digital whisper
Technologic Papers

Introduction to Bro scripts

מאת יואב קמיר

הקדמה

בשנה האחרונה יצא לי להתנסות לא מעט בכלים רבים מעולם ה-Network forensics, אחד הכלים שיצא לי לעבוד איתם הוא Bro.

Bro הוא אחד הכלים החזקים ביותר שיצא לי לפגוש בתחום החקירה התקשורתית. עולם ה-Scripting שהוא מכניס לתחום הפקטות וה-IDS הופכים אותו לכלי ייחודי בשוק שלדעתי הכרחי לכל חוקר תקשורת להכיר.

במאמר הזה אציג סקריפט שכתבתי ב-Bro על מנת לזהות ולספק מידע אודות ICMP Tunnelling. במהלך המאמר ארחיב לגבי נושא ה-Tunnelling, אך במקרה זה הוא משמש אותי ככלי כדי להציג את היכולות של Bro, כיצד איך הוא עובד ולעזור ל-Bro scripter המתחיל.

אז בקצרה, מה זה בכלל Tunnelling?

בצורה הפשטנית ביותר - שימוש בפרוטוקול מסוים על מנת להעביר מידע או פרוטוקול אחר.

לרוב השימוש ב-Tunnelling יהיה תמים וחוקי, כאשר נרצה לקשר בין רשתות שונות על גבי רשת ה-WAN (במצב בו אנו לא רוצים שהתעבורה שלנו תעבור בצורה חשופה) ונשתמש בפרוטוקול שיעביר לנו את המידע בתוכו (פרוקטוקלים כמו PPTP או L2TP) אפשר גם לקרוא לתהליך זה VPN.

השימוש ה"אפל" יותר ב-Tunnelling הוא כאשר נשתמש בפרוטקול טריוואלי, כגון ICMP, וב-Payload שלו נכניס את הפקטה המקורית של פרוטוקול אחר. מה זה אומר? המקור יכניס את הפקטות שברצונו להעביר לתוך פקטות ICMP, והיעד יחלץ מתוך ה-Payload של הפקטות שהוא מקבל את הפקטה החבויה. לרוב אני אוהב לדמות את זה למשאית ליגיטימית, המחביאה בתוכה גנבים בכדי להכנס למפעל שהם אינם מורשים להכנס אליו:

למה שנרצה לעשות זאת?

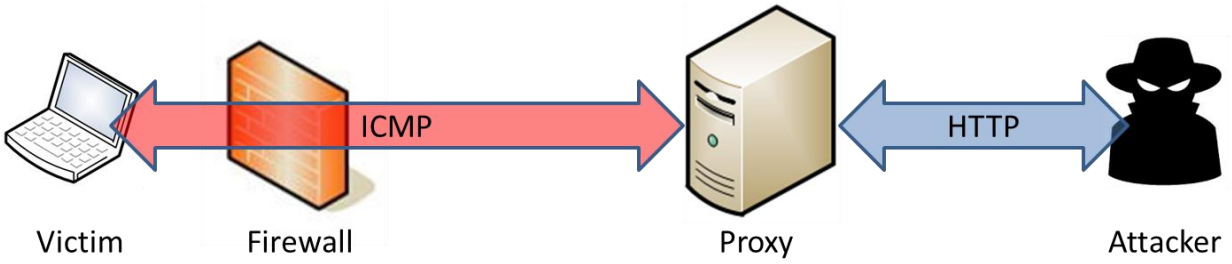
ישנן סיבות רבות למה נרצה לבצע Tunnelling:

- במצבים בהם לא ארצה שתעבורה זדונית מסוימת תהיה חשופה לקורבן,
- כאשר התעבורה המקורית אינה מאופשרת על ידי FW או רכיב אבטחה אחר
- כמו שאמרנו מקודם, כאשר אבצע VPN בין שתי רשתות מרוחקות

כמובן שקיימות סיבות רבות נוספות, אך ללא ספק אלה הן העיקריות.

באיזה כלי נשתמש כדי לממש ICMP Tunnelling?

