אבטחת מערכות ויישומים ברשת - שיעור 1#

הקדמה:

- המצגות הן חומר מספיק לקורס (לא יותר).
- אתר הקורס יעודכן בהמשך.
- תרגילי הבית: לא תרגילי תכנות, טרם נקבע כמה.

נושא היום: מבוא ומושגים חשובים להמשך הקורס:

הקדמה ל-Web Applications:

בעולם הארגון הפנימי, כל מערכות ה-IT עברו webification – הפיכת מערכות למערכות על ה-web. כיום .web app היא תשתית נרחבת של רוב מערכות מידע בארגונים.

ארכיטקטורת .Web App:

- משתמשים העובדים עם איזשהו ממשק משתמש, יכול להיות browser למשל. לא רק PC – גם mobiles מכילים דפדפנים. זוהי ה- presentation layer – מציג אינפורמציה למשתמש ומקבל ממנו פקודות. הדפדפן עובד מהצד השני של ה-representation layer מול web server. כלומר, שכבה זו רצה חלקית מול המשתמש וחלקית מול השרת.
- שכבה שניה: ה-business logic – תרגום כל הקלטים לפעולות על ה-data.
- שכבת המידע: ה-data tier, repository. ישנם סוגים שונים של מחסני מידע.

.legacy app – לרוב מערכות mainfraim ישנות המחזיקות הרבה מידע ארגוני שרוצים לתת למשתמש ממשק נוח ועדכני ואז ה-business logic יודע לתרגם את פקודות המשתמש לפקודות מתאימות ל-legacy.

הדפדפן מריץ קוד שיורד אל ה-web server. הדפדפן התחיל כמעיין terminal, וכיום הוא מסוגל להריץ תוכנה והוא מעיין מיני-מערכת הפעלה, כיוון שבדפדפן רצים המון תוכניות (ב-JavaScript או Flash). תוכניות / דפי ה-html יורדים אליו מה-web server. ה-web server אחראי על ה-rep. layer בתיאום מול הדפדפן ומקבל את הקלט שהמשתמש מקיש ומעביר ל-front systems.

ה-front systems מבצעת בדיקות ראשוניות על הקלט – תקינות, איזה business logic יש להפעיל (טכנולוגיות ב-asp, jsp וכו'). מהצד השני יש את ה-backend systems – החלק היותר כבד של התוכנה (SAP, ORACLE). אנחנו נתמקד יותר בחלק הקדמי ומטה.

מושגים:

HTTP: hypertext transfer protocol – הפרוטוקול הבסיסי להעברת מידע באינטרנט. אבחנה: האינטרנט היא רשת המחשבים העולמית, ה-web היא אפליקציה הרצה עליו. העולם ה-Webי הקלאסי עובד עם פרוטוקול זה. התכונה המרכזית שחשובה לנו היא שפרוטוקול זה הוא stateless – לא מנהל זיכרון, שולח request ומקבל response שאלו היחידות האטומיות שלו (בעיקרון). עיקרון זה חשוב בעולם ה-web כיוון שבהגדרה עוברים משרת לשרת, ולכן ל-state אין ערך. בעולם ה-web שולח ה-requests הוא הדפדפן אל ה-web server, והוא מחזיר לדפדפן response. Documents: ה-client (לרוב דפדפן) מבקש doc מהשרת והוא עונה לו. זו רמת פרוטוקול התקשורת ב-layer 7 במודל השכבות.

HTML: שפה המתארת לדפדפן כיצד להציג את המידע. זו שפה דומה ל-XML היודעת לשלב בתוכנה אלמנטים של קוד, גרפיקה, וידאו וכו'. הדפדפן מקבל מסמך HTMLי ואחראי להציג אותו למשתמש, ומצד שני לקבל דרכו קלט מהמשתמש וליצור request ל-web server המתאים (לאו דווקא זה שממנו קיבל את המסמך).

JavaScript: שפה לכל דבר, התחילה במקור כדי לאפשר ויזואליזציה מורכבת יותר. בפועל שפה זו הפכה לשפת תוכנה מלאה, השפה המובילה ב-web ברמת ה-presentation layer (המוצאה ע"י Netscape).

Form: בתוך דפי HTML, בפרט שמדברים על אפליקציה ולא דפים סטטיים בלבד, נרצה לתת למשתמש לבקש בקשה ולא רק להקליק על לינק (זו בקשה פשוטה ביותר). אופן קבלת הקלט מהמשתמש היא ה-form. המשתמש מזין קלטים דרך אלמנט הטופס, ותפקיד הדפדפן הוא לקחת את הקלטים וליצור מהם request מתאים. הקלט מועבר דרך query-string או POST-data, וזה מה שנשלח.

