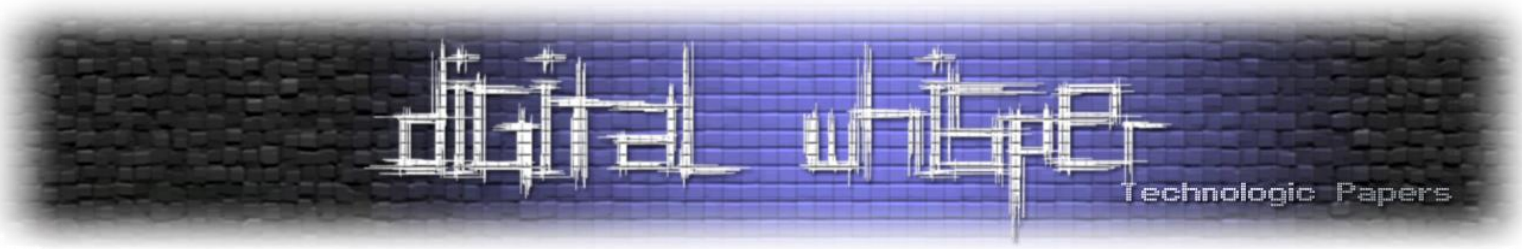

באגים נסתרים

מאת עידו קנר

הקדמה

בעקבות דיונים ארוכים אשר נכחתי בהם מספר פעמים, החלטתי להעלות למאמר את דעתי בנושא שאני מוצא כי יש בו מחסור רב בהבנת המשמעות של הכלים בזמן פיתוח תוכנה.

למרות כי על פניו, נראה כי כל מה שצריך הוא פוסט בבלוג, חשבתי כי הנושא חשוב מספיק בשביל להיות נגיש ליותר אנשים ועל כן החלטתי לפרסמו כמאמר במגזין.

במאמר קצר זה אנסה להציג מדוע לדעתי יש בעיות בתפיסה בה ביצוע בדיקות על תוכנה נחשבות להעדר באגים, ומה המשמעות של אמונה זו מבחינת בעיות אשר כן יכולות להגיע לתוכנה.

חשוב לי להדגיש כי זו הראיה שלי בלבד, המגיעה מניסיון שלי ושל מכרים בתחום התוכנה, וכתיבת מאמר זה אינו מגיע לתקוף דעות המנוגדות לשלי, אלא להציג כשלים שונים אשר אני חוויתי בעת אותם הדיונים.

תפיסת עולם

באפריל 2018 העברתי הרצאה למספר ארגונים שונים אודות מסדי נתונים רלציונים המתוחזקים כקוד פתוח. אחת השאלות הראשונות אשר נשאלתי היתה "איך זה לא מסוכן לדעת מה מנגנון האבטחה של מסד הנתונים, כאשר הקוד שלו גלוי?"

התשובה שלי הייתה מאוד פשוטה: "האם הקוד של iOS חשוף לעולם?" נעניתי בשלילה. האם עדיין יש פריצות אשר מאפשרות לקבל גישת ניהול על המכשיר? התשובה היתה חיובית. כך שבעצם Security through Obscurity יכול להקשות, אבל לא למנוע מגילוי בעיות.

מימוש טוב, ללא באגים ידועים גורמים לכך שהאלגוריתמים, ללא קשר אם חושפים את הקוד שלהם או לא, מקשים מאוד על פריצה כלשהי. כמובן שאין זה אומר כי אין חולשות, אלא כי הן אינן מוכרות לציבור.

כלומר החולשה היא לא הידע כיצד מומש האלגוריתם, אלא בניצול של פער הקיים במימוש, בין אם מדובר במימוש כושל או במימוש למנגנון כושל (כשל לוגי בדרך כלל). ניתן לעשות למימוש האלגוריתם Reverse Engineering גם אם אין את קוד המקור, אך ניתן למצוא בעיות גם כאשר קוראים את קוד המקור.

הנה הסבר מוחשי יותר: כאשר יודעים כי יש מימוש קוד המממש את x509 למשל, כל עוד הקוד של x509 ללא באגים ידועים, וכל עוד אין כשל לוגי בפרוטוקול, יהיו חייבים לשנות את הקוד בשביל שיהיו בעיות אבטחה. על מנת לנסות ולמצוא בעיות, יהיו חייבים לחקור אותו הרבה מאוד זמן בשביל למצוא מה אולי מומש לא טוב ונכון. במידה ולא נמצאה חולשה כלשהי במימוש, לא תהיה פריצה אליו, אולי ניצול של דברים מסביב, אבל לא של המימוש שבוצע ל-x509.