ישנו כלי בשם ptunnel אשר מבצע ICMP Tunnelling בצורה אוטומטית. זה כלי זמין להורדה ומותקן על כל מכונת kali בצורה דיפולטית. הכלי דורש שימוש ב-Proxy. כלומר התעבורה תעבור בין הלקוח לשרת ה-proxy ומשם, תגיע ליעד. בין המקור ל-proxy התעבורה תהיה מתועלת תחת ICMP. ניתן לראות זאת בתרשים הבא:



כמו שאנחנו יכולים לראות בתרשים: התעבורה מתועלת בין הקורבן לשרת ה-Proxy, בין ה-Proxy לתוקף התעבורה בצורתה המקורית. להמשך המאמר, התעבורה שמעניינת אותנו היא זו שבין הקורבן לבין שרת ה-Proxy, בעצם הפקטות שעברו Tunnel.

כל חבילות המידע שיצאו מהמקור ומגיעות ל-Proxy, יכנסו לתוך הודעות echo-request וכל ההודעות שיצאו משרת ה-proxy ומיועדות למקור יכנסו לתוך הודעות echo-replay. למראית עין הודעות ה-ICMP נראות ליגיטימיות לחלוטין, החלק המעניין הוא מה שקורה בתוך ה-Payload. שם ptunnel משתמש בפורמט ייחודי לו כדי להעביר את הנתונים.

כדי לבחון את הפקטה ולהבין את המבנה שלה נשתמש ב-scapy.

הכנתי מראש pcap המכיל חבילות HTTP שעברו ICMP Tunnelling ובעזרת קריאה של הקובץ ב-scapy אפשר לקרוא לכל אחת מהפקטות ולהציג אותן:

```
>>> packets = rdpcap('/icmp_tun.pcap')
>>> packets[0].show()
###[ Ethernet ]###
  dst       = 00:0c:29:f2:61:d0
  src       = 00:0c:29:dc:8e:04
  type      = 0x800
###[ IP ]###
     version   = 4L
     ihl       = 5L
     tos       = 0x0
     len       = 56
     id        = 27005
     flags     = DF
     frag      = 0L
     ttl       = 64
     proto     = icmp
     chksum    = 0x1ca1
     src       = 192.168.153.170
     dst       = 192.168.153.171
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xb4c7
        id        = 0x6582
        seq       = 0x0
###[ Raw ]###
           load      = '\xd5 \x08\x80\xc0\xa8\x99\x99\x00\x00\x00P@\x00\x00\x00\x00\x
>>>
```

בתמונה זו ניתן לראות את החבילה הראשונה שנשלחה מהמקור לשרת ה-Proxy. אפשר לראות כי מדובר בחבילת ICMP ליגיטימית לחלוטין בעלת כל ה-Headers המוכרים, אך מה שמעניין בה זה ה-Payload שהיא מחזיקה. רצף של מידע ב-Hexdecimal שמאוחסן ב-data של הפקטה (ב-load).

באותו Payload הכלי ptunnel שולח את המידע במבנה מאוד מסוים. זהו המבנה:

magic	Ip	port	state	Ack	length	Seq	rsv	Data
4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	2 bytes	2 bytes	

כלומר אם נסתכל אך ורק על ה-Payload של הפקטה נוכל להתאים כל אחד מהשדות:

```
>>> hexdump(packets[0][Raw].load)
0000   D5 20 08 80 C0 A8 99 99  00 00 00 50 40 00 00 00   . .........P@...
0010   00 00 FF FF 00 00 00 00  00 00 65 82               ..........e.
```

- ארבעת הבתים הראשונים ("D5 20 08 80") הם מה שנקרא ה-"magic number", והם מה שמזהה את הפקטות ICMP שהכלי ptunnel מייצר. שדה זה חשוב ביותר כי בעזרתו נוכל לזהות את פקטות אלה ולהבדיל אותן מפקטות ICMP אחרות.
- ארבעת הבתים לאחר מכן מציינים את כתובת ה-ip שהמקור מעוניין להגיע אליה (כלומר היעד המקורי, לא שרת ה-Proxy).