ה-server מקבל את ה-request, יבצע עליו Parsing, יוציא מתוכו את קלט המשתמש, יבצע interpretation כדי לפרש את משמעות הקלט והבנת ה-business logic, יבצע את הפעולות שצריך, יבנה דף תשובה ויחזיר אותו. במצגת שקופית HTTP Request/Response ממחיש זאת ויזואלית.

מומלץ לקרוא ולהרחיב עצמאית על HTTP, HTTP-headers (מבנה ה-HTTP), מושג ה-Cookies, HTML, JavaScript (הבנת סקריפטים בשפה זו), ארכיטקטורת ו-infrastructure של web application.

Cookies: בתרגום אפליקציה לעולם ה-web, ישנה שבירה של אלמנטים אפליקטיביים כגון סדר פעולות או זיהוי משתמש. הדרך בה ה-web מתאים לעולם האפליקציות בו יש session של משתמש מול האפליקציה היא Cookies.

מושגים באבטחת מידע ביישומי הרשת:

אנו רוצים לשמור על המידע:

- סודיות המידע – שיהיה נגיש רק למי שמורשה לגשת אליו.
- Integrity של המידע – שלמות המידע, המידע שנשלח מצד אחד הוא זה שהתקבל בצד השני.
- Authenticity – מי יצר את המידע ועומד מאחוריו.
- Availability – זמינות המידע. נגישות המידע מן הסתם הכרחית להתנהלות תקינה.

Vulnerability: חולשה במערכת או באפליקציה המאפשרת, לפחות תיאורתית, ניצול וייצור התקפה על אותה מערכת או asset.

Attack: האופן בו מנצלים את הפגיעות על מנת לתקוף את המערכת, ניצול החולשה.

דוגמאות:

- סיסמא חלשה: סיסמא שקל לנחש אותה, או סיסמת default שלא השתנתה. זוהי חולשה כיוון שניתן לבצע עליה התקפה שבאופן קל יחסית תמצא את הסיסמא. למשל: dictionary attack – התקפה לדוגמא על סיסמא חלשה.
- באגים בתוכנה: למשל, העתקה של נתונים למקום בו דורסים מקום בזיכרון. החולשה: לא נבדק שיש מספיק מקום, חוסר הבדיקה. ההתקפה: התקפות buffer-overflow.
- קלטים מהמשתמש שלא נבדקים: חולשה טרוויאלית ומסוכנת. העובדה שקלט לא נבדק ע"י האפליקציה פותחת פתח להתקפות רבות.
- מערכת הפעלה עם פורטים פתוחים שלא צריכים. כאשר יש פורטים פתוחים, שלרוב לא יודעים שהם פתוחים ואין עליהם לפיכך הגנה, הם פתוחים להתקפה.
- קונפיגורציה לא נכונה: למשל הגדרת הרשאות לא נכונות. ההתקפה: ניצול העובדה על ההרשאות כדי לבצע התקפה.
- Trap-doors: מפתחים משאירים "דלת אחורית" כדי שיוכלו לטפל במערכת תוך הנחה שגויה שרק הם מודעים אליה ואף אחר לא ימצא זאת (הנחה שגויה). למשל: תעבורה לא מוצפנת. ההתקפה: האזנה לאותה תעבורה.

דוגמאות התקפות:

- Dictionary attack – כמצויין לעיל.
- Worms
- Trojan horses: דוגמא לפגיעות הקשה ביותר – הפגיעות האנושית.
- Phishing: גם כאן הפגיעות היא חולשה אנושית.
- הקלטת אינפורמציה לא מוצפנת.
- Denial of service: לגרום למידע ולאתר להיות לא זמינים.

הערות כלליות על הקורס:

- המבחן יהיה כנראה אמריקאי ולא מכשיל (!)
- שבוע הבא (10-2-28) פורים, לא יהיה שיעור.

Threat:

ההתקפה היא האמצעי, והיא מגדירה threat. למשל, אם סיסמא היא חלשה, וניתן לתקוף אותה, האיום הוא שמי שמשיג את הסמא יוכל להתחזות. או למשל, אם מידע לא מוצפן, וניתן להאזין לו, ה-threat הוא שהמידע יחשף.

כאשר האיום נעשה מוחשי, מגיעים למצב security incident / breach – כאשר ההתקפה ממומשת.

