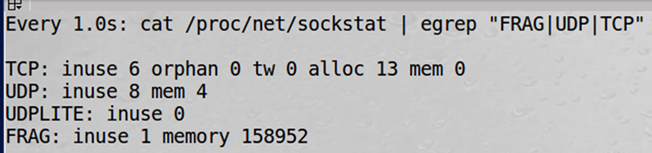
רואים בקוד שיצרנו שיש הרבה Packets שמתחילות ואומרות שיש להן המשך אבל ההמשך לא יבוא. לכל אחת מהן אנחנו צריכים UDP header, ואז אנחנו שולחים את הPacket.

נריץ את הקוד:

נראה את הwireshark של צד השרת:

רואים שאכן מתקבלות אצלו הPackets.

נריץ פקודה שמראה את הזיכרון ([בbytes](https://serverfault.com/questions/628635/what-is-the-unit-of-mem-and-memory-in-proc-net-sockstat)) בcache : (תודה יאנה):

ואחרי ההרצה:

אפשר לראות שיש שימוש בזיכרון אבל בימינו זה לא הרבה (בערך 1 מגהביט)

1.3 סיכום המשימה:

ראינו איך אפשר לשלוח Fragmented Packets, ואז ראינו איך באמצעות הידע הזה אנחנו יכולים לבצע התקפות שונות:

Teardrop – שליחה של Packets עם Overlap בין הFragments. לאחר שעשינו את זה בכמה וריאציות ראינו שגם אם אנחנו משנים את הסדר, הPacket תיבנה לפי הFragments offset ואם יש מידע "כפול" רק החלק הראשון יישאר. כלומר כל פעם שיש overlap, השני שמגיע נמחק או מתעלמים ממנו.

Bonk – שליחה של Packet גדול יותר מהמותר (65535) באמצעות חלוקה לFragments כדי לנסות לגרום למערכת ההפעלה לקרוס, פעם מערכות ההפעלה לא ידעו איך לאכול את זה. ראינו שבwireshark שמתקבלת שגיאה (BAD UDP LENGTH) ושלמערכת ההפעלה לא אכפת מעבר לזה.

Ping of death: שליחת Packets עם Fragments חסרים כדי לתפוס מקום בCache של מערכת ההפעלה, ראינו שגם אם אנחנו שולחים הרבה Packets כמות המידע שנלקח בזיכרון היא זניחה אז בימינו ההתקפה לא רלוונטית.

Task 2: ICMP Redirect Attack

# 2.1 מבוא:

בחלק זה של המטלה אנחנו רוצים לשלוח Spoofed Packetשל Icmp Redirect כדי לגרום למחשב אליו שלחנו לחשוב שדרך המחשב שלנו הנתיב שלו לכתובת באינטרנט יותר יעיל, ואז התעבורה שלו תעבור דרכנו, ונוכל להפוך לMan In The Middle (עם ip-forward)

נרצה לזייף את שדה הSrc לשדה של הGateway, ולשלוח Packet עם ICMP Type 5.

כל העסק יעבוד רק אחרי שנוריד ידנית הגנה בLinux שמבטלת קבלה של ICMP Redirect Packets.

# 2.2 ביצוע המשימה:

נתחיל בלשנות את ההגדרה במחשב הServer שמבטלת קבלת Icmp Redirect Packets:

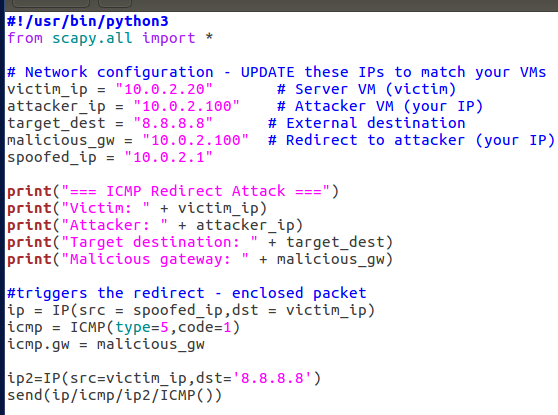


עכשיו בServer כאשר תתקבל Packet של ICMP Redirect היא לא תיזרק אוטומטית.

