מבוא

מערכת בטוחה: מערכת שבה עבור כל מרכיבי המערכת נשמרות, לאורך כל זמן חיי המערכת 3 תכונות יסודיות (CIA):

1. **Confidentiality (חשאיות):** זהו מצב של המערכת שבו איננה מתאפשרת זרימת אינפורמציה אל גורמים שאינם מורשים, בכדי למנוע חשיפת מידע שעלול לגרום נזק (לארגון).
2. **Integrity (שלמות):** זהו מצב שבו המערכת וכל חלקיה במצב חוקי ועקבי- המערכת לא ניזוקה.
3. **Availability (זמינות):** זהו מצב של המערכת של רכיביה נמצאים, נגישים ומוכנים לשימוש.

שלמות ונכונות:

נכונות: המערכת עובדת לפי מפרט המערכת (נניח כי הוא נכון).
שלמות: המערכת עקבית ולא ניזוקה.
הערה: קשה למדוד/להוכיח נכונות של מערכות ולכן אנחנו מתמקדים בשלמות.
שלמות התחלתית נובעת מ: אימון ביצרן, אימות נכונות פורמלי או סטיפיקציה וסקירה ע"י הקהילה.
מדידת שלמות נדרשת בזמן: התקנה, הוספת רכיבים למערכת, boot, ריצה.

חשאיות ופרטיות:

חשאיות נדרשת כדי לספק פרטיות, אך איננה בהכרח מספקת פרטיות.
אם נדרשת פרטיות-מדיניות אבטחת המידע צריכה לציין זאת מפורשות.
הערה: קיימת סתירה מובנית בין הדרישה לפרטיות, ובין הדרישה לשלמות מבוססת 4C
"זליגת" מידע פיסית: זמן החישוב/צריכת האנרגיה מעידים על הפעולה המבוצעת, רעש מהמעבד יכול לשמש למציאת אינפורמנציה סודית המעובדת בתוכו, חום הנפלט/רעש מהמעבד מעידים על הפעולה שהוא מבצע.
אמצעי הגנה ל"זליגה": יצירת פעולות המסוככות על דברים אלה (צריכת אנרגיה קבועה לפעולה, צריכת זמן קבועה לפעולה, משתיק קול).
Electro-magnetic Pulse: פצצות שמתפוצצותמעל האטמוספירה יכולות לייצר קרינה שהורסת את כל המערכות האלקטרוניות, מבלי לפגוע בבני אדם ובעלי חיים. פיתרון אפשרי: כלוב פרדיי.

בניית תוכנה בטוחה

נקודות תורפה:

בכל מערכת (תוכנה או חומרה) קיימות נקודות תורפה- חורי אבטחה. יצרניות מערכות תוכנה/תוכנה-חומרה, כמו גם חוקרים מקהיליית האבטחה, מחפשים באופן תמידי אחר נקודות תורפה במערכות. מירב נקודות התורפה הן תוצאה של באגים בתכנות או בתכנון. בעיקר עקב חוסר בדיקת תקינות של מספר, תחום ואורך של פרמטרים.

דריסת מחסנית (Stack overrun): בדרך-כלל משנה התוקף בדריסת מחסנית את כתובת החזרה כך שתצביע לקטע קוד, שיושב בעצמו בחוצץ "הנדרס". לדוגמא: בלינוקס ניתן להריץ shellחדש שיורש את ההרשאות של התהליך הנתקף.
EGG- הקוד המותקף: צריך להיות Position Independent Code, איננו יכול להכיל תווים מיוחדים (לדוגמא: \n\) והדירסה מתבצעת באמצעות פונקציות מחרוזות כדוגמאת str*(), gets(), scanf().
פיתרון: הוסיפו דגל לחומרת ה-MMU שבעזרתו ניתן לציין כי דף בזיכרון מכיל נתונים בלבד, ואיננו ניתן לביצוע. בנוסף, יש אזורים במ"ה שאי-אפשר לעשות בהם exec code.
Return to libc() attack: שינוי מהלך הביצוע של התוכנית ע"י קפיצה למקומות שונים בזיכרון. בפרט התוקף יכול לקפוץ ישירות לתוך שירותי מערכת.

©ענת עציון

פיתרון: Adress Space Layout Rndomization (ASLR)- מנגנון שמשנה זמן Link את מיקום הספריות בזיכרון. ואז תוקף לא יכול לתשול מיקום סטטי של שירות מערכת במקום כתובת החזרה.
פיתרון ל-Buffer Overflow – קנרית: בסוף כל אזור משתנים שמים מחרוזת ייחודית ארוכה דיה, ומידי פעם בודקים שהמחרוזת תואמת לציפייה. במקרה שלא כך הדבר, יש בזיכרון דליפה מסוימת המאפשרת כתיבה על הנתונים. ברמת קומפילציה יש קוד שבודק את ה-Canary אבל אם המתקיף יודע על זה, הוא יכול לשים Canary במקום המתאים.

איך מייצרים אקראיות?

מחשבים הם דטרמיניסטיים ולכן לא יכולים לייצר אקראיות אמיתית. במקום זה הם משתמשים ב-**Pseudo-random number generators (PRNG)** כדי לייצר אותם. חושפים להתקפות (ניחוש) ולכן יש לבחון את בטיחותם.
seed: קלט נתון שממנו מייצרים מחרוזת גדולה יותר בעלת תכונות אקראיות.
אתחול seed: אתחול לפי זמן, כתובת IP, שמות וכן הלאה בעייתי- הנתונים הללו קלים לניחוש.
חלופות ל-seed: חומרה מיוחדת, קלט משתמש (קצב הקשה), מדידת זמני גישה לדיסק, שילוב...

התקפות-Exploits:

Local Exploits: התקפות שמחייבות גישה אל המערכת (כדי להפעילן צריך להיות משתמש לגיטימי של המערכת).
Remote Exploits: התקפות שניתן להפעילן "מרחוק" מבלי להיות משתמש של המערכת.
הערה: מירב ההתקפות (וירוסים, תולעים) הגדולות ניצלו חורי אבטחה שפורסמו עבורם תיקונים, זאת מכיוון שברגע שמשתחרר patch, בודדים מעדכנים, ופרטי חור האבטחה ידועים לכל.
Zero day attack: התקפה או חור אבטחה שטרם התפרסם עבורם תיקון (patch).

מדדים לביטחת של מערכת:

שטח פני ההתקפה (Surface of attack): סך כל הפעולות ומשאבי המערכת שחשופים למשתמשים. ישנן פעולות ומשאבים שמהווים מטרות יותר "מפתות", למשל כאלה שרצות בהרשאות damin או root.
וקטורי ההתקפה (Vectors of attack): דרכים אפשריות להגיע לפני השטח של ההתקפה: גישה מקומית למערכת, גישה דרך ה-web, או email.
הערה: בדרך-כלל המטרה היא להפחית ככל שניתן את שטח הפנים "המפתה" שפתוח להתקפה דרך וקטורים מרוחקים.
דרכים לצמצם את שטח פני ההתקפה:

- Firewall מפחית את מספר הדרכים בהן תוקף יכול להתחבר למערכת.
- תיעוד פעילויות במערכת (logging) יכול להפחית את הזמן הנתון לרשות התוקף לחדור למערכת ויכול לעזור לגלות בעיות אפשריות וקונפיגורציות שגויות.

מדוע עדיין יש תוכנות מזיקות:

מקומות שקוד זדוני מתחבא בהם:

- יכול לפעול כתהליך עצמאי- רוב המשתמשים לא יודעים להבחין בין תהליכים שצריכים לרוץ במערכת לאלה שלא. ולא ניתן להרשות זיהוי שגוי של תהליך תקין כתהליך זדוני (false positive)
- בתוך (הזיכרון של) תהליך רגיל- הוירוס יכול לשנות את עצמו עם הזמן כדי להימנע מגילוי. זיכרון עשוי להיות מדופדף לדיסק וגישה לקובץ הדפדוף בדיסק היא לרוב קלה יותר מגישה לזיכרון (למשל, לאחר כיבוי המחשב, קובץ הדפדוף עדיין מכיל את המידע).

©ענת עציון

- קובץ הנגיש רק ע"י התהליך: אחסון הקוד הזדוני על קובץ היא דרך טובה להבטיח את השפעתו על המערכת לאחר האתחול הבא.
- בסביבת הגרעין (kernel mode): החדרת קוד זדוני ל-kernel משמעותה שליטה מלאה במערכתץ ב-kernel mode הקוד הזדוני יכול לגשת לכל ההתקנים של המערכת.
- ב-BIOS או רכיבי זיכרון בלתי נדיפים אחרים: וירוס שפוגע ב-BIOS יכול לשנות את פעולתה של המערכת ע"י הטעיית ה-OS loader. וכאשר הקוד הזדוני נטען בשלב כה מוקדם, הוא יכול להתחבא.

עקרונות מנחים לתוכנה בטוחה:

1. **אבטחו את החוליה החלשה:** כיום קריפטוגרפיה לרוב אינה הבעייתית, אלא הנדסה חברתית.
2. **הגנה בשכבות:** אם שכבה אחת נפרצת, שכבה נוספת תמנע פריצה מלאה. יתירות במנגנוני הגנה דורשת מהתוקף להערים על כל אחד מהם כדי להשיג גישה למשאב המותקף. כדי להשיג הגנה טובה בשכבות, כל מנגנון הגנה צריך להיכתב מתוך הנחה שמנגנוני ההגנה האחרים כבר נפרצו.
3. **היכשל באופן בטוח:** להימנע מבעיות אבטחה הנוצרות בזמן כישלון. תוקף עשוי לייצר מצב כזה במכוון כדי להחליש את המערכת. צריך לדאוג לנקות, ולצאת מהתוכנית בצורה "יפה" ו"בטוחה".
4. **עיקרון מזעור הזכויות:** יש לספק את הגישה המינימאלית הדרושה לביצוע הפעולה, ויש לתת אותה לזמן המינימאלי הדרוש.
setuid: הרשאה מיוחדת ב-unix המאפשרת להכריז כי כל מי שמריץ את הקובץ נהנה מהזכויות של בעליו בעת הרצתו. כל משתמש שמריץ את התוכנית לשינוי סיסמא, מקבל הרשאות root ויכול לשנות את קובץ הסיסמאות.
5. **מידור:** בידוד קוד בעל הרשאות גבוהות יותר. חלוקה לתתי-משימות בלתי תלויות. במערכת הפעלה יש חלוקה ל-User mode ו-Kernel mode.
6. **Keep It Simple (KISS):** מורכבות מעלה את הסיכוי לבעיות. קוד מורכב נוטה להכיל יותר באגים.
7. **שמרו על פרטיות:** הימנעות מפעולות שיגרמו לפגיעה בפרטיות המשתמש.
8. **זכרו שקשה לשמור סודות:** שמירה סודות אינה פשוטה- קוד בינארי הוא לא סודי.
קוד סגור: הגנה ע"י הסתרה. קוד המהודר לצורה בינארית אינו סודי- Reverse engineering.
ערבול קוד: טכניקות שמקשות על התוקף- הוספת קוד שלא רץ, הזזת קוד, קידוד מידע באופן שונה, הצפנת חלקי תוכנית
קוד פתוח לעומת קוד סגור:
   - קוד פתוח בטוח יותר: ככל שיותר אנשים יראו את הקוד, כך רבים הסיכויים שחורי האבטחה יימצאו ויתוקנו.
   - קוד פתוח בטוח פחות: כאשר הקוד ניתן לשינוי, תוקפים יכולים לנצל זאת לרעה.
9. **אל תבטחו בקלות:** תמיד לבדוק שהפרמטרים נכונים, לא לסמוך על הצד השני.
10. **נצלו את משאבי הקהילה:** מומלץ להסתמך על אלגורתמים קריפטוגרפיים מקובלים שנשחנו היטב ע"י הקהילה. עדיף להשתמש בספריות עם קוד אבטחה שנעשה בהן שימוש רב.

©ענת עציון

מושגי בסיס בקריפטוגרפיה

עיקרון קירכהופס: הצופן ידוע ופיסת המידע היחידה שחסרה למתקיף היא המפתח שהשתמשו בו להצפנה.
הערה: מערכת שבטוחה גם כאשר השיטה ידועה, בטוחה גם כאשר השיטה איננה ידועה.

צופן מושלם:

- בצופן מושלם אורך המפתח חייב להיות גדול או שווה לאורך ההודעה.
- כל מפתח יכול לשמש להצפנה של לכל היותר הודעה אחת.
- לצפנים מושלמים בטיחות מושלמת אך השימוש בהם קשה מאוד מבחינה מעשית. אורך המפתחות ותדירות החלפת המפתחות מהווים מכשול.

מפתח חד-פעמי:

- אסור להשתמש באותו מפתח יותר מפעם אחת, גם לא כדי להצפין מפתחות אחרים.

