אבטחת מערכות ויישומים ברשת, שיעור 3#

נושא היום: Web Confidentiality and Data Integrity

בהרצאה הקודמת דיברנו על firewall וההגנה שמספק על גישה בין שתי רשתות ובתוך רשת; דיברנו על מנגנון ה-NAT וה-packet filtering ותצורת ה-DMZ היושב בין הרשת הפנימית לחיצונית. כמו כן דיברנו על כך שה-firewall לא נותן הגנה ברמה האפליקטיבית.

מבוא לקריפטוגרפיה:

הקריפטוגרפיה מאפשרת הפיכת מידע קריא למידע לא קריא פרט למחזיק במפתח באמצעותו הוצפן המידע. סימונים: הטקסט הלא מוצפן הוא plaintext והמוצפן הוא cyphertext. הקריפטוגרפיה עוזרת למנוע את זליגת המידע לגורם שאינו מורשה, שכן גם אם מידע מוצפן מגיע לגורם לא מורשה, הוא אינו רלוונטי עבורו (כל עוד לא מחזיק במפתח).

מפתחות:

מפתח קריפטוגרפי הינו רצף אקראי של ביטים. אקראיותו חשובה לחוזק המפתח, כדי למנוע מהיריב לנחש את הסיסמא ביעילות. סטטיסטיקות שנעשו על סססמאות שהרוב בוחרים מראים שמרחב המפתחות הוא בסד"ג של מליון אפשרויות (שכן מבוססים על מחרוזות שאינן לגמרי אקראיות, מניפולציה על דברים קרובים לנותן הססמא), וזאת לעומת מרחב המפתחות האמיתי למפתחות בגודל 8 תוים אלפא-נומריים – 2^{48}.

חוזק הצפנה הוא כחוזק המפתח, ולכן רוצים שהמפתח יהיה ארוך ככל הניתן, שכן אורכו קובע את חוזקו. חוזק ההצפנה תלוי כמובן גם באלגוריתם ההצפנה. כיום מפתח בגודל 128 ביטים נחשב סטנדרטי וקשה לפיצוח בהצפנה סימטרית. בהצפנה אסימטרית המפתחות יהיו לרוב ב-1024 ביט ומעלה.

אלגוריתם הצפנה הוא שביר אם ניתן למצוא את המפתח המוצפן בפחות ממניה על מרחב המפתח. מפתח לא שביר – דורש סטטיסטית מניה מלאה על מרחב המפתחות השלם.

הצפנה – encryption: תהליך הצהפנה הוא לקיחת מידע גלוי ומפתח, הפעלת פונקציה מתמטית ויצירת סתר – cyphertext שאינו קריא אלא אם יש בידי את המפתח.

פענוח – decryption: הפונקציה ההופכית שבהינתן מפתח ו-cyphertext מניבה את ה-plaintext, ועבור כל מפתח שאינו נכון לא נותנת plaintext קרוב למקור. אם התנאי האחרון לא מתקיים, אזי פונקצית ההצפנה/פענוח אינה חזקה. נדרוש שאם המפתח שונה אפילו בביט אחד מהנכון, התוצאה תהיה אקראית לחלוטין ולא תרמז דבר על הטקסט המקורי. מטרות:

- הפיכת מידע ללא קריא, ולפיכך ללא חשש לזליגתו.
- מתן אינפורמציה על השולח ווריפקציה שהוא אכן השולח.

שתי משפחות ההצפנה:

הצפנה סימטרית: אותו מפתח משמש גם להצפנה וגם לפענוח. פונקציות ההצפנה והפענוח אינן זהות, לרוב אחת היא ההופכית המתמטית של השניה. הדרישה בהצפנה סימטרית היא ששני הצדדים (שולח ומקבל) יסכימו ביניהם על אותו מפתח סודי. כלומר עליהם לתאם מראש את המפתח.

דוגמאות: DES (64 ביט) ו-AES (128 ביט). ה-AES נחשב לחזק ומהיר, וסטנדרט להצפנה סימטרית כיום. ל-AES וריאנט שכיח של 128 ביט, וקיימים גם 192 ו-256 ביט.

