אבטחת מערכות ויישומים ברשת - שיעור #2

חשיבות ה-firewall:

כאשר מחברים רשת פנימית (מוגדרת לקבוצה מסויימת של משתמשים) לרשת חיצונית, למשל האינטרנט (או חיבור רשת פנימית אחת לאחרת), אותם vulnerabilities של הרשת הפנימית ניתנים לניצול לא רק ע"י המשתמשים של הרשת הפנימית (שהנחת סיכוי ניצול משתמשיה הוא נמוך), אלא גם ע"י משתמשים חיצוניים מרשת האינטרנט – כמות המשתמשים שיכולים לנצל את הפגיעויות והסיכוי שישתמשו בהם גדל משמעותית.

סטטיסטית, כאשר מחברים רשת פנימית לאינטרנט, תוך 8 שעות כתובות כל המחשבים בה יוכרו באינטרט ותוך 24 שעות יותקנו maleware על מחשבים אלו.

שרתים על רשת פנימית מכילים את ה-data assets החשובים לארגון, ולכן יש צורך בשליטה על התעבורה אל תוך הרשת ומהרשת החוצה. כמו כן רוצים למנוע חשיפת מידע לא נחוץ על הרשת הפנימית שלנו שכן כל מידע ניתן פוטנציאלית לניצול ע"י גורם עוין. סיבות להגנה על היציאה מהרשת:

- יתכן שהרשת הפנימית שומשה ע"י גורם תוקף.
- יתכן שגורם חיצוני נכנס לרשת הפנימית כדי להוציא החוצה מידע סודי אל מחוץ לרשת. data leackage prevention – DLP – מניעת זליגת מידע החוצה.

אז מטרת ה-firewall היא לאפשר תקשורת מצד אחד אך לבודד / להגביל את התקשורת תחת הגדרות מדוייקות מצד שני. מטרת ה-firewall הוא נקודת החיבור בין הרשת הפנימית לחיצונית.

אנו נתייחס לרמת ה-transport – העברה של packets, כאשר messages לרוב מורכבות מכמה packets ברמת הרשת (packet = data grahm). ה-packets יבחנו ע"י סט חוקים לפיהם יחולקו ל-packets שמותר להן לעבור לעומת כאלו שאסורות למעבר.

הגדרות ל-firewall:

ניתנת למימוש בחומרה, תוכנה או שילוב של שניהם ומטרתה למנוע מתוכנות לא מורשות או משתמשי אינטרנט מחדירה לרשת פנימית או מחשב בודד.

ה-firewall יכול להיות ברמת רשת או ברמת מחשב בודד – personal firewall היושב בין הרשת למחשב, לעומת network firewall – בין רשתות. ה-firewall מייעלת ומפשטת את תהליך האבטחה בכך שמצמצמת לנקודה אחת את ניהול חוקי אבטחת הרשת, לעומת טיפול בלקוחות רבים (מחשבים).

כיום ב-firewalls מתקדמים ישנם מנגנונים מתקדמים לניתור תעבורה לפי סט חוקים שמוגדר ע"י מנהל הרשת – טיב ה-firewall הוא לכל היותר כטיב סט החוקים / קונפיגורציה שלו. ה-firewall יושב בקצה הרשת בנקודה היציאה / מעבר לרשת אחרת, למשל בין LAN ל-WAN, או בין LANs.

Intro to TCP/IP:

ה-IP stack: מודל הדומה למודל ה-5 שכבות: רמת האפליקציה, למשל FTP או HTTP (מקביל לרמות 5-7); לאחר מכן רמת ה-transport – TCP; תחתיה רמת ה-protocol – IP; לבסוף הרמה הפיסית, רמת ה-link. ה-drivers של ה-TCP רצים ברמת ה-kernel הרץ ברמת יעילות גבוהה ולכן נרצה שה-firewall ירוץ ברמות אלו, כדי שירוץ ברמת הקרנל. בעבר רצו ברמת ה-user ביעילות נמוכה יותר. לכן, ה-firewalls רצים ב-kernel mode, לכן ברמת ה-TCP/IP.

- קובץ ברמת האפליקציה
- ברמת ה-presentation נוסיף Header ו-Trailer
- ברמת ה-session נפרק לחלקים רטנים עם header ו-trailer כל אחד.
- ... התהליך חוזר על עצמו בכל אחת מהרמות הבאות: transport, network, data link ו-physical. כל חלק מתקבל עם סדרת headers ו-trailers.