בטוח באופן מושלם: צופן שבהינתן אינסוף משאבים, ומספר אינסופי של זוגות של הודעות (P,C), לא ניתן למצוא גלוי בהינתן הסתר הבא. הוא בפרט צריך להיות צופן מושלם.
הערה: מכיוון שצפנים כאלה אינם מעשיים נסתפק ב**בטיחות חישובית**.
חיפוש ממצה: חיפוש על מאגר המפתחות. ניתן להתגונן ע"י מאגר מפתחות אפשריים גדול מאוד שאף כוח חישובי לא יכול לעבור עליו, למשל: מאגר מפתחות אפשריים של 2^{128}.
בטוח באופן חישובי: אלגוריתם הצפנה שקשה מאוד לשחזר את הטקסט המקורי בהינתן הטקסט המוצפן (למשל, הזמן הדרוש לכך הוא ארוך מידי). בהיעדר הוכחה לבטיחות, צופן חשב חזק אם הרבה זמן לא הצליחו לשבור אותו.

התקפות על צפנים:

התקפת Ciphertext only: לתוקף קבוצת כתבי סתר {C} והוא ינסה להשיג את הכתבים הגלויים המתאימים.
התקפת Known plaintext: לתוקף זוגות של כתבי סתר והכתבים הגלויים שלהם {(P,C)}. לרוב התוקף ינסה להשיג את המפתח, או להיות מסוגל להצפין ולפענח הודעות שאינן בקבוצת ההודעות שהוא מכיר.
התקפת Chosen Plaintext: התוקף בוחר קבוצת כתבים גלויים {P} ומבקש מהמותקף להצפין לו אותם תחת המפתח הלא ידוע. בהינתן כתבי הסתר, לרוב התוקף ינסה להשיג את המפתח.
התקפת Adaptive Chosen Plaintext: התוקף מבקש הצפנה של קבוצת הודעות, מקבל את כתבי הסתר, מבקש הצפנה של הודעות נוספות (על סמך מה שקיבל) וכן הלאה.

צפני בלוקים וצפני שטף:

DES: גודל בלוק: 64 ביט. אורך מפתח: 56 ביט. Triple DES: 3 שכבות הצפנה.
AES (Rijndael): גודל בלוק: 128 ביט. אורך מפתח: 128, 192 או 256 ביט. מספר שלבים משתנה.
צפני שטף: כאשר המידע מגיע בשטף (לדוגמא: שיחת טלפון) אז מוצפנים "בלוקים" קטנים- 1,2,4 או 8 ביט. ההצפנה תלוייה גם במספר הבלוק ובבלוקים קודמים (צופן בעל "זיכרון", לעומת צופן בלוקים שהוא חסר זיכרון). צפנים אלה מהירים יותר אך פחות בטוחים.
דוגמא- RC4: בהינתן מפתח K באורך עד 256 בתים, הצופן עובר אתחול ופולט מפתח מורחב, שהוא רצף ביטים פסאודו-אקראי (לא ליניארי). ואז נעשה xor עם המפתח. שימוש ב-xor הוא מאוד מהיר אבל לא מספיק חזק.

©ענת עציון

RSA:

מפתח פומבי: (n,e). הצפנה: $c = m^e \pmod n$. מפתח פרטי: d. פענוח: $m = c^d \pmod n$.

- בהינתן n,e קשה למצוא את d.
- אם n=pq שהם זוג ראשוניים גדולים, אז אם יתקיים: $ed = 1 \pmod{(p-1)(q-1)}$ אז מובטח שלכל הודעה יתקיים: $(m^e)^d \pmod n = m$.
- בהינתן p,q,e, מציאת d שיקיים את הדרישות היא בעיה פשוטה (ע"י אלגוריתם אוקלידס).

יצירת מפתחות: בוחרים זוג ראשוניים גדולים (512 ביט או יותר) p,q. מוצאים זוג מספרים e,d המקיים את הקשר. מפרסמים את (n,e) כמפתח פומבי ושומרים את d כמפתח פרטי.
הקושי של RSA: קיים קושי למצוא את d ללא ידיעת הפירוק לגורמים של n. בעית הפירוק נחקרת כבר שנים והיא נחשבת בעיה קשה.

צפנים סימטריים לעומת אסימטריים:

- צפנים סימטריים מהירים יותר מצפנים אסימטריים (בסדרי גודל).
- צפנים סימטריים מבוססים על פעולות שנחשבות בטוחות, ולרוב אינם תלויים בהנחות סיבוכיות. צפנים אסימטריים מבוססים לרוב על בעיות מתמטיות הנחשבות קשות.
- במערכות גדולות, ניהול המפתחות של צפנים סימטריים הינו מסובך- כל זוג צריך להחליט על מפתח. ניהול המפתחות של צפנים אסימטריים הוא פשוט יותר.

Hybrid: בהתחלה שני הצדדים מסכימים על מפתח סימטרי תוך שימוש בצופן מפתח פומבי. לאחר-מכן הצדדים משתמשים במפתח זה בצפנים סימטרייסץ בצורה זו מרוויחים את נוחות ניהול המפתחות של צפני מפתח פומבי ואת מהירות החישוב של צפני מפתח סימטרי.

חתימה דיגטלית:

אלגוריתם חתימה: $sig = S(M, \Pr_U)$, אלגוריתם בדיקה: $V(M, sig, Pub_U) = true / false$.

- בהינתן M ו-$\Pr_U$ קל לחשב את $S(M, \Pr_U)$.
- בהניתן Pub_U קשה לחשב מתוכו את $\Pr_U$.
- ללא ידיעת $\Pr_U$ קשה למצוא זוגות (sig,M) כך שהאימות שלהם מצליח.
- בהינתן סט גדול של זוגות {(M,sig)} ו-Pub_U קשה לייצר זוג חוקי שאינו בסט.

חתימה דיגיטלית מספקת: **אימות, אי הכחשה ושלמות המידע.**
הערה: לאף שיטת חתימה אין הוכחה מלאה כי היא עומדת בכל הדרישות.
זמן חישוב: זמן החתימה ארוך מאוד ותלוי באורך ההודעה. גם זמן הבדיקה ארוך ותלוי באורך ההודעה. פיתרון: במקום לחתום על כל ההודעה, חותמים על תמצית שלה.
פונקציה חד-כיוונית: קשה למצוא מקור x' כלשהו עבור ערך y נתון.
פונקציית תמצות קריפטוגרפית: מבצעת כיווץ- אורך הפלט קבוע, חד-כיווניות, חסרת התנגשויות- קשה למצוא שתי הודעות בעלות אותה תמצית.
הערה: הדרישה היא שקשה למצוא התנגשויות, אך הן קיימות מעיקרון שובך היונים.
בעיה בחתימות על תמצות: פונקצית תמצות עלולה להכניס התנגשויות לחתימה.
פרדוקס יום ההולדת: בהינתן הודעה m קשה למצוא m' כך ש- h(m)=h(m'). אבל הרבה יותר קל למצוא מראש שתי הודעות m_1 ו-m_2 כך ש-$h(m_1)=h(m_2)$, ואם למתקיף יש יכולת לבחירת ההודעה שתחתם, הוא יכול להחליף את ההודעה מבלי שהתמצות ישתנה.
כל פונקצית תמצות חשופה להתקפת יום הולדת
מסקנה: כדי שפרדוקס יום ההולדת לא יפגע בבטיחות החתימה, גודל התמצית צריך להיות לפחות 128 ביט. התוקף יצטרך להשקיע 2^{64} הפעלות של פונקציית התמצות כדי למצוא התנגשות ($2^{n/2}$).

©ענת עציון

Message Authentication Code (MAC):

הרעיון: לשני הצדדים יש מפתח סודי K משותף. כדי לאמת שהודעה M הגיעה מאחד מהם, הם יחשבו פונקציה מסויימת שתלוייה ב-K. $MAC = f_K(M)$.

- ללא ידיעת K לא ניתן לייצר MAC חוקי על הודעה M.
- גם אם לתוקף יש הרבה הודעות ו-MAC-ים תואמים הוא איננו מסוגל לייצר זוג חדש של הודעות ו-MAC שאינו בסט.
- לא ניתן לשחזר את המפתח K מתוך הודעה M וה-MAC שלה (וגם לא מסט גדול של הודעות ו-MAC).

MAC מספק: **אימות ושלמות מידע**. לא מספק אי-הכחשה, כי כל אחד משני הצדדים יכל לייצר את ההודעה החתומה.

MAC מבוסס פונקציות תמצות: מתמצתים את ההודעה M והמפתח הסודי ביחד, למשל: $Keyed\ MAC: h(K, M, K)$, $HMAC - h: h(K \oplus opad, h(K \oplus ipad, M))$.

MAC מבוסס צפני בלוקים: לדוגמא- CBC-MAC שם הבלוק האחרון של ההצפנה במוד CBC משמש כ-MAC ונשלח ביחד עם ההודעה לצורך אימות.

אין להשתמש באותו מפתח להצפנה ב-CBC ולאימות ב-CBC-MAC! האימות מוסיף יתירות למידע המועבר העשוייה לעזור לתוקף.

השימוש ב-MAC מהיר יותר משימוש בחתימה דיגטלית, אך מצריך מפתח סודי משותף.

©ענת עציון

ניהול מפתחות

תכונות של מפתחות קריפטוגרפיים: אורך המפתח ואורך החיים.
עדכון מפתחות: תוקפם של מפתחות יכול לפוג, ואז יש צורך לעדכנם. המפתח הציבורי צריך להיות כזה שניתן לעשות לו Revocation- רישום כמפתח שאינו חוקי יותר אף-על-פי שהוא עדיין בתקופת הקיום החוקית שלו (לדוגמא: אם המפתח הפרטי נחשף בטעות).

אלגוריתם Diffie-Hellman להסכמה על מפתח סודי משותף:

לכל אחד מהצדדים מפתח פומבי ומפתח פרטי מתאים. כל אחד מהצדדים מחשב את המפתח הסודי המשותף תוך שימוש במפתח הפרטי שלו, ובמפתח הפומבי של הצד השני.
הפרמטרים: מספר ראשוני-p והיוצר של $g\text{-}Z^*_p$, ידועים לכל משתמשי המערכת.
מפתחות פרטיים: x, y **מפתחות פומביים:** $g^x \pmod p$, $g^y \pmod p$. **המפתח המשותף:** $g^{xy} \pmod p$.
בעיילת הלוג הדיסקרטי: בהינתן g^x, p ו-g קשה לחשב את x.
אזהרות: אלגוריתם DH משמש אך ורק להסכמה על מפתח סימטרי משותף, **אינו** יכול לשמש עבור:

1. הצפנת מפתח פומבי.
2. חתימות דיגטליות.
3. כל דבר שאיננו הסכמה על מפתחות.

סרטיפקטים:

Certification Authority (CA): גורם מאשר שכולם סומכים עליו שתפקידו להנפיק סרטיפיקטים. כל המשתמשים במערכת מכירים את המפתח הפומבי של ה-CA.
סרטיפיקט: מחרוזת שמכילה: שם משתמש, המפתח הפומבי שלו, תאריך התפוגה של הסרטיפיקט, שם ה-CA המנפיק. כל השדות הללו חתומים ע"י ה-CA שהנפיק את הסרטיפיקט.
הערה: הגורם המאשר **אינו יודע** לפענח או לחתום תחת המפתחות אותם הוא מאשר!
Public Key Infrastructure (PKI): תשתית להעברה פרסום ואימות של מפתחות ציבוריים. המפתחות הפומביים בדרך-כלל מפורסמים בעזרת סרטיפיקטים.
הגדרות של PKI:

- מתן סרטיפיקט- מי מנפיק סרטיפיקט למי, תהליך הוידוא, תהליך ההנפקה, ואיפה הסרטיפיקט שמור.
- אימות מפתח פומבי- כיצד מוצאים סרטיפיקטים של ישויות במערכת וכיצד מאמתים סרטיפיקט (האם הוא תקף).
- ביטול הסרטיפיקט- כשקיים סיכון שמפתח פרטי מסויים נחשף.

מסלולי אמון: רוב ה-PKI-ים מכילים מספר גורמים מאשרים. משתמשים יכולים לשמש כגורמים מאשרים. משתמשים שאינם גורמים מאשרים נקראים: "ישויות קצה". גורם מאשר יכול להנפיק סרטיפיקט לגורם מאשר אחר או לישות קצה. ואז ניתן לבנות מסלול אמון בין שני משתמשים.
ארגון הגורמים המאשרים: מי חותם למי, איך נראה מסלול אמון וכיצד מוצאים אותו.

- מבנה היררכי: הגורמים המאשרים מאורגנים בעץ. כל צומת פנימי בעץ חותם לילדיו ולאביו. העלים- ישויות קצה.
- מבנה קליק: כל גורם מאשר חותם לכל אחד מהגורמים המאשרים האחרים (ובכל מבטיח שאורך מסלול אימות יכיל תמיד שני גורמים מאשרים).
- קיצורי דרך: עץ שיש בו קיצורים (מעגלים).
- Web of trust: אין מבנה מסודר של גורמים מאשרים (לדוגמא- PGP), על כל משתמש לאמץ מדיניות לפיה יקבל או ידחה סרטיפיקטים.

Scalability: PKI שיישאר מעשי גם עבור מספר רב של משתמשים. בפרט מסלולי האמון יהיו קצרים יחסית וקלים למציאה ובדיקה.

©ענת עציון

דוגמא- מבנה היררכי: אין בעיית של scalability, אבל המערכת יותר מידי תלוייה בגורמים המאשרים שנמצאים ברמות העליונות של העץ. חשיפה של מפתח פרטי של אחד מהם תגרום לנזק רב. לכן גורמים מאשרים אלה מהווים יעד מפתה להתקפה.