השלכות ה-security incident:

- Data corruption - פגיעה בנגישות המידע.
- Data disclosure – חשיפת מידע, פגיעה באבטחה.
- Loss of authentication – תוקף שיכול להתחזות ל-root למשל.
- Defacement: שינוי חזות אתר, למשל העלאת עמוד מחתרתי על גבי אתר קיים כלשהו.
- תקיפת אתר לצורך שימוש בו לצורכי תקיפה: למשל, תקיפת אתר והכנסת קוד עוין לתוכו, מה שיגרום למשתמשים באתר זה להוריד למחשבם קוד עוין.

Business risks:

כתוצאה מ-vulnerabilities במערכות, נגרמים נזקים אדירים לאמון משתמשים במערכת, וההשלכות העיסקיות יכולות להיות חמורות ביותר.

דוגמאות לבעיות ברמה האפליקטיבית:

- שינוי ערך ב-form המוגדר כ-hidden, למשל עלות מוצרים באינטרנט. לפעמים דבר זה יתגלה במערכות ה-backend ולפעמים לא.
- Parameter tempring: id של session המוקצה עבור משתמש מוצג בשורת האתר. החלפתו בכוכבית תתפרש מאחור כקריאה לכל הנתונים – נתוני כל המשתמשים.
- SQL injection: הכנסת פרטים תוך ביצוע מניפולציות על SQL queries כדי להשיג מידע מוצפן.

אלו דוגמאות ל-coding vulnerabilities. ישנן גם logics-vulnerabilities: אמצעים מקובלים למניעת brute-force enumeration: הגבלת מספר האפשרויות לנסות להיכנס. דוגמא לשימוש: אדם ששם הצעה לקניה ב-auction, יכול לנסות להיכנס עם ססמא שגויה ל-auction עם ה-users של כל מי שעומד מולו – וכך למנוע מאחרים להציע מולו כי כולם יהיו חסומים – בעיית לוגיקה, לא בעיית קוד. פתרון: לא להציג את שם המשתמש – הצגת nickname אקראי שאינו ה-user name.

מושגים נוספים בעולם אבטחת המידע:

Security policy: מגדיר על מה רוצים להגן ומפני מה. תמיד עובדים עם משאבים מוגבלים, ולכן יש להגדיר על מה חשוב להגן, ומפני אילו איומים. הפתרון צריך להיות תפור על האיום – משתמשים פנימיים, חיצוניים וכו' – כל איום דורש פתרון שונה. על אילו information assets באים להגן. הגדרת ה-what.

לרוב נבנה מתוך ניתוח סיכונים של המערכת, דירוג הסיכונים כדי לאתר את הסיכונים המרכזיים והקצאת המשאבים להגנה. לא ניגע בקורס זה בניתוח סיכונים.

Security Mechanism: הגדרת המנגנונים המשמשים להגנה – הגדרת ה-how. בקורס זה נבין את עולם ההתקפות ופגיעויות, וה- security mechanisms הבאים למנוע אותן.

Integrity attack:

שלמות המידע; נניח מעבירים מידע מ-A ל-B וישנו תוקף באמצע המיירט את ההודעה, משנה אותה ושולח אותה הלאה. צד B חושב שקיבל תוכן מ-A, אך למעשה קיבל מידע שונה. המטרה כאן: למנוע ממישהו שאינו מורשה לשנות מידע. דוגמא ממשית: חברה משחררת מידע לבורסה, מידע זה משתנה ע"י גורם חיצוני עוין, יכול להיגרם נזק כלכלי כבד.

מנגנונים למניעה:

- חתימות דיגיטליות. יעילות על העברת מידע ברשת וגם על קבצים – לדעת שלא שונו.
- Access control – יעיל רק למידע סטטי, לא למידע שעובר ברשת. למידע שעובר ברשת יש צורך בשימוש בחתימות דיגיטליות.

פעמים רבות רוצים להגן על מידע ביותר מאמצעי אחד, הגנה בשכבות. שימוש בשני המנגנונים לעיל יכולים למנוע ממי שאינו מורשה לשנות את המידע, ובנוסף החתימה הדיגיטלית יכולה לספק מידע על מי שינה את המידע ומתי. שתי רמות מידע אלו יחד מאפשרים גם מניעה וגם מעקב אחר שינויים במידע.

Eavesdropping – message interception:

גורם עוין מאזין למידע העובר מ-A ל-B. יכול להיות מידע סודי, קבצים, וכו'. בכדי להגן על סודיות המידע רוצים להגיע למצב שבו מי שאינו מורשה למידע לא יוכל לקרוא אותו. בניגוד ל-Integrity, כאשר מגלים שמישהו טיפל בהודעה, ניתן לבקשה שוב. כאן המצב אינו כזה.