מתוך מבנה ה-IP header (מצגת 2, שקופית 13) נשים לב ל-destination ip address ו-source ip address – זאת ברמת ה-network. ברמה מעל, רמת ה-transport, פרוטוקול ה-UDP לא דואג ל-guarinty delivery אך הוא יעיל ביותר. תחת הנחה שאין אובדן packets גבוה, ולכן פרוטוקול זה נמצא בשימוש נרחב (כמו למשל ע"י ה-DNS). ב-header של ה-UDP, שהוא חלק מה-IP יש את ה-source port ו-destination port. (הערה: ה-header של ה-UDP נמצא כחלק מה-DATA של רמת ה-IP).

ה-TCP דומה רק שהוא נותן guarinty delivery לעומת ה-UDP. מה שחשוב לנו מה-header של רמת ה-TCP הוא ה-source port, destination port ושני flags – syn, ack. כדי להבטיח העברה נכונה מהמקור ליעד, ישנו בפרוטוקול ה-TCP שלב handshake בו בונים את ה-connection, המתחיל בשליחת syn message, בה דולק ביט ה-syn. ב-syn msg יש sequence num – הודעה על מספר אקראי כלשהו שנבחר כאינדקס התחלה ל-packets הנשלחות ל-server. ה-Server עונה ב-ack ל-client כתגובה על ה-syn, ושולח בעצמו syn לכיוון השני.

- $client \xrightarrow[syn_c]{} server$
- $server \xrightarrow[ack_s,syn_s]{} client$
- $client \xrightarrow[ack_c]{} server$

Packets constraints:

ב-UDP או TCP לכל packet יש ip, port ל-source ול-destination. ה-IP נלקח מה-ip header, הפוטים נלקחים מה-UDP/TCP header. בנוסף, ב-TCP יש את ה-syn/ack המרמזים על packet ראשון ליצירת connection.

Packet filter:

במודל פשטני של firewall, נניח שמאפשרים יצירת connection ע"י מישהו חיצוני. זיהוי syn מ-source שמקורו מחוץ לרשת הפנימית – יצירת connection ע"י חיצון לרשת – יסונן. אם ה-syn בא ממישהו פנימי ברשת – התקשורת תאופשר. בעיה: דחיית כל incoming TCP connections יכולה לעצור תקשורת HTTP – הפועלת מעל ה-TCP. לכן יש צורך ב-**exceptions.**

דוגמא נוספת לבעיה: שירות ה-DNS המתרגם שמות לוגיים לכתובות IP (domain name service). שירות זה צריך להיות מהיר ביותר ולכן נשלח על packet בודד ועובד מעל פרוטוקל UDP בפורט (בד"כ) 53. אם נחסום את ה-DNS, הדרך היחידה לתקשורת המשתמשים לשרתים חיצוניים היא ע"י ידיעת כתובות ה-IP שלהם, ולכן יש צורך ב-exception גם ב-UDP, ולא רק מעל TCP. ה-DNS הוא דוגמא לפרוטוקול אפליקטיבי מעל רמת ה-transport, כמו HTTP.

לרוב פרוטוקולים אפליקטיביים מזוהים ע"י פורטים סטנדרטיים, כמו 80 ל-HTTP ו-53 ל-DNS. כך ברמת ה-transport מזהים רמה אפליקטיבית. בעקרון כל רמה לא יודעת מה יש מעליה, אך כן רוצים יכולת סינון מסויימת ברמת ה-transport עבור רמת ה-application. ה-port נותן יכולת הבנה לגבי הפרוטוקול האפליקטיבי דרך רמת ה-transport דרך הקונבנציות הנ"ל. כאשר packet אינו חוקי, ניתן לעשות לו drop – השמטתו ללא הודעה לצד השולח, או reject – זריקה והודעה לצד השולח שה-packet לא מועבר.

ישנם שני סוגים של packet filters:

1. Standard packet filter: stateless:

מחלץ פר packet את המידע הנחוץ לו לצורכי סינון תוך התעלמות מ-scope רחב יותר. החוקים מוגדרים כאמור לרוב בהקשר לפרוטוקול האפליקטיבי, המזוהה ע"י ה-port. במצגת יש פירוט על הפרטים עליהם מסתכל הפילטר הסטנדרטי (שקופית 28).