- לאחר מכן ניתן לראות כי הכתובת ב-Hex היא: C0 A8 99 99, ושהתרגום שלה הוא: 192.168.153.153. ניתן להבחין כי כתובת היעד של החבילה באופן כללי היא 192.168.153.171 כך שאפשר להניח כי זו הכתובת של שרת ה-Proxy.
- לאחר מכן אפשר לראות את הפורט: 50 בהקסה זה 80 לכן אנחנו מבינים שהכתובת הקודמת היא כנראה שרת WEB.
- השדה שמגיע לאחר מכן חשוב במיוחד לסקריפט שאנחנו הולכים לראות בהמשך, השדה של ה-State. השדה הזה מצויין את סוג ההודעה. ישנם ארבעה סוגים ומספר שמייצג כל אחד מהם:

1. **KProxy_start** - הודעה המתחילה את השיחה עם שרת ה-Proxy ומעבירה לו את הפרטים הנחוצים. רק בהודעות אלה יעברו כתובת ה-IP והפורט (!) המספר שמעיד על ה-State הזה הוא: 0
2. **KProxy_ack** - הודעת אישור שמידע מסוים הגיע, המספר שמעיד על State זה הוא: 1
3. **KProxy_data** - הודעה המכילה את המידע עצמו, כלומר, המידע שעובר בין המקור ליעד, במספר שמעיד על state זה הוא: 2
4. **KProxy_close** - הודעה המעידה על סיום שיחה. המספר שלה הוא: 3

אם נחזור לחבילה בתמונה הקודמת, אפשר להבין שזוהי חבילה של תחילת שיחה, כי ה-state שלה הוא 0, ולכן גם אפשר למצוא בה את כתובת ה-IP והפורט היעד.

אם נציץ בחבילה אחרת לדוגמא (state = 2, כזו שמעבירה את המידע בפועל), נוכל לראות שעובר גם תוכן לאחר כל שדות החובה: במקרה זה בקשת GET:

```
>>> hexdump(packets[2][Raw].load)
0000   D5 20 08 80 00 00 00 00  00 00 00 00 40 00 00 02   . ..........@...
0010   00 00 FF FF 00 00 01 60  00 01 65 82 47 45 54 20   .......`..e.GET 
0020   2F 62 57 41 50 50 20 48  54 54 50 2F 31 2E 31 0D   /bWAPP HTTP/1.1.
0030   0A 48 6F 73 74 3A 20 31  32 37 2E 30 2E 30 2E 31   .Host: 127.0.0.1
0040   3A 38 30 30 30 0D 0A 55  73 65 72 2D 41 67 65 6E   :8000..User-Agen
0050   74 3A 20 4D 6F 7A 69 6C  6C 61 2F 35 2E 30 20 28   t: Mozilla/5.0 (
0060   58 31 31 3B 20 4C 69 6E  75 78 20 78 38 36 5F 36   X11; Linux x86_6
0070   34 3B 20 72 76 3A 35 32  2E 30 29 20 47 65 63 6B   4; rv:52.0) Geck
0080   6F 2F 32 30 31 30 30 31  30 31 20 46 69 72 65 66   o/20100101 Firef
0090   6F 78 2F 35 32 2E 30 0D  0A 41 63 63 65 70 74 3A   ox/52.0..Accept:
00a0   20 74 65 78 74 2F 68 74  6D 6C 2C 61 70 70 6C 69    text/html,appli
00b0   63 61 74 69 6F 6E 2F 78  68 74 6D 6C 2B 78 6D 6C   cation/xhtml+xml
00c0   2C 61 70 70 6C 69 63 61  74 69 6F 6E 2F 78 6D 6C   ,application/xml
```

על המידע הזה בניתי את הסקריפט שלי.

מה הוא Bro?

Bro הינו כלי לניתוח תעבורת רשת, מעין IDS. הוא מאזין לרשת בצורה פסיבית ומפיק קבצי לוג כלפי התעבורה שהקשיב לה. כל קבצי הלוג ש-Bro מפיק מיוצרים על ידי סקריפטים אשר רצים על התעבורה שאליה מאזינים.

מה שהופך את Bro להיות שונה מכל IDS אחר, הוא שבעוד שרוב ה-IDS שאנחנו מכירים עובדים עם חוקים וחתימות, Bro מספק שפה שלמה שכל מטרתה היא סביב תעבורת תקשורת. השימוש ב-Bro הוא מאוד נח, ובנוסף האתר מכיל הסברים ומדריכים רבים ל-Bro Scripter המתחיל. אני משתמש ב-Bro על המכונה Security Onion בה הוא מגיע כבר מותקן ומוכן לשימוש.