נתחיל במעקב אחרי טבלת הRoute לכתובת 8.8.8.8:

Watch -n 2 “ip route get 8.8.8.8” בServer:

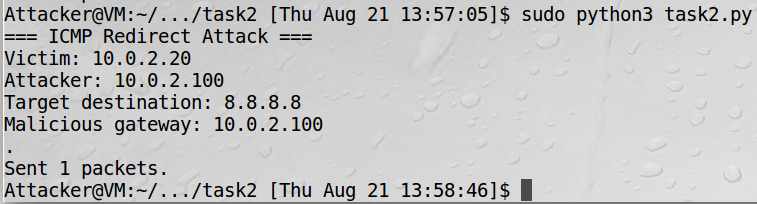
עכשיו נוכל לראות שינויים במקביל לביצוע המשימה.

במחשב התוקף ניצור את הקוד שישלח Spoofed ICMP Redirect Packet:

הגדרנו את השדות בהתאם למה שאנחנו צריכים: כתובות IP של הServer, הAttacker והיעד של הServer.

בונים את הPacket בחלקים: IP, Icmp, ip2:

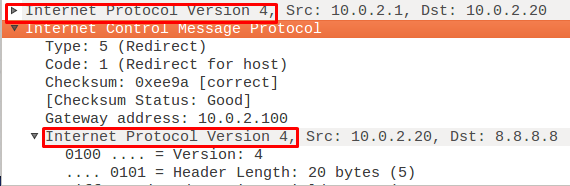
IP הוא הheader החיצוני, ICMP הוא החלק הפנימי ובתוכו הוא מכיל את IP2 שמגדיר מה היעד והמקור לPacket הICMP שאנחנו שולחים.

נריץ את הקוד:

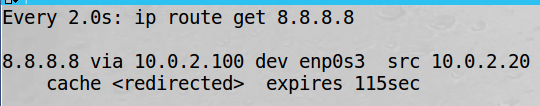
ואכן רואים שנשלחה Packet לServer.

בWireshark:

בפרטים:



נראה שנשלחה Icmp Redirect Packet מהGateway לServer , שתגרום לServer לעבור דרך נתיב טוב יותר, הAttacker.

נחזור לServer ונראה שהשתנה הroute המיטבי:

ואכן רואים שהPackets יעברו דרך הAttacker ולא דרך הGateway.

הדלקנו IP forward בAttacker, אבל כששלחנו הודעות הAttacker שלח בעצמו icmp redirects לServer ו"שרף" את הMITM. לא מצאנו תשובה טובה מספיק למה זה קורה. אנחנו מניחים שמוגדר אצלו הנתיב הטוב יותר והוא נחמד ומעדכן, וצריך לבטל את הנחמדות הזאת.

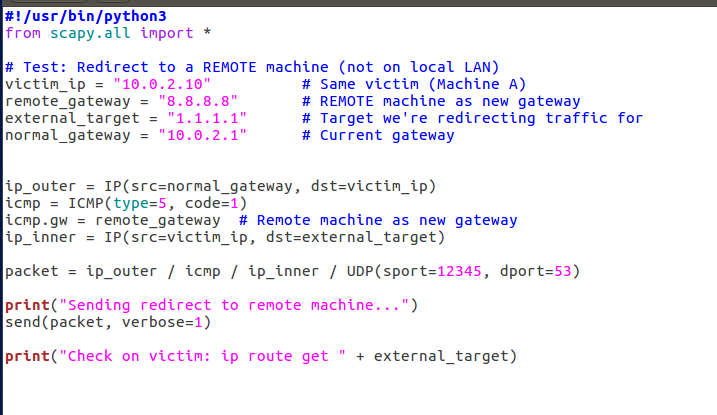
נמשיך לשאלות במטלה:

1. Can you use ICMP redirect attacks to redirect to a remote machine? Namely, the IP address assigned

to icmp.gw is a computer not on the local LAN. Please show your experiment result, and explain your observation.

