Port Knocking

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

השלב הבא בפריצה לאחר שהתוקף הגדיר לו מטרה הוא להשיג עליה כמה שיותר נתונים. אחד החלקים העיקריים בשלב הזה הוא למפות את המטרה, מבחינה תשתיתית. מיפוי המטרה יכול לכלול מספר נסיונות:

- נסיונות לגלות אילו **פורטים** פתוחים על השרת.
- נסיונות לגלות אילו **פרוטוקולים** משתמשים בפורטים האלו.
- נסיונות לגלות לאילו **שירותים** משמשים אותם פרוטוקולים.
- נסיונות לגלות את **גירסאותיהם** של השירותים.
- נסיונות לגלות **חולשות** באותן גירסאות.

ברור שקיימים עוד מהלכים שבהם תוקף יכול להשתמש בכדי למפות מבחינה תשתיתית את מטרתו, נתמקד במסמך זה באלה.

למה אנחנו צריכים לדעת את זה?

נניח תוקף קבע מטרה, שרת מרכזי של איזה ארגון, הוא סורק את השרת בעזרת NMAP או משהו בסיגנון, מקבל רשימת פורטים פתוחים, מגלה שהפורט 22 פתוח על השרת, מה שאומר שאם מנהל הרשת לא היה חכם מדי, כניראה מדובר בשירות SSH, התוקף יודע היטב שבכדי להתחבר לשרת ה-SSH הוא צריך שם משתמש וסיסמא, אבל מה, אחרי נסיון ההתחברות, הוא פתאום מגלה מהבאנר, שמדובר בשרת OpenSSH v3.3.

זהו, Game-Over, כולנו מכירים את ה-Buffer Overflow שקיים בגירסאות 2.9.9-3.3 של OpenSSH. המרחק מכאן עד להשתלטות מלאה של התוקף על השרת- קצר מאוד.

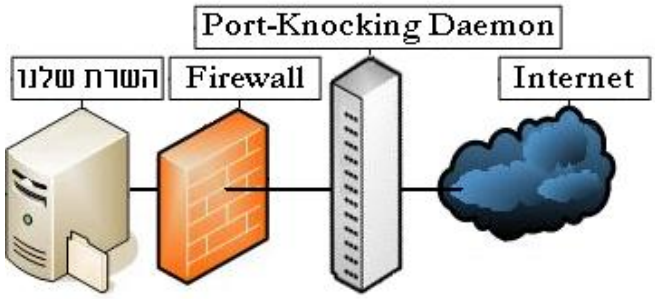
איך בכל זאת אפשר להתגונן מפני מקרים כאלה? חוץ מלעדכן את כל הגירסאות ולהיות רשום במליון ואחת רשימות-מיילים לעדכוני אבטחה- יש אפשרות פשוט להשתמש ב-Port -Knocking.

הרעיון הכללי

הרעיון הוא כזה- תשאיר כמה פורטים פתוחים על השרת שאתה רוצה, תשתמש באיזה גירסאות שאתה רוצה - שום תוקף לא יוכל לדעת איזה פורטים פתוחים על השרת שלך, שלא נדבר על לגשת אליהם.

איך בדיוק זה עובד? כל מי שירצה להתחבר לפורט 22 על השרת שלנו- בכדי לתקשר עם שירות ה-SSH, יהיה חייב קודם לכן להתחבר למספר פורטים קבועים מראש בסדר מסויים ורק אז פורט 22 יהיה פתוח לסייבר-ספייס ויהיה אפשר להתחבר אליו ולתקשר איתו.

היישום הוא כזה- לפני שכבת ה-Firewall שעל השרת, מתקינים Port Knocking Daemon שמנתר את כל התקשורת המגיעה לשרת, באופן הבא:



ציירתי את ה-Port-Knocking Daemon כחומרה נפרדת, אך כמו ה-Firewall, יש אפשרות שהוא יהיה אפליקטיבי ולאו דווקא כחומרה חיצונית.

