אבטחת מערכות ויישומים ברשת - שיעור 5#

נושא היום: Introduction to Web Application Security:

תזכורת: SSL Summary:

אספקטים נוספים בפרוטוקול:

ה-SSL נותן תווך תקשורתי מאובטח מעל TCP/IP. ה-SSL חייב לעבוד מעל TCP שנותן guarinty delivery, שליחה מחודשת של פקטות שלא הגיעו, סידור פקטות. בהצפנה ישנה חשיבות עליונה להגעת כל הודעה as-is, שכן אם נעלמת אפילו פקטה אחת ההצפנה לא תעבוד. לכן ה-SSL **לא** עובד מעל פרוטוקול כמו UDP. כדי להגן על מידע מעל UDP צריך להשתמש בפרוטוקולים אחרים.

ה-SSL יושב מעל שכבה 4 (מעל ה-TCP) תחת השכבה האפליקטיבית. הוא לא משרת רק את פרוטוקול ה-HTTP (הפוקוס שלנו בקורס). הוא הפרוטוקול הנפוץ להגנה על תעבורת WEB ולכן חלק אינטגרלי בכל דפדפן כיום, ושרתי WEB. ב-SSL באים לידי ביטוי מרבית טכנולוגיות ההצפנה: הצפנה סימטרית – העברת ה-DATA בין השולח למקבל, והיא יעילה יותר מהצפנה אסימטרית. הצפנה אסימטרית משמשת בפרוטוקול להעברת מפתח סימטרי ראשונית. נעשה כמו כן שימוש בחתימה דיגיטלית להבטחת DATA Integrity – נעשה שימוש בחתימה דיגיטלית סימטרית (ה-MAC שהוא חלק מה- SSL Record Protocol) שהיא יעילה חישובית, ונעשה שימוש בחתימה דיגיטלית אסימטרית, שכן זיהוי השרת כלפי הלקוח (ה-web server מול ה-browser) נעשה באמצעות certificate, החתום באופן אסימטרי ע"י ה-CA שהנפיק אותו. נעשה שילוב של כמה טכנולוגיות. ישנם סטים של אלגוריתמים לכל אחת מהטכנולוגיות הנ"ל, ולכן בעת בניית SSL session / connection נעשה handshake בו מוחלט על האלגוריתמים בהם ישתמשו.

תהליך ה-handshake ברמה הכללית:

- הלקוח שולח בקשת https – בקשה להעביר את בקשת ה-http מעל SSL. אז ה-browser בונה SSL session בינו ובין ה-web server, ולכן מבצע client hello ל-SSL Daemon על ה-web server. שם מציע סט אלגוריתמים ומודיע על הפרוטוקול בו עובד.
- השרת בוחר את סט האלגוריתמים (החזק מכולם), ואז השרת מעביר ללקוח את ה-certificate שלו ומוכיח שהוא בעל התעודה ע"י הצפנה של אינפורמציה מוסכמת ע"י ה-private key שלו, שמתפענח באמצעות המפתח הציבורי ע"י הלקוח – וכך מוודא ומזהה את ה-entity אליו פנה. כך השרת מוכיח ללקוח שהוא אכן הגיע אל אותו שרת שאליו התכוון לפנות.
- Client authentication: דילג.
- הלקוח מגריל מידע אקראי ומצפין באמצעות ה-public key של השרת, ולכן יכול לשלוח אותו ע"ג הערוץ למרות שטרם מאובטח (שכן רק השרת יוכל לפענח זאת). השרת מפענח את ההודעה המוצפנת ומוציא מתוכה את ה-pre master secret – כך נוצר שיתוף המפתח הסודי בין השרת ללקוח, ושני הצדדים מחשבים מתוכו את ה-master ואת סט המפתחות איתו טוענים את אלגוריתמי ההצפנה איתם יעבדו מעל SSL Recored layers.