תשובה: אם נגדיר כGateway יותר טוב host שלא נמצא ברשת שלנו, איך הVictim יידע לתקשר איתו? הוא ינסה לשלוח ARP כדי לדעת מה הMAC שמקושר לIP, אבל בגלל שהם לא באותה הרשת הבקשה תיכשל, אז התשובה היא לא.

נכין קוד כזה:

****

הגדרנו כמו מקודם את הScript של הSpoofed ICMP redirect packet.

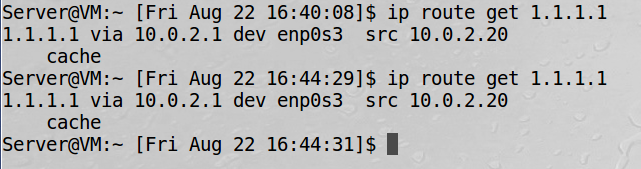
**תמונה שמכילה טקסט, צילום מסך, גופן, לבן

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.**נריץ:

וכצפוי קיבלנו שגיאת ARP, כי הם לא באותה הרשת.

נראה Wireshark:

גם פה יש את הheader הפנימי, נדלג על להדביק אותו.

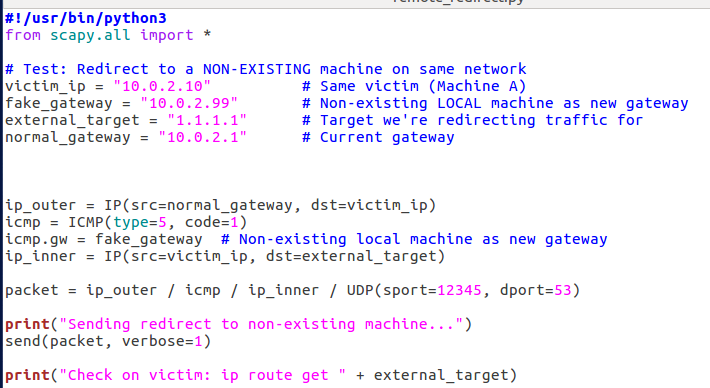
****בServer נראה את הRoutes:

אפשר לראות שהroutes לא השתנו בין הרצת הקוד, כך שהתשובה היא רשמית לא.

בנוגע לשאלה השנייה:

Can you use ICMP redirect attacks to redirect to a non-existing machine on the same network? Namely, the IP address assigned to icmp.gw is a local computer that is either offline or non-existing. Please show your experiment result, and explain your observation.

גם כאן התשובה תהיה לא, מאותה הסיבה: הVictim ינסה לשלוח בקשת ARP ולא יצליח ואז כל העסק ייכשל.

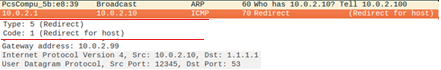
****ניצור את הקוד:

אותו קוד רק לgateway לא קיים.

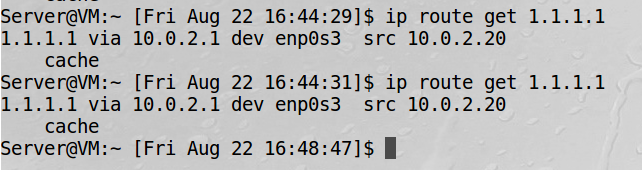
**תמונה שמכילה טקסט, צילום מסך, גופן, מידע

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.**נריץ את הקוד:

רואים שנשלחה הPacket לServer.

נראה בWireshark: (רק את החלקים הרלוונטיים מהתמונה, חתוכים)

רואים שנשלחה הPacket בהצלחה.

****נראה את התוצאה בServer:

אפשר לראות, כמו מקודם, שאין השפעה על הServer.

לכן גם כאן, התשובה הרשמית היא לא.