מודל של אמון- דוגמאות:

- לכל משתמש קיים לפחות גורם מאשר אחד שעליו המשתמש סומך לגמרי. אם גורם מאשר זה סומך על CA' אז המשתמש סומך גם על CA' באופן אוטומטי.
- סומכים על גורמים מאשרים מסויימים רק למטרות מסויימות, למשל: מאמינים רק לסרטיפיקטים שמונפקים ע"י ה-CA, בהם זהות המשתמש היא כתובת דוא"ל, סומכים על סרטיפיקטים על מפתחות חתימה אך לא על מפתחות הצפנה.

ביטול סרטיפיקט (Certificate revocation): דרוש במספר מקרים:

- המפתח הפרטי (המתאים למפתח הפומבי שבסרטיפיקט) התגלה.
- המפתח הפרטי של ה-CA התגלה.
- ה-CA כבר לא מחזיק סרטיפיקטים עבור משתמש זה.

Certificate Revocation List (CRL): כל גורם מאשר מפרסם רשימה של סרטיפיקטים שהונפקו על ידו ובוטלו. הרשימה חתומה ע"י הגורם המאשר ומכילה: מספר סידורי, תאריך פרסום הרשימה והתאריך בו תפורסם הרשימה הבאה.

- לפעמים יש צורך לפרסם CRL לפני התאריך המיועד של ה-CRL הבא.
- על המשתמש להחליט לבד כל כמה זמן הוא בודק את ה-CRL.

קבלת סרטיפיקט: משתמש צריך להיפגש פיזית עם הגורם המאשר שיקבל את פרטי הסרטיפיקט (שם מפתח פומבי וכו') ויבדוק את זהותו, ואם הוא יודע את המפתח הפרטי (ולא ע"י שיגלה מהו).

שירותי מדריך:

מדריך: שרת או קבוצה מבוזרת של שרתים ששומרים מבנה נתונים עם מידע על המשתמשים.

דוגמא למדריך – X.500:

- בכל כניסה במדריך שמורות תכונות שונות, אחת מהן היא ה-public key של הישות המתאימה.
- המדריך מתוחזק ומופעל ע"י היראכיה (עץ) של תתי-מדריכים.
- לכל ישות בעץ יש שם ייחודי שונה משל אחיו.
- השם הייחודי Distinguished Name (DN) מתקבל ע"י שרשור השמות במסלול מן השורש אל הצומת המתאים. זוהי קבוצה סדורה של זוגות: מילת מפתח שמורה ומחרוזת שמכילה ערך.

דוגמא למדריך – X.509:

- לא מגדיר כיצד ה-CA-ים מאורגנים אבל ממליץ לשמור על המבנה ההיררכי של X.500.
- דרכים לקבלת סרטיפיקט:
  - Central: הגורם המאשר מייצר את המפתח הפרטי והציבורי עבור הלקוח (subject), ושולח אותם (בצורה מוגנת) אל הלקוח.
  - Distributed: הלקוח מייצר את המפתח הפרטי והציבורי ושולח בקשה אל הגורם המאשר שמחזיר לו סרטיפיקט עבור המפתח הציבורי. בשיטה זו על הגורם המאשר לוודא כי הלקוח יודע את המפתח הפרטי המתאים.

EV Certificates: סרטיפיקט X.509 עם שדות נוספים ייחודיים. גורמים מאשרים המנפיקים אותם צריכים לעמוד בדרישות מחמירות יותר מגורמים מאשרים אחרים.

דפדפנים וסרטיפיקטים: כל הדפדפנים מכילים מאגר סרטיפיקטים של CAs שאמינים על משתמש הדפדפן, לצורך הקמת קשר SSL. ניתן להוריד ולהוסיף לרשימה סרטיפיקטים של שרתים או CAs באופן ידני.

©ענת עציון

PGP:

- אינו מניח קיום של רשות שכל המשתמשים סומכים עליה (CA).
- כל משתמש משמש כגורם מאשר ויכול לחתום על סרטיפיקטים של משתמשים אחרים כרצונו.
- כל משתמש מחליט לבד בצורה מפורשת על מי הוא סומך ועל מי לא.

מחזיק המפתחות הפרטיים: כל משתמש ב-PGP מחזיק מבנה נתונים שנקרא: "מחזיק המפתחות הפרטיים" (Private Key Ring) ובו שמורים המפתחות הפרטיים שלו. זוהי טבלה שבה כל שורה מתאימה למפתח פרטי.

הצפנת המפתח הפרטי: צופן סימטרי- המשתמש מקליד סיסמא pass, מיוצר hash על הסיסמא באורך 160 ביט. מתוכו נלקחות 128 סיביות כמפתח לצופן הסימטרי.

- המפתחות הפומביים נשלחים בין המשתמשים בעזרת סרטיפיקטים.
- סרטיפיקט מכיל כתובת דואר אלקטרוני ומפתח פומבי וחתום ע"י משתמשים שונים.

מחזיק המפתחות הפומביים: מבנה נתונים בו שמורים מפתחות פומביים של משתמשים שונים. זוהי טבלה שבה כל שורה מתאימה למפתח פומבי. קיימים שם גם השדות:

Owner Trust: שדה שנקבע ע"י המשתמש ובו הוא מציין את מידת האמון בחתימות של משתמש אחר עם המפתח הפומבי הזה.

Signatures: חתימות על הסרטיפיקט שנעשו ע"י משתמשים אחרים.

Signatures trust: לכל חתימה בשדה הקודם, נשמר ערך ה-owner trust של המפתח הקודם.

Key Legitimacy: שדה בינארי אשר קובע האם המשתמש אכן מאמין שזה באמת המפתח הפומבי של המשתמש האחר. ערך שדה זה מחושב ע"י PGP על סמך השדות האחרים.

חישוב השדה Key Legitimacy: לפי השדה signature trust, Unlimited: משקל 1, Always trust: משקל- 1/X, usually trust: משקל 1/Y, not trusted: מקבל 0. X ו-Y הם פרמטרים שנקבעים ע"י המשתמש. אם לא נאמר אחרת, נניח כי: X=1, Y=2. אם סכום כל המשקלים גדול מ-1, אז המפתח הוא לגיטימי, אחרת- לא לגיטימי.

©ענת עציון

בקרת כניסה

1. **זיהוי המשתמש- Identification:** המשתמש מזהה את עצמו למערכת ע"י הקשת שמו (User ID).
2. **אימות זהות המשתמש – Authentication:** המערכת מבצעת בדיקה, שמאמתת את זהות המשתמש (באימות הדדי גם המשתמש מבצע אימות של המערכת).
הערה: בעבר לא נדרש אימות הדדי מכיוון שהמשתמש היה מחובר פיסית למערכת או מחייג אליה- דבר שהקשה על התחברות מתקיף ביניהם.
האימות מבוסס על אחד או צירוף של: ידע של המשתמש, מכשיר בבעלות המשתמש, תכונה פיזית של המשתמש (what you know, what you have, what you are).
הערה: מעשית, אמצעי האימות הפופלארי ביותר כיום באינטרנט (וברוב רשתות המחשבים) הוא סיסמא, שניתנת לאחסון בזיכרון האנושי.

פרוטוקול בסיסי מבוסס סיסמא – TELNET Auth:

- השרת שולח למשתמש prompt
- המשתמש מגיב עם זהותו
- השרת מבקש מהמשתמש סיסמא
- המשתמש מגיב עם הסמא pass.
- השרת מאמת את הסיסמא אל מול רשומת המשתמש בקובץ הסיסמאות.

הערה: בקובץ הסיסמאות שבשרת מאוחסן h(pass, salt) יחד עם ה-salt (ערך אקראי שנוצר מקומית לכל סיסמא). השרת מחשב מחדש את h(pass, salt) ומשווה את התוצאה עם הערך ששמור בקובץ.
גם אם קובץ הסיסמאות נחשף, התוקף צריך להשקיע מאמץ רב בשחזור הסיסמא.
מוגן מפני: חשיפת קובץ הסיסמאות, התחזות ללקוח.

יצירת סיסמאות מסונכרנות:

מנגנון פשוט: לכל צד יש רשימה ארוכה של סיסמאות. בכל ניסיון כניסה למערכת, המשתמש מקיש את הסיסמא הבאה בתור.
יתרון: משתמשים ב-OTP, לכן לא ניתן לבצע התקפת שידור חוזר.
חיסרון: יש להחזיק רשימות ארוכות (קל לאבד וקשה לשמור בצורה בטוחה). תוך זמן קצר יש להחליף את הרשימות ע"י פגישה פיזית.
SecureID: יחיה פיזית שמחוללת מחרוזת פסאודו אקראית בכל דקה, ומציגה אותה על מסך זעיר.

- כאשר משתמש נדרש להכניס סיסמא, הוא מקיש את המחרוזת שמוצגת משורשרת ל-PIN.
- המערכת המאמתת מחוללת את המחרוזת תוך שימוש ביחידה זהה לחלוטין הנמצאת ברשותה ומאמתת את הערך שהכניס המשתמש.

Two factor authentication: האימות מתבצע על סמך what you have ו-what you know.
יתרון: בעית השידור החוזר נפתרה.
חסרון: לא מתבצע אימות המערכת. ניתן לבצע session hijacking.
מוגן מפני: התחזות ללקוח, שידור חוזר והתקפת מילון.
חטיפת הקשר- Session hijacking: המתקיף מחכה עד שתהליך אימות המשתמש החוקי מסתיים ולאחר מכן מנתק אותו (ע"י שליחת הודעה למשתמש שהקשר נותק) וממשיך את ה-session במקום המשתמש החוקי.
הערה: זוהי מעין התקפת MITM על פרוטוקולי אבטחת כניסה.
התגוננות: תוך כדי תהליך האימות מסכימים על מפתח סודי וחד-פעמי (session key) ולאחר-מכן כל ה-session מוגן עם המפתח הנ"ל.

©ענת עציון

פרוקוטולי Challenge-Response:

- בכל ניסיון כניסה של משתמש למערכת, המערכת שולחת לו challenge חדש (למשל מחרוזת אקראית).
- המשתמש מחזיר response (למשל חישוב פונקציה קריפטוגרפית מתאימה על ה-challenge ועל הסיסמא של המשתמש), ובכל מוכיח שהוא יודע את הסיסמא.
- גם המערכת מבצעת את אותו החישוב ומשווה עם מה שהתקבל מהמשתמש.
- הערה: הפרוטוקול הוא על בסיס OTP ולכן מוגן משידור חוזר.

Challenge Handshake Authentication Protocol (CHAP): לשני הצדדים יש סיסמא משותפת password.

- האתגר שהמערכת בוחרת הינו מחרוזת אקראית המוגרלת בכל ריצה מחדש.
- המשתמש מחשב ושולח את המענה response = h(challenge, password).
- המערכת מחשבת את מחרוזת התגובה, ובודקת אם היא זהה לערך שהחזיר המשתמש. לצורך כך שומרת המערכת בקובץ הסיסמאות את סיסמאות כל המשתמשים.

יתרון: פרוטוקול מסוג OTP, מוגן מפני שידור חוזר.

חסרון: חשיפת קובץ הסיסמאות מהווה פריצה. ניתן להתחזות לשרת (פיתרון: challenge-response דו-כיווני). ועוד שתי התקפות אפשריות: חטיפת הקשר, התקפת מילון.

התקפת מילון: מסתמכת על כך שרוב הסיסמאות שהאנשים ממציאים/מסוגלים לזכור הם מילים/צירופי מילים ומספרים- סיסמאות חלשות, ולכן אין צורך לעבור על כל הצירופים האפשריים של מחרוזת חוקית, אלא רק על צירופי הסיסמאות החלשות, מכין חיפוש ממצה על סיסמאות "שכיחות".
מניחים כי המתקיף השיג מחרוזת y כך ש-y=f(password) או y=f(password,x) ו-x ידוע לתוקף.

הערה: ב-CHAP מספיק למתקיף לראות זוג challenge-response אחד לביצוע התקפת מילון.

התקפת מילון מקוונת: לתוקף אין מחרוזת y כך ש-y-f(password,x) . בדיקת מילה בודדת מהמילון דורשת התקשרות למערכת. פחות יעילה מהתקפת off-line. מערכות בקרת גישה נדרשות לזהות ניסיונות להתקפת מילון מקוונת ולחסום את חשבון המשתמש המותקף.

מוגן מפני: התחזות ללקוח, שידור חוזר.

EKE-Encrypted Key Exchange:

המטרה: להגן מפני התקפת מילון מקוונת וכך לאפשר שימוש בסיסמאות חלשות לצורך בקרת כניסה.

הרעיון: גזירת מפתח הצפנה w מהסיסמא אשר יצפין מידע אקראי (או קרוב לאקראי).

- ביצוע אימות של שני הצדדים (למניעת התחזות לשרת).
- במהלך הפרוטוקול נגזור גם session key שבעזרתו נגן על כל ה-session בהמשך, למניעת חטיפת הקשר.

צעד ראשון: המתמש מחשב w=f(passwd) ושולח למערכת: את שמו ביחד עם מפתח DH פומבי מוצפן תחת w.

הרעיון: אפילו שהסיסמא passwd חלשה, לא ניתן לבצע חיפוש ממצה (לא מקוון) עליה מכיוון בהמפתח הפומבי הוא מחרוזת אקראית לגמרי שאינה ידועה לתוקף. ולכן בעת חיפוש ממצה אין לתוקף דרך לדעת אם עלה על הסיסמא הנכונה.

