



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



המערכת. אביא כאן שלוש דוגמאות.

פרוטוקול תקשורת



כמעט כל אפליקציה משתמשת בפרוטוקול תקשורת. בין אם זה לקבל מידע או שירותים מישויות אחרות או על מנת לתקשר בין חלקים שונים של אותה מערכת. בגדול, פרוטוקולים מעבירים שני סוגי מידע: (א) המטען (payload) – זה המידע שאנו רוצים להעביר (למשל: מה היתרה שלי בחשבון הבנק). (ב) מידע הנדרש על מנת שהמטען יועבר כמו שצריך ממחשב אחד לשני (למשל, כתובות).

חלקים גדולים מכל מערכת אינם מופיעים בצורה גלויה

כבודקים, אנחנו בראש ובראשונה מתרכזים במטען. ניקח לדוגמה מסך login. נניח שאנו בודקים שרת שדורש סיסמה מהמשתמש (אתר

הבנק שלנו, למשל). בבדיקות נתרז בווידוא שמשתמשים חוקיים שמספקים סיסמה נכונה מקבלים אישור כניסה, ומשתמשים לא קיימים, או שמספקים סיסמה שגויה, לא נכנסים. יש עוד הרבה דברים אחרים סביב לזה. חלק מזה אולי אפילו מתואר בדרישות: הסיסמה צריכה להיות בעלת חזק מינימלי; צריך לטפל בלקוח ששכח את הסיסמה; צריכים להיות מנגנונים שמאטים את תגובת המערכת אחרי מספר ניסיונות כניסה שגויים, וכו'.

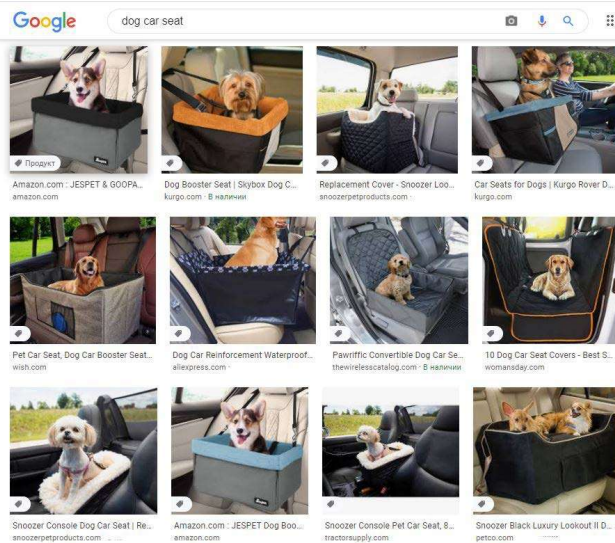
עד כאן – הכל גלוי וברור. אבל מה קורה בקרביים של הפרוטוקול? הרבה מאוד. למשל: הסיסמה עצמה לעולם לא נשלחת מהמחשב של המשתמש אל השרת. במקום זה, נשלח משהו שמוכיח שהמשתמש יודע את הסיסמה – אבל בצורה שמאזין מהדד לא יכול לזקק מתוכו את הסיסמה. שיטה מקובלת עובדת ככה: השרת שולח מסך אקראי למחשב של המשתמש. בצד של המשתמש, התוכנה מצרפת את המספר האקראי לטקסט של הסיסמה, ומכניסה את התוצאה לפונקציה חד-כיוונית (זוהי פונקציה שקשה מאוד לשחזר את הקלט שלה, בהינתן הפלט שלה; פונקציות hash הן דוגמה נפוצה). התוצאה נשלחת לשרת. השרת, שמכיר את הסיסמה של המשתמש, מבצע את אותה פעולה מתמטית, ומשווה את התוצאה המקומית שחישב למה שהמשתמש שלח. אם התוצאות זהות, הרי שהמשתמש הוכיחו שהם מכירים את הסיסמה.