SSL limitations:

- חייב לרוץ מעל TCP.
- אינו תומך במניעת התכחשות בגלל שימוש בהצפנה סימטרית – למרות שהבקשה עוברת מעל SSL, אז אכן מובטחת סודיות ושלמות, אך ה- web server לא יוכל למנוע מהדפדפן או המשתמש מאחוריו להתכחש לעובדה שהוא שלח זאת. זאת כיוון שבשביל יכולת מניעת התכחשות, דורשים שהמפתח הסודי יהיה ידוע אך ורק לגורם אחד – ואז אם מתפענח אז זה מעיד בהכרח על מקור יחיד. זאת כאשר עובדים מעל הצפנה אסימטרית, אך כיוון שאת המידע בפועל ב-SSL מעבירים בהצפנה סימטרית, אז ה-pre master secret / master secret ידוע לשני הצדדים, ומול צד שלישי צד אחד לא יוכל להוכיח שהודעה כלשהי התקבלה מצד שני. אם נרצה זאת נזדקק למימוש ברמה האפליקטיבית.
- ה-SSL מבוסס על אלגוריתמי הצפנה וחתימה, וחוזקו לכל היותר כחוזק כל אחד מהמרכיבים הללו. לכן, חולשה באחד מהמרכיבים הללו יפגע בחוזקו של ה-SSL.
- לא מגן על המידע מפני צד שלישי מעבר למעבר בתווך בין צד אחד לשני. מרגע שהגיע לצד השני הוא לא מוגן ע"י ה-SSL, ולכן נזדקק להגנה בדרכים אחרות.

SSL Performance Degradation:

- עלויות השימוש ב-SSL מבחינת ביצועים תהיה בעיקר בצד השרת, שכן מטפל בו"ז באלפי requests. ה-overhead היחסית קטן ב-browser מתורגם לסדרי גודל יותר בשרת. הירידה בביצועים יכולה לרדת אף ב-50% בשרת. העלות החישובית הכבדה היא לא בהצפנת המידע, אלא בפתיחת SSL session. לכן קונפיגורציה של הרבה connections מעל session יחיד תתן ביצועים טובים יותר מאשר session לכל connection
- Load blanacer – מחלק בקשות בין שרתים לחלוקת העומס. אותו load balancer יוכל לבצע את פענוח והצפנת ה-SSL, וכך גם מקטינים את הסיכון בכך שגורם אחד מחזיק במידע הפרטי, לעומת כל השרתים.

SSL to secure e-Commerce Applications:

<u>בחנות וירטואלית</u>:

רוב זמן השהיה של משתמש בחנות, המידע שמבקש אינו מידע סודי. אין צורך לבזבז משאבים על SSL. ההצפנה חשובה כאשר המשתמש רוצה לבצע רכישה של מוצר וצריך לתת את פרטי אמצעי התשלום החסויים. עבור הטרנסאקציה הזו בלבד נרצה להשתמש ב-https. אבל, זה לא מבטיח דבר לגבי גורל הפרטים החסויים בעת הגעה לשרתי החנות. כיום ישנו סטנדרט PCI-DSS וזהו סטנדרט אבטחה שחברות אשראי בינלאומיות גיבשו הכולל תקנים לחנויות המעוניינות לעבוד עם חברות כרטיסי האשראי.

Server authentication: יכולת ההוכחה של השרת שהוא אכן השרת אליו התכוון הלקוח להגיע. מנגנון זה מונע phishing – התחזות אתרים לאתרים לגיטימיים, על מנת להוציא ממשתמשים פרטים חסויים. ה-SSL נותן אפשרות זו כי הדפדפן אמור לבצע בדיקות אלו (בדיקת התעודה). הבעיה המרכזית היא שדפדפנים לא תמיד מיישמים בצורה מלאה את כל הבדיקות, ובעיקר: כאשר הדפדפנים מוצאים בעיה ב-certificate, הם מתריעים למשתמש שיש בעיה עם התעודה. רוב המשתמשים במקרה זה יבצעו אישור מבלי להתייחס לבעיה. רוב המשתמשים לא מבינים את הבעיות הללו. בתקופה האחרונה הגדירו EV-certificate, ואלו תעודות עליהם נערכות בדיקות הרבה יותר מחמירות, ושורת ה-URL צבועה בירוק או אדום או משהו (פעם היה ציור של מנעול שבור) – כך ההצגה למשתמש יותר מושכת לעין ואליה יותר משתמשים ייתיחסו.