<u>דוגמאות לבניית חוקים</u>:

מה-security policy נגזר ה-security mechanism. מה-policy תבנה הקונפיגורציה של ה-mechanism. היישום ב-firewall:

- ה-policy: איסור גישת רשת חיצונית; ה-firewall setting – זריקת כל ה-packets היוצאים לכל IP שהוא, בפורט 80.
- ה-policy: תקשורת חיצונית לשרת רשת פומבי בלבד; ה-setting: זריקת כל TCP SYN packet לכל IP פרט ל-IP מסויים (זה של השרת המוגדר) בפורט 80. כך מתאפשרת פתיחת connections מבחוץ רק לשרת מסויים ורק בפורט מסויים.
- ה-policy: מניעת IPTV מתפיסת ה-bandwidth; ה-setting: זריקת כל פקטת UDP נכנסת פרט ל-DNS ותשדורת נתב.

דוגמאות ICMP (internet control message protocol): מניעה מהרשת לשימוש ל-smurft DoS attack – זריקת פקטים של ICMP לצאת ל- broadcast ip – כתובת אליה מופנים תשובות שרתים אליהם ניגשים מהמחשב הנוכחי (הפניית תשובות במקום אלי למישהו אחר – יכול להיות כתוצאה מתקיפת עומס על אותו שרת אחר).

<u>Access control lists</u>:

כאשר דיברנו על Positive security logic לעומת negative, אמרנו שב-positive אנו מגדירים מה מותר, וכל מה שלא הוגדר כמותר הוא מהגדרה אסור. במצגת ניתן לראות חוקים המוגדרים כ-allow, פרט לאחרון שהוא deny על הכל – כל מה שלא הוגדר עליו ספציפית כמותר.

כאשר מגיעה packet, ה-firewall עובר חוק חוק בטבלה, ומחפש חוק שיאפשר לו מעבר. אם לא נמצא חוק מעבר, החוק האחרון מגדיר לאותה packet כ-denied. כיוון שה-firewall עובד ב-positive security logic, מה שלא הוגדר כמותר למעבר ייחסם (סוג של חיסרון).

ניתן לעבוד גם לפי negative security logic, בה יש חוק default allow policy – כאן מגדירים מה נחסם, וכל מה שלא הוגדר כחסום מועבר. כמובן, ה-default deny policy יותר בטוח, ודורש ניהול מסודר של הרשת. ה-default allow policy יותר גנרי והרבה פחות בטוח.

נשים לב שניתן בקלות לבטל את פעולת ה-firewall ע"י הפיכת הכלל האחרון מ-deny ל-allow. ניהול firewall בסביבה אמיתית הוא מאוד מורכב. כאשר מערבבים חוקי allow ו-deny במקום הגדרת default, ניתן להגיע לסתירות. כאמור, ניהול ה-firewall מגדיר את טיב ההגנה שמספק, ולכן ניהולו מאוד משמעותי.

2. Statefull firewalls:

אלו firewalls המסתכלים על scope רחב יותר מאשר packet בודד. למשל, כאשר נבנה connection, אם בניית ה-connection אושרה, כל ה-packets על אותו connection לא צריכים להיבדק, כי הוא כבר נבדק. לכן תוחזק טבלה של connections פתוחים שעברו תהליך אישור, וכאשר יש טבלה כזו, ברגע שמתקבלת packet תחילה בודקים את הימצאותה בטבלה. אם נמצאה – אין צורך לבדוק אותה (כמובן, עבור תעבורת TCP – רוב התעבורה). הטבלה תהיה hash table כדי שהתהליך יהיה יעיל ומהיר. במצב זה יש יכולת בצורה מהירה יחסית לזהות שיוך ל-connection קיים, לפי זה ניתצן לדעת את ה-state של ה-connection. אם זוהי תקשורת חדשה, כל החוקים של קודם חלים.

יתרון: ייעול התהליך; חסרון: זיכרון רב, ולכן המנגנון פגיע להתקפת availability. יש צורך לפיכך במנגנון ניקיון הטבלה. ישנם connections שלא יסגרו באופן מסודר עם FIN flag, ולכן יש צריך לנהל inactivity timeout לכל connection לצורכי ניקיון הטבלה. בעיה: תתכן עבודה עם אפליקציה שלא היתה בשימוש זמן כלשהו, ה-TCP connection פתוח, אך ה-firewall כבר סגר את ה-connection. כעת, ב-packet הבא שישלח, שישלח ללא syn, יזרק - כיוון שאין connection כנגדו (נזרק ע"י ה-firewall מהטבלה). במקרים כאלו צריכים שימוש בשליחת keep alive בין ה-client ל-server ולהיפך כדי לשמור על חיות ה-conn.