קצת על השימוש ב-Bro

כמו שהזכרתי קודם, ברגע שנפעיל את Bro הוא יתחיל להפיק לוגים על פי התעבורה שהוא מאזין לה. מבנה הלוגים הוא טבלאות גדולות המכילות נתונים על נושא מסוים. מאחורי כל קובץ לוג כזה עומד סקריפט שעבר על הפקטות ולפיהם ייצא את קובץ הלוג. להלן דוגמה של כמה לוגים ש-BRO מייצר:

```
ine:~/Desktop/icmp$ ls -l *.log
 1885 מצד  7 17:45 conn.log
 1189 מצד  7 17:42 dhcp.log
 1128 מצד  7 17:45 dns.log
 1935 מצד  7 17:23 files.log
 2903 מצד  7 17:23 http.log
  253 מצד  7 17:45 packet_filter.log
  361 מצד  7 17:45 reporter.log
  662 מצד  7 17:42 ssl.log
  587 מצד  7 17:45 weird.log
 1547 מצד  7 17:23 x509.log
```

אפשר לראות שיש לוג המכיל מידע על השיחות שעברו בהקלטה (conn.log) לוג המכיל מידע על תעבורת ה-DHCP ועוד.

נקודה חשובה - BRO מייצר לוגים רק של התעבורה שעברה דרכו, כלומר אם לא היו פקטות DHCP בזמן הריצה של Bro, קובץ הלוג הזה לא היה מיוצר.

בואו נסתכל על קובץ לוג לדוגמה:

```
#path   http
#open   2017-12-07-17-23-04
#fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p       tra
#types  time    string  addr    port    addr    port    count   string  string  string  str
1512667384.581385       CVqt6C1mWYnymqE2k1      192.168.153.128 39842   172.217.17.142  80
1512667384.792091       CZvDEQ2y5WpLI0fs48      192.168.153.128 43022   81.218.16.208   80
1512667385.264446       CVqt6C1mWYnymqE2k1      192.168.153.128 39842   172.217.17.142  80
1512667386.580751       CLKVTA3I23DfUE7IWi      192.168.153.128 57312   212.179.154.208 80
```

בכל שורה אפשר למצוא פרטים אודות הבקשה, התשובה מהשרת ואפילו נסיון חילוץ של שמות משתמש וסיסמה מגוף הבקשה.

מה הייתה המטרה שלי

המטרה שלי הייתה לכתוב סקריפט Bro שמייצר קובץ לוג המכיל מידע לגבי השיחות שעוברות ב-ICMP Tunnelling. קובץ הלוג יכיל נתונים כגון: לאן החבילות מיודעות (שאת זה צריך לחלץ מגוף החבילה, כמה שראינו בהתחלה) ומאיפה הן מגיעות. בנוסף, מעבר לפרטים שיכתבו לקובץ לוג, רציתי להשתמש ב-Bro כדי לכתוב לקובץ את כל המידע עצמו שעובר בפקטות האלה, כלומר המידע שמתועל בתוך הפקטת ICMP.

נעבור לסקריפט

השפה של Bro דומה מאוד לכל שפת תכנות אחרת שאנחנו מכירים וחוץ כמה ניואנסים קטנים קל מאוד להתחיל לכתוב בה. לפני שנעבור לקוד שלי, כמה דברים שכדאי להכיר:

תחילה נכיר את הקונספט Event. Event הינו סוג של פונקציה, אשר מופעלת מטריגר מסוים. אותו Event מספק לי מידע על הפקטות שגרמו לו לפעול, שבעצם היו הטריגר שלו. בואו נקח את ה-Event הבסיסי ביותר: Event בשם "new connection". אפשר להבין מהשם שלו שזהו Event המופעל כל פעם כשיש connection חדש.

המבנה שלו בסקריפט יהיה כזה:

```
event new_connection(c: connection)
{

}
```

לכל connection חדש שיפתח ה-event - new connection יספק לי אובייקט בשם c מסוג connection. אובייקט זה מכיל פרטים רבים כלפי הטריגר של ה-event (ה-connection שנפתח).