<u>באפליקציה בנקאית</u>:

כאן נרצה להגן על התקשורת לכל אורך ההתקשרות, החל מהכנסת המשתמש והססמא וכלה בטרנסאקציות של המשתמש והחזרת התשובות של השרת. לכל אורך ה-session האפליקטיבי נעשה שימוש ב-SSL. הוא כאמור מספק confidentiality, data integrity. ה-SSL מכיל בתוכו את האופציה של client authentication. בפועל הזדהות המשתמשים מול אפליקציות בנקאיות אינה ברמה האידיאלית (לא מזדהים עם תעודות). בסופו של דבר שימוש ב-client authentication ב-SSL מינורי, ורוב הזיהוי של המשתמשים נעשה ברמה האפליקטיבית.

Insecure Communications:

במהלך הקורס נדבר רבות על פגיעויות שקיימות במערכות web. ישנו ארגון בשם OWASP שקם בשנת ~2003, שקידם בצורה משמעותית את המודעות לנושא זה, ומפרסם אחת ל-3 שנים דירוג של ה-vulnerabilities top 10. נושא ה-insecure com- נמצא ברשימה זו.

Securing Network Communications:

הנחת העבודה בעבודה מעל הרשת היא שישנם גורמים שעלולים לפגוע או לחדור למידע שלנו. כמו כן ה-web server לרוב הוא רק ה-front end של צד השרת. יש להתחשב גם בתקשורת מעבר לזו שבין הדפדפן לשרת. בניתוח סיכוני מערכת, יש לבחון את אופן התקשורת של ה-web server מול ה-database server, והאם התקשורת ביניהם מאובטחת כראוי. בתנאים מסויימים, למשל בהעברת מידע של כרטיסי אשראי, הדרישה לתוך הרשת הארגונית היא העברת מידע מוצפנת. כלומר המידע צריך להיות מוגן גם בנבכי השרתים של צד הספק, ולא רק בין הלקוח לספק. הדרישות מורכבות גם מדרישות רגולטוריות - כמו הרגולציה של חברות כרטיסי האשראי.

<u>EncryptionMUST be used</u>:

פעמים רבות מגנים על המידע בשלב האותנטיקציה, שם מוגן באמצעות SSL. לאחר שלב ה-login, פעמים רבות האפליקציה לא אוכפת את השימוש ב-SSL, בעיקר משיקולי ביצועים. כך יתכן שבמהלך פעולות האפליקציה יתכן שפרטי מידע חסויים ופרטיים יעברו בצורה לא מאובטחת. אבל, פעמים רבות גם אם לא מועבר מידע סודי, כיוון שבעולם ה-web כל request הוא עצמאי, אזי בכל אחת מתבצעת אותנטיקציה מחדש (stateless). כל request כולל אלמנט אותנטיקציה, ולכן בדיוק כמו שצריך להגן על דף ה-login, צריך להגן על כל request נוסף באותו session, ואם לא יגנו על כך, מאזין יוכל לקחת את האינפורמציה הזו ולהמשיך את התקשורת תחת התחזות למשתמש האמיתי.

Insufficient Transport Layer Protection: שקף שלקוח מדו"ח כלשהו.

Insecure Communications: How to Protect Against: ...

ה-SSL לא מגן על השרת מפני משתמש עויין, הוא נותן הגנה רק בפני צד שלישי שמנסה להאזין לתעבורה. משתמש עויין יכול לנצל vulnerabilities של ה-web application כדי לפגוע בו. מדוע? ה-SSL מעביר מידע בצורה מאובטחת, אך תוכן המידע לא נבדק ולא מעניין ברמה הזו. לכן אם נשלחת בקשה עם תוכן עויין מעל SSL, לא תהיה בעיה להעבירו. אם השרת לא ידע להתגונן בפני זאת ברמה האפליקטיבית, האפליקציה תפגע.