ראו כמה דברים עובדים כאן "מתחת למים": ייצור המספר האקראי (אם אינו ממש אקראי זה יכול להקל על פורצים לחנש סיסמאות); המימוש של הפונקציה החד-כיוונית (טעויות במימוש עלולות לפתוח פירצה); מערכת שמירת הסיסמאות בשרת ומעל הכל: הפרוטוקול עצמו שבעזרתו עוברים המסרים בין המחשבים. בעיני רוחנו השרת מקבל שתי פיסות מידע: שם וסיסמה. אבל אם נחבר רחרון (sniffer) לקו התקשורת, יתברר לנו ששני השדות האלה נשלחים בתוך מבנה נתונים מסובך למדי: שדות שמגדירים את הפרוטוקול, את אורך המטען, כתובות, פרטים על הפרוטוקול, ועוד ועוד. לא רק זה: בדרך כלל מעורבים מספר פרוטוקולים בעניין. למשל: הודעת HTTP נשלחת כמוסה (encapsulated) בתוך פרוטוקול TCP, שכמוס בפרוטוקול IPv4, שכמוס בפרוטוקול Ethernet. לפעמים זה לא ממש מעניין – במקרים שהשרת משתמש במערכת הפעלה מלאה שמספקת את כל התשתית לטיפול בתקשורת, האפליקציה

עיקרון הקרחון



המשפחה שלי מתמחה בהמצאת רעיונות להתעשרות מהירה. לא שאנחנו מתכוונים אי פעם לעשות עם זה משהו, אבל זה שעשוע לא מזיק, כמו למשל הרעיון הנפלא לייצר כסאות בטיחות לכלבים. הרי בזמן תאונה הכלב יכול להתעופף בחלל המכונית ולפגוע באנשים! ובכלל עדיף שלא יתרוצץ וידחוף, באופן בוגדני, לשון קרה לאוזן של הנהג. רגע לפני שהשקעת את חסכונו המשפחה בסטארט-אפ, עשיתי חיפוש מהיר בגוגל:



מאז אותו אירוע הפנמתי את מאמר קהלת: "אין חדש תחת השמש", או בשפה מודרנית יותר: תחפש בגוגל לפני שאתה מתפטר מהעבודה. זה נכון לגבי בדיקות (ראה את "כל פעם אותו סיפור", גיליון 10), וגם לגבי העיקרון החדש שהמצאתי לאחרונה: "עקרונות הקרחון". אז זהו... שגם עקרון זה אינו חדש. הקופירייט על תאוריית הקרחון אמנם שייך לארנסט וקשור לסגנון הכתיבה שלו, אבל זה לא הפריע לאחרים להשתמש במונח לתיאור העובדה שחלקים גדולים מכל מערכת אינם מופיעים בצורה גלויה. על אף שאינם גלויים, לחלקים אלה משמעות רבה בפעולת המערכת ואין הבנה מלאה של המערכת בלי הבנת מה ש"מתחת למים".

כיוון שרבים לפני כבר ניכסו את הביטוי של המינגווי להעברת רעיונות, אני מרגיש בסדר להצטרף לחבורה, ולהראות את הרלוונטיות של עקרון הקרחון לתחום בדיקות התוכנה.

אנחנו בודקים את המערכת מבחוץ, ולא חשים כמה העסק מסובך בפנים

הדוגמה הפשוטה והזמינה ביותר היא היחס בין ממשק המשתמש למה שקורה "מתחתיו". קחו למשל את ממשק המשתמש של Google: מסך כמעט ריק (שדה אחד!). כל מה שהמשתמש עושה הוא להכניס כמה תווים לשדה זה. עוד לפני ההקלדה על כפתור החיפוש, המנועים של גוגל כבר בעבודה, ומציגים את החיפושים הפופולריים שמתחילים במילים שהקלדנו. נמובן שאחרי הלחיצה על "חיפוש", חוות רשתים עצומות אי שם בעולם מריצות אלגוריתמים משוכללים ומקבילים ומביאות לנו את התוצאות בפחות משיניה. זה עובד טוב כי (בין השאר) מישהו היה מודע לחלק שהמשתמש לא רואה – החלק שמתחת למים – ובדק אותו.

אפשר לטעון שאין כאן צורך במודעות השונה ממה שנדרש בבדיקת כל מערכת אחרת. יש דרישות פונקציונליות (מה יתקבל בחיפוש) ודרישות לא פונקציונליות (באיזה מהירות נקבל תוצאה), והבודקים צריכים לוודא אותן. אבל יש מקרים שבהם החלקים הנסתרים באמת נסתרים – בעיקר כי הם לא מיידית מתקשרים לפונקציונליות של