סיכום: Stateless דורשים פחות זיכרון ולא חשופים לבעיות availability, אך sessions שעובדים לאורך זמן פחות יעילים, בפרט כאשר יש חוקים רבים. ישנן כמה בעיות ביניהן למשל ההנחה על הקשר בין פורט לאפליקציה. זה מביא אותנו למנגנון הבא:

NAT (Network Address Translation):

מטרתו של המנגנון להסתיר את הכתובות הפנימיות מהעולם החיצוני. תחילה, המנגנון הוקם בשל צורך גודל מרחב הכתובות – שני שרתים ברשת בעלי כתובת זהה מהווים בעיה על אותה רשת. פתרון ה-NAT: ניהול כתובות פנימיות ברשת הפנימית באופן נפרד ממרחב כתובות איתם יוצאים החוצה לרשת החיצונית, רשת האינטרנט. ה-ISP (internet service provider) מקבל מארגון בינלאומי כלשהו מרחב כתובות המוקצות לו, והוא מקצה אותם באופן ייחודי למשתמשיו. ה-NAT מתרגם בכל יציאה לרשת האינטרנט כתובת פנימית לחיצונית.

היתרון מבחינת אבטחת מידע: רואים כתובות חיצוניות ולא פנימיות, ומבחוץ לא ניתן לדעת את הכתובות הפנימיות של השרתים. נשים לב שהכתובת החיצונית היא כתובת דינאמית המוקצית ברגע נתון לשרת היוצא החוצה, ולכן לא ניתן לשייך כתובת חיצונית לשרת פנימי מסויים. ה-NAT שומר טבלת מיפוי בין כתובות פנימיות לכתובות חיצוניות, וטבלה זו היא דינמית. בכל גישת שרת פנימי לרשת, ה-NAT מקצה אד-הוק כתובת חיצונית, ישמור את המיפוי ויחליף את הכתובת ב-packet (שינוי ה-source ip). ה-NAT מחזיק מיפוי IP וגם PORT – גם רמת ה-network וגם ה-transport.

ה-NAT כיום הוא לרוב חלק מ-router. ה-firewalls כיום מורכבים משני מנגנונים לפחות – packet filtering ו-NAT. שני מנגנונים אלו משלימים אחד את השני – פתרון בעיית ניהול והסתרת הכתובות הפנימיות וסינון פקטות לא מאושרות.

סוגי NAT:

נבדלים בחומרתם על המקורות המורשים לגשת לכתובות החיצוניות.

- Full cone: המקל ביותר; כל כתובת פנימית ופורט ממופים לאותה כתובת חיצונית ופורט. כל IP וכל פורט יכולים לשלוח חזרה לכתובת החיצונית.
- Restricted cone: מאפשר קבלת packets רק מכתובות IP אליה יצאתי; עדיין יהיה מוכן לקבל מכל הפורטים באותו IP. בניגוד לקודם, ה-IP מוגבל אך הפורטים לא.
- Port restricted cone: מאפשר לקבל פקטות רק מ-IP ופורטים מסויימים, אלו שנשלחו כ-destination מהרשת הפנימית. רק הם יוכלו לגשת חזרה.
- Symmetric: בתרגום מכתובת חיצונית לפנימית, התרגום אינו מבוסס רק על הכתובת הפנימית, אלא גם על ה-destination. המשמעות היא שיציאה לשני שרתים שונים בחוץ מאותו שרת תהיה דרך IP חיצוניים ופורטים שונים.

סיכום: טבלה נוחה בשקופית 44.

בעיה: כאשר רוצים לחבר בין שני מחשבים ברמת האפליקציה, למשל בהעברת מדיה, ברמה האפליקטיבית כתובת ה-IP נשארת הכתובת הפנימית, וה-NAT לא נוגע ב-packet (וגלא יודע איך) ברמת ה-DATA. כך, אפליקציה בשרת החיצוני אליו ניגשים חושב שה-IP שלו הוא הפנימי. דבר נוסף, אפליקציה לא יכולה לראות את הכתובת החיצונית שלו. לכך ישנו שירות המאפשר לאפליקציה לראות את הכתובת שלה כלפי חוץ וכך לאפשר ברמת האפליקציה לראות את הכתובת החיצונית של המחשב הפנימי. <u>הערה</u>: לא יעבוד ב-symmetric (מן הסתם).