ההצפנות הסימטריות מחולקות לשתי תתי-משפחות:

- Stream cipher: הכנסת מפתח ל-pseudo random generator המייצר סדרת ביטים ארוכה, כאשר ה-ciphertext הוא תוצאת ה-XOR PRG plaintext. אלגוריתם ההצפנה הוא למעשה אלגוריתם ה-PRG. הערה: OTP (one time pad) הוא הצפנה שלא ניתנת לפענוח.
- Block cipher: ההצפנה נעשית פר בלוקים של ביטים, לרוב חזקות של 2. המפתח, נניח בעל k ביטים, מוכנס עם כל בלוק לאלגוריתם ההצפנה ומתקבלים מכל בלוק גלוי בלוק מוסתר. כל ביט בסתר צריך להיות תלוי בפוקנציה לא לינארית (אך הופכית) של n ביטי הבלוק ו-k ביטי המפתח.

הצפנה אסימטרית: הצפנה המורכבת משני מפתחות – מפתח הצפנה ומפתח פענוח. מה שמוצפן באמצעות מפתח 1 יפעונח באמצעות מפתח 2, ולהיפך. אותו מפתח לא ישמש גם להצפנה וגם לפענוח.

בהצפנת מפתח ציבורי, מפתח אחד יהווה מפתח פרטי, ויישמר בידי המצפין, והמפתח השני יהווה מפתח ציבורי משותף לכלל. כל אחד יוכל להצפין באמצעות המפתח הציבורי ולשלוח למחזיק במפתח הפרטי, ורק הוא יוכל לפענח ולקרוא את ההודעה. שימוש נוסף הוא להזדהות השולח – חתימה דיגיטלית אסימטרית.

לסיכום:

- בהצפנה סימטרית אותו מפתח משמש גם להצפנה וגם לפענוח, והצפנה זו מהירה ויעילה ביחסית להצפנה האסימטרית.
- בהצפנה אסימטרית יש שני מפתחות הקשורים מתמטית, אחד פרטי ואחד ציבורי. הצפנה זו בטוחה יותר אך איטית יותר חישובית מאשר סימטרית.

מדוע להשתמש ב-public key cryptography:

ההצפנה הסימטרית היא פתרון מאובטח להעברת מידע ומפתחות דרך הרשת מבלי הצורך ששני הצדדים המשוחחים ייפגשו לצורך החלפת מפתחות.

Message digest and Hash Functions:

בהינתן רצף ביטים, נרצה ליצור עבורו מזהה חד ערכי כך שעבור כל רצף ביטים יהיה מזהה קצר וייחודי. נרצה שהמזהה יהיה תמיד בעל אורך נתון, ללא תלות באורך הטקסט, והייחודיות נדרשת מבחינה פרקטית (לאו דווקא מתמטית).

Hash function: פונקציה דטרמיניסטית המספקת את הנ"ל. ה-hash value נקרא ה-message digest. תהי H הפונקציה, h הפלט, M ההודעה, נאמר כי $H(M) = h$, ואחת הדרישות היא שבהינתן h יהיה בלתי אפשרי למעט מניה מלאה למצוא את M. ה-M היא ה-preimage של h. אם עבור M יש קלט אחר M' כך ש-$H(M) = H(M')$ אזי M' היא ה-second preimage של h. אם ישנן M,M' כאלה אזי זוג זה הוא collision ביחס ל-H.

פונקצית hash טובה תחשב כאשר הסיכוי ל-collision הוא $\frac{1}{\sqrt{2^n}}$ כאשר n הוא מספר הביטים ב-h (הפלט), וזה נובע מפרדוקס היומולדת. פונקציות ידועות לדוגמא: MD5, SHA1. דרישות מ-H:

- חישוב $H(M)$ יהיה יעיל.
- Preimage resistance: בהינתן h, יהיה קשה למצוא M שהוא המקור שלו.
- Second preimage resistance: בהינתן M יהיה קשה למצוא M' כך ש-$H(M) = H(M')$.
- Collision resistance: יהיה קשה למצוא זוג כלשהו של M,M' כאלה (בניגוד לדרישה הקודמת, שם נתונה M).

בסיס פונקציית hash היא פונקציית דחיסה של ביטים, כאשר העקרון הוא לקיחת רצף הביטים M ולחלקו לבלוקים (לרוב בגודל 512 ביטים). לוקחים initial value קבוע וידוע, ואז מפעילים את $f(k_0, m_0) = k_1$, $f(k_1 m_1) = k_2$ וכן הלאה, כאשר הפלט האחרון הוא תוצאת ה-hash.