אם אדפיס את המשתנה c:

```
event new_connection(c: connection)
{
      print "------new connection------";
      print c;
}
```

הפלט יהיה כזה:

```
------new connection------
[id=[orig_h=192.168.153.128, orig_p=52710/tcp, resp_h=216.58.210.14,
9:81:29:59], resp=[size=0, state=0, num_pkts=0, num_bytes_ip=0, flow_

}, history=, uid=Cz9haASUSdY9ZAu97, tunnel=<uninitialized>, vlan=<uni
F, extract_resp=F, thresholds=<uninitialized>, dce_rpc=<uninitialized
initialized>, dns=<uninitialized>, dns_state=<uninitialized>, ftp=<un
zed>, irc=<uninitialized>, krb=<uninitialized>, modbus=<uninitialized
<uninitialized>, sip=<uninitialized>, sip_state=<uninitialized>, snmp
ninitialized>, syslog=<uninitialized>]
------new connection------
[id=[orig_h=192.168.153.128, orig_p=36706/tcp, resp_h=216.58.208.42,
9:81:29:59], resp=[size=0, state=0, num_pkts=0, num_bytes_ip=0, flow_
```

אפשר לראות שהאובייקט c מכיל בתוכו הרבה משתנים שלכל אחד מהם יש ערך. כלומר במידה וארצה להדפיס רק את כתובת המקור של ה-connection החדש, אעשה זאת כך:

```
Print c$id$orig_h;
```

Event-ים נוספים שכדאי להכיר הם: Bro_init ו-Bro_done. Bro_init הוא event הפועל בכל פעם ש-Bro מופעל. כשארצה לכתוב בסקריפט פעולות שיפעלו בהתחלה, ללא שום טריגר ספציפי מחבילה, אשתמש ב-Event זה.Bro_done הינו Event הפועל כל פעם ש-Bro נסגר. Bro מספק ספריית Events ענקית, כמעט לכל פרוטוקול מוכר וזאת בלי להזכיר את ה-Events שניתן ליצור בעצמנו.

ה-Event-ים שהשתמשתי לסקריפט שלי הם Icmp_echo_request ו-icmp_echo_replay. כמו שאפשר לנחש, ה-Event-ים האלה מופעלים כאשר מגיע הודעת echo request או replay. בנוסף, הם מספקים פרטים אודות ה- connection וה-Payload של כל אחת מחבילות המידע. כמו שהבנו קודם לכן, ה-Payload הוא המידע החשוב, מפני ש שם יש את חבילת המידע המקורית. בואו נשתמש בסקריפט בכדי להדפיס את ה-Payload.

בדוגמא הבאה כתבתי event של הודעת echo request ובכל פעם שהיא תופעל היא תדפיס את ה-Payload:

```
event icmp_echo_request(c: connection, icmp: icmp_conn, id: count, seq:
count, payload: string)

{
    Print ------request payload------;
    print payload;
}
```

זהו הפלט:

```
--------request payload--------
\xd5 \x08\x80\xc0\xa8K\x88\x00\x00\x00P@\x00\x00\x00\x00\x00\xff\xff\x00\x
--------request payload--------
\xd5 \x08\x80\x00\x00\x00\x00\x00\x00\x00\x00@\x00\x00\x02\x00\x00\xff\xf
 localhost:8080\x0d\x0aAccept: */*\x0d\x0a\x0d\x0a
--------request payload--------
\xd5 \x08\x80\x00\x00\x00\x00\x00\x00\x00\x00@\x00\x00\x03\x00\x00\x00%\x
--------request payload--------
\xd5 \x08\x80\x00\x00\x00\x00\x00\x00\x00\x00@\x00\x00\x03\x00\x00\x00%\x
```

ה-Payload שאנחנו רואים כאן הוא אותו Payload שניתחנו קודם עם scapy. לכן עכשיו, אחרי שיש לנו אותו בידיים, נרצה לייצא ממנו את כתובת היעד, הפורטים וכמובן - התוכן.

את הסקריפט שלי ניתן לחלק לשלושה שלבים.

שלב ראשון - זיהוי פקטות ICMP Tunnel:

כדי לזהות חבילות אלו נגדיר ל-Bro להתייחס רק לחבילות אשר מכילות את ה-flag "D5 20". לשם כך אנו נדרשים לקחת את ה-Payload ולקרוא מהמקום בו יש את ה-flag (ה-4 בייטים הראשונים), כדי לעשות זאת הפכתי את ה-Payload לטיפוס מסוג string, ואז בעזרת הפונקציה "sub_bytes" הוצאתי אך ורק את ה-2 בייטים הראשונים והכנסתי אותם למשתנה בשם check:

```
local pk = string_to_ascii_hex(payload);
local check: string = sub_bytes(pk,1,4);
```