אז כמו שאמרנו, כל מידע שמגיע למחשב שלנו עובר דרך שירות ה-Port Knocking [מעכשיו: PK] - נניח והפורט 22 אכן פתוח על השרת שלנו, וב-PK הגדרנו שה-Sequence לפורט 22 הוא:

```
50222<-102<-3302<-56432
```

ב-knockd.conf, זה נראה ככה פחות או יותר:

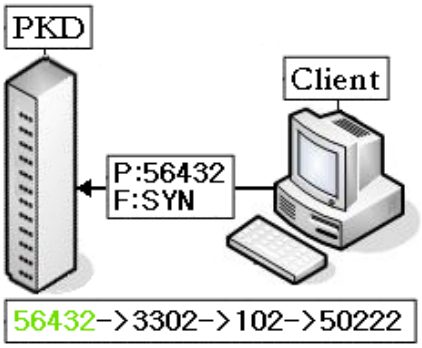
```
[SSH]
sequence    = 56432,3302,102,50222
seq_timeout = <timed out>
command     = <command to Bind the SSH to this connection>
ACCEPT
tcpflags    = syn
```

אם נשלח SYN ל-22 בכדי לנסות להתחיל את ה-Handshake של ה-TCP/IP אנחנו לא נקבל שום תשובה, למה? כי שירות ה-PK לא יעביר את המידע ששלחנו ל-22, פורט 22 על השרת שלנו אף פעם לא קיבל שום מידע!

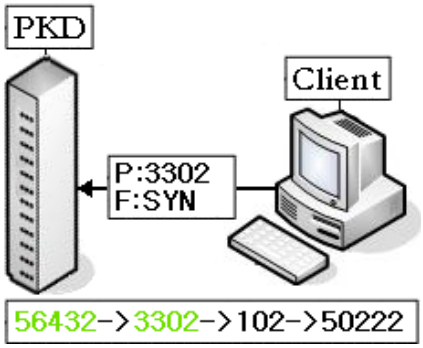
אופן ההתחברות

בכדי כן להצליח להתחבר אליו, אנחנו נאלץ לשלוח SYN לכל פורט שהוגדר ב-Sequence של ה-PK לפי סדר ה-Sequence, בצורה הבאה:

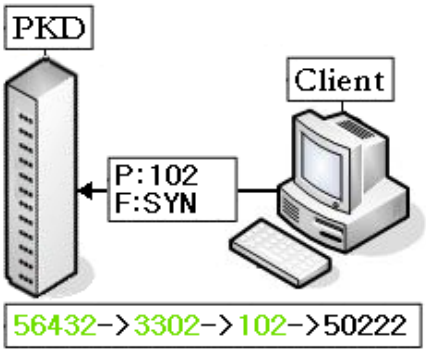
- שלב ראשון- שליחת SYN לפורט 56432:



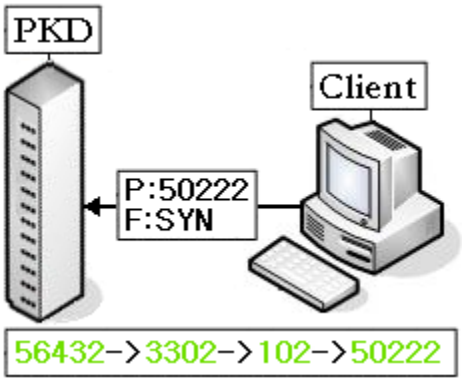
- שלב שני- שליחת SYN לפורט 3302:



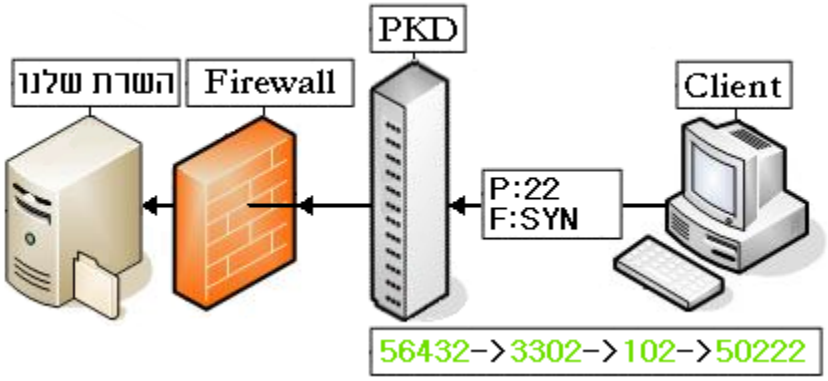
- שלב שלישי- שליחת SYN לפורט 102:



- שלב רביעי- שליחת SYN לפורט 55022:



- שלב חמישי- ה-PKD מאפשר את פתיחת הפורט ב-Firewall וככה ה-Client יכול לתקשר עם שירות ה-SSH:

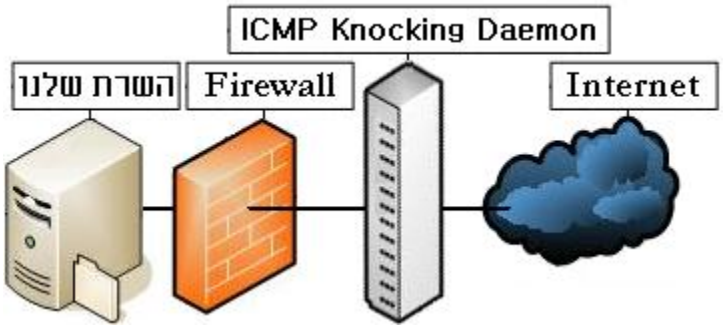


ורק כך בעצם מתחיל כל ה-TCP/IP Handshake לרוץ.

כמו שראינו- בלי לדעת את ה-Sequence הנכון, אין לנו סיכוי לדעת בכלל שפורט 22 בכלל פתוח!

- **ICMP Knocking:**

ICMP Knocking נקראת כך מפני שהיא מממשת את עקרון ה-Knocking כל הפורטים סגורים עד שמתבצעת הקשה כלשהיא – רק בהתבסס על בקשות "פינג" ולא שליחת דגלים לפורטים ב-Sequence ידוע מראש. בהתחלה, המצב בדיוק אותו דבר כמו ב-PK:



כל הפורטים סגורים ושרת ה-ICMP מאזין לכל התקשורת הנשלחת למחשב.
לקוח ה-ICMP Knocking שולח מספר "בקשות פינג" (ICMP) לפני כל בקשת חיבור לפורט מסויים בשרת ICMP, כל בקשת פינג שנשלחה בעלת Payload-length שונה.
הבקשות נבדקות על ידי שירות ה-ICMP Knocking ואם ה-Payload-length מתאים לגדלים שנקבעו מראש והפורט שנתבקש לפתוח אכן מתאימים ב-Sequenc שירות ה-ICMP Knocking מאשר ל-Firewall לפתוח את הפורט המבוקש, אם לא-הפורט נשאר סגור.
הרעיון הוא שמגדירים את ה-Payload-length Sequence כמו שמגדירים את ה-Port Sequence.

- **Single Packet Authorization:**

מתבצע באופן דומה מאוד ל- Port Knocking רק מתבצע ע"י שליחת חבילת מידע (Packet) אחת שנקראת "SPA Packet" שנכון לעכשיו נתמכת רק ע"י FWKnop.

תצורת ה-Packet נראת כך: [נלקח מהקונפיג של FWKnop-]:

```
16 bytes of random data
local username
local timestamp
fwknop version
mode (access or command)
desired access (or command string)
MD5 sum
```

- ה-16bytes הראשונים מכילים תווים רנדומאלים, המקושרים ל-Local TimeStamp, שניהם ביחד נועדו בכדי למנוע Replay Attack.
- ה-Local username קיים בכדי לאפשר מידור הרשאות.
- ה-Mode אומר לשרת האם ה-Packet מבקש להריץ פקודה או לקבל גישה למשאב מסויים.
- ה-Desired Access זאת בעצם הפקודה או הבקשת גישה עצמה.
- ה-MD5sum הוא כמובן ה-sum של כל ה-Packet עצמו ונועד להבטיח את שלמות ה-Packet.

כל המידע הזה נכתב בשורה אחת והערכים מובדלים בעזרת ":" (נקודותיים), את השורה מקודדים בBase64 - ואת המחרוזת שקיבלנו מצפינים ב-Rijndael בעזרת מפתח שידוע מראש גם ללקוח וגם לשרת.