דיברנו על deep packet inspection ב-firewalls המנסה לזהות כוונות זדוניות בתוך פקטות. אבל, כאשר המידע מועבר מוצפן ב-tunnel דרך ה-firewall, אז מערכות זיהוי תכנים עויינים אינן רלוונטיות, ולא ניתן לאתר תכנים עויינים במידע מוצפן. כך אותם intrudion detection לא יעיל אל מול כך. כיום, כפתרון לכך, ישנם firewalls ברמה האפליקטיבית שנמצאת לרוב על ה-load balancers או application delivery controllers (ADC) (נושא זה יילמד בהמשך).

פגיעויות הקוד האפליקטיבי הן פגיעויות שניתן לנצל על מנת לתקוף את התשתית עליה רץ הקוד האפליקטיבי, או לנצל את היכולת לפנות למערכות אחרות ברשת הארגונית ולהעזר בו על מנת לתקוף אותם. פגיעויות אלו בקוד האפליקטיבי חושפות גם את התשתית וגם המערכות הארגוניות איתם מסוגל לתקשר.

Insecure (Cryptographic) Storage:

הצפנות משמשות לשמירה על confidentiality של המידע. כמו כן, יש צורך גם לשמור אותו בצורה מאובטחת על ה-web server / database server. אחת הדרכים לכך היא להצפין אותו. כעת נדבר על פגיעויות בתחום זה:

Protecting Data Confidentiality is required:

כיום מבחינת PCI-DSS, שמירת מידע על כרטיסי אשראי חייב להיות מוצפן, ונעשה שימוש נרחב יותר ויותר ב-database encryption. כיום ברוב ה-frameworks המקובלים ישנה תשתית של crypto API, כלומר לא נדרש לפתח עמאית את קוד ההצפנה. אבל רוב המפתחים לא מבינים את יסודות ההצפנה, ולכן נבחרים הרבה אלגוריתמי הצפנה לא טובים, או טובים שבהם נעשה שימוש בצורה שגויה.

האתגר בפני מפתחים הוא כיצד לשמור או להצפין נכון מידע עליו מדרשים להגן. בעיות:

- אי הצפנת מידע קריטי או רגיש: פעמים רבות לא מזוהה מידע כזה כמידע שיש להגן עליו.
- אחסון מפתחות וססמאות בצורה לא מאובטחת: המקום בו יוחזקו המפתחות קריטי ולא טרוויאלי. הבנה מעמיקה של הארכיטקטורה נדרשת, כדי לאתר את המקום האופטימלי לאחסון.
- שמירה לא מאובטחת של מפתחות בזיכרון: memory dump עלול לחשוף מהזיכרון לדיסק מידע סודי. תוקף שמשכפל את הדיסק (למרות שמלכתחילה לא אוחבן עליו המפתח), יכול להשיג את המידע הסודי. פתרון: החזקה בזכרון לפרק זמן מינימלי שנדרש – ומיד לאחר מכן למחוק אותו.
- Hard-coding keys: החזקת executable על השרת, ללא source – כיום ישנם כלים המסוגלים לנתח קבצי הרצה ולהוציא מהם מפתחות. לשים מפתחות בתוך קוד הוא עקרונית לא נכונה, כיוון שהחלפת מפתחות תדרוש החלפת קוד, והיא פחות טרוויאלית מאשר להחליף רק מפתחות.

Insecure Cryptographic Usage:

- מקורות לא טובים לראנדומיות בעת בחירת מפתחות: למשל rand ב-C היא פרדיקטיבית, ולא קריפטוגרפית. בקריפטוגרפיות דורשים אקראיות סטטיסטית ואי יכולת חיזוי בין ערך אחד לבא אחריו. פתרון: לקחת מספיק מידע אקראי ולהעבירו דרך פונקציה חד כיוונית, ניתן לקבל תכונות יחסיות אקראיות.
- בחירת אלגוריתמים שאינם בטוחים יותר: ישנם אלגוריתמים שבעבר נחשבו בטוחים וכיום כבר אינם, למשל MD-5 או RC4. קבוצת חוקרים הראתה כיצד שימוש ב-MD-5 ב-digest של חתימה דיגיטלית אינה בטוחה. הוכיחו כי לאגל' זה יש חולשה בתחום collision resistance.
- שימוש באלגוריתם שפותח עצמאית: הנחת העבודה צריכה להיות שהאלגוריתם ידלוף, ואז המצב יהיה כנראה גרוע יותר מה-APIs המוכרים. זאת כיוון שפיתוח אלג' הצפנה קריפטוגרפי דורש הרבה זמן ומשאבים לפיתוח נכון של כזה.
- ביצוע backup למערכות: אם ה-backup אינו מוגן, הגישה למידע שאמור להיות חסוי היא פשוטה.
- הצפנת המידע נייח חשוב, אך לא עונה על כל האיומים: חשוב להגן על המידע גם on transit וגם on rest אך ישנן פגיעויות נוספות.

Insecure Cryptographoic Storage: עוד קטע מהדו"ח:

יתכנו משתמשים פנימיים שירצו לגשת למידע פנימי רגיש, כמו גם internal administrators.

How to Protect Against:

- יש לנתח על איזו אינפורמציה צריך להגן: הגישה היא למזער את כמות האינפורמציה הרגישה הנשמרת במערכת. דמידע רגיש שלא חייבים, רצוי לא לשמור כדי להקטין את הסיכונים ולהפוך את המערכת לפחות אטרקטיבית כלפי תוקף.
- לאחר שהמידע עליו צריך לשמור מוצפן הוגדר, יש לבדוק שאכן כך בפועל. צריך לבדוק שהמידע מוגן לאורך כל מסלול החיים שלו. צריך לוודא שאין נקודות פגיעות בהן המידע לא מאובטח.

Best practices for Crypto Usage:

- כאשר ניתן, יש לשמור hash-value במקום encrypted value. הסיסמאות יוצפנו באופן חד כיווני, כך שלא ניתן לחזור אחורה (בדיקת סמסא היא ע"י חישוב hash עליה והשוואה).
- יש להשתמש באלגוריתמים מוכרים וידועים כבטוחים קריפטוגרפית, למשל SHA-256.
- חשוב לבחור ספריות קריפטוגרפיות שהן public domain, כיוון שככל שיש יותר משתמשים באותה ספריה קריפטוגרפית, הסיכון שיהיה בה פגיעות לא ידועה יותר קטן. ככל שיש יותר אנשים, הסיכוי שפגיעות תתגלה, תפורסם ותתוקן גבוה (למשל openSSL).
- חשוב לעקוב אחר עדכניות האלגוריתמים בהם משתמשים, כדי להיות מכוסים מבחינת Patches ועדכוני תוכנה למניעת כל הפגיעויות הידועות.

Securing secrets in Memory:

- ישנה סכנה של חשיפה מהזיכרון בעיקר כאשר אפליקציה קורשת ויש dump לדיסק. סיכון נוסף הוא שחרור mem segment והשארת מידע סודי בו (העלול להגיע לקטע קוד אחר באפליקציה אחרת). לכן כאשר כותבים סוד לזיכרון, יש להשאירו בזיכרון לפרק הזמן הקצר ביותר הנדרש, ובסיום יש לדאוג לדרוס אותו ע"י תוכן כלשהו לפני שחרור הזיכרון (לא לסמוך על GC).
- כאשר מצפינים ושולחים, יתכן מצב בו תהליך כותב לבפר, תהליך שני מצפין את המידע ותהליך שלישי שולח. יתכן מצב בו התהליך המצפין לא עומד בקצב של השולח. אז יתכן שכתוצאה מ-race condition התהליך שמצפין לא יספיק וישלח מידע לא מוצפן. לשם כך יש להשתמש בבפר נוסף בין המצפין לשולח, ואז מובטח שרק מידע מוצפן ישלח, כי כל עוד לא מוצפן לא מגיע לבפר ממנו נשלח המידע.

דוגמא: חנות ויטואלית: לא יסוכם, להסתכל המצגת.