צעד שני: המערכת מחשבת את K המפתח DH הסודי המשותף ומגרילה מחרוזת אקראית $challenge_s$ ומצפינה אותה תחת K. בנוסף היא שולחת את המפתח DH הפומבי שלה מוצפן תחת w.

צעדים 3 ו-4: עד כה לא נעשה שום אימות. הצעד 3 המשתמש יישלח $E_K(challenge_s,challenge_u)$ ויוכיח בכך שהוא יודע את K ולכן לא מבצע שידור חוזר. בצעד 4 תשלח המערכת את $E_K(challenge_u)$ ותוכיח בכך שהיא יודעת את K ולכן לא מבצעת שידור חוזר.

- K משמש להגנה על ה-session שנוצר.

מוגן מפני: שידור חוזר, התקפת מילון (לא מקוונת), התחזות לשרת או ללקוח, חטיפת הקשר.

©ענת עציון

הערה: אם משתמשים בצופן שטף, או בהצפנת בלוקים במוד ECB (הצפנת כל בלוק בלתי תלוייה באחרים), כאשר גודל ה-challenge הוא כפולה של אורך הבלוקים, ניתן להתחזות עד לסיום EKE אבל אי-אפשר לנהל קשר לאחר-מכן כי K עדיין אינו ידוע.

SRP- Secure Remote Password Protocol:

- פרוקוטול בקרת גישה המבוסס על בעיית הלוג הדיסקרטי.
- עמיד בפני כל ההתקפות שתוארו.
- מוגן ע"י פטנט.

Kerberos:

זהו פרוטוקול בקרת גישה של clients ל-servers.

- ה-clients מבקשרים מה-Authentication server **כרטיסים**. שרת מאשר את כניסתו של client רק לאחר שוידא את חוקיות הכרטיס.
- נעשה שימוש בקריפטוגרפיה סימטרית- ביצועים טובים.

היישויות ב-Kerberos:

Principles: לקוחות ושרתים.

לקוחות: מעוניינים לקבל שירות משרתי אפליקציה, מסומנים ב-c.

שרתים: נותנים שירות ללקוחות. מסומנים ב-s.

Key Distribution Center (KDC): מבצע אימות משתמש ונותן לו כרטיס "אוניברסאלי" – כרטיס TGT (Ticket Granting Ticket). הוא שומר את כל המפתחות הסודיים של כל הלקוחות והשרתים ברשת. מייצר session keys לשימוש הלקוחות והשרתים ומפיץ אותם (ומכאן שמו).

Ticket Granting Server (TGS): מייצר כרטיסים (Tickets) לשרתים, על סמך הכרטיס האוניברסאלי של המשתמש.

הפרוטוקול הבסיסי:

- הלקוח c מבקש מה-KDC כרטיס אל השרת s.
- ה-KDC שולח כתגובה מפתח סודי session key- $K_{c,s}$ (בין ה-client ל-s), מוצפן תחת K_c, שנגזר מהסיסמא של הלקוח וכרטיס $Ticket_{c,s}$.
- ה-nonce מספק ללקוח הוכחה שה-KDC יודע את K_c.

הערה: בסיום התהליך ה-KDC לא יודע האם הלקוח מתחזה או לא. אך על-מנת שהלקוח יוכל לגשת לשרת s עליו לפענח את ה-session key שמוצפן עם המפתח הסודי של המשתמש החוקי. השרת s יקבל את ה-session key בתוך הכרטיס כשהמשתמש יפנה אליו.

- הכרטיס מוצפן תחת המפתח הסודי של השרת, K_s ומכיל בין היתר את ה- session key, $K_{c,s}$.
- הלקוח שולח לשרת את הכרטיס $Ticket_{c,s}$ ואת ה-$Auth_{c,s}$.
- השרת מוודא את ה-Authenticator בעזרת ה-session key (בינו לבין ה-client) שבכרטיס ומאפשר (או לא מאפשר) כניסה.

תכולת הכרטיס:

- Validity period: התקופה שבה הכרטיס חוקי (שרשור של זמן התחלה ומשך התוקף).
- $K_{c,s}$: מפתח משותף לשרת וללקוח (session key).
- K_s: מפתח סודי של השרת.

השימוש בכרטיס הוא רב-פעמי.

הפרוטוקול המלא:

- קיים TGS והשרתים מתחלקים בין ה-TGS-ים השונים.
- ה-KDC נותן ללקוח כרטיס ל-TGS המתאים. TGS נותן ללקוח כרטיס לשרת s. ל-KDC צריך לפנות רק לצורך חידוש ה-TGT.

מוגן מפני: שידור חוזר, התחזות , חטיפת הקשר. **לא מוגן מ:** חשיפת קובץ סיסמאות ומילון.

©ענת עציון

בקרת גישה

בקרת גישה: מניעת שימוש לא מורשה במשאב, כולל מניעת שימוש במשאב באופן לא מורשה.

מדיניות בקרת גישה:

מכתיבה איזה סוגי גישה מותרים, באיזה נסיבות וע"י מי. בד"כ מסווגים סוגי מדיניות באופן הבא:

בקרת גישה מבוססת שיקול דעת (Discretionary Access Control-DAC): בקרת גישה מבוססת על זהות המבקש ועל כללי גישה המציינים מה המבקש רשאי (או לא) לעשות. המדיניות היא מבוססת שיקול דעת מכיוון שלכל ישות יש הרשאות המרשות לה לאפשר ע"פ רצונה גישה של גורם אחר למשאב השייך לה.

מימוש: מטריצת גישה (בד"כ מטריצה דלילה). מימוש ע"י חלוקת המטריצה לעמודות נותן: ACL- Access Control List. מימוש ע"י חלוקת המטריצה לשורות נותן: capability tickets.

בקרת גישה כפויה (Mandatory Access Control-MAC): בקרת הגישה מבוססת על השוואת תוויות אבטחה (המציינות את רגישות המשאב) עם היתר אבטחה. בקרת הגישה היא כפויה, כיוון שישות בעלת היתר לגשת למשאב לא יכולה לתת אפשרות כזו לישות אחרת.

בקרת גישה מבוססת תפקידים (Role-Based Access Control – RBAC): בקרת הגישה מבוססת על תפקידים שיש למשתמשים במערכת, ועל הכללים המציינים את הגישות המותרות למשתמשים בתפקידים נתונים.

מימוש: במטריצות גישות שבה מקצים לכל משתמש את התפקידים שהוא מקבל. ובמטריצה אחרת מקצים לכל תפקיד את הגישות שהוא מקבל

הערה: מנגנון בקרת גישה יכול לשלב בין שתיים או שלוש מהגישות עבור סוגים שונים של משאבי המערכת.

עצם (object): משאב במערכת שהגישה אליו מבוקרת.

נתין (subject): ישות היכולה לגשת לעצמים. בדרך-כלל תהליך (הפועל בשם המשתמש). מחולקים ל-3 קבוצות: בעלים (owner), קבוצה (group) ועולם (world).

הרשאת גישה (access right): מתארת את הדרך בה נתין יכול לגשת לעצם. לדוגמא: Read, Write, Execute, Create, Search.

אבטחה עם מספר רמות:

מקרה פרטי של MAC בו תוויות האבטחה מאורגנות בהיראכיה.

דוגמא: רמת סודיות הנתונים- Top Secret > Secret > Confidential > Unclassified.

פעולה של נתין על עצם מותרת רק אם מתקיים יחס מסויים בין רמות האבטחה המיוחסות לשניהם.

שתי גישות מרכזיות להגדרת היחסים הנדרשים:

דגש על סודיות:

- Read down: לנתין מותר לקרוא רק עצמים ברמת ההרשאה שלו ומטה.
- Write up: לנתין מותר לכתוב רק עצמים ברמת ההרשאה שלו ומעלה.

גישה זו מבטיחה סודיות כיוון שהמידע יכול לזרום רק מרמות הרשאה גמוכות לרמות הרשאה גבוהות.

דגש על שלמות:

- Read up: לנתין מותר לקרוא רק כצמים ברמת ההרשאה שלו ומעלה.
- Write down: לנתין מותר לכתוב רק עצמים ברמת ההרשאה שלו ומטה.

גישה זו מבטיחה שלמות, כיוון שמשתמשים לא אמינים (הרשאה נמוכה) אינם יכולים לכתוב מידע לעצמים אמינים (הרשאה גבוהה).

©ענת עציון

בקרת הגישה ב-Windows:

בקרת הגישה נעשית באמצעות מנגנון DAC על בסיס ACL-ים.
הרחבה של המנגנון הבסיסי עם תוויות- Integrity Level (IL). משתמשים ב-4 רמות:
System Integrity: מוקצית לתהליכי מערכת.
High Integrity: מוקצית ל-administrators.
Medium Integrity: מוקצית למשתמשים מאומתים.
Low Integrity: מוקצית ל-everyone.

- גם לכל עצם מוגדר IL (כאשר לא מוגדר IL ברירת המחדל היא Medium).
- תהליך יכול לבצע פעולת כתיבה ומחיקה רק לעצמים בעלי IL זהה לשלו או נמוך יותר (Write down). דוגמא: התהליך iexplorer.exe רץ ב-Low Integrity. באופן, לדפדפן יש גישה מוגבלת לקבצי המשתמשים במערכת (שלרוב יוגדרו כ-Medium Integrity).
- התהליכים IEUser.exe ו-IEInstal.exe רצים ברמות הרשאה גבוהות יותר ומספקים ממשק לצורך פעולות נקודתיות הדורשות הרשאות גבוהות יותר (לדוגמא: שמירת קובץ למדריך המשתמש).
- כאשר נדרשות פעולות בעלות הרשאות גבוהות שאינן נתמכות ע"י הממשק של IEUser.exe ו-IEInstal.exe, ניתן לייצר תהליך נפרד ולבקש את אישור המשתמש להעלאת ההרשאה.

סוגי בקרת גישה מבוססת תפקידים:

$RBAC_0$-המודל הבסיסי: מציג יחסים בין משתמשים, תפקידים והרשאות.
$RBAC_1$- היראכיות תפקידים: הרחבה המאפשרת להגדיר היראכיות בין תפקידים. מאפשר ירושה של הרשאות.
$RBAC_2$- הגדרת אילוצים: ניתן להגדיר אילוצים מהסוגים הבאים:
מניעה הדדית: בכל זמן נתון, ניתן להקצות רק את אחד מהתפקידים למשתמש (עיקרון הפרדת משימות). דוגמא: משתמש לא יכול להיות גם בתפקיד סטודנט וגם בתפקדי מרצה.
Cardinality: יש מגבלה על מספר המשתמשים שיכולים להחזיק בתפקיד מסויים. לדוגמא: רק משתמש אחד יכול להיות בתפקיד מתרגל ראשי.
דרישות קדם: ניתן להקצות תפקיד למשתמש רק אם הוקצו לו לפני כן תפקידי קדם דרושים. דוגמא: משתמש יכול להיות בתפקיד מרצה רק אם הוא בתפקיד סגל קורס.
$RBAC_3$- מודל משולב: שילוב אילוצים והיראכיית תפקידים.

©ענת עציון

התקפות ואיומים ברשתות מחשבים

מבנה החבילה:

MAC header: משתנה לפי המדיה הפיסיקאלית. מכיל את כתובת ה-MAC, את זהות הפרוטוקול הבא (IP) ואינפורמציה נוספת.

IP header: מכיל בין היתר: כתובות IP של השולח והמקבל, TTL, הפרוטוקול הבא (TCP/UDP או אחר), וגרסת פרוטוקול IP.

UDP/TCP header: מכיל: מספרי הפורטים של השולח והמקבל, ואינפורמצית בקרת שגיאות. TCP מכיל בנוסף חיוויים (ack) ומספרים סידוריים.

דוגמא לרשת מקומית- Ethernet:

רשתות מקומיות נפוצות ביותר. בפרוטוקל משלוח חבילה הבסיסי ביותר של הרשת, החבילה משודרת בתפוצה כוללת (broadcast), כאשר כל כרטיס רשת בודק האם החבילה מיועדת אליו, על סמך כתובת MAC בת 48 סיביות. אם החבילה מיועדת אל המחשב הנוכחי- החבילה מועברת לשכבת ה-IP, אחרת היא נזרקת.

ברשתות Ethernet רגילות קל מאוד להאזין לתעבורה- ניתן לבקש מכרטיס הרשת להעביר את כל החבילות לשכבת ה-IP. בצורה זו עובדים מרבית ה-sniffers אשר מאזינים לכל התעבורה שעוברת.

בגרסאות "חדשות" יותר של Ethernet הדורשות חומרה מתאימה, הרשת היא switched, ואז ההודעות עוברות דרך מרכזייה (switch) אל היעד בלבד.

Internet Protocol (IP):

- הניתוב בין הרשתות מבוסס על טבלאות ניתוב הבנות תוך שימוש בפרוטוקולי ניתוב שונים.
- חבילות שונות עשויות להישלח במסלולים שונים, ולכן סדר הגעת חבילות יכול להיות שונה מסדר שליחתן.
- התרגום מכתובות לוגיות (t2.technion.ac.il) לכתובות IP נעשה באמצעות מערכת שרתים בשם: **Domain Name Service (DNS)**.