# 2.3 סיכום המשימה:

ראינו שאפשר באמצעות ICMP Redirect Packets לגרום לשינוי של הנתיב של הVictim כך שיעבור דרך Host שאנחנו בוחרים ושזה ייראה כמו שליחה לגיטימית. כן היינו צריכים להוריד הגנה קודם בלינוקס שזורקת בקשות ICMP שנוגעות לRouters. לפי חיפוש, בWindows אין את ההגדרה הזאת.

לאחר מכן שאלו אותנו אם אנחנו יכולים לבצע את אותו דבר על Host מרשת אחרת או אחד שלא קיים ברשת שלנו, והתשובה היא לא, כי בקשת הARP תיכשל מראש.

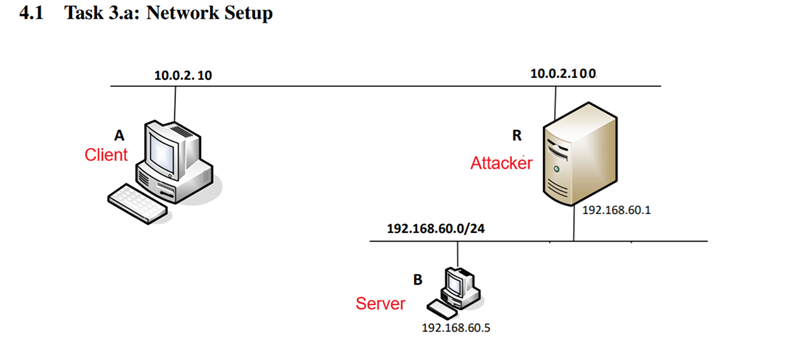
Task 3: Routing and Reverse Path Filtering

# 3.1 מבוא:

במטלה זו נרצה לבדוק את השימוש בפועל של מנגנון בשם Reverse Path Filtering, שבודק אם Packet שנשלחה אליו מבחוץ חוזרת דרך אותו Interface. אם המסלולים א-סימטריים, היא זורקת את הPacket, מה שמונע התחזות מבחוץ לHost ברשת הפנימית.

יהיו לנו מלא הגדרות, כי אנחנו רוצים ליצור שתי תתי רשתות שונות שמחוברות עם Router אחד, ולראות מעבר של Packets ביניהם ואיך הRouter מגיב לשלושה סוגי Packets: אחד מהרשת החיצונית, אחד מהרשת הפנימית (שמגיע מבחוץ), ואחד מהרשת הכללית (האינטרנט).

נצפה שבגלל המנגנון הPacket הראשון והשלישי יעברו בכיף, אבל הPacket השני ייזרק.

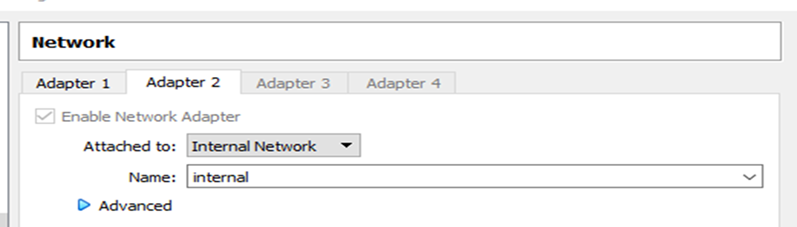
שרטוט הרשת:

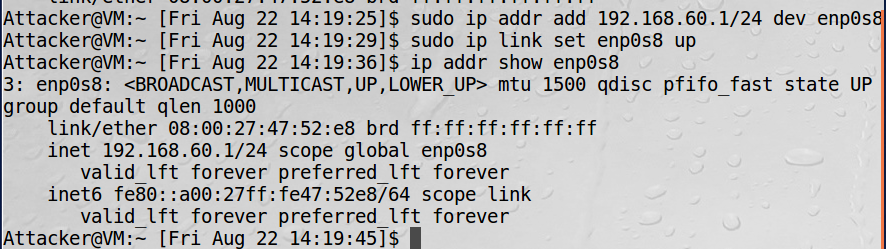
השמות לא מדויקים, השתמשנו במכונות שהיו לנו בשביל הנוחות, בפועל הן יעבדו.

