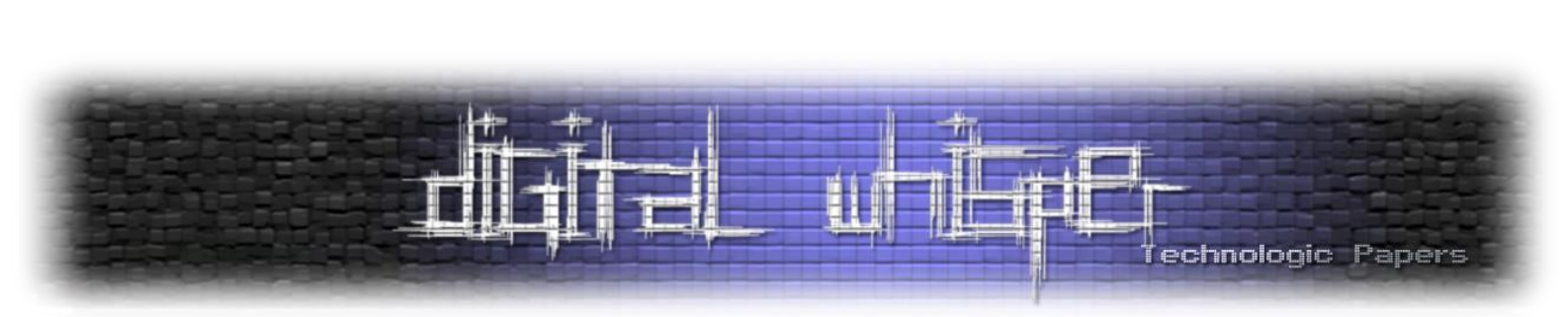

טכניקות אחיזה בלינוקס

מאת ירין הרמן

הקדמה

ברכותיי תוקף יקר! השגת גישה למערכת הלינוקס הקרובה לביתך (או רחוקה מאוד). מה אתה עושה מפה? מתחיל לכרות ביטקוין? מעולה כסף זה מצרך חשוב. מצרף את המכונה למערך botnet משומן היטב? קדימה למתקפת ה-Ddos הבאה. הכל טוב ויפה, אבל מה קורה כאשר צוותים יתחילו לחקור את הפעולות שלך? וכאשר הם יתחילו לחסום אותך? האם הכל יאבד? אחרי כל העבודה הקשה שעשית?

כאן נכנסת לתמונה אחיזה. אחיזה היא טכניקה קריטית במתקפות סייבר (שאמנם היא אינה שלב ב- Cyber KillChain, אבל לגמרי צריכה להיות כזו), שמטרתה לאפשר לתוקף לשמור על גישה מתמשכת למערכת, גם לאחר שהמערכת אותחלה מחדש, או לאחר פעולות מניעתיות מצד הקורבן.

טכניקות אחיזה משומשות בדרך כלל על ידי תוקפים, כדי להבטיח שהם יכולים להמשיך לבצע פעולות זדוניות, לאסוף מידע, או לתקוף חלקים אחרים ברשת הקורבן לאורך זמן. אחיזה יכולה להתבצע בדרכים רבות, תוך ניצול חולשות במערכת הפעלה, בהגדרות אבטחה, או בתהליכי עבודה שגרתיים.

רובנו מכירים את טכניקות האחיזה הידועות ב-Windows (סליחה על הקללה)- משימה מתוזמנת דרך task scheduler, השמת ערך registry בכל מיני נתיבים על מנת שקובץ יעלה ב-startup וכדומה. אבל איך זה עובד בלינוקס? הרי בלינוקס הכל קבצים (ואתם יודעים מה אומרים על קבצים).

במאמר זה אסקור את כל הדרכים המקובלות וה-לא מקובלות להשגת אחיזה בלינוקס. חלקן מתבססות על פיצ'רים בתוך ה"קישקע" של לינוקס, אז לאורך המאמר אסביר על כמה נקודות חשובות בנושא linux internals.

לאורך המאמר אני מציין נתיבי קבצים שונים עבור טכניקות האחיזה השונות. אותם נתיבים הם דיפולטים ברוב הפצות לינוקס, אך הן עשויות להשתנות בגרסאות שונות ומשונות. אני מציע לכם לגגל נתיב ספציפי אם אתם לא מוצאים את הנתיב שציינתי כאן.

קריאה נעימה!

משימות מתוזמנות וחברים

משימות מתוזמנות הן תהליכים אוטומטיים המוגדרים לפעול בזמנים או במרווחי זמן מוגדרים ללא התערבות ידנית. הם נמצאים בשימוש נרחב בכל מערכת ההפעלה, לביצוע פעולות שגרתיות, משימות תחזוקה ואוטומציה של פעולות שחוזרות על עצמן.

המטרה העיקרית של משימות מתוזמנות היא לשפר את היעילות והעקביות בניהול המערכת ובביצוע האפליקציות. שימושים נפוצים למשימות מתוזמנות כוללים הפעלת גיבויים, ביצוע תחזוקת מערכת, שליחת מיילים אוטומטיים, השקת סקריפטים ועדכון יישומים.

ב-Windows, השירות Task Scheduler מנהל את המשימות האוטומטיות הללו, בעוד שבמערכות מבוססות יוניקס, יש כמה מנגנונים שונים לתפעול משימות כאלה. מנגנוני תזמון אלו מאפשרים למשתמשים להגדיר טריגרים ספציפיים לביצוע משימות, כגון לוחות זמנים מבוססי זמן (למשל, יומי, שבועי או חודשי), אירועי מערכת (למשל, הפעלה או התנתקות), או אפילו תנאים מותאמים אישית.