שכבת התובלה (Transport layer):

- כתובת באמת התובלה היא **מספר הפרוטוקול (Protocol Number)**.
- שכבת התובלה מספקת שרותים לכל האפליקציות במכונה.
- מחלקת את התעבורה (multiplexing) בין כל האפליקציות על-פי מספרי פורטים.

UDP (User Datagram Protocol): מספק שירות connectionless ו-unreliable. לא דואג לסדר ואיננו מבחין בחבילות חסרות. מותיר את האחריות לאפליקציה. מתחזק מינימום משתני state. מתאים להשתמש בו כאשר לא נורא אם נפספס אחת מהחבילות.

TCP (Transport Control Protocol): מספק שירות שהוא reliable ו-connection-oriented. מתחזק session- מסדר את החבילות המתקבלות לפי הסדר ודואג לשידור מחדש של חבילות שהלכו לאיבוד. מספק לאפליקציה stream של בתים מסודרים. לשם כך מתחזק ה-TCP חוצצים ומונים שמוקצים עם תחילת ה-session (די הרבה משאבים פר session).

TCP-Three way handshake: לחיצת יד משולשת. לקוח: SYN, שרת: SYN, ACK, לקוח: ACK.

©ענת עציון

הערה: TCP גם תומך בלחיצת יד מרובעת שבה השרת מגיב על ה-SYN בשתי חבילות נפרדות, הראשונה- SYN והשנייה-ACK.

שכבת האפליקציה (Application layer):

- התקשורת ברמת האפליקציה מתבצעת בעזרת פורטים.
- בכל מחשב קיימות שתי סדרות של פורטים: UDP ו-TCP.
- ברמת האפליקציה מתקיים session בין שני שותפים והפורטים שלהם מייצגים את זהות התהליך (או האפליקציה) וזוהות הקשר המסויים בכל מחשב.
- בתוך כל חבילה כתובים פורט המקור ופורט היעד, והם מהווים חלק מכתובת המקור והיעד של החבילות ברמת האפליקציה.

איומים על רשתות תקשורת:

ניתן לתקוף כל אחת מהשכבות במודל. לכל שכבה שתוקפים יש משמעות שונה, דרישות שונות וכלים שונים. פעמים רבות ניתן לתקוף את החיבורים בין השכבות.

IP Spoofing: שינוי כתובת ה-IP של ה-host לצורך התחזות. מתקיף יכול לשכנע שהחבילות שהוא שולח נשלחות ממחשב אחר, אבל חבילות התשובה לא יגיעו אל המתחזה, אלא אל המחשב האחר.
הערה: במקרה של UDP לא תהיינה בהכרח חבילות תשובה.

- **הגנה- cookies:** עם תחילת ה-session נשלח ליוזם הקשר cookie שהיא מחרוזת בלתי ניתנת לחיזוי מראש. לאחר מכן נדרוש לקבל אותה בחזרה מיוזם הקשר. אם הוא מחזיר אותה, הרי שהיא בהכרח הגיעה ליוזם הקשר, כלומר: היוזם אכן נמצא ב-IP שהוא טוען ששייך לו.
- **הגנה-שימוש ב-TCP:** לחיצת יד משולשת, בסיומה ידוע כי ה-IP של הצד השני אכן מאויש ביוזם הקשר.

IP Spoofing דו-כיווני: מאפשר למחשב מתחזה לשלוח חבילות מכתובת IP שאיננה שלו ואף לקבל עליה תשובה. הוא יותר קשה מ-IP Spoofing רגיל כי הוא מצריך שינויים ברשת (בנתבים), ולא רק במחשב המתחזה- התוקף צריך לדעת איך ההודעה מנותבת ברשת. במהלך הקורס התקפה זו לא תחשב לסבירה- נניח שהיא קשה ולא אפשרית.

DNS Poisoning: מכיוון שהקישור בין שם domain וכתובת ה-IP שלו דורשת גישה לשרתי ה-DNS ומכיוון שפרוטוקול ה-DNS איננו בטוח, ניתן לתקוף את קו התפר הזה. התוקף במקרה זה שולח חבילה שמתחזה לתשובה משרת ה-DNS. יש לפרוטוקל ה-DNS מעין מנגנון הגנה מינימאלי נגד זה אבל הוא לא מספק.

התקפות על TCP:

בתחילת session מערכת ההפעלה מקצה משאבים (כולל הקצאות זיכרון). המשאבים הללו משוחררים בזמן סגירת ה-session. בפתיחת session קיים timeout של כשתי דקות לקבלת המענה: אם אין מענה ה-session מבוטל והמשאבים משוחררים. פתיחה של TCP sessions רבים מידי יכולה להפיל את המחשב.

- **Syn attack:** המתקיף שולח כמות גדולה מאוד של חבילות ראשונות ב-TCP session. למערכת ההפעלה תורים מיוחדים (וקטנים) לשמירת הנתונים של החיבורים באמצע הלחיצת יד. שליחת כמות גדולה של חבילות, גורמת לניצול כל המקומות בתורים הללו והמערכת לא יכולה לקבל חיבורים נוספים.

הגנה אפשרית: הגבלת מספר ה-sessions המורשים עבור כל כתובת IP. לא מגן מפני התקפה זו בתוספת IP Spoofing.

הגנה אפשרית טובה יותר: שימוש ב-cookies.

©ענת עציון

- **RST attack:** המתקיף שולח כמות גדולה מאוד של חבילות TCP במטרה לאתחל מחדש חיבור TCP קיים. ל-TCP קיים מנגון לסגירת session וכן מנגנון לאתחול session. שליחת הודעת RST אמינה לאחד הצדדים תגרום לו לסגור את החיבור בצורה מסודרת. זה פוגע בחיבור TCP קיים בין שני משתמשים.
- **פרטי מימוש:** מכיוון שיש צורך לבצע injection של חבילות ל-TCP session, יש לנחש את ה-sequence number ואת הפורט אליו יש לשלוח את ההודעה. המספר הסידורי שיש לנחש צריך ב"חלון" ולא ערך אחד נתון. חבילות RST מועברות מיידית עם הגעתן, אם מספרן בחלון. בנוסף, הרבה מימושים של TCP לא באמת מגרילים מספר סידורי ראשוני אלא מתחילים מ-0.

התקפות על UDP: UDP לא שומר מצב על connection, ולכן הוא חשוף יותר להתקפות המשולבות עם IP Spoofing דו-כיווני. חלק גדול מהתקפות ה-DNS poisoning מבוססות על כך שפרוטוקול ה-DNS רץ מעל UDP, ולכן ניתן לפגוע בשרתי DNS מרוחקים.

Ping Flood: מעל שכבת ה-IP קיים פרוטוקול ה-ICMP (Internet Control Message Protocol). פרוטוקול זה עוזר לשכבת ה-IP, אך הוא נמצא מעט מעליה. כשמריצים ping לשרת מרוחק, למעשה שולחים הודעות ICMP מסוג ping לשרת, ומחכים לתגובת pong ממנו. שליחת הודעות אלה יוצרת עומס על חביור האינטרנט של המותקף. אם החיבור של התוקף מהיר ושל המותקף איטי אז זהו DoS על לקוח.

התקפות ברמת האפליקציה:

- באגים
- שימוש בפונקציות לא בטוחות (כמו gets, scanf).
- שימוש במוד debug.
- Session hijacking.
- DoS
- DDoS
- **התקפות על אפליקציות web:** חלק השרת של האפליקציות נכתב בכלים סטנדרטיים ורץ מעל שרתי web. האפליקציות אינן דורשות התקנת לקוח- הלקוח ממומש באמצעות קוד נייד, לדוגמא: Java ו-ActiveX. ואז כשקוד הלקוח יורד למחשב ניתן להכניס בו שינויים זדוניים.
- **Cookie poisioning:** http מאפשר לשרת לשמור מידע אצל הלקוח במכניזם שנקרא- cookies. העוגיות נשמרות אצל המשתמש והוא יכול לשנות את תוכן וערכן.
- **SQL injection:** במקרים רבים שאילתות ברשת מועברות למנגנון SQL שמטפל בבקשה. אם המכניזם שמפעיל את ה-SQL לא בודק שהשאילתא "תקינה" אז ניתן להעביר ל-SQL פקודות או הוראות.
- **Cross Site Scripting (XSS):** הרשת מלאה בהרבה scripts בשפות שונות. מנצלים חוסר בדיקת תקינות בצד של השרת, כדי להכניס שורות קוד לביצוע אל תוך דפי ה-html שהשרת מציג.
  הגנה: בדיקות פרמטרים.
- **Cross site request forgery (CSRF):** התוקף משתמש בנתונים שנלקחים מהמשתמש (בד"כ cookies authentication data) בכדי לבצע טרנזקציה בשרת תחת זהות משתמש מאומתת.
  הגנה: אימות דינמי (בניגוד לסטטי)- אפשרי רק ברמת האפליקציה.
  הערה: ב-XSS הדפדפן בוטח בקלט שמגיע מהשרת- ההתקפה מכוונת נגד הדפדפן במטרה להשיג מידע רגיש של המשתמש. ב-CSRF השרת בוטח בנתונים שמגיעים מהדפדפן.
  הערה: הרבה פעמים התקפות משמשות ב-XSS בכדי להשיג מידע וב-CSRF בכדי לבצע פעולות כמשתמש מאומת.
- **התקפות המבוססות על social engineering:** דואר זבל, הונאות אתרים (phishing), סוסים טרויאנים, ניצול רשתות P2P ותוכנות מסרים מיידיות ועוד...

©ענת עציון

Firewalls

Firewall: כלי למימוש מדיניות הנה, בעל יכולת (מוגבלת) להגן מפני חלק מההתקפות על הרשת.

- הוא ממוקם בנקודה בה עוברת כל התעבורה בין הרשת של הארגון לבין העולם החיצוני (choke point). הדבר מאפשר פיקוח על התעבורה וסינון שלה.
- זוהי מערכת שמפרידה בין שני אזורים ברשת וכופה מדיניות אבטחה על התקשורת ביניהם.
- המערכת יכולה להיות מורכבת ממחשב או נתב בודד, או מאוסף מחשבים, נתבים וכו'.
- Firewall שאיננו על מחשב אישי (משתמש קצה), חייב להכיל לפחות multi-homed host אחד (קופסה עם חיבור לשתי רשתות ויותר).

חשוב: יש למקם את ה-Firewall בצורה כזאת שכל התעבורה בין שני האזורים תעבור דרכו.

עקרונות תיכון Firewalls:

- זכויות מזעריות: אין ל-FW הרשאות (גישה לרשת) שאינן הכרחיות.
- Choke point: ניתוב על התעבורה דרך נתיב צר.
- Fail safe: נפילה איננה פריצה- במקרה שה-FW נופל, כל התקשורת תתנתק.
- מבנה פשוט.
- חסימת כל מידע, אלא אם הוא מורשה במפורש.

מיקום FW: בין הרשת הפרטית לציבורית, ובין שתי רשתות פנימיות בארגון.

Bastion Host (BH): מחשב שניתן לגשת אליו מהרשת הציבורית (בד"כ מכיל שרתים שמשרתים את הרשת הציבורית, ולפעמים גם את הרשת הפרטית). בדרך-כלל יושב מחוץ לרשת הפרטית. מחשב זה צריך להיות מאובטח היטב. יש לדאוג לכך שתוקפים לא יוכלו לנצל אותו לצורך התקפות על הרשת הפרטית. ייתכנו מספר BH לאותה רשת פרטית.

סוגי BH:

Non-routing dual-homed host: מחשב שמחובר למספר רשתות (למשל הרשת הפרטית והרשת הציבורית), נותן להן שירותים, אך **לא** מעביר חבילות בין הרשתות.

BH קורבן: שרת זה מיועד לכלל המשתמשים באינטרנט ולכן קשה לאבטח אותו וה-FW לא יכול להוסיף לו הרבה אבטחה. חשוב לדאוג שלא ניתן יהיה לתקוף דרכו את הרשת הפרטית.

BH פנימי: ממוקם פיסית **בתוך הרשת הפרטית ונותן שירותים למשתמשים מהאינטרנט**. יש לאבטח בצורה מיוחדת. רצוי להימנע מ-BH מסוג זה.

האזור המפורז (DMZ): האזור בו נשים את השרתים הבעייתיים (ה-BH-ים שלנו). זוהי רשת בתוך הארגון המפרידה בין הרשת הפרטית, המוגנת, ובין הרשת הציבורית. DMZ מוסיף עוד שכבה של הגנה על הרשת הפרטית. המחשבים בתוכו יכולים לשמש כ-proxies.

טכניקות למימוש Firewall:

Stateless Packet Filtering: Packet filter סורק כל חבילה שעוברת דרכו, ומחליט האם להעביר אותה הלאה או לא. ההחלטה מבוססת על **טבלת חוקים** המגדירה מתי להעביר את החבילה (forward).

- הטבלה נכתבת ע"י מנהלי הרשת בהתאם למדיניות ההגנה. זו טבלת חוקים סטטית- לא מתעדכנת בזמן הפעולה (ולא על-פי תוכן החבילות).
- ה-TCP/IP headers של כל חבילה נבדקים בטבלה, לפי סדר השורות בטבלה, וללא קשר לחבילות אחרות.
- החבילה מועברת הלאה או נזרקת לפי **השורה הראשונה בטבלה** שמתאימה לחבילה. אם אף שורה לא מתאימה, החבילה נזרקת. על אף שזוהי ברירת המחדל, מומלץ להוסיף כלל דיפולטי בתחתית הטבלה שדוחה כל חבילה.