שמורות



כשאנחנו מריצים בדיקה, אנו מגדירים תוצאות צפויות ומוודאים בקפדנות שתוצאות אלה התקבלו. אבל מעבר לתוצאות אלה, יש המון דברים שאמורים להישאר בדיוק כמו שהיו לפני הרצת הבדיקה. אלה נקראים בעברית צחה "שמורות" (invariants). אפשר כמובן לתת אין-ספור דוגמאות: אם אני מחובר לבנק שלי דרך כרום וגם דרך פיירפוקס, הרי שיציאה מהחשבון בכרום לא אמורה להשפיע על החיבור דרך פיירפוקס. למעשה, כל מה שקורה בפיירפוקס הוא אינווריאנטי לפעולות בכרום. השמורות הן עוד דוגמה לעקרון הקרחון: הרבה פעמים אנחנו לא מודעים לצורך לחשוב על מה שלא אמור להשתנות, ומה, מכל הדברים שלא אמורים להשתנות, צריך לכסות על ידי מקרה בדיקה. אמנם אין סוף דברים שאמורים להישאר כמו שהם, אבל עם קצת מחשבה אפשר לזהות מה נמצא בסיכון לשינוי לא רצוי.

דוגמה מוכרת (ומעצבנת ביותר) היא הכנסת נתונים לטופס ברשת: אני מכניס נתון שגוי בשדה מסוים, ומנסה להתקדם למסך הבא. אני מצפה שהנתונים שהכנסתי ישמרו – גם אם הייתה טעות באחד השדות.

תוצאה צפויה: הודעת שגיאה וציון השדה הבעייתי.

שמורות: הנתונים בכל שדה שלא היה שגוי נשארים עם הערך שהוכנס.

אחרי דוגמה זאת אני מניח שלא צריך להסביר כמה החיים היו יותר טובים אם המושג של "שמורות" היה מוכר לכל הבודקים!... אלה היו רק שלוש דוגמאות. כיוון שחלקים גדולים מכל מערכת אינם מופיעים בצורה גלויה, סביר שגם במערכת שלכם יש לא מעט דברים מתחת למים. מתי בפעם האחרונה צללתם והסתכלתם מה יש שם?

באמת מקבלת בסופו של תהליך התקשורת רק את המטען. אבל מה אם המוצר שלנו ממש שרת מינימלי במערכת משובצת, ששם כל בייט זכרון חשוב, ולכן כתבנו בעצמנו את הקוד של הפרוטוקול? עכשיו מסתבר שהפעולה הפשוטה יחסית של קבלת שני פרטי מידע ממשמש כוללת בעצם עשרות שדות של מידע. עלינו לבדוק שהקוד בשרת מפרק (parse) נכון את התשדורות המתקבלות ומתנהג נכון בכל המקרים שתשדורות מגיעות עם פרטי מידע שגויים, לא בתחום, חסרים וכו'. הזנחת הבדיקה של חלקים אלה בקוד מעמידה את השרת בסכנה שיתמוטט בכל פעם שיגיע משהו לא מזוהה דרך הרשת. זה גם פותח פתח להתקפות זדוניות, שכן אם קלט מסויים מקריס את המערכת, קל לייצר התקפת (DOS denial of service).

פעולה פשוטה של קבלת שני פרטי מידע ממשמש כוללת בעצם עשרות שדות של מידע

כאילו שזה לא מספיק, יש עוד דברים בתחום ה"בלתי נראה". כמשתמשים, יש הרגשה שמה שאנחנו שולחים ממשק המשתמש, זה מה שמגיע לשרת. למעשה, במקרים רבים, ההודעה של המשתמש עוברת עיבוד לפני

השליחה (הצפנה; או תוספת פרטים כמו שעת המשלוח). ממשק המשתמש מגן עלינו ומונע מאיתנו שליחת מידע שגוי. למשל, שעת המשלוח תהיה תמיד השעה שעל שעון המחשב, בלי יכולת לשלוט על זה. כבודקים, אנחנו צריכים להיות מודעים לתכולה המלאה של המסרים שנשלחים לשרת ולייצר מצבים פתולוגיים גם בתוספות הבלתי נראות. זה אומר שאנו לא יכולים להסתפק ביצירת בדיקות דרך ממשק המשתמש, אלא צריכים לכתוב קוד שישלח הודעות לא סטנדרטיות. כיוון בדיקות כזה נראה אולי מוגזם: למה לייצר הודעות שאין יכולת לממשק המשתמש לייצר? כיוון שמדובר בתקשורת, עלינו לקחת בחשבון שהודעות יכולות להיות מיורטות בדרך בין המשתמש לשרת, ולעבור שינוי (לא אכנס כאן לפרטים – חפשו בגוגל man in the middle attack). זהו סיכון אמיתי ולכן בבדיקת המערכת צריך לשלוח נתונים לא רק דרך ממשק המשתמש, אלא לרדת אל "מתחת למים" ולייצר הודעות שגויות וזדוניות באופן מלאכותי.

