



מיכאל שטאל

ארכיטקט בדיקות תוכנה באינטל, ישראל, עוסק בעיקר בבדיקות מערכות משובצות מחשב. במסגרת תפקידו, מיכאל מגדיר שיטות בדיקה ומתודולוגיות עבודה, עוסק בהדרכה ולפעמים אפילו מרשים לו לבדוק משהו (שזה הכי כיף). מיכאל מציג תכופות בכנסים בארץ ובח"ל ומלמד בדיקות תוכנה בפקולטה למדעי המחשב באוניברסיטה העברית. ניתן לראות חלק מהמצגות והמאמרים שלו באתר www.testprincipia.com



michael.stahl@noWhere.com אינו קיים - ולכן הפורץ יכול לדלג על שם זה ולא לבזבז זמן בנסיונות ניחוש של הסיסמה שלו. לעיתים עצם העובדה שיש לי חשבון באתר מסויים יכולה לגרום לי מבוכה או צרות (ראו [בקישור](#))

”
עצם העובדה שיש לי חשבון באתר מסויים יכולה לגרום לי מבוכה או צרות
”

דרך אגב: בדקתי, וגם הכניסה לגוגל מתנהגת ככה... אני אתן לחברות מכובדות אלה להנות מהספק ולומר שכיוון שהאימייל שלי הוא לא בדיוק סוד, אין בעייה עם זה. או שהחבר'ה האלה יודעים מה הם עושים, ויש להם הגנות מספקות אחרות מעבר לשם המשתמש. אבל בעצם עדיף שמסך הכניסה ידרוש גם שם משתמש וגם סיסמה, וכשאחד מאלה אינו נכון, יתן הודעה כללית, כמו שאני מקבל בהכנסת סיסמא שגויה אחרי שם משתמש נכון (ראו איור 2)

בדיקות אבטחה



כשמדברים על בדיקות אבטחת תוכנה, האסוציאציה היא מיד ל - white-hat hacking, penetration testing וכדומה. אבל מסתבר שגם בודקי התוכנה ה"רגילים" – בני תמותה כמונו - יכולים לתרום הרבה לשיפור הקשיחות של התוכנה והעמידות שלה לפריצה. בדיקות אבטחה מסוג זה הן במידה רבה הרחבה של בדיקות פונקציונאליות שאנו ממילא מבצעים, ולא תחום חדש לגמרי. לעיתים זה פשוט עניין של שימת לב נוספת להתנהגות התוכנה תוך כדי הרצת בדיקות שממילא אנו מתכננים.

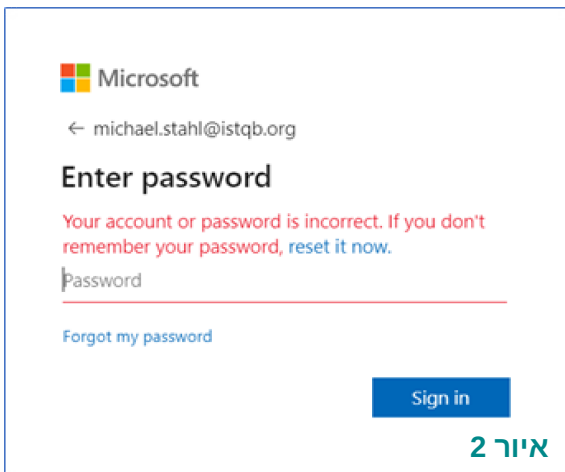
אחת הפונקציות שהקשר שלהן לאבטחת תוכנה ברור ומוכן אינטואיטיבית היא ה-log in. כל אתר שמספק שירות וחלק מאתרי התוכן כוללים פונקציה זו, וכל סט בדיקות פונקציונאליות של האתר מכיל מן הסתם גם בדיקות של מסך הכניסה למערכת. למשל, בדיקה שמשתמש רשום יכול להכנס רק אם סיפק סיסמה נכונה; משתמש לא רשום לא נכנס, ושיש יכולת שיחזור סיסמה ("שכחתי סיסמה").