©ענת עציון

- בחבילה שיוזמת TCP session, דגל ה-ack=0. ובשאר החבילות של ה-session, דגל ה-ack=1 ולכן ניתן לדעת האם חבילה יוזמת session חדש ולהתייחס לכך בחוקים בטבלה. מדיניות הגנה נפוצה היא לא לאפשר ליזום sessions מבחוץ אל תוך הרשת הפרטית.

FTP-פרוקוטול בעייתי: פרוטוקול שעובר מעל TCP, ומשתמש בשני פורטים קבועים לשרתים:

- 21- command port.
- 20- data port.

בהגדרה המקורית של ה-FTP הלקוח מעביר ב-command session את מספר הפורט שישמש אותו ב-data session. השרת פותח את ה-data session מפורט 20 שלו אל מספר הפורט שקיבל מהלקוח. צורה זו נקראת FTP אקטיבי.

בעיה ב-FTP אקטיבי: השרת מאתחל את ה-data session.

פיתרון- FTP-פאסיבי: הלקוח שולח את פקודת pasv ב-command session. השרת מגיב עם מספר פורט אקראי (גדול מ-1023). הלקוח פותח data session מפורט אקראי אל הפורט שקיבל מהשרת.

הערה: במקרה של FW המסוגל לבחון את תוכן ה-session ניתן להמשיך להשתמש בגרסה האקטיבית.

Stateful Packet Filters: לגבי החבילה הראשונה שפותחת session נעשה שימוש בחוקים סטטיים על-מנת להחליט האם זהו session חוקי. כל חבילה שאינה מתחילה session חדש ואינה שייכת ל-session פתוח- תיזרק.

Stateful inspection: שיפור זה מסוגל לסרוק כל שדה בחבילה, בפרט הוא מסוגל לסרוק את שדות האפליקציה (הספציפיים לכל אפליקציה). מכיוון ששדה מסויים באפליקציה יכול לחצות גבולות חבילה, הרי שכדי שה-packet filter יוכל לבצע סריקה אחרי שדות בעלי משמעות ברמת האפליקציה, הוא חייב לזהות את כל החבילות ששייכות לאותו ה-session, ולסרוק אותן לפי הסדר-Inspection.

הערה: Stateful inspection יחד עם שמירת ה-session context מאפשרים להתגבר על אי-הידידותיות של FTP- כאשר נפתח ה-command session, הלקוח שולח לשרת את מספר הפורט שלו עבור ה-data session. מערכת ה-stateful inspection מציצה לתוכן ההודעה וזוכרת את מספר הפורט המועבר. לאחר מכן, היא תאפשר את מעבר החבילה שפותחת את ה-data session מהשרת אל הלקוח.

Proxy servers:

תכניות שרצות במערכת ה-firewall. משמש כ-Man In The Middle חוקי. הלקוח פונה ל- proxy server וזה פונה בשמו לשרת. הוא מאפשר מידע אך ורק דרך ה-proxy applications.

הערה: בגלל מורכבות הבדיקות שהוא מבצע, הוא יותקן על מחשב **בתוך** רשת הארגון ולא ביציאה מהרשת. אפשר להשתמש ב-FW packet filtering שהחוקים בו יאפשרו רק תקשורת שנעשית דרך ה-proxy servers וכך ימנעו עקיפה שלו.

- מכיוון שהוא עובד ברמת האפליקציה, הוא יכול לבצע בדיקות נוספות כמו: אנטי וירוס.

דוגמא- SMTP: SMTP מאפשר לכל שרת לקבל הודעות עבור כל שרת אחר, ולכן ניתן לשים mail gateway ארגוני ולדאוג כי תעבורת הדואר הנכנסת תכנס אלו. שרת הדואר המרכזי יכול להריץ אנטי וירוס, מסנן דואר זבל. הוא גם יכול לדאוג שהודעות עם מידע סודי לא יוכלו לצאת מהארגון

יתרון: אפשרות לבקרת גישה על ישויות שקיימות רק ברמת אפליקציה. אין קשר TCP ישיר בין שרת ללקוח.

חיסרון: לכל אפליקציה חדשה יש לכתוב proxy חדש. לרוב אין שקיפות לקצוות.

הערה: במהלך הקורס נניח ש-proxy server לעולם לא יהיה מותקן על אותו המחשב כמו Packet filtering firewall וכי כל proxy server מותקן על מחשב נפרד.

©ענת עציון

IPsec – הגנה בשכבת הרשת

ארכיטקטורת IPsec:

- מכיל שני פרוטוקולים לאבטחת תעבורת IP:
  ESP: מספק הצפנה ו/או אימות.
  AH: מספק אימות בלבד.
- פרוטוקול לניהול והחלפת מפתחות IKE.

ניתן להשתמש ב-IPsec בשני אופנים: Transport mode ו-Tunnel mode.
הקמת ערוץ IPsec: רק בין שתי מכונות שיש ביניהן אמון הדדי (למשל ע"י החלפת מפתחות ארוכי טווח, פומביים או סימטריים). האמון ההדדי בין המכונות מצמצם החשיפה ל-syn attack ו-DoS.
Security Association Database (SAD): כל מחשב שמותקן בו IPsec, מחזיק מבנה נתונים זה. המבנה מחזיק רשומות שנקראות- Security Association (SA).
Security Association (SA): רשומה כזו מכילה את כל הנתונים הדרושים לקיום קשר IPsec בין שני מחשבים: האלגוריתמים המשמשים להצפנה/אימות, המפתחות הקריפטוגרפיים, אורך החיים של ה-SA, מונה מספר סידורי (sequence number), גודל חלון עבור מספרים סידוריים, מוד הפעלה (Tunnel/Transport mode ,ESP/AH).
הערה: כל SA מגדיר קשר חד-כיווני. את ה-SA מגדירים בזוגות, אחד לכל כיוון.
Security Policy Data (SPD): מאפשר למשתמש לבחור את שירותי האבטחה שיינתנו לשירות מסוים, ואת האלגוריתמים הקריפטוגרפיים שבאמצעותם ימומשו שירותי האבטחה. ה-SPD מוזן ע"י המשתמש ומייצג את חשיבות האבטחה לכל חיבור אפשרי.

מבנה ה-SAD:

- רשומות SA.
- SPI- אינדקסים של הרשומות.
  הערה: מכיוון שהתוכן של SA משותף לשתי מכונות (צד שולח וצד מקבל), לכל SA יש שני SPI, אחד במכונה A והשני במכונה B. ה-SA מ-A יכיל את כל המידע הדרוש ל-A כדי לשלוח חבילה מאובטחת ל-B ואת כל המידע הדרוש ל-B כדי לקרוא ולאמת חבילה זו.
- לכל כיוון של תעבורה בין A ל-B מתאים SA אחד.
- ה-SAD מחולק לשתי טבלאות: אחת לתעבורה יוצאת ואחת לתעבורה נכנסת.

Security Parameter Index (SPI): מסמר באורך 32 ביט שמשמש כמצביע אל העותק של ה-SA אצל הצד המקבל. ערך ה-SPI לתעבורה מ-A ל-B מוקצה ע"י B. הגישה לטבלת ה-SA הנכנסים נעשית לפי ה-SPI. כאשר A שולח חבילה ל-B בתוכה רשום ה-SPI של B כדי שהוא יידע עם איזה SA לפתוח את החבילה.
הערה: לצורכי יעילות ניתן לשמור מצביע בין שני SA של ניתוב בשתי הטבלאות.

Authentication Header (AH):

מספק: הגנה כנגד שידור חוזר, אימות ושלמות (גם של ה-IP header).

Authentication data: שדה שהוא ה-MAC המחושב על שרשור של ה-IP header החיצוני, ה- AH header וה-data payload. השדות ב-IP header החיצוני שמשתנים בכל hop (דוגמת ה-TTL) מאופסים לשם חישוב ה-MAC. כמו-כן מאופס השדה שיכיל את ה-MAC.

תהליך ה-Encapsulation:

1. מציאת ה-SA.

©ענת עציון

2. שליפת ה-SPI של הצד השני.
3. שליפת ה-sequence number והגדלה ב-1.
4. בניית ה-AH header.
5. בניה (מוקדמת) של ה-IP header.
6. חישוב ה-MAC (תוך איפוס זמני של חלק מהשדות).

תהליך ה-Decapsulation:

1. מציאת ה-SA (על-סמך ה-SPI) ובדיקה ש-SA זה משותף עם השולח ויכול לשמש לצורך הנוכחי.
2. בדיקת ה-Sequence number (שהוא בחלון).
3. חישוב ובדיקת ה-MAC (תוך איפוס זמני של חלק מהשדות).
4. אם בדיקה כלשהי נכשלת- זרוק את החבילה.
5. אחרת, הגדל את ה-sequence number ב-SA ב-1.
6. המשך עיבוד לפי ה-Next protocol.

Encapsulating Security Protocol (ESP):

מספק: סודיות, אימות, שלמות והגנה כנגד שידור חוזר.

שדות ב-ESP שמופיעים גם ב-AH:

- **SPI**- מצביע ל-SA אצל המקבל.
- **Sequence number**- מונה שמשמש להגנה מפני שידור חוזר.
- **IV**- במידה ואלגוריתם ההצפנה דורש זאת.
- **Payload data**- המידע עליו מגן ה-SA.
- **Padding**- במידה ודרוש, אם ה-SA מצפין בעזרת צופן בלוקים.
- **Pad length**- אורך ה-padding, מופיע תמיד (גם אם אין צורך).
- **Next protocol**- מספר הפרוטוקול הבא.
- **Authentication data**- תוצאת MAC על המידע ועל ה-ESP headers.

הערה: ה-IP header אינו מוגן ע"י ה-ESP.

תהליך ה-Encapsulation:

1. מציאת ה-SA.
2. שליפת ה-SPI של הצד השני.
3. שליפת ה-sequence number והגדלה ב-1.
4. לצורך ההצפנה: קביעת ה-IV (אם נדרש) ודיפון המידע (אם נדרש).
5. שרשור ה-Pad length וה-Next protocol number.
6. הצפנה באלגוריתם שנקבע ב-SA.
7. חישוב ה-MAC.

תהליך ה-Decapsulation:

1. מציאת ה-SA (על-סמך ה-SPI) ובדיקה ש-SA זה משותף עם השולח.
2. בדיקת ה-Sequence number (שהוא בחלון).
3. חישוב ובדיקת ה-MAC (תוך איפוס זמני של חלק מהשדות).
4. אם בדיקה כלשהי נכשלת- זרוק את החבילה.
5. אחרת, הגדל את ה-sequence number ב-SA ב-1.
6. פענוח
7. בדוק ש-SA זה יכול לשמש לצורך הנוכחי, וזרוק את החבילה אם לא (בדיקה זו אינה יכולה להעשות לפני הפענוח).
8. המשך עיבוד לפי ה-Next protocol.

©ענת עציון

Security Policy Data (SPD):

מבנה: רשימה של חוקים (דומים לחוקי FW) שמוזנים ע"י המשתמש. כל חוק מורכב מ-selectors ומפעולה.

Selectors: יכולים להיות כתובות IP, טווחים של כתובות IP, מספרי פורטים TCP/UDP ועוד.

פעולה: יכולה להיות- זרוק את החבילה, שלח חבילה רגילה, שלח חבילה מוגנת ב-SA נתון או- יצר SA ושלח חבילה מוגנת. במקרה האחרון הפעולה מפנה לתכונות (אלגוריתמים קריפטוגרפיים) שיש להשתמש בהן ב-SA.

עיבוד החבילה ביציאה:

1. ניגשים עם החבילה ל-SPD.
2. אם תשובת ה-SPD: זרוק או שלח חבילה רגילה, יש לזרוק את החבילה או להעבירה לשכבה הבאה, בהתאמה.
3. אם תשובת ה-SPD: צור SA ושלח חבילה מוגנת- מפעילים את IKE לשם בניית ה-SA בהתאם לכללים שה-SPD החזיר.
4. אם התשובה: שלח חבילה מוגנת ב-SA נתון- מבצעים עיבוד החבילה בהתאם.

עיבוד החבילה בכניסה:

1. ניגשים ל-SAD עם האינדקס SPI שבחבילה ומאמתים/מפענחים את החבילה. אם אין SA מתאים, זורקים את החבילה.
2. ניגשים עם החבילה וה-SPI ל-SPD. ה-SPD בודק התאמה ביניהם. אם אין התאמה, זורקים את החבילה.
3. מעלים את החבילה לשכבה הבאה.

אופני התפעול של IPsec:

Transport mode: שני מחשבי הקצה מתקשרים ביניהם באמצעות IPsec באופן ישיר. מספק שירותי אבטחה מקצה לקצה (end to end) לכן, מחייב את שני הקצוות להפעיל את IPsec בעצמם.

Tunnel mode: השימוש ב-IPsec לא נעשה ע"י משתמשי הקצה, הם מקיימים ביניהם קשר IP רגיל וישנם Security gateways שמפעילים את ה-IPsec.