הבעיה הבאה איתה צריך להתמודד: הגבלת outgoing:

DMZ: demilitarized zone – "אזור מפורז", לא נמצא ברשת החיצונית אך גם לא בפנימית, אלא יושב ביניהם. מצד אחד הוא נגיש לעולם החיצון, והסיכון שלו להיחשף להתקפה גבוה, ולכן ננתר את התקשורת בינו לרשת הפנימית. מטרת ה-DMZ להגן על הרשת הפנימית ע"י ניתור התעבורה בין שרתים חיצוניים לפנימיים.

תוקף המגיע מהעולם החיצון יכול לקבל נגישות רק לשרתים הנמצאים ב-DMZ ואין לו גישה לשרתים ברשת הפנימית. גם אם יצליח להשתלט על שרת ב-DMZ, כיוון שהתעבורה בין ה-DMZ לפנימיים מוגבלת, הסיכוי שיצליח לנצל את זה כדי לתקוף את הרשת הפנימית קטן משמעותית.

ישנן שתי תצורות מרכזיות ל-DMZ:

1. <u>Dual firewall architecture</u>: השמת שני firewalls שונים ומופרדים בין הרשת החיצונית ל-DMZ ובין ה-DMZ לרשת הפנימית. הרווח: השימוש בשני firewalls נפרדים מאפשר כתיבת חוקים שונים בין אחד לשני. החיצוני, frontend, מקונפג לאפשר גישה רק לשרתי ה-DMZ והוא יקבל בד"כ יותר traffic (המגיע מהעולם החיצון. על ה-backend יוגדרו כללים נוקשים יותר, לא תתאפשר incoming conn מה-DMZ לרשת הפנימית, אלא רק הפוך. אם נשתמש ב-firewalls מספקים שונים, אז אם לאחד יש פגיעות כלשהי, הסיכוי שלשני תהיה אותה פגיעות נמוכה. החסרון הוא המורכבות והעלות של אחזקת שני firewalls משני vendors שונים. אבל, כיוון שכל אחד מנוהל בנפרד מוריד את הסיכוי שטעות בקונפיגורציה תחלחל מאחד לשני. הערה: אין יוזמת תקשורת מה-DMZ פנימה, אלא רק מהפנים החוצה. כך רק שרתים פנימיים יכולים ליזום את התקשורת החוצה (ואז יקבלו response וכן הלאה).
2. <u>Single firewall</u>: יתרונות: עלות נמוכה יותר, פחות מורכב מבחינת אחזקת ה-firewall. ל-firewall הזה יהיה 3 "רגליים" – 3 network interface – אחד ברשת החיצונית, אחד ב-DMZ ואחד ברשת הפנימית. חסרון: טעות בקונפיגורציה של ה-firewall עלולה לחשוף את כל הרשת, אין 2 רמות אבטחה.

בשקופיות יש דוגמא לקונפיגורציה של firewall. חשוב לשים לב לכך שה-firewall מבצע access control ברמת הרשת, ולכן עקרון ה-least privileges תקף כאן.

Firewall between network segments:

ה-Firewall יכול לשבת על רשת עם "הרבה רגליים", אך ניהול כל רגל הוא בפני עצמו, נפרד, וישנו מקום רב לטעויות בקונפיגורציה.

ניהול firewall: הנקודה החשובה ביותר: סט כללים ברור הכרחי לפעולה נכונה של ה-firewall, וללא סט כללים מוגדר פעולתו מיותרת.

<u>מדוע firewall לא מסייע לחסימת פגיעות ברמת האפליקציה</u>:

ה-firewall מגדיר חוקים עד רמת הפורט, ולכן אם פורט כלשהו פתוח, ישנה פתיחות להתקפה – אין ל-firewall חוקים לבדוק האם ה-HTTP request המתקבלת היא חוקית או לא, ה-firewall יעביר בין כה וכה. כיום ישנם deep-packet inspection firewalls, המתייחסים ל-data בצורה מוגבלת. אבל, כדי להסתכל על ה-DATA ולחפש request לא חוקי, יש צורך לחבר Packets לכדי request message מלאה, וגם תחת הנחה שה-firewalls יודעים לבצע זאת, ה-firewalls מעבירים הרבה פרוטוקולים, וכאשר התעבורה מוצפנת ה-DATA סגור לחלוטין מול ה-firewall.

העובדה שה-firewalls פותחים פורטים רבים, וכאשר מעבירים תעבורה מוצפנת ב-SSL, אז ברמה האפליקטיבית ה-firewall לא מגן מפני ניצול פגיעויות.