לאחר מכן ביצעתי את הבדיקה בעזרת שימוש ב-if פשוט:

```
if (check == "d520")
{
    .....
}
```

שלב שני - הוצאת נתונים רלוונטים מתוך ה-payload לקובץ log:

כמו שהזכרתי בהתחלה, ב-Payload נוכל לאתר את כתובת ה-IP, ה-Port וה-State אשר מעיד על סוג ההודעה. בגלל שכתובת ה-IP וה-Port יהיו רק בפקטות שה-State שווה 0 (תחילת שיחה) אז נעשה תנאי שבודק את את ערך הבייטים של ה-State, ואז נייצא את כתובת ה-IP וה-Port לפי המיקום שלהם ב-Payload (גם פה אשתמש ב-sub_bytes כדי להוציא אותם):

```
local flag: string = sub_bytes(pk,31,2);

if (flag == "00")
{
      local dst_addr: string = sub_bytes(pk,9,8);
      local st_port: string = sub_bytes(pk,17,8);
}
```

אני מזכיר שהמידע שאנחנו עובדים איתו מה-Payload הוא הקסה-דצימלי. לצערי ל-Bro לא היו פונקציות מובנות להפיכתו לכתובת או מספר לכן כתבתי אותן בעצמי.

פונקציה ראשונה מקבלת רצף הקסה-דצימלי ומחזירה אותו כטיפוס מסוג כתובת IP. פונקציה שנייה גם היא מקבלת רצף הקסה-דצימלי ומחזירה אותו כטיפוס מספרי (count).

בגדול, שתי הפונקציות הפכות את הרצף ההקסה דצימלי לרצף של בייטים, ואז לטיפוס המבוקש (כל זה בעזרת פונקציות מובנות של Bro). לבסוף קראתי ל-2 הפונקציות עם המשתנים שייצאתי קודם:

```
function hex_to_addr(s: string): addr
{
      local ip_bytes: string = hexstr_to_bytestring(s);
      local ip_dest: addr = raw_bytes_to_v4_addr(ip_bytes);
      return ip_dest;
}

function hex_to_port(s: string): count
{
      local port_bytes: string = hexstr_to_bytestring(s);
      local port_dst: count = bytestring_to_count(port_bytes);
      return port_dst;
}
print hex_to_addr(dst_addr);
print hex_to_port(dst_port);
```

ומה שיודפס למסך הוא:

```
listening on eth0

192.168.153.153
80
```

עד שלב זה אספתי את כלל הנתונים שברצוני לכתוב לקובץ ה-log. כל הנושא סביב קובץ ה-log בסקריפט די מורכב, מורכב בעיקר ברמה ה"ביורוקרטית". כדי ליצור קובץ לוג חדש שאליו אכתוב את הנתונים הרלוונטים, יש כמה שלבים: תחילה אצטרך להצהיר על קובץ הלוג החדש, ובעצם אוסיף אותו לרשימה של כלל שמות קבצי הלוג הקיימים:

```
redef enum Log::ID += { ptunnel };
```

בפקודה זו הוספתי סוג של קובץ לוג חדש לרשימת ID קיימת, הסוג שהוספתי הוא בשם ptunnel. כמו שראינו בעבר, הנתונים נרשמים לקובץ הלוג בצורה של עמודות. לכן, כדי לכתוב לקובץ הלוג נתונים - עלינו ליצור תחילה אובייקט חדש מסוג record שהאיברים שלו הם העמודות של קובץ הלוג.

טיפוס מסוג record זהו טיפוס דומה ל-dictionary אך כל איבר בו יכול להיות מסוג שונה, כלומר הוא יכול להכיל בתוכו ערכים מסוג מספרי, string וכל ערך אחר שאבחר. כל פעם שארצה להכניס שורה חדשה לקובץ הלוג, אכניס את הנתונים לאובייקט ה-record ואתו אכתוב לקובץ הלוג.

ה-record שיצרתי נקרא info ומכיל בתוכו זמן, כתובת מקור, כתובת יעד ופורט יעד (שימו לב שליד כל איבר ברשימה מצוין הסוג שלו):

```
type info: record { i_time: time &log; source: addr &log; dest: addr
&log; dst_port: count &log; };
```

לכל איבר יכול להיות ערך מסוג אחר, לדוגמה הערך של האיבר source יהיה מסוג addr.