תכונה נוספת: הסתרת כתובות ה-IP אמיתיות של מערכות הקצה, בזמן מעבר החבילה ברשת.

הערה: ייתכן שעל אותה חבילה, בזמן הניתוב, מפעילים קודם את IPsec ב-transport mode במחשב הקצה ולאחר-מכן מספר פעמים IPsec ב-tunnel mode ב-Security gateways בדרך.

Virtual Private Network (VPN):

רשת פרטית וירטואלית- רשת הנבנית מעל רשת ציבורית (לדוגמא האינטרנט) באופן המשמש את פרטיות משתמשי ה-VPN. הרעיון הוא שרק המשתמשים המורשים של VPN מסוגלים לדבר אחד עם השני, ורק הם מסוגלים לשלוח ולקבל הודעות ב-VPN.

Network Address Translation (NAT):

פרוטוקול שמאפשר לרשת גדולה לעבוד עם מספר קטן של כתובות IP.

- בתוך הרשת ייעשה שימוש בכתובות IP "לא חוקיות".
- כאשר מחשב מהרשת הקטנה יוצר קשר עם האינטרנט, ה-gateway מתרגם את כתובתו לכתובת IP חוקית וכך גם בכניסת מידע מהאינטרנט.
- לא ניתן להשתמש ב-AH ב-transport mode יחד עם NAT כי יש אימות על שדה ה-IP.

©ענת עציון

IKE- Internet Key Exchange

הגדרות כלליות של IKE:

- IKE הוא פרוטוקול ובניית וניהול IPsec SAs בין שני מחשבים שממשים IPsec.
- IKE הוא הפרוקוטול הסטנדרטי היחיד לבנית IPsec SAs (מימוש סטנדרטי של IPsec חייב להכיל מימוש של IKE).
- IKE (כמו IPsec) יכול להתנהל בין זוג hosts, זוג security gateways או בין host ו- security gateway.
- IKE הוא צירוף של שני פרוטוקולים, ISAKMP ו-Oakley.
- IKE ממומש כאפליקציה שעובדת מעל UDP, פורט 500.

תכונות פרוקוטול להחלפת מפתחות:

IKE הוא פרוטוקול להחלפת מפתחות- מטרתו הסכמה על מפתח משותף והוא צריך לקיים:

Authenticity: אימות הזהות של המשתתף השני (מניעת התקפות Main in the middle).

Secrecy: רק שני המשתתפים הלגיטימיים יודעים את המפתח הסודי המשותף.

מפתחות ארוכי טווח: מפתח משותף סודי או מפתח פומבי של הצד השני.

מפתחות קצרי טווח: המפתחות עליהם מסכימים במהלך הפרוטוקול.

הערה: על-מנת להריץ פרוטוקול להסכמה על מפתחות עם אימות, הצדדים צריכים להסכים מראש על מפתח משותף (או להחליף מפתחות פומביים) בצורה בטוחה- שלא דרך הרשת.

מפתחות ארוכי טווח:

למה צריך פרוטוקול להסכמה על מפתחות אם קיימים כבר מפתחות ארוכי טווח?

תשובה: לא רצוי להצפין מידע רב עם אותו מפתח, ובפרט רצוי להפריד בין מפתח להצפנה למפתח לאימות.

Perfect Forward Security (PFS): נאמר שפרוקוטול להסכמה על מפתחות מקיים את תכונת ה-PFS אם חשיפה עתידית של מפתחות סודיים ארוכי טווח עכשוויים לא תגרום לחשיפת המפתחות קצרי הטווח שנוצרים בביצוע הנוכחי של הפרוטוקול. כלומר, גם אם המפתח ארוך הטווח נשבר, המידע שכבר עבר מוצפן, נותר בטוח.

הפאזות של IKE:

פאזה I: שני הצדדים מייצרים ISAKMP SA (כולל אלגוריתמים, שיטת אימות ומפתחות)- ערוץ מאובטח המגן על הפאזה השנייה.

פאזה II: מיוצרים ה-IPsec SA שיאוחסנו ב-SAD- Quick mode.

מדוע יש שתי פאזות?

- על-מנת להשיג בטיחות טובה יותר, בדרך-כלל יש לבצע חישובים יקרים. לכן: מריצים את הפאזה הראשונה (היקרה יותר מבחינה חישובית) לעיתים רחוקות יחסית (למשל פעם ביום).
- לאחר-מכן משתמשים ב-ISAKMP SA למספר רב של הרצות הפאזה השנייה (שהיא יעילה יותר).

הפאזה הראשונה:

1. החלטה על תכונות ה-ISAKMP SA (אלגוריתמים, זמן חיים, צורת אימות וכו').
2. הכנת keying material סודי משותף לשני הצדדים, ורק להם, שממנו ייגזרו מפתחות ל- ISAKMP SA.
3. אימות שני המשתתפים בפרוטוקול.

©ענת עציון

Main mode:

מורכב מ-3 שלבים, בכל אחד שתי הודעות. בכל שלב רמת האבטחה גדולה יותר, לכן חישובים כבדים נדחים לשלב מאוחר ככל שניתן.

שלב 1- הסכמה על SA: המאתחל שולח מספר הצעות ל-SA, הצד המגיב שולח בחזרה את ההצעה שבה בחר. ה-SA מכילים את כל המידע הדרוש אך לא את המפתחות.

שלב 2- החלפת מפתחות DH: שני הצדדים מחליפים את מפתחות DH הציבוריים שלהם (מפתחות אלה הם קצרי טווח ומתחלפים בכל הרצה של IKE בפאזה הראשונה). כל צד שולח מחרוזת אקראית (nonce) נגד שידור חוזר.

שלב 3- אימות זהויות: כל צד שולח לצד השני את זהותו (למשל כתובת IP), מחרוזת Auth וסרטיפיקט (רק אם נעשה אימות בעזרת חתימות, ואז יש צורך לאמת את המפתחות הציבוריים). כל אלה נשלחים מוצפנים תחת המפתח- $SKEYID_e$ שמצריך ידיעת הזהויות במקרה של PSS.

HDR- cookies שהן מחרוזות אקראיות. יש להן שני תפקידים:

1. משמשות כ-session identifier (מכיוון שב-UDP אין זיהוי של sessions).
2. מונע IP spoofing (ולכן מקטין את הסיכון של DoS): כל משתתף בפרוטוקול שולח מחרוזת אקראית לצד השני, הצד השני נדרש להחזיר את ה-cookie ולהוכיח בכך שכתובת ה-IP שלו היא אמיתית.

הגנה כנגד DoS: צוואר הבקבוק החישובי של IKE הוא חישוב מפתח ה-DH המשותף וה-Authenticator. רצוי למנוע התקפת DoS בה התוקף מבצע IP Spoofing. בהתקפה זו מציף התוקף את הקורבן בבקשות IKE ומכריח אותו לבצע המון חישובים יקרים. לפני ביצוע החישובים היקרים נרצה לוודא שהצד השני אכן נמצא בכתובת IP שמופיעה ב-IP header. ולכן רק לאחר שהצד השני שולח cookie ומוודאים שהוא נכון, מחשבים את המפתח DH הציבורי.

דרך יצירת ה-cookies: הפעלת פונקציית תמצות על: כתובת ה-IP, ערך סודי מקומי, תאריך ושעה (למניעת שידור חוזר).

Identity protection: זהויות המשתתפים בפרוטוקול נשארות מוגנות- הן נשלחות בשלב השלישי בצורה מוצפנת. לכן התוקף לא יודע מיהם המשתתפים בפרוטוקול.

Aggressive mode:

זהו פרוטוקול החלפת מפתחות שאינו מספק identity protection. הוא מבצע את שלושת השלבים המבוצעים ע"י main mode ב-3 הודעות. הוא מהיר יתר אך גם חשוף יותר להתקפות.

- אין הצפנה ולכן אין identity protection.
- ב-HDR עדיין יש cookies אך הן אינן מגנות מ- DoS אלא רק מגדירות את ה-session.
- החישובים הכבדים מתבצעים בשלב מוקדם יחסית (לפני קבלת ה-cookies)

המפתחות ארוכי הטווח:

המפתחות שמשמשים לאימות שני המשתתפים ולאבטחת הפרוטוקול. קיימים מספר סוגים:

Pre-shared secret (PSS): מפתח סימטרי סודי הידוע לשני הצדדים (דורש הזנה ידנית).

Signatures: לכל משתמש זוג מפתחות פומבי-פרטי ארוך טווח.

בפאזה הראשונה המפתחות ארוכי הטווח משמשים לחישוב ה-authenticators.

PFS: המפתחות קצרי הטווח נגזרים ממפתח ה-DH המשותף עליו מסכימים הצדדים. מפתח DH של כל צד הוא חד-פעמי וחשיפתו של המפתח ארוך הטווח איננה גורמת לחשיפתו של המפתח קצר הטווח.

חישוב בשיטת PSS:

- ה-Authenticator הוא חישוב של MAC על מידע שעבר בפרוטוקול עם המפתח SKEYID.
- רק מי שיודע את ה-PSS מסוגל לחשב את SKEYID. לכן חישוב נכון של ה-authenticator מוודא: ידיעת ה-pss, אמיתות כל המידע החתום (מתקיף לא שינה אותו...).

©ענת עציון

PRF-Pseudo Random Function. $SKEYID = PRF_{pss}(N_i \mid N_r)$

ומחושבים AUTH שהם PRF_{SKEYID} על חלק מהמידע שעבר בפרוטוקול.

חישוב בשיטת Signatures:

- ה-SKEYID אינו מבוסס על סוד משותף: $SKEYID = PRF_{N_i|N_r}(g^{x_i x_r})$
- ה-AUTH הוא ייצור חתימה בעזרת מפתח פרטי על ה- PRF_{SKEYID} .
- החתימות מוכיחות אותנטיות של המידע וזהות (רק בעל המפתח הפרטי יכול לייצר חתימה).

הערות:

- באימות עם PSS, רק המשתתפים האמיתיים מסוגלים לחשב את ה-SKEYID (מפני שחישוב זה מצריך את ידיעת ה-pss). חישוב ה-hash מוכיח גם את זהות המשתמש וגם שצד זה אכן שלח את הערכים עליהם מחושב ה-hash. ה-authenticator הוא התמצות עצמו.
- באימות עם חתימות, כל מי שהשתתף בביצוע הפרוטוקול יכול לחשב את SKEYID ואת ערך ה-hash (גם אם הוא איננו מי שהוא טוען שהוא). הוכחת הזהות של צד מסויים מתבצעת ע"י החתימה שלו על ה-hash. ה-authenticator כאן הוא החתימה על ה-hash.

גזירת מפתחות:

לכל צד ישנה מחרוזת SKEYID. המחרוזת אף-פעם לא נשלחת ברשת. מלבד חישוב ה-AUTH היא משמשת לגזירת המפתחות עבור ה-SAs הנבנים. נגזרים 3 מפתחות:

$SKEYID_d$: משמש לגזירת שאר המפתחות וגם לגזירת המפתחות בפאזה השנייה (המפתחות עבור ה-IPsec).

$SKEYID_a$: מפתח לאימות בתוך ה-SA שנבנה.

$SKEYID_e$: מפתח להצפנה בתוך ה-SA הנבנה.

הערה: המפתח שאיתו מוצפנות שתי ההודעות האחרונות של ה-main mode הוא ה-$SKEYID_e$.

בעיה- אין Identity protection עם PSS וזהויות שאינן כתובות IP:

- בכדי למצוא מהו ה-PSS שבעזרתו מחושב המפתח יש לדעת את הזהות, אך זו מוצפנת.
- אם הזהות היא כתובת IP אז היא מופיעה ב-IP header.
- ניתן לתקן את הבעיה ע"י למשל הוצאת ה-pss מחישוב ה-$SKEYID_e$ אבל אז ההגנה על הזהויות תהיה רק נגד התקפות פאסיביות.

התקפה אקטיבית על Identity protection: באימות עם חתימות ניתן לחשוף את זהות היוזם ע"י התקפה אקטיבית. המגיב- מוגן. מתחזים למגיב ושולחים מפתח DH פומבי אחר.

הפאזה השנייה:

- קיים exchange בודד שנקרא Quick mode, באורך 3 הודעות.
- ההודעות מוגנות ע"י המפתחות שנוצרו בפאזה הראשונה- $SKEYID_e$
- Quick mode בונה את ה-IPsec SA.
- אם לא מבוצע DH ב-Quick mode, המפתחות נגזרים מה-DH שחושב בפאזה הראשונה.
- פיצוח ה-DHבפאזה הראשונה יחשוף את כל מפתחות ה-IPsec SA.
- אם מעוניינים ב-PFS יש לבצע DH בכל Quick mode.

©ענת עציון

Wireless LAN security

רשת מקומית אלחוטית: נקבעת ע"י אזור גיאוגרפי פיזי (שנמצא כולו בטווח קליטה/שידור) וטווח אורכי הגל שעליהם עובדת הרשת. ייתכן מצב בו באותו אזור פיסי, מעל אותם אורכי גל, עובדות מספר רשתות מקומיות. לכל רשת כזאת יש מחרוזת יחודית שנקראת SSID (service net identifier). כאשר משתמש מתחבר לרשת, עליו לבחור את ה-SSID של הרשת שברצונו להתחבר אליה.