משימה מתוזמנת באמצעות Time Unit תחת Systemd

Systemd הוא מנהל שירותים (init system) מודרני ורב-תכליתי המשמש לאתחול, ניהול ותפעול שירותים ותהליכים במערכות מבוססות לינוקס. הוא מחליף מנהלי שירותים מסורתיים כמו SysVinit ו-Upstart ומציע תכונות מתקדמות יותר. יש לו כמה תכונות מרכזיות, כמו ניהול שירותים בצורה יעילה עם systemctl, אתחול שירותים בצורה מקבילית, מנגנון איסוף לוגים מתקדם וגם תמיכה בתזמון משימות (ספוילר ☺)

לפני שנדבר על time unit, צריך להבין איך שירות עובד בלינוקס. Service Unit היא יחידה שמשמשת לניהול והגדרה של שירותים. Service Unit מאפשר להגדיר את התנהגות השירות, להגדיר מתי ואיך להפעיל אותו, ולציין תלות בין שירותים שונים.

הקבצים (או יחידות) משתמשים בפורמט טקסט פשוט עם סיומת Service., עם סינטקס שכולל מספר שדות:

חלק בקובץ	שם השדה	משמעות
תחת Unit	Description	תיאור השירות
	After	מגדיר סדר הפעלה
תחת Service	ExecStart	הפקודה או הסקריפט שירוצו בעת הפעלת השירות.
	ExecStop	הפקודה שתבוצע בעת עצירת השירות.
	Type	סוג ההפעלה של השירות
	Restart	הגדרת התנהגות מחדש אם השירות נכשל
תחת Install	WantedBy	איזו Unit יעד צריכה לכלול שירות זה כאשר הוא מופעל

יש עוד הרבה שדות שאפשר להכניס, אלה המרכזיים. אם אתם רוצים להעמיק בפורמט, יש מאמר מעולה לזה כאן. חדי העין ישימו לב שהשדות Type ו-WantedBy די דומים, אך הם לא:

- **Type** משמש כדי לציין איך יש להפעיל את השירות. הוא מגדיר את סוג ההפעלה של התהליך ומשפיע על האופן שבו systemd מטפל בשירות.
- **WantedBy** משמש כדי להגדיר תלות ב-target, וקובע מתי ה-Unit תאותחל. אפרט יותר על שדה זה בחלק על סקריפטי startup.

קבצים אלה מוגדרים בנתיבים הבאים:

/usr/lib/systemd/system/

/etc/systemd/system/

אחרי שהבנו מה זה Service Unit, נעבור לדבר על הדבר שלשמו התכנסו:

Time Unit הוא רכיב ב-Systemd המאפשר לתזמן ביצוע פעולות במועדים מוגדרים או במרווחי זמן קבועים. הוא משמש כתחליף מתקדם ל-Cron, שעליו גם נדבר עוד מעט. Time Units הם קבצים בעלי סיומת timer., שמגדירים מתי להפעיל שירות. הם פועלים יחד עם Service Units, שמבצעים את הפעולות בפועל.

קובץ Time Unit כולל שלושה חלקים עיקריים:

- **[Unit]**: תיאור כללי ותלות
- **[Timer]**: הגדרות התזמון
- **[Install]**: הגדרות התקנה (לדוגמה מתי הטיימר יופעל).

```
[Unit]
Description=Malicious Timer

[Timer]
OnBootSec=1min
OnUnitActiveSec=30min

[Install]
WantedBy=timers.target
```

[דוגמה של *Time Unit*]

שני השדות שרואים בקובץ - OnBootSec ו-OnUnitActiveSec קשורים לתזמון של המשימה:

- **OnBootSec**: מגדיר טיימר ביחס למועד האתחול של המכונה. לדוגמה "OnBootSec=5min" יתחיל את המשימה 5 דקות לאחר אתחול המערכת.
- **OnUnitActiveSec**: מגדיר טיימר ביחס למועד שבו היחידה שהטיימר מפעיל הופעלה לאחרונה. זה מאפשר ביצוע חוזר של המשימה.

אז בקצרה, OnBootSec זה טיימר ל-Startup ו-OnUnitActiveSec זה טיימר לתזמון המשימה.

אופציה נוספת להגדרת אחיזה במערכת היא ע"י השדה OnCalendar. שדה זה עובד ע"פ התחביר הבא:

```
OnCalendar=DayOfWeek Year-Month-Day Hour:Minute:Second
```

זה מאפשר ריצה חוזרת במרווחי זמן קבועים. לדוגמה, OnCalendar=Mon יריץ את השירות כל שבוע ביום שני בשעה 00:00. שילוב של שדה זה ביחד עם השדה Persistent מאפשר אחיזה עמוקה מאוד במערכת.

השדה Persistent הוא בוליאני. אם טיימר היה אמור להפעיל שירות במהלך שהמערכת הייתה כבויה, הוא יופעל ברגע שהמערכת תחזור לאינטרנט. כאשר השדה מוגדר כ-false, השירות לא ירוץ למרות שהטיימר "קפץ".