# 3.2 ביצוע המשימה:

הגדרות למיניהן בVM:

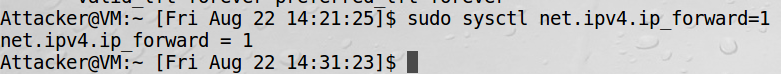
נגדיר את הAttacker להיות הRouter עם ממשק רשת נוסף:



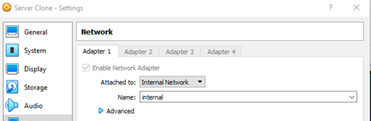
לאחר שהוספנו את הממשק, נגדיר כתובות IP סטטיות לרשת הפנימית:

הוספנו כאן את הRouter לרשת עם הכתובת 192.168.60.1, בפועל התחלנו רשת סטטית.

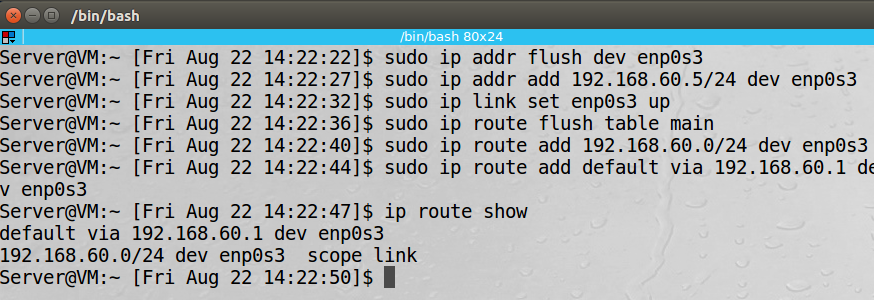
לאחר מכן נפעיל IP Forwarding כדי שהRouter יוכל להעביר Packets ובאמת להיות Router:



אכן רואים שההגדרה עבדה.

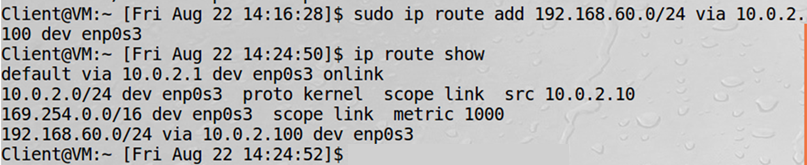
אותו סגנון הגדרה נעשה גם לServer, שיהיה ברשת הפנימית:

שינינו מNAT לInternal Network.

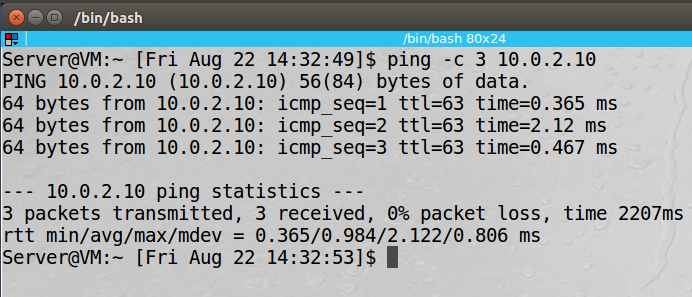
עכשיו נוסיף IP סטטי ונגדיר נתיב לשרת (שיעבור דרך הRouter):

אפשר לראות שהוספנו כתובת IP לממשק החדש שלנו (162.168.60.5), וגם שכRoute ברירת מחדל נעבור דרך הRouter בכתובת 192.168.60.1.

מן הסתם היינו צריכים למחוק Routes ומידע ישן כדי שהHost ישתמש רק במידע החדש.

במכונה של הClient (שנמצא מחוץ לרשת) נגדיר נתיב לרשת דרך הRouter:

הגדרנו Route לרשת הפנימית והגדרנו שצריך לעבור דרך הAttacker כנתב לשם.

נראה בדיקת קישוריות מהServer לClient:

רואים שאכן יש מעבר מידע ביניהם.