802.11:

תקן לרשתות מקומיות אלחוטיות (רמת MAC) שפורסם ע"י IEEE. נקרא גם Wi-Fi. שני אופני תפעול:
Ad-hoc: קשר נקודה-לנקודה בין שתי תחנות ניידות.
Infrastructure: התחנות הניידות מתקשרות עם תחנה מרכזית נייחת, תחנת בסיס. תחנה זו נקראת גם נקודה גישה (Access point) מאר והיא מספקת גישה אל מחוץ לרשת המקומית. התחנות הניידות חייבותל להימצא בטווח קליטה ושידור מתחנת הבסיס.

- כמות איבוד החבילות והשגיאות גבוהה.
- אין כבלים.
- פרישת הרשתות זולה.
- חשופות להתקפות "מגרש חניה"- תוקף מסוגל להקשיב לרשת הארגונית גם ממגרש חניה (לא צריך כניסה פיזית לארגון).
- חשופות להתקפת ע"י rouge access point (מתחזה ל-access point).

שירותי אבטחה ב-802.11:
Wired Equivalent Privacy (WEP): אמור לשמור על: סודיות, שלמות ובקרת גישה. בפועל- לא ממלא אף-אחד מתפקידיו!

802.11i:

תקן חדש שנועד להתמודד עם בעיות האבטחה ב-802.11. הוא מגדיר פרוטוקולי אימות משתמש, ניהול מפתחות ואבטחת תעבורה חדשים. הוא מגדיר שני פרוטוקולים לאבטחת שכבת ה-MAC:
TKIP (Temporal Key Integrity Protocol), מבוסס על WEP. מוסיף ניהול מפתחות, מחליף את ה-CRC ב-MIC (message integrity code) ועוד.
CCMP – CRT with CBC-MAC protocol (מוגדר בתקן WPA2)- מבוסס על AES.
כיום: אחד ממרכיבי האכיטקטורה הם פרוטוקולי האימות ובקרת הגישה בתקן 802.1X ובקרת גישה המבוססת על EAP.

בקרת כניסה ברשת אלחוטית:

Roaming: נדידה היא מעבר בין רשתות פיסיות שוות תוך שמירה על זהות ותכונות המשתמש כמו: הרשאות, השירותים להם הוא זכאי ואינפוקמציית הזהות שלו.
התחברות לרשת מקומית- תרחיש 1: הקודת הגישה תומכת במשתמשים שנמצאים באותו מקום פיסי. אם נקודת הגישה היא נתב, היא יכולה לספק חיבור לאינטרנט. אין תמיכה ב-roaming.
התחברות לרשת מקומית-תרחיש 2: יש נקודות גישה בכמה מיקומים פיסיים מרוחקים זה מזה וכולן מחוברות לאינטרנט. האימות מתבצע ע"י שרת אימות יחיד. התקשורת בין gateways לשרת אימות נעשית בפרוטוקול RADIUS.
RADIUS (Remote Access Dial In User Service): פרוטוקול שרץ מעל UDP. נועד להעביר אינפורמציית אימות מעל האינטרנט. המשתתפים הם נקודת הגישה ושרת האימות. הוא מבצע אימות של שני הצדדים.

©ענת עציון

802.1X:

המצב לפני:

- לא נעשה אימות משתמש ברמת ה-MAC.
- ברגע שמשתמש מתחבר, נקודת הגישה מקצה לו כתובת IP, אך אינה מאפשרת לו יציאה אל מחוץ לרשת.
- לאחר קבלת כתובת IP, מבצע המשתמש אימות מעל SSL אל נקודת הגישה. נקודת הגישה מוודאת את אינפורמצית האימות מול שרת אימות מרוחק, מעל RADIUS.
- ברגע שהאימות מסתיים בהצלחה, המשתמש מקבל גישה לאינטרנט דרך ה-ISP.

בעיות:

- נקודת הגישה מקצה משאבים (ערוץ תקשורת, כתובת IP) מבלי לבצע כל בדיקה.
- למשתמש אין דרך לוודא שנקודת הגישה משתמשת ב-SSL.
- התחזות של נקודת הגישה.
- חטיפת הקשר.
- נקודת הגישה חייבת לממש SSL ובקרת כניסה.

פיתרון- 802.1X:

זהו תקן לאימות משתמשים ברמת ה-MAC. האימות מתבצע לפני קבלת כתובת IP ע"י המשתמש. יש המכנים אותו כ-EAPoL.

- אין בזבוז משאבים על משתמשים לא מורשים.
- תומך ב-Roaming.
- אכיטקטוררה לאימות המשתמשים המתבססת על EAP.
- קיים אוסף של שיטות אימות שנכתבו עבור 802.1X. המרכזיות הן LEAP, PEAP ו-EAP-TLS.

EAP:

פרוקוטוקול נושא (Carrier) לפרוקוטוקולי אימות, הוא אינו מגדיר את שיטת האימות (Method). הוא רץ מעל TCP ומעל PPP.

יכול לרוץ מעל שכבות שונות, כולל MAC ו-Application.

המשתתפים ב-802.1X וב-EAP:

Supplicant: יחידת המשתמש, המבקש להתחבר לרשת המקומית.

Authenticator: יחידת הגישה אליה מבקש המשתמש להתחבר (בד"כ נקודת הגישה). ממנה ניתן להתחבר לאינטרנט. זוהי המערכת שמבקשת לזהות את הלקוח ולא זו שמבצעת את הזיהוי.

Authentication Server: שרת האימות, שבו יושבים נתוני האימות של המשתמשים.

מודל האימות ב-802.1X:

ה-Authenticator (יחידת הגישה) משמש כתחנת ממסר להודעות פרוטוקול האימות (EAP) מבלי שתבין את תוכנן. פרוטוקול האימות מתבצע בין יחידת המשתמש ובין שרת האימות. בסיום הפרוטוקול מקבל ה-Authenticator משרת האימות תשובה ובהתאם מתאפשר או לא, חיבור המשתמש לאינטרנט.

אכיפה מתבצעת בשילוב ע"י:

- ה-Authenticator שמחליט האם הגישה ל-port מוגנת או לא.
- שרת האימות (AS) שמחליט מהי צורת האימות.

©ענת עציון

- תמיכה ב-roaming: אפשרותת לגישה לשרת אימות מרכזי מכל הרשתות המקומיות שמחוברות לאינטרנט.
- אי-תלות שיטת האימות ביחידת הגישה המקומית: ניתן לשנות שיטת אימות ע"י שינוי השרת ויחידת המשתמש, ללא שינוי יחידת הגישה ברשת המקומית.
  הערה: שרת אימות ויחידת קצה ניתנים לשליטה ע"י ארגון בודד, ה-access point לא!

MitM ב-802.1X:

בעיות:

- מכיוון שה-authenticator אינו שותף לתהליך האימות ובפרט איננו מאומת ע"י יחידת המשתמש, 802.1X חשופה להתקפה ע"י rouge access point.
- המתקיף מצליח להתחזות למשתמש האמיתי, ואף לקיים קשר ברמות IP ומעלה.

פיתרון: שימוש בשיטת EAP שגוזרת מפתחות:

- המתחזה איננו מסוגל לייצר את המפתחות. המפתחות ישמשו להגנת הקשר בין יחידת הגישה ויחידת המשתמש ברמת ה-MAC.
- מכיוון שגזירת המפתחות היא מול שרת האימות, הוא יעביר אותם לנקודת גישה.
- הקשר בין יחידת הגישה ושרת האימות חייב להיות מאומת, הדדי וסודי.
- 802.1X מייצר גם מפתח סודי PKM (Pair-wise Master Key) שמועבר ל-Authenticator.
- WPA2 מגדיר כיצד גוזרים יחידת המשתמש ויחידת הגישה מה-PMK את המפתחות שמשמשים להגנת התעבורה באמצעות CCMP.

איך WEP עובד:

ניהול מפתחות:

- לכל רשת ישנו מפתח K משלה.
- בכל תחנה נייחת מוגדר מפתח לכל רשת שמעוניינים להתחבר אליה ב-WEP.
- כל המשתמשים הניידים המנסים להתחבר לנקודת הגישה חייבים לדעת את K. כלומר, כל המשתמשים מקבליפ ממנהל המערכת את אותו מפתח משותף K.
- המפתח K הוא קבוע- התקן אינו מגדיר מנגנון לשינוי והפצת המפתחות. לכן המפתחות מתחלפים לעיתים רחוקות, אם בכלל.

ההצפנה ב-WEP:

- משתמש בצופן השטף RC4:
  - בהינתן מפתח (קצר יחסית) K, מייצרים מפתח מורחב KS (Key Stream).
  - ההצפנה מתבצעת ע"י $C = M \oplus KS$.
  - הפענוח מתבצע ע"י $M = C \oplus KS$
- בעיה: אם משתמשים באותו מפתח K יותר מפעם אחת, מקבלים את אותו מפתח מרוחב KS. קל לגלות תוכן של הודעות שהוצפנו תחת אותו מפתח.
- החלפת מפתחות היא פעולה יחסית ארוכה ומורכבת, ולכן לא רצוי להחליף מפתחות בכל חבילה.
- הפיתרון של WEP: שימוש במספר סידורי-IV, אורכו: 24 ביט.
- לשני הצדדים מפתח משותף K- שאורכו 40 ביט (או כיום 104 ביט).
- מפתח ההצפנה (שממנו מיוצר ה-key stream) מתקבל ע"י שרשור המפתח המשותף וה-IV.
- **בהצפנה:** משתמשים ב-IV חדש. המפתח המורחב מתקבל ע"י $KS_{IV} = RC4(IV, K)$. ה-IV נשלח בצורה גלוייה ביחד עם ההודעה המוצפנת.
- **הפענוח:** מחשבים את $KS = RC4(IV, K)$ ואז מקסרים עם ההודעה המוצפנת. זה מועיל רק אם דואגים שה-IV לא חוזר על עצמו בזמן החיים של המפתח K.

©ענת עציון

פרוטוקול אימות המשתמש:

- תחנת הבסיס מוודאת את זהות המשתמש של התחנה הניידת.
- המשתמש נדרש להוכיח שהוא יודע את המפתח המשותף הסודי K.
- תחנת הבסיס שולחת Challenge בצורה גלויה. המשתמש נדרש להחזיר Response מוצפן ב-RC4 תחת K.

חולשות WEP:

נקודת תורפה 1 - מפתח קצר:

- גודל המפתח הרשמי ב-WEP הוא 40 ביט.
- המאמץ החישובי הדרוש לביצוע 2^{40} נסיונות הצפנה של RC4 (כדי למצוא את המפתח בחיפוש ממצה) הוא קטן- פחות מיממה.

הפיתרון הפשוט המתבקש: הארכת המפתח (ואכן נהוג להשתמש במפתח באורך 104 ביט).

נקודת תורפה 2 - ערכי IV חוזרים על עצמם:

- המפתח המשותף של כל המשתמשים הניידים ותחנת הבסיס הוא קבוע, זהה וידוע לכולם. כלומר: כל המשתמשים יכולים לשמוע את כל ההודעות העוברות ברשת (בדומה ל-Ethernet).
- חייבים להבטיח שה-IV לא ייחזרו על עצמם כדי שה-key stream לא יחזור על עצמו.
- אורך שדה ה-IV ב-WEP הוא רק 24 ביט, ולכן לאחר 2^{24} הודעות (16 מיליון) מובטח שערך IV יחזור על עצמו.
- בנוסף: כרטיסי רשת 802.11 מאפסים את ה-IV בכל הפעלה מחדש ולאחר-מכן מגדילים ב-1 בכל צעד ולכן ערכי IV קטנים חוזרים בהתחלה של כל session.

מסקנה: למרות ש-WEP משתמש באלגוריתם קריפטוגרפי ידוע (RC4), השימוש הוא שגוי ולכן הסודיות לא מובטחת.

נקודת תורפה 3- התקפה על פרוטוקול אימות המשתמש:

- מתקיף שמאזין לנסיון התחברות של תחנה ניידת חוקית אל תחנת הבסיס יכול להשיג זוג (Challenge,Response).
- זה מאפשר לתוקף לגלות את המפתח המורחב $K_{IV} = Challenge \oplus Response$ ולהתחזות לתחנה ניידת ע"י שימוש באותו IV: בהינתן Challenge2 עונים עליו:

$$Response_2 = Challenge_2 \oplus K_{IV}$$.

נקודת תורפה 4- (חוסר) סודיות:

- אם יודעים את ערך ה-key stream שנוצר עבור ערך IV נתון, הרי שאפשר לפענח כל הודעה המוצפנת ע"י ערך ה-IV הזה.
- לכן אפשר להכין מילון המכיל: IV, K_{IV} .

נקודת תורפה 5- שלמות הודעות ב-WEP:

- לפני הצפנת ההודעה מחשבים CRC של ההודעה. לאחר מכן, מצפינים את ההודעה יחד עם ה-CRC.
- CRC הוא קוד ליניארי לגילוי שגיאות. ליניארי מקיים: $CRC(x \oplus y) = CRC(x) \oplus CRC(y)$.
- CRC נועד לגילוי שיגואת ולא להגנה מהתקפות מכוונות. מי שמתקיף ומשנה את ההודעה יכול גם לחשב CRC בעצמו ולשנות את הביט המתאים בהצפנה של ה-CRC.

©ענת עציון