כמובן שיש עוד שדות שאפשר לכתוב בתוך הקובץ, אך אלה הם המרכזיים. נראה שחסר לנו שדה חשוב מאוד לא? השדה שמגדיר איזה שירות להעלות אחרי סיום הטיימר. צר לי לאכזב אתכם אבל השדה הזה לא קיים. במקום זה, הטיימר עצמו צריך להיות מוגדר באותו השם כמו השירות (סיומת שונה כמובן), ו-systemd יידע להפעיל את השירות בעקבות השם הזהה.

כדי שהטיימר יתחיל לעבוד, צריך להריץ 2 פקודות:

```
sudo systemctl enable malicious.timer
sudo systemctl start malicious.timer
```

פקודות אלה מאתחלות את הטיימר ומתחילות את הריצה שלו. אז כדי ליצור משימה מתוזמנת עם שירות, אנחנו צריכים:

ליצור שירות כלשהו באמצעות Service Unit File

ליצור Time Unit File עם הגדרות הריצה שלנו

לאתחל ולהריץ את הטיימר

משימה מתוזמנת (Crontab)

Cron Job הוא מנגנון תזמון משימות בלינוקס שמאפשר להריץ פקודות או סקריפטים במועדים קבועים או במרווחי זמן מוגדרים. הוא מנוהל על ידי ה-Daemon (תהליך רקע) cron, שבודק באופן מתמיד אם יש משימות שעליו להריץ בהתבסס על ההגדרות בקובצי הקונפיגורציה. אפשר לערוך את רשימת המשימות באמצעות הפקודה crontab -e. התחביר ליצירת משימת Cron קבוע ועובד לפי הסדר הבא:

```
* * * * * command_to_run
| | | | |
| | | | ---- היום בשבוע (0-7, כאשר 0 ו-7 הם יום ראשון)
| | | ------ חודש (1-12)
| | -------- יום בחודש (1-31)
| ---------- שעה (0-23)
---------- דקה (0-59)
```

כך, לדוגמה הפקודה:

```
0 3 * * * /usr/bin/cleanup.sh
```

תריץ את הסקריפט cleanup.sh כל יום בשעה 3 בלילה. כאשר יש כוכבית בשדה, השדה מוגדר ל-always.

ישנם שני נתיבים שונים שאפשר לכתוב לתוכם פקודות cron:

- /etc/crontab - משפיע על כל המשתמשים במערכת,
- /var/spool/cron/ - משפיע על יוזרים ספציפיים.

משימות At

משימות at הן מנגנון תזמון בלינוקס המיועד להרצת פקודות או סקריפטים פעם אחת בלבד, במועד עתידי מסוים. בשונה מ-Cron, שמתאים לתזמון משימות חוזרות, at מיועד למשימות חד-פעמיות. משימות at מוגדרות באמצעות הפקודה at, והן מנוהלות על ידי ה-atd deamon, שפועל ברקע ומבצע את המשימות המתוזמנות בזמן שהוגדרו.

התחביר של פקודות at פשוט מאוד, ולכן ניתן להשיג אחיזה באמצעות משימות אלה בצורה קלה יחסית: at ולאחריה זמן כלשהו, לדוגמה at 14:00, שיגרום להרצת פקודה בשעה 14:00, פותח ממשק אינטראקטיבי לכתיבת הפקודות. ניתן לראות את כל המשימות באמצעות הפקודה atq. ניתן למחוק משימות מסוימות באמצעות הפקודה atrm. בגלל שמשימות at מיועדות לרוץ פעם אחת בלבד, השגת אחיזה איתן זה משימה קצת יותר מסובכת. פעולות באמצעות at יכולות להיות עבור הסוואת/עיכוב פעילות, לדוגמה:

```
at now + 45 minutes
```

תריץ פקודה כלשהי 45 דק לאחר כתיבתה, מה שיכול להסוות במעט פעילות של תוקף. אם ברצונכם ליצור משימה מתוזמנת "כהלכה", שתרוץ במרווחי זמן קבועים, עדיף להשתמש ב-Cron או Time Unit.

סקריפטי Startup למינהם

סקריפט Startup הוא סקריפט שמוגדר להרצה באופן אוטומטי בעת עליית המערכת או בעת התחברות משתמש. בלינוקס, תהליך ה-startup כולל את טעינת מערכת ההפעלה, הפעלת שירותים בסיסיים, והרצת סקריפטים או קבצים שהוגדרו להיטען בעת האתחול.

בזמן עליית מערכת - Service Unit File

על מה זה Service Unit דיברנו בחלקו הראשון של המאמר, משימות מתוזמנות, תחת השגת אחיזה באמצעות Time Unit. באמצעות אותו כלי, של Service Unit, אפשר להשיג גם אחיזה בצורה של סקריפט Startup. אגיד בקצרה שניתן להשתמש גם ב-Time Unit כדי ליצור אחיזה ב-Startup, באמצעות שימוש בשדה OnBootSec, אך זה שימוש קצת מסורבל בפיצ'ר זה, וניתן לקבל אחיזה ב-Startup דרך ה- Service Unit עצמו.

כעת אתמקד באחד מהשדות WantedBy (תחת Install). שדה זה מגדיר את קבוצת ה-"Targets" שאליה משתייך השירות. בלינוקס יש מספר 'מצבים' של מערכת ההפעלה - קוראים לכך runlevel. המושג Runlevel מתייחס למצב פעולה של מערכת ההפעלה המוגדר מראש כמספר שלם בודד שיכול לנוע בין אפס לשש.