כיוון ששדות הקלט (שם משתמש, סיסמה) מקבלות מחרוזות, הרי שגם נבדוק מקרי קצה: מחרוזת ריקה, מחרוזת באורך 1 ומחרוזת באורך מקסימלי. עוד סט בדיקות שעדיין "יושב טוב" בתוך התחום הפונקציונאלי הם בדיקות שהקוד מבטיח איכות מינימלית של הסיסמה (לפחות 8 תווים; לפחות ספרה אחת, אות גדולה ואות קטנה, וכו'). כללים אלה נועדו להקשות על ניחוש הסיסמה על ידי פורצים. עד כאן, הכל פונקציונאלי: עובד נכון או לא עובד. מכאן אציינ עוד מספר בדיקות ש"מתרחקות" מבדיקת היכולת הבסיסית של אימות המשתמש ומטרתן לוודא שמסך הכניסה לא סובל מחולשות שיקלו על האקרים לחדור לאתר. בדיקות אלו מתייחסות גם לדרך המימוש של אימות זהות המשתמש, ולא רק ל"עובד/לא עובד".

זליגת מידע



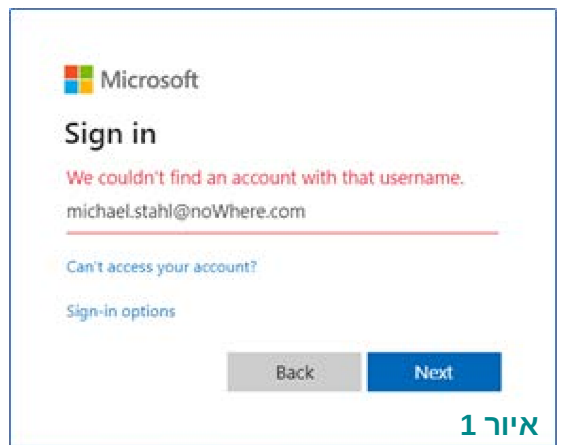
על מנת לפרוץ לחשבון של מישהו, צריך לפחות שני פריטי מידע: שם המשתמש וסיסמה. אם סתם אנסה המון צירופים של שמות משתמש וסיסמה, צפויה לי הרפתקה ארוכה ומתסכלת עד שבמקרה אפול על צירוף נכון. על כל שם משתמש שאנחש, עלי לנסות את כל הסיסמאות שאני מנחש. כמה נחמד היה אם יכולתי לקבל רמז ששם המשתמש שנתתי אינו תואם אף אחת מהשמות שרשומים באתר ולכן אין טעם לנסות שם זה עם המון סיסמאות! הנה דוגמה לא טובה: אני מנסה לעשות log-in לשרת של חברה קטנה מרדמונד, ומכניס שם משתמש שאינו קיים במערכת. בתגובה, המערכת טורחת להודיע לי ששם המשתמש אינו נכון! (ראו איור 1).



איור 2

שימו לב שההודעה לא מספרת לי מה לא נכון – שם משתמש, הסיסמה, או שניהם – וזו התנהגות נכונה. בדוגמה כאן זה קצת מצחיק כי אי אפשר להגיע למסך הסיסמה בלי להכניס שם משתמש נכון, אבל העקרון עדיין תופס – וצריך לבדוק שזו ההתנהגות של מסך הכניסה למערכת.

העקרון של מניעת זליגת מידע תופס גם לגבי התנהגויות אחרות במערכת: האם קריאה שגויה ל-API; מייצרת הודעת שגיאה שמספקת לי מידע מיותר? האם נסיון גישה לדף שאינו קיים במערכת מספרת לי משהו שאוכל להשתמש בו למציאת פירצה (כגון: כתובת IP, הוורסיה של תוכנת השרת; מיקום מדויק של קבצים על השרת, וכו')? כל אינפורמציה כזו מרחיבה את מה שפורץ פוטנציאלי יודע על האתר או התוכנה שלי, ואולי תספק משהו שניתן לנצל לצורך חדירה לאתר. בקיצור, כלל טוב (גם לחיים): עדיף לדבר מעט.