לאחר שני השלבים הללו, ניתן ישירות לעבור לכתיבה לקובץ ה-log. תחילה עלינו ליצור stream ל-log שאנו יוצרים. ב-stream אגדיר איזה ID קובץ הלוג שלי (ptunnel), מהן העמודות בקובץ ה-log שלי (info, ה-record שיצרנו) ואיך קובץ הלוג יקרא בפועל שיווצר:

את ה-stream יצרתי ב-event - Bro_init. ה-event שפועל כל פעם כש-Bro נתחיל לרוץ.

```
event Bro_init()
{
      Log::create_stream(ptunnel, [$columns=info, $path="icmp_tunnel"]);
}
```

כל פעם שנאסוף את הנתונים הרלוונטים לנו ונרצה להכניס ל-log, ניצור משתנה חדש מסוג info (ה-record שיצרנו) ונכתוב את המשתנה הזה לקובץ הלוג בעזרת פונקציית הכתיבה ל-log (Log::write):

```
local test: info = [ $i_time = network_time(), $source = c$id$orig_h,
$dest = hex_to_addr(dst_addr), $dst_port = hex_to_port(dst_port)];

Log::write(ptunnel, test);
```

בקוד אני מגדיר משתנה חדש בשם test ומכניס לכל אחד מהאיברים שלו את הנתונים הרלוונטים. קובץ ה-log שנקבל בסוף יהיה בשם icmp_tunnel.log ויראה כך:

```
#fields i_time  source  dest    dst_port
#types  time    addr    addr    count
1513265336.560221       192.168.153.170 192.168.153.153 80
1513265336.561105       192.168.153.170 192.168.153.153 80
1513265337.223729       192.168.153.170 192.168.153.153 80
```

באופן זה יצרנו קובץ לוג פשוט של Bro המספק לנו מידע אודות פעולה מסוימת אותה בחרנו לנטר.

שלב שלישי - כתיבת תוכן השיחה לקובץ:

לאחר שהסקריפט מגלה שהיה Tunnel, ומזהה את הכתובות שביצעו אותו - מעניין אותנו לדעת מה התוכן שעבר שם. שלב זה יחסית פשוט מפני שאנו כבר יודעים איפה נמצא ה-Data ב-Payload. כך שכל מה שנשאר לנו הוא לכתוב אותו לקובץ. הפקטות שמעניינות אותנו אלה הן שה-State שלהן שווה 1,2 שהן החבילות אשר עובר בהן מידע ו-Acknowledge.

תחילה אני פותח קובץ בשם data שאליו אכתוב את כל ה-data שאני מוציא. כדי לעשות זאת אני משתמש בפונקציה open_for_append כדי שאוכל להוסיף מידע לקובץ מבלי לדרוס. לאחר מכן, בעזרת sub_bytes אני לוקח מה-payload את ה-data על פי הבייטים שהוא נמצא בהם. את ה-data שאני מוציא

אני מנקה בעזרת split מכל מיני בייטים מיותרים ויוצר רשימה המכילה לי את כל השורות של ה-data. הרשימה הזאת נקראת בקוד שלי headers. על כל אחת מהשורות האלה אני רץ עם לולאת for וכותב אותן לתוך הקובץ שפתחתי (בעזרת הפונקציה write_file). לבסוף סוגר את הקובץ - שלב הכרחי לעבודה עם קבצים ב-Bro:

```
if (flag == "01" || flag == "02")
{
      local f = open_for_append("/home/r123/Desktop/icmp/data");
      local d: string = sub_bytes(payload, 29,200);
      local headers = split(d, /\x0d\x0a/);
      local i: count = 1;
      while ( i <= |headers| )
      {
            print headers[i];
            write_file(f, headers[i]);
            write_file(f, "\n");
            i +=1;
      }
      close(f);
}
```

הקובץ שאקבל בסוף יראה כך:

```
data - Mousepad
File  Edit  View  Text  Document  Navigation  Help

GET /flag.jpg HTTP/1.1
User-Agent: curl/7.26.0
Host: localhost:8080
Accept: */*



HTTP/1.1 200 OK
Date: Tue, 29 Sep 2015 16:49:52 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Tue, 29 Sep 2015 16:48:06 GMT
ETag: "8f44-520e594065d58"
Accept-Ranges: bytes
Content-Length: 366

Filetype: None | Line: 1 Column: 0 | OVR
```

קובץ נקי המראה לי בדיוק את תוכן השיחה המתועלת שעברה.