כאשר מדובר במערך בדיקות, אני מגלה כי יש תפיסה שגויה בהרבה פעמים אודות המשמעות של בדיקות.

הרעיון של ביצוע בדיקות אינו משקף הרבה מאוד דברים להם יש יחס כאשר מדברים על אותו מערך בדיקות, ואני חושב כי הגיע הזמן לנסות לפקס ולמקד קצת מעבר לכך מה התפקיד של אותו מערך בדיקות עצמו. זאת לטובת כך שמערכי הבדיקות לא יספקו תחושות שגויות באשר למה שהן מספקות.

כאשר אני מדבר על מערך בדיקות תוכנה, אני מדבר **רק** על שלושת הבדיקות הבאות:

- יחידות בדיקה (unit test): בדיקת פונקציונליות של פונקציה/מתודה מדוייקת, לתת אפשרות מסוימת.
- בדיקת השתלבות (integration test): בדיקת שילוב של מספר פונקציות לקבל פונקציונליות רצויה.
- בדיקות קצה לקצה (end to end/e2e): בדיקת תוצר רצוי כאשר מבצעים פעולות מלאות.

למרות שישנם סוגים נוספים של מערכי בדיקות, אינני הולך להתייחס אל אותם המערכים...

תפקיד הבדיקות

אז מדוע בכלל לבצע בדיקות תוכנה? התשובה לכך תלויה בסוג הבדיקה. למשל בדיקות של unit test, מבצעות בדיקה כי קלט או פלט מסוימים עושים את העבודה כמצופה בנקודה מסוימת. כלומר נקודת לוגיקה אחת בתוך פונקציה אחת בלבד נבדקת, ולא בדיקה של כלל הפונקציה.

כאשר משנים פונקציה כלשהי, רוצים בנוסף לבדיקה גם לראות שלא נשבר משהו שעבד בעבר (או אולי כן במידה וצריך), והתפקיד של unit test הוא לספק את עצם הוודאות כי אנו יודעים שכל מה שרצינו לקבל מתקבל, כל עוד אנו יודעים מה אנו רוצים לקבל.

זו הסיבה גם כי בודקים נקודת לוגיקה אחת פשוטה בפונקציה ולא את כולה.

כאשר מדובר ב-integration test, אנו רוצים (לצורך ההסבר) לדעת אם אנו מצליחים לקבל דוא"ל, לדעת כי הדוא"ל קיים במערכת ולשלוח הודעה בהצלחה (או לקבל כישלון נכון). אנו משלבים מספר פונקציות שאנו יודעים מה התוצר שלהן אמור להיות בכל שלב, ובכך יודעים כי לוגיקה מסוימת מתקיימת לפי

הצורך. הבדיקה היא רק על התנהגות מאוד מוגדרת של דוא"ל, אך לא מעבר. זה תוצר של מספר פונקציות, אשר אחראיות לכך. למשל גם פונקציות המספקות פרסרים, פונקציות היוצרות פרוטוקול SMTP וכו', עד שמגיעים למצב בו מתקבלת הודעת דוא"ל ומפורשת בהצלחה.

כאשר מדובר בבדיקות e2e, אנו רוצים לבדוק כי אנו מקבלים מסך login, וכאשר מזינים משתמש וסיסמה נכונים, עוברים לדף הבא. בדיקות e2e למעשה מקבלות תוצר של מספר לוגיקות אשר יוצרות תוצר אותם המשתמשים מקבלים.

כלומר הבדיקה היא קבלה של מסך מסוים, האם יש שם שדות שמחפשים אותן, האם יש כפתור לכניסה, האם לחיצה עם שדות מלאים מספקים את מה שציפינו לו, כגון כניסה בהצלחה או הזדהות שגויה. בדיקת e2e היא למעשה בדיקה מה החוויה מבחינת סדר פעולות אשר המשתמש נדרש לעשות והאם פעולות אלו עוברות בהצלחה.

כעת, לאחר שעברנו יישור קו עם המונחים וכיצד אני מתייחס אל אותם המונחים, ניתן להמשיך.

מדוע יש בעיות עם בדיקות?

לפני מספר שנים הייתי בהרצאה על כמה חשוב לקבל מערך בדיקות (התשובה: כי הוא למעשה 'התנ"ך' של המערכת). במידה ויש כיסוי מאוד מסיבי, סביר להניח כי אפשר להבין דברים רבים על המערכת, אבל מערך בדיקות אינו מראה את איכות המערכת, ובוודאי שלא מצביע על העדר בעיות ובאגים.