מנגנונים למניעה:

- Read access: לתת הרשאות קריאה רק למורשים לכך.
- Encryption: הפיכת המידע לרצף אקראי של ביטים שלמי שאין את המפתח לא יוכל לקרוא את המידע.

ההצפנה מתאימה גם למידע בתנועה וגם למידע נייח. המנגנון הראשון מתאים למידע נייח. במידע נייד נרצה כמובן לאפשר הגנה בשכבות (defense in depth) ע"י החלת שני המנגנונים.

Privacy: רוצים להגן על מידע פרטי של אדם, לרוב מידע אישי של אנשים.

Anonymity: ניתוק הקשר בין מבצע הפעולה לפעולה עצמה, למשל קישור בין מצביע להצבעה.

Attack on Availability:

זמינות המידע או המערכת. פגיעה בסיסית: פגיעה בחומרה, בתשתית התקשורת; ניתן גם ברמה הלוגית – פגיעה בקבצי הקונפיגורציה. מערכת יכולה לאתר שינויים אלו ולא להעלות את המערכת. ניתן לפגוע ב-packets ברשת, מנגון ה-integrity יזהה זאת וה-Packets יזרקו. נתמקד באספקט היישום עצמו: באג בתוכנה שבתסריט נדיר מאוד יכול לגרום לאפליקציה "להתכופף". הסיכוי שקומבינציה זו תוכנס לאפליקציה נמוכה, אך אם הבאג נופל לידי תוקף והוא מגלה אותו (יש דרכים שונים לגלות זאת) הוא יכול לכופף את התוכנה.

גם אם אין באגים כאלו, המערכת יכולה לקרוס אל מול עומס הודעות. המטרה היא שהאפליקציה יהיו זמינים למשתמשים החוקיים. לכן אם רוצים לתקוף יישום, ניתן להציפו בהודעות סרק שיגזלו מהיישום משאבים רבים והוא לא יוכל לטפל בבקשות אמיתיות, וכך נפגת זמינות האפליקציה. בקשות סרק אלו יכולים לגזול משאבי רשת (מילוי ה-bandwidth), משאבי מערכת הפעלה (CPU, MEM, DISK SPACE). מערכות יכולות או להתמלא או פשוט לקרוס.

DoS/DDoS – denial / distributed denial of service – מחזיקים את הרשת תפוסה. כדי לסכל נסיונות זיהוי התקפה מ-IP מסויים למשל, תוקפים יכולים לזהות באמצעות כלים אוטומטיים מחשבים לא מאובטחים, לשתול בהם תוכנה עויינת שלא מפריעה למשתמש אותו מחשב כלל, וכעת לתוקף יש אוסף מחשבים (מאות אלפי ויותר) המכילים את התוכנה העויינת שרוב הזמן רדומה. כאשר התוקף רוצה לתקוף אתר מסויים ברשת, הוא יתן את הכתובת שלו ואיזו פקודה לשלוח אליו. כל המחשבים ישלחו הודעה זו למערכת המותקפת ברגע נתון, ונוצר עומס גדול – וזהו distributed denial of service. אז משתמשים חוקיים לא יוכלו לגשת לשרת. המחשבים המרוחקים הם zombies, ורשתות אלו נקראות BotNet.

אתרים רבים, בעיקר העוסקים בהימורים, מאבדים כספים רבים על דקות שהן לא פעילים בהם, וישנם ארגונים הגובים "protection" מאתרים כאלו כדי שלא יציפו אותם ויקריסו אותם.

Authentication Attack:

ישנו צורך במנגנון שידע לזהות מי בצד השני של הקו. שני מנגנונים מרכזיים:

- Authentication: בא לענות על השאלה מי מזדהה, והוכחה שאותו אדם הוא אכן מי שהוא טוען שהוא.
- Authorization: מנגנון הבודק שאדם המבצע פעולה אכן מורשה לבצע את אותה פעולה. בא לענות על השאלה האם אותו משתמש מורשה לבצע את הפעולה שאותה מבקש לבצע. כאן נכנסים מנגנוני ה-access control שדוברו קודם.

כדי לאכוף את מנגנוני ה-access control, כמובן שיש צורך ב-authentication.