תמונה שמכילה טקסט, צילום מסך, גופן, מספר

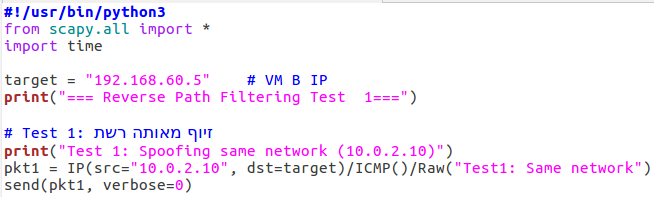
תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.את אותו דבר נראה בין הClient לServer:

רואים שגם כאן יש מעבר מידע, כמו שציפינו שיקרה.

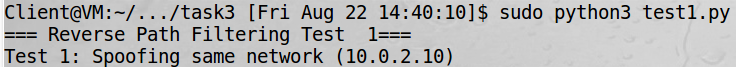
לא עשינו telnet, אנחנו מצפים שיעבוד כי פינג עובד.

כעת נעבור לעיקר, נראה את Reverse Path Filtering בפעולה:

מקרה ראשון: נשלח הודעה מהרשת החיצונית לרשת הפנימית ונראה אם היא מתקבלת.

ניצור את הקוד שישלח Packet מהClient לServer: (קראנו לזה Spoof אבל בפועל זה לא, זה סתם מילה ליופי)

יוצרים Packet עם Source IP ששייך לרשת החיצונית.

נריץ את הקוד:

(verbose = 0 אז לא רואים הדפסת שליחה, לא קריטי)

שלחנו Packet לServer בתור הClient שידוע שהוא ברשת החיצונית. נראה את זה בWireshark:

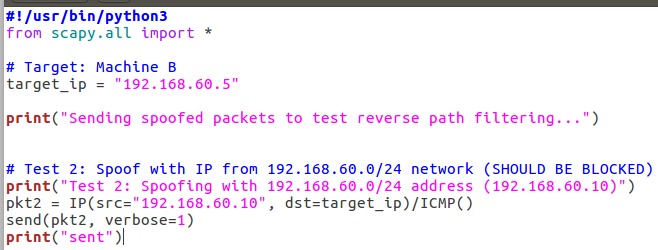


אפשר לראות (בקושי) שאכן נשלח פינג (Echo request) והתקבל reply, כלומר הPacket לא נזרקה כי היא באה מבחוץ.

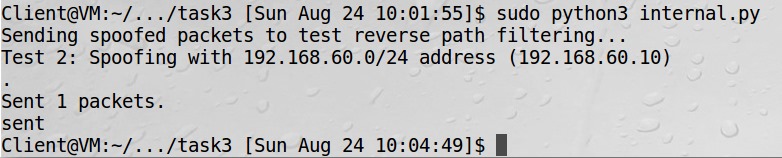
בServer:

גם כאן רואים את הRequest והReply שהתקבלו.

מקרה שני: נשלח הודעה מזויפת (מתחזה לשייכת לרשת הפנימית) מהרשת החיצונית לרשת הפנימית ונראה אם היא מתקבלת.



רואים שהגדרנו כתובת מזויפת לרשת הפנימית ומנסים לשלוח אותה מבחוץ.

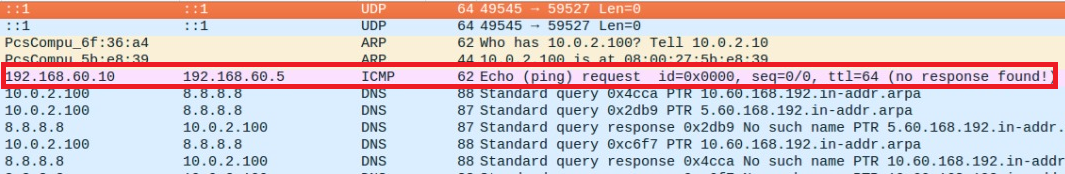
נריץ את הקוד:

רואים שנשלחה הPacket בהצלחה.