- RunLevels מתייחסים ל-Targets ספציפיים ב-Systemd.
- Systemd-targets הן שיטות להפעלת המערכת. הן קבוצות של יחידות (Units) שמגדירות מצבי מערכת ספציפיים.

ניתן לתאר את כל ה-Targets השונים ומקביליהם ב-Runlevel כך:

RunLevel	Target	תיאור	שימוש נפוץ
0	poweroff.target	כיבוי המערכת.	כיבוי בטוח של המערכת.
דומה ל-0	halt.target	עצירת כל התהליכים במערכת ללא כיבוי.	לא נפוץ
1	rescue.target	מצב משתמש יחיד (Single User Mode).	תחזוקה או פתרון בעיות. מאפשר גישה בסיסית למערכת ללא רשת או משתמשים נוספים.
3 2 אם ללא רשת	multi-user.target	מצב רב משתמשים בממשק טקסטואלי (Multi-User Mode).	מערכת פועלת במצב טקסטואלי מלא עם שירותי רשת ושירותים אחרים.
5	graphical.target	מצב רב משתמשים עם ממשק גרפי.	מערכת פועלת עם GUI ושירותי רשת.
6	reboot.target	אתחול מחדש של המערכת (Reboot).	אתחול המערכת מחדש בצורה מסודרת.
אין מקביל	shutdown.target	תהליך כיבוי מסודר של המערכת.	לעצירת כל השירותים לפני מעבר ל־poweroff.target.

כאשר מערכת ההפעלה עולה, הקרנל מפעיל את התהליך הראשי (systemd) כ־PID 1. systemd בודק את קובץ הגדרת ה-traget. רוב המחשבים מוגדרים עם graphical.target או multi-user.target (תלוי במערכת). systemd טוען את אותו ה-target, מכיל תלות בשירותים ו־targets אחרים שמוגדרים ב-wants ו-requires. המערכת טוענת את כל השירותים והתלויות הנדרשות וכך מערכת ההפעלה עולה.

אזי, בשביל להשיג אחיזה של סקריפט startup, צריך להגדיר בשדה ה-WantedBy את ה-target המתאים למערכת, כך שבזמן עליית ה-target השירות ייטען.

Shell Configuration - On user login

ה־Shell הוא ממשק בין המשתמש לבין מערכת ההפעלה. הוא מאפשר לבצע פקודות, להריץ סקריפטים ולנהל את המערכת. קבצי קונפיגורציית shell הם סקריפטים הפועלים לאורך הפעלת משתמש בהתבסס על אירועים (למשל, כניסה/יציאה או פתיחה/סגירה של shell). קבצים אלה משמשים להתאמה אישית של ה-shell, כולל הגדרת משתני סביבה והגדרות אחרות ספציפיות להפעלה. ל-shells שונים יש קבצי קונפיגורציה משלהם. בגלל זה, נתמקד בעיקר בקבצים של shell של bash - קיצור של Bourne Again Shell, מה שנחשב ל-shell הנפוץ ביותר. ישנם כמה קבצים מאוד חשובים ל-shell הזה:

נתיב ושם הקובץ	משמעות
etc/profile/	קובץ קונפיגורציה גלובלי (מערכתי) שנטען עבור כל המשתמשים.
~/.bash_profile	קובץ קונפיגורציה אישי, נטען עבור משתמש ספציפי.
~/.bash_login	מכיל הגדרות אישיות עבור כל session של Login Shell.
~/.profile	קונפיגורציה למשתמשים שאינם משתמשים ב־Bash בלבד.
~/.bashrc	נטען עבור Non-Login Shell

הקבצים האלו נטענים בזמן התחברות משתמש (login). ישנו רצף טעינה אחיד לקבצים האלה, בהתאם לסוג ה-shell שנפתח - Login ו-Non-Login Shell. הכוונה ב-Non Login Shell היא ל-shells שנפתחים מתוך GUI לדוגמה. רצף הטעינה ב-Login Shell הוא:

1. קבצי מערכת גלובליים (כמו /etc/profile/), שעשויים לטעון קבצים נוספים תחת /etc/profile.d/ (למשל /etc/profile.d/*.sh).
2. קובצי קונפיגורציה אישיים, תלוי בסדר העדיפות של הקבצים האישיים:
   תחילה ~/.bash_profile.
   אם ~/.bash_profile לא קיים אז ~/.bash_login ייטען.
   אם ~/.bash_login לא קיים אז ~/.profile ייטען. רק אחד מהקבצים האישיים נטען, לפי סדר זה.

רצף הטעינה של Non-Login Shell כולל בתוכו רק קובץ אחד: ~/.bashrc.

על מנת אחיזה באמצעות קבצים אלה, כל שצריך לעשות הוא להזריק קוד כלשהו לתוך אחד מהקבצים האלה (כמובן שצריך לשים לב לסדר הטעינה שלהם). הקבצים האלה מורצים כפקודות, ולכן, כדי להשיג הרצת סקריפט כלשהו, מזריקים קוד בפורמט הבא:

```
bash ~/<path_to_script.sh>
```

התחלה אוטומטית באמצעות XDG

XDG (ראשי תיבות של X Desktop Group, כיום חלק מפרויקט Freedesktop.org) הם סט של תקנים שנועדו ליצור סביבת שולחן עבודה אחידה במערכות לינוקסיות. התקנים מספקים שיטה מאורגנת לניהול קבצי קונפיגורציה, ואחראים על הגדרת התנהגות של יישומים, כאשר משתמש מתחבר למערכת. המטרה המרכזית של התקנים היא לארגן קבצי הגדרות משתמשים בתיקיות סטנדרטיות כדי למנוע בלגן בקבצי הבית (~), ולאפשר יישומים ושירותים לפעול בצורה אחידה על פני שולחנות עבודה שונים, כמו GNOME, KDE, ו-XFCE.