יותר מכך, יכול להיות שכלל המערך בודק דברים שגויים, וככזה הוא מקשה על הבנה כיצד המערכת בנויה, וזאת - במקום לספק את היכולת להבין נכון את המערכת.

ואם לא די כאן, כבר נתקלתי במערך בדיקות אשר הקוד היה מאוד קריפטי ורק מי שכתב את הבדיקות הבין כיצד הוא מבצע את הבדיקות בפועל. כלומר הבנתי לפי השם של הבדיקה מה היא בודקת, אבל לא היה ברור בכלל כיצד זה אמור להגיע לידי ביטוי, אך איכשהו הבדיקה עבדה, ולי לא היתה דרך להבין אם היא באמת בודקת את מה שרצה אותו מפתח או לא.

בדיקות תוכנה הן למעשה עוד עטיפת קוד, אשר מחביאות בעצמן בעיות נוספות. הבדיקות מציגות כי מה שנבדק עובד, אך בצורה אשר רק מי כתב אותם רצה. וזאת כל עוד המערך בדיקות אינו מכיל באגים ובעיות אחרות בעצמן.

בדיקה לא נכונה, או הבנה לא נכונה של תפקיד הפונקציה, יכולה לתת תוצאה נכונה מסיבות לא נכונות. ובכך הבדיקות למעשה אינן משקפות את תקינות המערכת. כלומר בדיקות יכולות גם לספק false positive או true negative. הסיבות לכך הם שיש מספר רב של נעלמים שאיננו מודעים להם.

למשל עם עורך טקסט, כאשר הוא מקבל תו null, מה קורה? נגיד והתו הראשון הוא null, האם העורך יגיב בצורה זהה כמו התו שני? מה קורה אם רק בתו ה-10,000,005 יקרה משהו? סדרה של כמה null

עלולה לגרום לערוך הטקסט להתנהג בצורה לא צפויה? פשוט לא ניתן לחזות כל דבר. זו הסיבה כי באגים כדוגמת הבאג הזה, לפעמים מתגלים רק במצבים קיצוניים בלבד.

עוד הדגמה היא, יצירת מערכת שתוכל להתמודד עם Yottabyte של מידע. בזמן כתיבת שורות אלו, אני חושב כי מדובר במשהו תיאורטי בלבד, אבל כיצד המערכת תתמודד עם גודל מידע אסטרונומי שכזה?

הבעיה על Yottabyte נשמעת כנושא תיאורטי בזמן כתיבת שורות אלו, אך בפודקאסט ששמעתי לפני הרבה מאוד שנים, דובר על באג אשר היה במימוש של XFree ששם ה-Buffer לא בדק חריגה של גודל מידע מסוים אשר בזמן הכתיבה של המערכת, זה היה גודל תיאורטי בלבד, ולא היו בכלל מחשבים אשר הכילו את אותו הגודל, וכאשר הטכנולוגיה סיפקה גודל שכזה, התגלה באג שתורגם בסופו של דבר ל-root elevation עם buffer overflow ומספר בעיות נוספות.

כאשר אני מציג בעיות כאלו, התשובות המתקבלות הרבה פעמים הם: "אז אתה אומר כי בדיקות תוכנה אינן צריכות להתקיים?". ובכן, התשובה היא שיש להן הרבה מקום, אבל חובה להבין בדיוק את תפקידן, והוכחה של היעדרות בעיות אינן אחת מהן.

התשובה היא לא קיצונית, אין כאן מצב של "הכל או כלום", אין מקום לגישה שכזו, היות ויש מקום למערך הבדיקות, אך חשוב להבין מה התפקיד שלהן ואיפה הוא נותן תחושה שגויה של ביטחון.

כמה בדיקות צריך בשביל להחליף נורה?

אנחנו מכירים את זה: מגיע מנכ"ל של חברה, עומד על במה, מדגים את המוצר החדש של החברה לקהל גדול, גם אנשי תקשורת, כולם בהתלהבות של ההשקה, אותו מדגים מדבר כי על כל 10,000 שורות קוד של המערכת יש כ-100,000 בדיקות, וזה המוצר עם הכי הרבה בקרת איכות שאי פעם בוצע בחברה, ו... אז תוך כדי הצגה של המערכת - היא קרסה למדגים.

אם יש בדיקות רבות כל כך, איך עדיין יש באגים? או לחילופין:

מדוע בדיקות אינן מוכיחות העדר באגים?