תמונה שמכילה טקסט, צילום מסך, גופן, מידע

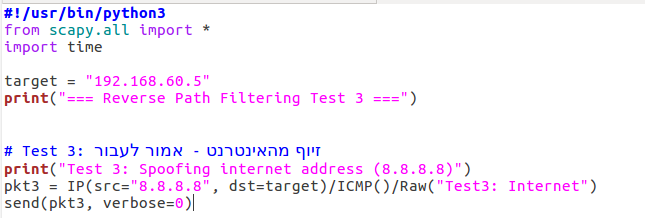
תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.במחשב הAttacker (שמתפקד במטלה זו כRouter) רואים שהתקבלה הPacket:

רואים רק את הRequest, כמו שציפינו, הRouter לא מעביר את הPacket הלאה בגלל Reverse Path Filtering.

בWireshark:

רואים שנשלח Request אבל אין Reply, כי הAttacker (Router) זורק את הPacket, הודות לReverse Path Filtering.

מקרה שלישי: נשלח הודעה מהרשת החיצונית שמתחזה לכתובת מהאינטרנט לרשת הפנימית ונראה אם היא מתקבלת. נצפה שכן, כי Reverse path filtering לא יפריע פה.

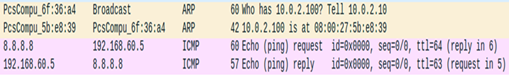
ניצור את הקוד:

אותו סגנון כמו מקודם, רק עם כתובת מהאינטרנט ולא מהרשת החיצונית הישירה.

תמונה שמכילה טקסט, גופן, צילום מסך, לבן

תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.נריץ את הקוד:

(verbose = 0 אז לא רואים הדפסת שליחה, לא קריטי)

נראה בWireshark של הAttacker:

אפשר לראות שאכן בוצעה שליחה והתקבלה תגובה גם

נראה בWireshark של הServer:

גם פה אפשר לראות שהתקבל Request ונשלח reply חזרה.

# 3.3 סיכום המשימה:

בחלק הזה של המטלה ראינו איך להגדיר רשת חיצונית ורשת פנימית עם ראוטר ביניהם, ולאחר מכן ראינו את Reverse Path Filtering בפעולה:

המנגנון בודק אם הPacket חוזרת לאותו Interface ממנו היא טוענת שהגיעה, ואם לא – זורק אותה. כך הוא מונע זיוף של הודעה מהרשת הפנימית אם אנחנו מגיעים מבחוץ (לדוגמא לטובת Fraggle/Smurf Attacks)

בדקנו שלושה מקרים:

הראשון, שליחה הודעה שמתחזה להודעה מהרשת החיצונית אבל לא לפנימית, וזה עבד.

השני, שליחת הודעה שמתחזה להודעה מהרשת הפנימית, ולא עבד, בגללReverse Path Filtering.

השלישי, שליחת הודעה שמתחזה להודעה מרשת האינטרנט, ועבד, בגלל אותן סיבות שהמקרה הראשון עבד (הPacket חוזר לאותו Interface ממנו הוא הגיע).

בעיות:

קושי ראשון היה בהגדרה של כל העסק בVMs, אבל הסתדרנו

קושי שני היה להבין שצריך לבצע Flush להגדרות האינטרנט בServer כדי שיהיה "נקי" ולא ישמור הגדרות קודמות

סיכום המעבדה

במעבדה היו כמה חלקים, שנבנים יחד להבנה של IP, Fragments, וICMP.

בחלק הראשון ראינו איך אפשר לבצע Fragmentation בעצמנו ולשלוח Packets בצורה הזאת, ואז באמצעות הידע הזה ניסינו לבצע התקפות שונות:

Teardrop – שליחה של Packets עם Overlap בין הFragments. ניסינו כמה וריאציות וראינו שהן לא משנות ושמערכת ההפעלה מחברת את הPackets לפי הOffset.