הסקריפט המלא:

```
redef enum Log::ID += { ptunnel };
type info: record { i time: time &log; source: addr &log; dest: addr &log; dst port: count
&log; };

function hex to addr(s: string): addr
{
    local ip_bytes: string = hexstr_to_bytestring(s);
    loca ip dest: addr = raw bytes to v4 addr(ip bytes);
    print ip dest;
    return ip_dest;
}

function hex_to_port(s: string): count
{
    local port bytes: string = hexstr to bytestring(s);
    local port dst: count = bytestring to count(port bytes);
    print port_dst;
    return port_dst;
}

event Bro_init()
{
    Log::create_stream(ptunnel, [$columns=info, $path="icmp_tunnel"]);
}

event icmp_echo_reply(c: connection, icmp: icmp_conn, id: count, seq: count, payload: string)
{
    local pk = string_to_ascii_hex(payload);
    local check: string = sub bytes(pk,1,4);
    if (check == "d520")
    {
        local flag: string = sub_bytes(pk,31,2);
        if (flag == "00")
        {
            local dst addr: string = sub bytes(pk,9,8);
            local dst_port: string = sub_bytes(pk,17,8);

            local test: info = [
            $i_time = network_time(),
            $source = c$id$orig h,
            $dest = hex to addr(dst addr),
            $dst_port = hex_to_port(dst_port)];
            Log::write(ptunnel, test);
        }
        if (flag == "01" || flag == "02")
        {
            local f = open for append("/home/rt/Desktop/icmp/data");
            local d: string = sub_bytes(payload, 29,200);
            local headers = split(d, /\x0d\x0a/);
            print |headers|;
            local i: count = 1;

            while ( i <= |headers| )
            {
                print headers[i];
                write file(f, headers[i]);
                write file(f, "\n");
                i +=1;
            }
            close(f);
        }
    }
}

event icmp_echo_request(c: connection, icmp: icmp_conn, id: count, seq: count, payload: string)
{
    local pk = string to ascii hex(payload);
    local check: string = sub_bytes(pk,1,4);
    if (check == "d520")
    {
        local flag: string = sub_bytes(pk,31,2);
        if (flag == "00")
```

```
        {
            local dst_addr: string = sub_bytes(pk,9,8);
            local dst_port: string = sub_bytes(pk,17,8);
            local test: info = [
            $i time = network time(),
            $source = c$id$orig h,
            $dest = hex_to_addr(dst_addr),
            $dst_port = hex_to_port(dst_port)];
            Log::write(ptunnel, test);
        }
        if (flag == "01" || flag == "02")
        {
            local f = open_for_append("/home/rt/Desktop/icmp/data");
            local d: string = sub_bytes(payload, 29,200);
            local headers = split(d, /\x0d\x0a/);
            print |headers|;
            local i: count = 1;

            while ( i <= |headers| )
            {
                print headers[i];
                write file(f, headers[i]);
                write file(f, "\n");
                i +=1;
            }
        close(f);
        }
    }
}
```

לסיכום

במאמר זה למדנו לעבוד עם הכלי Bro, כיצד הוא עובד וכיצד ניתן לרתום אותו לטובתנו בעזרת כתיבת סקריפטים. דיברנו על ICMP Tunnelling בעזרת ptunnel, וכמובן - כיצד ניתן לזהות תקשורת זו באמצעות Bro. עיקר מטרת המאמר שלי היא לתת היכרות עם Bro ולהראות כמה השפת סקריפטים שהוא מספק מכניסה דינמיות וגמישות שלא קל למצוא בכלים אחרים. עם שימוש נכון ב-Bro כנראה שאין תרחיש אותו אי אפשר לתפוס...

תודות

עמית פורת - על העזרה והליווי בכתיבה

על המחבר

יואב קמיר בן 22, כארבע שנים בתחום, עוסק בעיקר בעולם ה-network forensics. לשאלות / הערות / תיקונים אשמח לקבל מייל בכתובת: yoavkamir@gmail.com

מקורות

ה-Documentation מהאתר הרישמי של Bro:

https://www.Bro.org/documentation/index.html

קישור להורדת הכלי ptunnel:

http://www.mit.edu/afs.new/sipb/user/golem/tmp/ptunnel-0.61.orig/web