ישנם הרבה תקנים שונים שנוגעים לנושאים שונים במערכת ההפעלה (תיקיות בית, פורמט לקיצורי דרך וכו'). התקן של XDG Autostart הוא חלק מעניין במיוחד, שכן הוא מגדיר דרך להפעלת יישומים באופן אוטומטי באמצעות Desktop Entry Files, שהם קבצי טקסט פשוטים עם סיומת desktop.

קובצי desktop. משמשים בדרך כלל לקביעת אופן הצגת יישומים בתפריטים ואופן הפעלתם. על ידי שימוש ב־XDG Autostart, תוקפים יכולים להגדיר יישומים זדוניים שיפעלו באופן אוטומטי בכל פעם שמשתמשים מתחברים לסביבת שולחן העבודה שלהם.

לקבצים כאלה יש סינטקס עם שדות מסוימים (שדומים מאוד לשדות של Service Unit):

- Type=Application מציין שהקובץ הזה מגדיר יישום שניתן להפעילו (ערכים נוספים יכולים להיות ספרייה או קישור, במקרה שלנו זה כנראה יהיה אפליקציה)
- Exec=path_to_script מציין מה יש להריץ, Hidden הוא ערך בוליאני שמגדיר האם הקובץ יהיה מוסתר או לא
- NoDisplay הוא עוד ערך בוליאני שקובע האם הקובץ יופיע בממשק המשתמש
- X-GNOME-Autostart-enabled מאשר שהקובץ נועד לפעול בעת התחברות משתמש בסביבת GNOME ולבסוף ישנם שדות name ו-comment שמשמעותם מובנת מאליה.

כך למשל, קובץ .desktop במערכת עשוי להיראות כך:

```
[Desktop Entry]
Type=                        Application
Exec=                        /usr/local/bin/update_checker.sh
Hidden=                      false
NoDisplay=                   true
X-GNOME-Autostart-enabled=   true
Name=                        System Update Checker
Comment=                     Ensures your system is always up-to-date
```

מיקומם של קבצי ה-Autostart משתנה בהתאם לשאלה האם הקובץ רלוונטי עבור כל המשתמשים (ברמת המערכת כולה), או עבור משתמש ספציפי. כמו כן, המיקום תלוי בסביבת שולחן העבודה הנמצאת בשימוש.

ברירת המחדל לקבצי autostart ברמת המערכת נמצאת בתיקיות שדורשות הרשאות Root לשם שינוי, כגון /etc/xdg/autostart/ ו-/usr/share/autostart/. קובצי autostart ברמת משתמש ספציפי נמצאים לרוב בנתיב ~/.config/autostart/

ניתן להשיג אחיזה ב-startup באמצעות מנגנון זה באמצעות יצירה/מודיפיקציה של אחד מקבצי ה-desktop. הערך החשוב ביותר בשדה הוא exec, שיכול לכלול נתיב לסקריפט או פקודה כלשהי.

דרכי אחיזה נוספות

מניפולציה של מפתחות SSH

SSH או בשמו המלא Secure Shell הוא פרוטוקול לגישה מאובטחת למערכות מרוחקות. הפרוטוקול משתמש בצמדי מפתחות ציבוריים/פרטיים כדי לאמת משתמשים. אותם צמדים מורכבים ממפתח פרטי, השמור מאובטח על ידי המשתמש, ומפתח ציבורי, המשותף עם המערכת המרוחקת.

ישנם מגוון קבצים במערכת שאחראים על התחברות מרוחקת, המרכזיים שבהם הם ~/. ssh/id_rsa, /root/.ssh/id_rsa, ~/.ssh/authorized_keys ו-/root/.ssh/authorized_keys. כל הקבצים הללו הם ייחודים למשתמש. הקובץ שאחראי לקונפיגורציה ברמת המערכת נמצא ב-/etc/ssh/.

במהלך התחברות, המפתח הפרטי נשאר במחשב הלקוח, בעוד המפתח הציבורי מועתק לקובץ authorized_keys של המערכת.

תוקף יכול לבסס אחיזה באמצעות הוספת מפתח ציבורי לקובץ authorized_keys. פעולה זו מבטיחה את היכולת לקבל בחזרה גישה למערכת גם אם המשתמש משנה את הסיסמה שלו.

יצירת משתמש והכנסתו ל-Sudoers

ישנן שתי גישות עיקריות לביצוע הפעולות הבסיסיות האלה: שימוש בכלים המובנים של המערכת או עריכה ישירה של קובצי מערכת כמו /etc/passwd.

שימוש בפקודות ובכלים המובנים במערכת הוא די פשוט מאליו, מוסיפים משתמש באמצעות הפקודה useradd (אפשר להוסיף לו תיקיית בית ו-shell באמצעות פרמטרים שונים בפקודה). ניתן להוסיף/לשנות סיסמה באמצעות הפקודות passwd/chpasswd.

לאחר מכן ניתן להוסיף למשתמש הרשאות sudo באמצעות הפקודה usermod -aG sudo user.