וידוא integrity של מידע: רוצים לודא שהמידע שנשלח הוא זה שהתקבל. ניתן להוציא מ-data את ה-hash value שלה, ונשלח את ה--data,hash value. כעת בצד השני, כאשר H ידועה, יקח המקבל את ה-DATA ויפעיל עליו את H, וישווה את התוצאה למה שקיבל מהשולח. אם תתקבל תוצאה זהה אזי המידע לא שונה בדרך.

Digital Signature:

אופן החתימה: הפעלת H על DATA וקבלת h, ואת h מייצר המידע יצפין עם המפתח הפרטי שלו – והפלט הוא החתימה הדיגיטלית של מייצר המידע על המידע. כעת מקבל המידע (שאינו מוצפן, אן לא מדובר על סודיות המידע) יכול לקחת את המידע ולהפעיל עליו את H, ובאמצעות המפתח הציבורי יפענח את החתימה של השולח וישווה למה שחישב קודם – אם הערכים זהים המקבל יודע: א) המידע לא השתנה ב) השולח זוהה אכן כשולח שהוא טוען שהוא.

Digital Certificates and PKI:

כיצד נתפרת זהות של entity ל-public key? ה-PKI – public key infrastructure, המאפשר לזהות שמפתח ציבורי אכן שייך ל-entity – תהליך, שרת, אתר, אדם. ה-PKI מתבסס על הצפנה אסימטרית – מפתח פרטי הנשמר למייצר, ומפתח ציבורי. המפתח הציבורי הולך לגוף שלישי המהווה מאמת לזהות, שישים אישור על הקשר בין דורש השירות למפתח הציבורי. למשל, אדם בעל תעודת זהות – מנפיק תעודת הזהות, משרד הפנים במקרה זה, הוא הגוף השלישי הנותן גושפנקא לכך שתעודת הזהות היא מזהה עבור אותו אדם.

Certificates:

לאחר יצירת זוג המפתחות יש צורך בקישור המפתח הציבורי ל-entity, וזאת נעשה דרך צד שלישי המנפיק digital certificate, כאשר X.509 הוא פורמט לתעודה זו. התעודה כוללת שם, Public key, מנפיק התעודה וחתימה דיגיטלית המאמתת את התעודה (נחתם ע"י המנפיק). אם סומכים על ה-issuer אז תעודה זו מהווה אימות של הקשר בין ה-entity ל-public key שלו.

Certificate verification: כאשר browser מקבל certificate מ-website, המכיל למעשה את ה-public key של האתר, הוא בודק באמצעות החתימה הדיגיטלית כמתואר לעיל (עם hash). אבל, מהיכן ידוע ה-public key של ה-issuer? המודל תלוי בנקודה זו.

המוסמך לחתום על certificates הוא קבוצה מצומצמת של entities, והם נקראים CA – certificate authorities, ותפקידם לוודא את הזהות האמיתית של מבקש התעודה, ולאחר בדיקה של קשר אמיתי בין אתר לבעליו, הגוף יוצר SSL certificate וחותם עליו עם ה-public key של ה-CA

והחתימה של ה-CA. ה-self signed certificate מובנים בתוך ה-browsers עבור CA מוכרים כלל עולמיים והם הבסיס לכל התהליך. אגב, ניתן להגדיר ב-browser הוספת או חסימת CA certificates.

בעיה: ניתן לשתול certificate ב-browser; אבל אם האקר הגיע לנקודה זו הוא כבר יכול לבצע הרבה יותר.

X.509 v.2 certificate format:

- מספר גרסה של X.509
- Serial number: לזיהוי התעודה, למקרה שתהיה לא תקפה בעתיד, ניתן יהיה לזהות זאת לפי מזהה זה.
- Signature algorithm identifier
- Issuer
- Validity period: אחת הטכניקות להתמודדות עם הבעיה של תעודות שהופכות ללא רלוונטיות כי ה-Private key שלהן נחשף, הוא ע"י קביעת תאריך תפוגה לתעודה.
- Subject (X.500 name) – למי מונפקת התעודה.
- Subject public key info: algorithm identifier, public key value.
- Unique identifier
- חתימה דיגיטלית שמיוצרת ע"H ה-CA.