Bonk – שליחה של Packet גדול יותר מהמותר (65535) באמצעות חלוקה לFragments כדי לנסות לגרום למערכת ההפעלה לקרוס. ראינו שבwireshark שמתקבלת שגיאה (BAD UDP LENGTH) ושלמערכת ההפעלה לא אכפת מעבר לזה.

Ping of death: שליחת Packets עם Fragments חסרים כדי לתפוס מקום בCache של מערכת ההפעלה, ראינו שגם אם אנחנו שולחים הרבה Packets כמות המידע שנלקח בזיכרון היא זניחה אז בימינו ההתקפה לא רלוונטית.

בחלק השני ראינו שאפשר באמצעות ICMP Redirect Packets לגרום לשינוי של הנתיב של הVictim כך שיעבור דרך Host שאנחנו בוחרים ושזה ייראה כמו שליחה לגיטימית (מוטיבציה: MITM). כן היינו צריכים להוריד הגנה קודם בלינוקס שזורקת בקשות ICMP שנוגעות לRouters.

לאחר מכן שאלו אותנו אם אנחנו יכולים לבצע את אותו דבר על Host מרשת אחרת או אחד שלא קיים ברשת שלנו, והתשובה היא לא, כי בקשת הARP תיכשל מראש.

בחלק השלישי של המטלה ראינו איך להגדיר רשת חיצונית ורשת פנימית עם ראוטר ביניהם, ולאחר מכן ראינו את Reverse Path Filtering בפעולה:

המנגנון בודק אם הPacket חוזרת לאותו Interface ממנו היא טוענת שהגיעה, ואם לא – זורק אותה. כך הוא מונע זיוף של הודעה מהרשת הפנימית אם אנחנו מגיעים מבחוץ (לדוגמא לטובת Fraggle/Smurf Attacks)

בדקנו שלושה מקרים:

הראשון, שליחה הודעה שמתחזה להודעה מהרשת החיצונית אבל לא לפנימית, וזה עבד.

השני, שליחת הודעה שמתחזה להודעה מהרשת הפנימית, ולא עבד, בגללReverse Path Filtering.

השלישי, שליחת הודעה שמתחזה להודעה מרשת האינטרנט, ועבד, בגלל אותן סיבות שהמקרה הראשון עבד (הPacket חוזר לאותו Interface ממנו הוא הגיע).

לימוד מעבר לדרישות המעבדה

[DoubleDirect](https://zimperium.com/blog/doubledirect-zimperium-discovers-full-duplex-icmp-redirect-attacks-in-the-wild) – התקפה מ2014 על מכשירי פלאפון ברשתות Wi-Fi, באה לנצל את העובדה שבהרבה מכשירים האופציה של Accept Redirects הייתה דלוקה.

בעצם ההתקפה השתמשה בICMP Redirect כדי להפוך לMITM בין המכשיר לאתר מסוים, אבל היא קצת יותר מתוחכמת: כדי לדעת לאיזה אתרים המשתמש עומד לגלוש, ההתקפה גילתה לאיזה אתרים הוא שולח בקשות DNS לפי שרתים שקל לנחש (לדוגמא, DNS משרת הDHCP המשותף ברשתות Wi-Fi, או DNS ברירת מחדל כמו 8.8.8.8 או 8.8.4.4.) לאחר מכן היה אפשר להעביר את כל בקשות הDNS דרך התוקף ואז לדעת את כתובות הIP שהמשתמש הולך לבקר בהן, ואז מבצעים עבורם ICMP Redirect.

לאחר מכן התוקף בעצם הופך לMITM בין האתר ללקוח ויכול לדלות פרטים כרצונו.

יש נתונים (בכתבה של ZIMPERIUM שיש לה פה HyperLink) על אתרים מובילים ושהם גילו את ההתקפה ב31 ארצות, כולל ישראל.

סה"כ זאת דוגמא להתקפה שמשתמשת בICMP Redirect כדי להפוך לMITM כמו שלמדנו בחלק השני של המעבדה, ונחמד לראות שהשתמשו בזה בפועל.