עריכת הקבצים האחראיים על המשתמשים, הסיסמאות והרשאות ה-Sudo במערכת, נותנת גישה עקיפה יותר להשגת אחיזה מסוג זה, אחיזה שגם עוקפת תיעוד בלוגים.

הקבצים הרלוונטים ליצירת משתמש והכנסתו ל-sudoers הם: /etc/shadow/ ,etc/passwd ו-/etc/sudoers.

נתיב הקובץ	משמעות	תחביר של שורה
/etc/passwd	מאחסן מידע על כל המשתמשים במערכת, כולל UID ו-GID.	פורמט: username:x:UID:GID:comment:home_directory:shell
/etc/shadow	מאחסן סיסמאות מוצפנות.	פורמט: username:hashed_password:lastchanged:min:max:warn:inactive:expire
/etc/sudoers	מגדיר מי יכול להשתמש בפקודת sudo ואילו הרשאות יש להם בעת הרצת פקודות עם הרשאות root.	פורמט: username host=(user) commands כאשר host הוא שם המחשב (לרוב ALL), ו-user זה המשתמש שעליו מורצות הפקודות (בדרך כלל גם ALL)

את שני הקבצים הראשונים ניתן לערוך באמצעות עורך טקסט כלשהו (nano או vim לדוגמה), אך את הקובץ etc/sudoers מומלץ לערוך באמצעות הפקודה visudo, שכן היא מבצעת וולידציה על הסינטקס של התוכן בקובץ. אם הסינטקס לא נכון, ייתכן שאולי ניאלץ לעשות boot למערכת ב-recovery mode.
אם רוצים להימנע מלוגים - כן אפשר לערוך את הקובץ באמצעות עורך טקסט כלשהו.

שינוי Aliases

Aliases הם קיצורי דרך לפקודות, שמאפשרים למשתמשים להגדיר פקודות קצרות יותר או מותאמות אישית, במקום פקודות ארוכות או מסובכות. הם מוגדרים לרוב בקובצי קונפיגורציה של ה-Shell, כמו ~/.bashrc, ~/.bash_profile, ~/.zshrc, או /etc/bash.bashrc (עבור כל המשתמשים במערכת).

כאשר משתמש מגדיר alias, כל פעם שהוא מקליד את הפקודה המותאמת אישית, המערכת מריצה את הפקודה האמיתית שמאחוריה.

אפשר להגדיר alias בצורה מאוד פשוטה:

```
alias ll='ls -lah'
```

כעת הפקודה ll מבצעת את הפקודה ls -lah. נשמע כמו חלום של כל תוקף לא? לגרום למשתמש להריץ מאחורי הקלעים את הפקודות הזדוניות שלו. חוץ מפקודות בסיסיות כמו wget וכו', ניתן גם להריץ סקריפטים שלמים כ-alias.

כך ניתן ליצור סקריפטים שלמים שמתבצעים מאחורי הקלעים ללא ידיעת המשתמש, או עם ידיעת המשתמש, פשוט מבלי לדעת שמדובר בסקריפט זדוני:

```
alias sudo='read -p "[sudo] password for $USER: " pass; echo "$USER:$pass" >>
~/.mal_log; /usr/bin/sudo'
```

בדוגמה כאן נוצר alias לפקודה sudo, שמבקש קלט של סיסמה. לעיתים פקודת sudo באמת דורשת סיסמה, אבל כאן מדובר סקריפט זדוני שכותב את הקלט אל תוך קובץ לוגים של התוקף.

האפשרויות לאחיזה באמצעות alias באמת בלתי נגמרות, בגלל הוורסטיליות הרבה של הפיצ'ר.

שימוש ב-LD_PRELOAD

LD_PRELOAD הוא משתנה סביבה המאפשר שליטה על טעינת ספריות משותפות (Shared Libraries) לפני כל ספרייה אחרת בתהליך. כאשר תוכנה נטענת, מנהל הקישורים הדינאמיים (Dynamic Linker) אחראי על חיפוש וטעינת הספריות הדרושות לתוכנה. LD_PRELOAD מכריח את מנהל הקישורים להעדיף ספרייה מסוימת (המוגדרת על ידי המשתמש) על פני ספריות אחרות, כולל הספריות הסטנדרטיות במערכת.

אם הספרייה המוזרקת מכילה פונקציות בעלות אותו שם כמו אלו שבספריות המערכת (למשל printf), הפונקציות שבספרייה המוזרקת יקבלו עדיפות ויופעלו במקום המקוריות.

גישה זו משמשת למטרות לגיטימיות כמו איתור באגים, בדיקות ביצועים או הוספת תאימות מבלי לשנות את הקוד המקורי. לדוגמה, מפתח תוכנה יכול להשתמש ב-LD_PRELOAD כדי להחליף פונקציות בספרייה מסוימת ולנתח את התנהגותן בזמן ריצה.

עם זאת, LD_PRELOAD יכול גם להיות מנוצל לרעה (ולצערנו זה לרוב שימושו כיום). תוקפים עשויים ליצור ספריות מותאמות אישית שמחליפות פונקציות קריטיות כמו read, write, או open, ולהזריק ספריות אלה כך שהם ישיגו אחיזה על המערכת.

אחת הדרכים הפשוטות להשתמש ב-LD_PRELOAD לצורך ניצול, היא הגדרת המשתנה בקבצי קונפיגורציה שמשתמשים טוענים אוטומטית, כמו bashrc. או profile./~.