לאחר שה-Packet נשלח השרת מפענח את המידע בעזרת המפתח, בודק את ה-MD5sum, בודק אם ה-TimeStamp וה-Ramdon-16bytes לא קיימים כבר ב-Cache שלו, ואם הכל תקין- הוא בודק את הרשאות המשתמש, ואם זה תקין- הוא מבצע את הפעולה ושומר את ה-TimeStamp וה-Random-16bytes ב-Cache למקרה שישלח אותו Packet ע"י תוקף שיבקש לבצע Replay Attack.
כמובן שאם המפתח של ה-Rijndael מתגלה לתוקף הוא יוכל להשתלט על השרת בקלות רבה.

גמישות

ישנם מספר דרכים בכדי לשלוח לאנשים מורשי גישה את ה-Sequence של הפורט הספציפי אליו הם מעוניינים להתחבר, אני אציין אחת מהן:

- **שלב ראשון:** לוקחים את ה- (56432,3302,102,50222) Sequence וממירים כל מספר לבינארית ומוסיפים אפסים לפני התוצאה כך שיהיה לנו 16 ביטים בכל ייצוג, נניח שהפורט הראשון שלנו הוא: 56432
  בבינארית זה יוצא:
  1101110001110000
  יש לנו כאן 16 ביטים ולכן לא צריך להוסיף שום 0.
  הפורט הבא שלנו הוא:
  3302
  בבינארית זה יוצא:
  110011100110
  יש לנו כאן 12 ביטים, ולכן אנחנו צריכים להוסיף עוד **4 אפסים**:
  0000110011100110

- **שלב שני:** מחברים את הבינארית של כל הפורטים שלנו, ונקבל את המספר הבא:

  1101110001110000000011001110011000000000001100110110001000010111O

  זה המפתח שלנו לפורט 22, מבצעים למפתח הזה XOR בעזרת מחרוזת (שנקבעה מראש עם הלקוח) כמובן שאפשר להשתמש ב-AES או כל אלגוריתם הצפנה מבוסס מפתח שהוא.

- **שלב שלישי:** הלקוח מבצע שאילתת בקשת מפתח מוגדרת מראש לפורט 22 ומקבל את הסייפר שיצרנו מה-Sequence.

- **שלב רביעי:** הקליינט מפענח את הסייפר בעזרת המפתח של ה- XOR או של האלגוריתם שקבענו מראש ומקבל את הצירוף המקורי.

- **שלב חמישי:** הקליינט מפרק את התוצאה למחרוזות בנות 16 ביטים ומקנפג בעזרתו את פורט 22 על ה-PK קליינט שלו וכך יוכל להתחבר לשרת.

אם תוקף מבצע שאילתת בקשת מפתח הוא יקבל את המחרוזת של המפתח הכניסה שלנו באופן מוצפן, ולא יוכל לעשות בה שימוש בלי המפתח והאלגוריתם שבהם השתמשנו בכדי להצפין את ה -Sequence לפורט הספציפי.

הרעיון בשימוש במנגנון כזה הוא שאם נרצה בעתיד לשנות את ה-Sequence של אחד הפורטים שלנו, לא תהיה לנו שום בעיה, כי בפעם הבאה שהלקוח ינסה להתחבר לשרת ויראה שהוא לא מצליח להתחבר בעזרת הצירוף הישן- הוא יבין שהצירוף כניראה שונה, והוא פשוט יבצע שאילתת בקשת מפתח חדשה.

חולשות

כיום לא ידועה שום מתקפה חיצונית המאפשרת לדעת אילו פורטים פתוחים ברשת מאחורי ה-PKD, אך המנגנון הזה לא מוגן מפני גורמים פנימיים – כגון מקרים שבהם תוקף הצליח לבצע מ תקפת MITM באופן מוצלח (למשל ע"י ARP Poisoning), אין לשירות ה-PKD אפשרות לדעת כי אכן מדובר בתוקף, גם אם חבילות המידע הנשלחות יהיו מוצפנות במפתח הציבורי של השרת ולתוקף לא תהיה אף אפשרות לדעת מה תכולתן- הוא עדיין יוכל לדעת לאילו פורטים מיועדות החבילות, וכך לעלות את ה-Sequence המקורי, וגם אם מדובר ב-ICMP Knocking.