קלט של פונקציות



תיאוריית בדיקות בסיסית ממליצה לבדוק שפונקציות מבצעות בדיקות על הקלט לפני שמשתמשים בו. כך נמנע מהפעלת הפונקציה על נתונים שהם מחוץ לתחום שהפונקציה מוגדרת בו. למשל, אם הקלט לפונקציה הוא מצביע (pointer), נבדוק שהקלט אינו NULL לפני שנשתמש בו. אם מתברר שהוא כן NULL, על הפונקציה להחזיר קוד שגיאה. כל זה טוב ויפה כשמדובר בפונקציה שמקבלת קלט פשוט. אבל לעיתים, קלט לפונקציה הוא מבנה נתונים. כלומר, לא נתון אחד או שניים, אלא אולי עשר או עשרים נתונים. לעיתים קרובות חלק מהשדות במבנה הנתונים של הקלט גם הם אינם קלטים פשוטים, אלא מבני נתונים... וכו' וכו'. אחרי מעקב ופריסה של כל הקלט, יתכן שמדובר בעשרות רבות של פרטי קלט – וכל זה על פונקציה אחת! יש כאן מספר בעיות (א) בדיקות מלאות של כל השדות יוסיפו המון עבודה והמון זמן בבדיקות. (ב) הכנסת קוד שיבדוק את הנכונות של כל הערכים בקלט יהפוך כל פונקציה למפלצת של בדיקות קלט, וישפיע לרעה על זמן העיבוד ועל גודל התוכנה לאחר קומפילציה (ג) עקרון הקרחון: לפעמים אנחנו לא מודעים כלל לעומק הסיבוך... אנחנו בודקים את המערכת מבחוץ, דרך ממשקים פשוטים יחסית (GUI או API) ולא חשים אמה העסק מסובך בפנים. גם אם היינו מודעים לכך, יהיה צורך במאמץ עצום לייצר בדיקות נגטיביות שיצליחו לחלחל קלט לא תקין לכל שדה במבנה מסובך.

מה עושים? מעבר למודעות, הערכת סיכונים וכתובת בדיקות שמיועדות לבדוק חולשות ספציפיות, אחת הדרכים להתמודד ולמצוא באגים במקרים אלה הוא שימוש רחב ב-fuzzing. תיארת את הטכניקה הזו במאמר בגליון הקודם (בדיקות אבטחה – גליון 24).

קול קורא

Testing & Automation

GeekWeek

2024

20-24 ביוני, מלון דניאל, הרצליה

שמחים לעדכן כי גם השנה נקיים את כנס הבדיקות המסורתי - המקום שלכם להתעדכן במה שקורה היום בעולם ה-QA, וליהנות מחוויות נטורקינג עם קהילת הבודקים והמומחים בתחום! הכנס יתקיים בהתאם להנחיות משרד הבריאות וניתן להתחבר גם מרחוק.

לא ניתן לפתח את עולם הבדיקות היום, מבלי לקחת בחשבון את התמורות בניהול פרויקטי התוכנה, הכניסה המוחלטת לעולם ה CI/CD ושילוב הבדיקות כחלק מתהליך ה-Continuous מחייב לימוד נושאים חדשניים שונים ומגוונים בתחום. בכנס השנה נדון בעיקר בתפקיד מנהל הבדיקות והבדוק בארגון, פתרונות חדשים, שילוב ה-Continuous כחלק מהבדיקות, בדיקות API Unit Test, שילוב ML/AI בתהליך הבדיקות, כלים פשוטים לבדיקות אוטומטיות אסטרגיות בדיקה חדשות ועוד...

במגוון סמינרים מרתקים ומעבודות מקיפות, נחשוף את השינויים הדרמטיים ואת המצופה מאנשי הבדיקות בעידן פיתוח תוכנה המודרני.

אנחנו מזמינים אותך להצטרף לשרת המרצים שלנו, המעוניינים להעביר את הידע המקצועי שלהם בכנס השנתי שלנו.

להגשת מועמדות ניתן ליצור קשר עם אושרת שם טוב | israeli@johnbryce.co.il