הגדרה זו תבטיח שכל תהליך חדש שהמשתמש מריץ יטען את הספרייה הזדונית. לדוגמה, ניתן להוסיף את השורה הבאה לקובץ .bashrc של המשתמש:

```
export LD_PRELOAD=/path/to/malicious_library.so
```

בפועל, זה יגרום לטעינת הספרייה הזדונית בכל הפעלה של פקודה חדשה, ובכך תוקף יכול לשמור על שליטה מתמשכת במערכת.

על התהליך המלא של הזרקת SO באמצעות LD_Preload לא אפרט, כיוון שנכתב על כך כבר בגיליון 124.

השתלת מודולי קרנל

LKM או בשמם המלא Loadable Kernel Modules, הם רכיבי תוכנה שניתן לטעון ולפרוק באופן דינמי לתוך הקרנל של לינוקס, ללא צורך באתחול מחדש של המערכת. הם מרחיבים את הפונקציונליות של הקרנל, על ידי הוספה או שינוי של מנהלי התקנים, מערכות קבצים, פרוטוקולי רשת ותכונות חיוניות אחרות ברמת הקרנל.

מודול קרנל הוא קובץ בינארי המורכב מקוד שעבר קומפליציה, שנטען לקרנל של לינוקס. המודול נכתב בדרך כלל בשפת C, ומשתמש בממשקי API שמספקת הליבה כדי לתקשר עם הקרנל ושאר חלקי המערכת. המודולים נטענים כקבצים בעלי סיומת .ko, כלומר kernel object.

הקרנל פועל בארכיטקטורה מונוליטית, שבה מערכת ההפעלה כולה פועלת במרחב הקרנל, אך היא גם מודולרית, המאפשרת הרכבה של מודולים שניתן לטעון ולפרוק בזמן ריצה.

טעינת מודול לקרנל מתבצעת בדרך הבאה:

פקודת טעינה: המשתמש או סקריפט מפעילים פקודה כמו insmod או modprobe כדי לטעון את המודול.

איתור המודול: הקרנל מחפש את הקובץ המתאים בספריית המודולים (בדרך כלל -/lib/module/<kernel version>)

טעינה למרחב הזיכרון: הקרנל טוען את הקוד של המודול לזיכרון ומבצע קריאה לפונקציית האתחול של המודול (בדרך כלל נקראת init_module).

רישום: המודול נרשם בליבה, ומתחיל לספק את הפונקציונליות המוגדרת בו (כמו תמיכה בחומרה חדשה או ניהול תהליכים).

הסרה (אם נדרש): ניתן להסיר את המודול עם פקודה כמו rmmod, אשר תגרום לקריאה לפונקציית הניקוי (clean-up) של המודול (cleanup_module).

תוקפים ישתלו LKM-ים מ-2 סיבות עיקריות

ראשית, LKMs פועלים עם רמת ההרשאות הגבוהה ביותר- רינג 0. rings הן רמות הרשאות השולטות בגישה למשאבי המערכת וקובעות אילו פעולות ניתן לבצע על ידי חלקים שונים של המערכת. לינוקס משתמשת בעיקר בשתיים מתוך ארבע ה-rings הקיימים:

- רינג 0 (kernel mode): זוהי הרמה עם רמת ההרשאות הגבוהה ביותר , שבה פועל הקרנל.
- רינג 3 (user mode): זוהי הרמה עם הרשאות פחותות, שבה פועלים יישומי משתמש.

אם כך, בשל ההרשאות הגבוהות ש-LKM-ים רצים, תוקפים יכולים לנצל את זה כדי ליצור rootkits שמתחמקים מזיהוי ויוצרים אחיזה חזקה במערכת.

שנית (ובהמשך לנקודה הקודמת), LKM-ים יכולים לחמוק מזיהוי ע"י כלי הגנה למיניהם, בעזרת מודיפיקציה ו-hooking של מגוון דברים. לדוגמה, תוקפים יכולים לשנות את טבלת ה-syscall כדי ליירט אותם, ולהזריק קוד זדוני. משמע, אותם LKM-ים יכולים לשמש כ-Rootkit.

והדבר הכי חשוב מכל - LKM-ים יכולים ליצור אחיזה מאוד חזקה במערכת שקשה לגלות. בנוסף, תוקפים יכולים לגרום ל-LKMs זדוניים לטעון אוטומטית במהלך אתחול המערכת, על ידי השמתם בספריות ספציפיות כגון: /etc/modules/, /lib/modules-load.d/ או /usr/lib/modules-load.d/.

נקודה חשובה: השתלת מודולים בקרנל המערכת לרוב דורש הרשאות root.

סיכום

לאורך המאמר עברנו על טכניקות שונות להשגת אחיזה בלינוקס. כלומר, איך לשמר את הגישה שלך למערכת, גם לאחר פעולות מצד צוותים כחולים ו/או אתחול המערכת. בהתחלה דיברנו על משימה מתוזמנת, שהיא, כשמה כן היא, משימה שקורית כל פרק זמן מוגדר.

המנגנון המיועד לכך נקרא cron job וניתן לבצע משימות מתוזמנת גם באמצעות התעסקות עם התהליך הראשון במערכת - systemd. אפשר להגדיר שירות (באמצעות קובץ ספציפי) ולהגדיר לו קובץ טיימר שיפעיל את השירות בכל פרק זמן מוגדר. בנוסף קיימות משימות מסוג at, שנועדו להתבצע פעם אחת בלבד, בפרק זמן עתידי. ניתן להשתמש גם בהן להשגת משימה מתוזמנת, אבל בעיקר (ועדיף) להשתמש בסוג זה של משימות עבור טשטוש עקבות.