מנגנון נוסף: Auditing: לפני שמעלים מערכת, רוצים לבדוק שהמערכת לא מכילה פגיעויות. ניתן להשתמש ב-scanners הסורקים אחר פגיעויות ברמת האפליקציה או הרשת. המטרה היא לתת למנהל המערכת את תמונת המצב של הפגיעויות במערכת. רמה נוספת ב-auditing: בזמן אמת, כל אירוע חשוד ידווח למרכז בקרה. שלב אחרון: כל האירועים החשודים מוכנסים למאגר נתונים, במטרה שאם יקרה אירוע אבטחת מידע, תהיה לפחות היכולת לנתח מה קרה ואיפה היא התחילה, לזיהוי מקור הבעיה והחולשה. ה-logs חשובים ל-tracing לאחור של אירועים כאלו.

Triple Gold Standard: AAA: בדיוק שלושת המנגנונים לעיל.

תכונה חשובה נוספת: מניעת התכחשות.

סיכום: מוצג גרפית בשקופית Classify Security Attacks as

- התקפות פסיביות: האזנה ל-traffic – שני צידי התקשורת לא יודעים כלל שההתקפה קרתה.
- התקפות אקטיביות: ניסיונות התחזות, התקפת reply, שינוי הודעות, DoS. ב-DoS ההתקפה לא מוסתרת, לעומת השלושה הראשונים.

Security and Risk Management:

כל התמודדות עם התקפות עולה משאבים – חומרה, תוכנה וכו'. אין אפשרות מעשית להגן על הכל. כמו כן לחלקים שונים ישנן רגישויות ברמות שונות, ולכן המנגנון צריך להיות מותאם לרגישות המערכת וחלקיה. מטרות:

- הסרת הגורם ל"משיכה" לתוקף.
- לגרום לתוקף ללכת לתקוף אחרים (למשל ע"י הגנה מוגברת).

כאשר מיישמים מערכות אבטחת מידע, עובדים לפי כמה כללים בסיסיים:

Positive Security logic: הכל אסור, פרט למה שהוגדר כמותר. דוגמא: מי שאין לו תו כניסה לחניון האונ', לא יכול להיכנס. כל מה שלא מותר במפורש, הוא אוטומטית אסור. החיסרון: כל מה שמותר צריך הגדרה ספציפית. דוגמא: Firewall – מגדיר מה מותר (source IP, destination IP and port). כל מה שלא מוגדר לפיירוול אסור. כנ"ל לגבי אותנטיקציה, רק משתמש שמוגדר כחוקי יאושר.

Negative Security logic: כמו אנטיוירוס, ניסיונות איתור קבצים חשודים כקבצים עויינים; תעבורת רשת בעלת מאפיינים של תעבורה עויינת. יתרון: פתרון זה הוא גנרי, מגדירים מה אסור ולא מה מותר. חסרונות:

- התקפות שניתנות לזיהוי הן רק כאלו שכבר מכירים, התקפות לא מוכרות יצליחו.
- לפיכך נדרש עדכון תמידי של מאגר ההתקפות המוכרות.
- ישנו פרק זמן בין תחילת הרצת תקיפה ברשת ועד עדכון תוכנת ההגנה – "zero day".
- False positive: המערכת יכולה לזהות חשד שלא לצורך.

ישנו trade-off בין false-negative – לא זוהתה התקפה, ובין false positive – משהו שאינו התקפה זוהה כהתקפה. דוגמא לכך: מה עושים במצבי קצה שלא הוגדר מראש טיפול בהם באפליקציה. מבחינת אבטחת מידע, לא מאפשרים את המצב, הולכים "על בטוח". מבחינת שימושיות, המשתמש נחסם – בעית שימוש. אבטחה ו-usability לרוב באות אחת על חשבון השניה.

עקרון ה-least privileges: כל משתמש יקבל לכל תהליך את ההרשאות הנדרשות לו לביצוע תפקידו בדיוק (לא פחות ולא יותר). אם לא יהיה כך, אדם יוכל לבצע פעולות שאסורות לו, או אדם המתחזה לו יכול להשתמש בהרשאות המקסימליות של אותו אדם בתהליך אותו מפעיל.

Separation of Duties: מניעת סיכונים מיותרים במערכת ע"י חלוקת סמכויות והרשאות. נושא זה חשוב כדי לבנות מערכת least privileges טובה כדי לנהל access control כמו שצריך.

עקרון Open Design: ככל שיותר אנשים מכירים את הקוד ובודקים אותו, הסיכוי שתהיה חולשה שלא תתגלה קטנה. חסרון: גם התוקף יכיר את קוד המערכת. ככל שהקוד סגור יותר, כך יש יותר סיכוי לבעיות ופחות insentive לתקנן. תפישה זו נעשית יותר ויותר פופולארית בעולם.