ישנן מספר דרכים להתגונן מפני התקפות כאלה. נציג אחת מהן: כל חבילת מידע שנשלחת מהקליינט ל-PKD תכלול בתוכנה גם TimeStamp ומפתח סודי שנקבע מראש (אם רוצים אפשר שלכל פורט ה-Sequence יהיה מפתח שונה), את חבילת המידע מצפינים בעזרת המפתח הציבורי של אותו השרת, כאשר החבילה תעבור אצל התוקף (MITM), הוא לא יוכל לקרוא את מה שכתוב בפנים מפני שאין לו את המפתח הפרטי של השרת, הדבר היחידי שהוא יוכל לעשות- זה להמשיך להעביר את החבילה הלאה.

השרת מקבל את החבילה, מפענח אותה, ובודק את ה-TimeStamp. אם היא אותנטית - הוא בודק את המפתח הסודי שנכלל בחבילת המידע, אם כן- הוא מחכה להמשך ה-Sequence.

במקרה כזה, לא מנענו מהתוקף לקבל את ה-Sequence הסודי, אך ידיעת הנתון הזה בלבד- לא תעזור לו להתחבר לשום פורט הפתוח בשרת שלנו, מפני שהוא צריך את המפתח לכל פורט ! זה נכון שהתוקף יוכל לדעת את ה-Sequence, אבל סתם לשלוח SYN ברצף הנכון כבר לא מספיק.

מה בדבר לבצע Replay Attack? הרעיון הוא כזה- התוקף לא יודע מה המפתח, אבל יש לו את חבילות המידע שנשלחו ובתוכם יש את המפתח לכל פורט- הוא יוכל לשלוח אותם וכך ליצור חיבור! אז זהו, שלא. כאן בדיוק נכנס הרעיון של ה-TimeStamp שהתפקיד שלה בדרך כלל הוא למנוע Replay Attack, החתימה נוצרת בעזרת השעון, ובשרת נקבע Time-out שלאחר זמן קצוב (וקצר מאוד) חבילת המידע הופכת ללא רלוונטית מבחינת השרת, גם אם כל המידע בפנים נכון ותקף.

סיכום + קישורים

הרעיון לא מסובך, לא להבנה ולא לביצוע, אבל אין ויכוח שאכן מדובר בפיתרון אלגנטי ויפה ב יותר להסתרת הפורטים הפתוחים על השרת שלנו, מה שמוסיף ללא ספק עוד שכבה באבטחתו.

בתיאוריה אפשר לשלוח SYN לפורטים שעל השרת באופן שיטתי (Brute-Force) וכך לנסות לגלות איזה פורטים אכן פתוחים, אבל גם אפשר להגביל את מספר נסיונות ההתחברות בשרת למספר קטן של פעמים וכך למנוע את זה, וגם בלי זה- קיימים 65535 פורטים שונים, התחברות שיטתית לכל הפורטים (בהתחשב ש-Sequence אחד יכול להיות גם עשרים, חמישים פורטים ויותר) היא מלאכה אשר תגזול יותר שנים ממה שהתוקף יחיה.

חשוב לציין ולהדגיש כי **ה-Port Knocking לא בא להחליף שום מנגנון אבטחה בשום פרוטוקול**. אם משתמשים ב-Port Knocking בשום פנים ואופן אין שום סיבה להסיר את מנגנוני ההזדהות הקיימים בפרוטוקולים!

קישורים:

- האתר של Martin Krzywinski, אתר שלם העוסק בנושא: http://www.portknocking.org
- עוד אתר מומלץ עם המון חומר בנושא, הוא Aldabaknocking:
  http://www.aldabaknocking.com/
- מאמר ב-Linuxjournal המסביר בין השאר איך להשתמש ב-PK בשילוב עם IPTables:
  www.linuxjournal.com/article/6811
- fwknop - הסבר ומימוש על Single Packet Authorization ב-FWKnop:
  http://www.cipherdyne.org/fwknop/
- Knockd - עמוד הבית של אחד משרתי ה-Port Knocking המוכרים ביותר:
  http://www.zeroflux.org/projects/knock
- מימוש נחמד ב-Python ל-ICMP Knock Server:
  http://homework.nwsnet.de/products/0d09_icmp-knock-server