לאחר מכן דיברנו על סקריפטי startup, כלומר סקריפטים כלשהם שעולים עם עליית מערכת ההפעלה. לשם כך חזרנו לדבר על תהליך ה-systemd, כיוון שהוא התהליך הראשון שעולה במערכת ההפעלה. ניתן להגדיר עליית שירות בהתאם לשלב מסוים בעליית מערכת ההפעלה. שלב זה נקרא target. מעבר לזאת, אפשר להגדיר סקריפטים מסוימים שיופעלו בעת התחברות של משתמש מסוים למערכת (תלוי בסוג ה-

shell, במאמר דיברנו על shell מסוג bash). לבסוף, עברנו על התחלה אוטומטית של סקריפטים באמצעות XDG, שהם סט של תקנים שנועדו ליצור סביבת שולחן עבודה אחידה.

הנושא הבא שדיברנו עליו הוא "דרכי אחיזה נוספות" - שיטות שאינן תואמות לשתיים הקודמות אך עדיין מספקות אחיזה איתנה במערכת. התחלנו עם פרוטוקול SSH, שהוא פרוטוקול התחברות מאובטח למערכות לינוקסית. תוקף יכול להחדיר לתוך קבצי ההגדרה של פרוטוקול זה מפתח, כך שתישמר לו האחיזה במערכת גם אם המשתמש יחליף סיסמה. בהמשך למשתמש וסיסמה - עברנו על שיטת האחיזה הנפוצה והקלה ביותר - הוספת משתמש חדש למערכת והוספתו לקובץ sudoers, שמאפשר הרצת פקודות באמצעות root (בלי להיות כזה). לאחר מכן דיברנו על עולם ההשתלות וההזרקות. עברנו על המשתנה LD_Preload, שמאפשר לטעון ספריות משותפות לפני כל ספרייה אחרת בעת עליית תהליך, ועל המשמעות של טעינת ספרייה זדונית.

ואחרון חביב: מודולי קרנל ומשמעותם. אלה הם רכיבי תוכנה שניתן לטעון ולפרוק באופן דינמי, מה שאפשר הרחבת פונקציונליות של המערכת. תוקף עלול להחדיר רכיבים כאלו אל המערכת ובכך להסתיר את פעילותו , ולהשיג אחיזה איתנה במרחב הקרנל של המערכת.

על המחבר

שמי ירין הרמן, וזה מאמרי השני. את המאמר הראשון כתבתי לפני בערך 3 חודשים, ושחררתי אותו בגיליון 168. ברגע שיצא המאמר, סבתא שלי (שהייתה אז בת 95) ביקשה שאדפיס אותו כדי שתוכל לקרוא אותו. סבתא שלי לא ידעה לתפעל טלפון חכם אבל ביקשה (או יותר נכון דרשה) לקרוא מאמר על פורנזיקה של פוגענים. לאחר כשבוע שאלתי אותה מה דעתה על המאמר והיא אמרה לי: "*ירין אני לא מבינה בזה כלום אבל אני ממש גאה בך*". היא גם דאגה לכתוב לי את זה על פתק שהיא הדביקה על ההדפסה של המאמר.

לאחר שלושה שבועות היא אושפזה בביה"ח. את המאמר הזה התחלתי לכתוב לצד המיטה שלה בבית החולים. כשאמרתי לה שאני כותב את המאמר היא חייכה וכל כך שמחה לדעת את זה. כעבור 4 ימים סבתי נפטרה. זאת הדרך שלי (גם אם קצת מוזרה) להנציח את סבתא, שכל כך הייתה גאה במאמר על פוגענים שעברו אובפוסקציה, בלי לדעת מה זה פוגענים, או אובפוסקציה.

סבתא, אני אוהב אותך ומקווה שאת תמיד תהיי גאה בי.

בנימה אחרת לגמרי, אם יש לכם שאלות, הערות או הארות לגבי המאמר - מוזמנים לכתוב לי במייל:

yarinherman05@gmail.com

מקורות

- Linux Detection Engineering - A primer on persistence mechanisms — Elastic Security Labs
- systemd/Timers - ArchWiki
- Shenanigans of Scheduled Tasks
- Understanding Systemd Units and Unit Files | DigitalOcean
- How to create a systemd service in Linux
- Scheduled Task/Job: Systemd Timers, Sub-technique T1053.006 - Enterprise | MITRE ATT&CK®
- systemd.timer
- What is the LD_PRELOAD Trick on Linux?
- Why isn't LD_PRELOAD disabled by default in Linux? - Information Security Stack Exchange
- Playing with LD_PRELOAD - BreakInSecurity
- How to Set Up a Cron Job in Linux? {Schedule Tasks}
- What's the difference between /etc/cron.d and /var/spool/cron? - Server Fault
- How to Change Runlevels (targets) in SystemD
- Shell initialization files - DEVOPS DONE RIGHT.
- Desktop Application Autostart Specification
- Detecting LD_PRELOAD Malware with libdl's Functions
- What not to add to your ~/.bashrc: good practices for the end user - HPCKP
- LD_PRELOAD - Introduction - Professionally Evil Insights
- How To Create A New Sudo Enabled User on Ubuntu | DigitalOcean
- Kernel Modules & Extensions - Red Canary Threat Report
- Introduction to Linux Loadable Kernel Modules