איור 1

התנהגות כזאת נקראת "זליגת אינפורמציה". המערכת מספקת מידע שהוא בעל ערך למי שמנסה לנחש שמות משתמשים. הודעה זו מדווחת שהשם



גלישת חוצץ (Buffer Overflow)



” יתכן מאוד שקוד – גם כשיש בו גלישת חוצץ – ירוץ בסדר גמור ”

”גלישת חוצץ” היא שגיאת תכנות המתבטאת בכך שתוכנית מחשב כותבת לאזור בזיכרון המחשב יותר מידע מאשר אותו אזור מסוגל להכיל. כתוצאה מכך "גולש" חלק מהמידע אל מחוץ לגבולות החוצץ, ומשנה נתונים שלא היו אמורים להשתנות. גלישת חוצץ עלולה, בין השאר, לאפשר הרצה של "קוד זדוני" הגורם לתוכנית לפעול באופן שלא תוכנן מראש. מבחינת אבטחת תוכנה, לא רק כתיבה אלא גם קריאה מאיזור שהוא מעבר למה שהוקצה לנתון מסוים, יכולה להוות פריצה הניתנת לניצול.

מה שמעניין הוא שהרבה פעמים הקוד – גם כשיש בו טעות שאורמת לגלישה – ירוץ בסדר גמור. יתכן שרק קומבינציה מאוד ספציפית של נתוני הרצה יגרמו לשגיאה שניתן לחוש בה באופן חיצוני. המשמעות של מצב זה היא שבמקרים רבים בדיקות פונקציונליות לא יזהו שיש בעיה של גלישה.

אחת השיטות לבדוק אם יש גלישה, היא שימוש בטכניקה של fuzzing בשילוב עם וורסיה של הקוד שעבר מיכשור (אינסטרומנטציה) מיוחד.

(א) Fuzzing היא טכניקת בדיקות שבהם אלגוריתם מייצר מספר רב של קלטים שונים עבור התוכנה הנבדקת, שולח אותם לתוכנה, ובודק שהתוכנה לא נתקעה או התרסקה. אין כאן בדיקה פונקציונלית האם התנהגות התוכנה מתאימה לקלט שנשלח – בעיקר כי הרוב המוחלט של הקלטים שהאלגוריתם מייצר הם קשקוש מוחלט ולא קלט הגיוני שאפשר לעשות איתו משהו. כל מה שאנו מנסים לעשות זה לייצר קלט שהתוכנה לא יודעת לזרוק או להתעלם ממנו. מצב כזה, שבו התוכנה "מתבלבלת" מהווה סיכון בחיבת אבטחת תוכנה, כי יתכן שתגובת התוכנה תהיה לקרוס תוך זליגה של מידע, או להתקע לפני שהספיקה למחוק סודות (למשל, מפתחות הצפנה) מהזכרון.

(ב) מיכשור: שתי הפונקציות של הקצאת זכרון ושחרור זכרון (malloc; free) מוחלפות בקוד שמאפשר לבדוק אם כתיבות או קריאות נעשות רק מאיזורים בזכרון שמותר לתוכנה לגשת אליהן. ברגע שהקוד מזהה גישה לאיזור אסור בזכרון (שהו מה שקורה בגלישת חוצץ), הוא מקריס את התוכנה. שיטה זו נקראת [AddressSanitizer](#) או בקיצור: Asan.

” לעיתים, כדי למצוא בעיות אבטחת מידע, כל מה שצריך זה מודעות ”

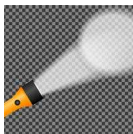
הרצה ארוכה של fuzzing (שעות או ימים!) מעלה את הסיכוי ליצירת שילוב של קלטים שיגרום לגלישת חוצץ. שילוב של הרצת fuzzing על קוד שעבר מיכשור של AddressSanitizer יגרום לקוד לקרוס ברגע שגלישה כזו קרתה, וכך נוכל לזהות שיש בקוד בעיית גלישה.

פעילות בדיקה מסוג זה נופלת בין אחריות של קבוצת הבדיקות "הרגילה" לבין אחריות של מומחים באבטחת תוכנה. גישה פרקטית היא שפיתוח ה-fuzzer והמכשור של הקוד נעשה על ידי מומחי אבטחת התוכנה, והאחריות להרצת בדיקות אלה מידי פעם (נגיד, על כל וורסיה חדשה של התוכנה) מוטלת על קבוצת הבדיקות. אצין גם שהתמחות בטכניקת fuzzing מספקת מסלול קידום לאנשי בדיקות, שמאפשר להם לקחת אחריות גם על הצד המתקדם יותר של פיתוח ה-fuzzer.