השינוי ב-v3: עדג אז היה חיבור רק בין ה-entity ל-public key, ובגרסה זו ניתן להוסיף אינפורמציה נוספת כ-extension, וכך ניתן לקבל מידע נוסף כמו תפקיד. זהו עיקר השינוי בגרסה זו לעומת הקודמת.

התפקיד המרכזי של ה-CA הוא אימות הקשר בין ה-entity לזהות שלו. ישנן סוגים שונים של תעודות בהתאם ל-level של הבדיקה שנעשתה על היחס בין הזהות למפתח הציבורי. כדי שנדע מהי רמת האמינות הגדירו EV certificates שהן תעודות ברמה גבוהה עבור מספר מצומצם של vendors המועמדים לניסיונות התחזות רבים, כמו למשל אתרי בנקים. התעודה במקרים אלו תוצג עם רקע ירוק, ואילו תעודות בעיתיות יוצגו עם רקע אדום.

כיצד נעשית וריפיקציה: כמו שתואר קודם, עם ה-hash.

נסתכל על סביבת ארגון בעל CA פנימי, הנפיק עבורו self signed certificate והפיץ אותו ברחבי הארגון. משתמש ניגש ל-CA עם מפתח ציבורי (את הפרטי שומר לעצמו) כדי לקבל תעודה חתומה ע"י certified administrator, וזאת לאחר שיצר הצלבה פיסית מול הדורש. אותו דורש יוכל להשתמש כעת בתעודה שניתנה לו – כל browser שיכיל את התעודה בתוך הארגון יוכל לאמת את זהות הדורש.

סיכום:

ה-CA מחבר בין entity ל-public key. הדורש צריך להזדהות מול ה-CA, וה-CA יוצר self signed certificate עבור אותו משתמש לאימות הקשר בינו ובין ה-public key שלו.

Salability:

נניח A יוצרת תעודה אצל CA כלשהו, נסמנו CA1, ו-B צריך אימות אודות אמינותו של CA1. לפיכך יש רמות נוספות, למשל CA2 שייתן certificate ל-CA1, וכך CA3 יתן תעודה ל-CA3 וכן הלאה. מבנה היררכי זה בנוי כך שבראשו נמצאים סט קטן יחסית של trusted CAs הנמצאים כ-default ב-browser.

Certificate Revocation: להשלים.

Distributions of CRLs:

- Polling: על ה-CAים להיות זמינים שכן כל CA בשרשרת צריך בדיקת ולידיות, ולכן עליהם להיות זמינים ולנהל רשימה עדכנית של תעודות. זאת ע"י דגימה של browsers ע"י ה-client. מודלים: אחת לכמה זמן נעשית דגימה, לעומת דגימות רבות. במודל בו נעשה polling אחת לפרק זמן ישנה בעיית validity בדלתא בין שתי הדגימות.
- Push: מנגנון דחיפה של CA לתוך ה-browser. מנגנון זה בעייתי שכן תופס רוחב פס מיותר.

חזרה לחתימות דיגיטליות:

חתימות דיגיטליות משמשות כעדות בטרנזאקציות, כך שהחותם אינו יכול להתכחש לפעולות שמבצע ועליהן חותם – שכן הוא היחיד שיכול לחתום בחתימתו. פרוטוקולים כאלו קבילים משפטית.

Insecure cryptographic storage:

כיום האינטרנט מלא במידע רגיש, אך מצד שני ישנן טכניקות הצפנה רבות בעלות אמינות גבוהה וכיום רוב האלגוריתמים שהוזכרו זמינים דרך ספריות קריפטוגרפיות שהן חלק אינטגרלי מכל סביבת פיתוח: CryptoAPI.

<u>בעיות קלאסיות בשימוש לא תקין ב-cryptoAPI</u>:

- לא תמיד כל המידע הנדרש להצפנה אכן מוצפן – בעיה ב-policy, לא ב-mechanism.
- שמירה בצורה לא סודית את המפתחות של מערכת ההצפנה.
- שמירה של אלמנטים סודיים כמו מפתחות וסיסמאות בזיכרון או בקוד האפליקציה. לרוב נובע בגלל הקושי למצוא מקום נכון לאחסן בו את המפתח.
- אי הגנה על קבצי dump לאחר crash – תוקף מיומן יוכל לגרום לקריסה, ולנסות לשחזר מתוך ה-dump את המידע הסודי.