

התקפות ערוצי צד



תרגול 7 - הנדסה לאחור - חורף תשפ"א

©עידן רז




התקפות ערוצי צד

התקפות המבוססות על זליגה של מידע הנובע מאופן היישום הפיזי או השימוש של המערכת.

ישנם הרבה ערוצי צד עליהם אנחנו יכולים להסתכל:



- מידע תזמון
- צריכת אנרגיה
- מידע אקוסטי
- תוכן ה-cache 

התקפות תזמון

- קחו לדוגמא את הקוד הבא:

```
1 bool validate_password(char* actual_pw, char* typed_pw){
2     if (strlen(actual_pw) != strlen(typed_pw))
3         return false;
4
5     for (int i = 0; i < strlen(actual_pw); ++i){
6         if (actual_pw[i] != typed_pw[i])
7             return false;
8     }
9     return true;
10 }
11
```

- מה הבעיה בו? איך תוקף יכול לנצל זאת על מנת לגלות את הסיסמא האמיתית?



התקפות מבוססות CACHE

- ב-2017, התגלו שתי חולשות מרכזיות ברוב המעבדים המודרניים שאפשרו הדלפה של מידע פרטי מכל תהליך בעל הרשאות נמוכות
- שתי החולשות התבססו על מנגנונים שונים של מעבדים מודרניים על מנת להכניס לcache מידע מכתובות שהתהליך לא אמור להיות מסוגל לגשת אליהן ואחר כך לגלות את המידע הזה בcache באמצעות התקפת תזמון



- [Meltdown](#) – התבססה על מנגנון Out Of Order Execution



- [Spectre](#) - התבססה על מנגנון Speculative Execution

MELTDOWN

- מתבססת על קטע הקוד הבא:

```
1 ; rcx = kernel address, rbx = probe array
2 xor rax, rax
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```



- גישה לכתובת לא חוקית תביא את המידע לcache לפני שהמעבד יבין שהגישה לא חוקית



SPECTRE

- מתבססת על קטע הקוד הבא:

```
24 void victim_function(size_t x) {  
25     if (x < array1_size) {  
26         temp &= array2[array1[x] * 512];  
27     }  
28 }  
29
```

- אימון של branch predictor על ערכי x קטנים יגרום לכך שבקריאה הבאה לפונקציה עם x גדול המידע יובא לcache למרות שהגישה לא חוקית