מנסיון, גם הרצה של הבדיקות הרגילות (בלי fuzzer) על קוד שעבר מיכשור של AddressSanitizer מוצאת בעיות של גלישות זכרון. חלק מהגלישות לא מחייבות קלט מאוד ספציפי, ויקרו גם על הקלט שכבר קבעתם לבדיקה מסויימת. זה הופך את השיטה לכלי יעיל מאוד במציאת חולשות בקוד.

יצירת הקוד המיוחד אינה קשה; העלות העיקרית של פעילות זו היא הצורך בהרצה של סבב בדיקות שאי אפשר להתחשב בתוצאותיו כיוון שהסבב מבוצע על קוד שעבר מיכשור והוא שונה מהקוד של המוצר הסופי (זו בדיקת אותה בעיה שיש כשמריצים בדיקות לצורך הערכת כיסוי הקוד – code coverage). שימו לב ששפת התכנות שבה מפתחים את המוצר משפיעה על הסיכון שהקוד סובל מגלישות. #C למשל כמעט ואינה רגישה לגלישה (צריך להתאמץ כדי לייצר את זה). C לעומת זאת, לא מוגנת כלל, אלא אם משתמשים בדגלים מסוימים בזמן הקומפילציה, כמו למשל -fstack-protector. מומלץ לחפש חומר בנושא ולראות מה הקומפיילר שלכם יכול לעשות בתחום זה (חפשו: "security hardening compilation flags").

מתחת לפנס



לפעמים לא צריך להתאמץ יותר מידי כדי למצוא בעיות אבטחת מידע בקוד – כל מה שצריך זה מודעות ופתיחת עיניים. הנה כמה דוגמאות:

- הריצו את הקוד להחלפת סיסמה, תוך שאתם מקליטים (עם sniffer) את התעבורה בין המחשב שבו אתם משתמשים והשרת. וודאו שהסיסמה החדשה מוצפנת לפני שהיא נשלחת דרך השרת אל השרת. לעיתים מה שעובר זה לא ממש הצפנה אלא digest (מעין "חתימה") של הסיסמה – שזה ממש לא הצפנה. אפשר להקליט את ה-digest, ושידור שלו ישירות לשרת (ולא דרך דף ה-log in) יעבוד יפה מאוד - השרת יחשוב שקיבל סיסמה נכונה.
- אם המוצר שלכם כותב קבצים זמניים או קבועים לדיסק, כדאי לפתוח אותם ולבדוק אם הם מכילים נתונים חסויים (נגיד, מספר כרטיס אשראי... מספר חשבון בנק...) ללא הצפנה.
- האם השרת שבניתם עובד עם cookies? איזה מידע נכתב בהם? האם יש שם מידע חסוי, שלא הוצפן ויכול להיקרא על ידי כל אדם עם גישה למחשב?

ישנן עוד הרבה בדיקות שחורגות מעבר לשאלה של "האם זה פועל". אפשר למצוא הרבה חומר [ברשת](#) שמלמד מה עוד כדאי לבדוק על מנת להבטיח שהתוכנה שלכם לא רגישה לפריצה או לשיתוף במידע חסוי.

לסיכום

שלושת הנושאים שהבאתי הם רק דוגמה לדברים שבעבר נחשבו כאחריות של מומחי אבטחת תוכנה (יש עוד!). החשיבות שיש למחשב ולחיבוריות בחיינו מצד אחד, והתגברות הסכנות לפריצה מצד שני, משמעותם שאבטחת מידע ותוכנה הם תחומים שכל בודק תוכנה צריך להבין. כמו כן סביר לדרוש שבדיקות בסיסיות של אבטחת תוכנה תהיינה כבר חלק מהבדיקות הרגילות של המוצר. לדאוג: עדיין תשאר עבודה למומחי הפריצות לחפש חורים במוצר – אבל זה יהיה משמעותית יותר יעיל אם הטעויות הפשוטות, שניתן לזהות בבדיקות ישירות ומתוך מודעות, מראש לא יכנסו למוצר שמומחים אלה יתקפו. תודה לעמיתי יאיר נצר על ההערות לטור, ועל הפניות למקורות ברשת.