בעולם אבטחת המידע יש כלי הנקרא fuzzer. התפקיד של הכלי הוא לייצר מידע רנדומלי בתבניות שונות על מנת למצוא בעיות בתוכנות. זה למעשה אמצעי בדיקות אוטומטי אשר תפקידו למצוא נקודות כשל בתוכנה שלא סביר לחשוב עליהן לבד.

כלומר, אני כבן אדם מתקשה הרבה פעמים לחשוב על הרבה מאוד תבניות למערכת שאני בודק, ונוסחאות אוטומטיות שהמחשב מבצע עבורי מוסיפות עוד מידע שניתן לבדוק אותו במערכת אותה אני מפתח.

כאשר, אני מתכנן ממשק משתמש, אשר ניתן להוסיף רשימה באמצעות פסיקים של כתובות IP. מגיע משתמש ולא קורא את ההוראות ושם לי רווח או שורה חדשה פר כתובת IP, מה עכשיו? כיצד המערכת שלי תגיב? ומה אם הוא הכניס לי מפריד אחר? האם אני יכול לנחש כל סוג מפריד שהוא?

היות ואין לי יכולת (או זמן) לבדוק כל סוג קלט אפשרי, בכל מבנה אפשרי, אין לי יכולת לכתוב בדיקות אשר יצליחו לבדוק כל מבנה אפשרי וסדר אפשרי של כל תו בכל אינדקס אפשרי, ארצה להשתמש ב-fuzzer על מנת לנסות ולכסות מקרים רבים יותר ככל הניתן.
אך גם כלים כדוגמת fuzzer אינם מסוגלים לייצר את כל האופציות. הם יכולים ליצור dataset מסוים ולהגיע לקלטים רבים יותר מבני אדם, אך ה-fuzzer עדיין לא יכול לספק מענה הולם לכל מקרה אפשרי. השימוש בבדיקות חייב להיות שונה בתפיסה שלו ובשימוש שלו. ההבנה כי באגים יכולים להתקיים, לא משנה מה כמות הבדיקות, היא חשובה מאוד בנושא.

סיבה לכך כי גישה הזו של "יש לי באגים במערכת", אומרת כי יצרן המערכת אינו נתפס בהפתעה, ובכך בניהול הבעיה הוא מגיע מוכן, במידה והיא פשוטה או במידה והיא חמורה עדיין יש מקום רב יותר להתמודדות עם הבעיה ותיקון טוב יותר שלה.

גם בדיקה היא קוד

"Quis custodiet ipsos custodes?" הוא משפט לטיני השואל "מי ישמור על השומרים?" כיצד אפשר להבטיח כי בדיקה אינה מכילה באגים בפני עצמה? הרי גם בדיקה היא למעשה קטע קוד, ויש סיכוי כי גם לה יש באגים. ולכן, בדיקות צריכות להתבצע, אך יש לקחת אותן בערבון מוגבל.

האם כאשר בדיקה נכשלת, זה בהכרח אומר כי הקוד שאותה הבדיקה בודקת שגוי? האם כאשר בדיקה עוברת, זה בהכרח אומר כי הקוד שאותה הבדיקה בודקת תקין?

התשובה לשתי השאלות הללו היא כמובן "לא".

אם עד עכשיו דיברתי באוויר, ארצה להדגים לכם את הבעיה, ואשתמש בקוד Go לשם כך:

```
func IsOne(input string) bool {
  return input == "1"
end
```

שורת לוגיקה אחת, בודקת האם מחרוזת מכילה רק את הספרה "1" כמחרוזת בתוכה, נכון? אצור לה כעת בדיקות:

```
func TestIsOneWithZero(t *testing.T) {
   result := IsOne("0")
   if result == true {
     t.Error("`result` expected to be `false`, but got `true`")
   }
}

func TestIsOneWithOne(t *testing.T) {
```

```
  result := IsOne("1")
  if result == false {
    t.Error("`result` expected to be `true`, but got `false`")
  }
}
```

האם הבדיקות אשר כתבתי למעלה בודקות את כל המצבים האפשריים עבור שורת לוגיקה אחת של IsOne?

תחושת הבטן אומרת לי שכן, אבל הניסיון אומר לי שלא!

אתחיל בהתחכמות, ואגיד כי אני בודק רק את התו "1" באסקי, אך למשל מה עם 'Mathematical bold digit one'

אני יודע, אפשר להגדיר כי התפקיד של אותה הפונקציה היא להחזיר רק אם הערך אסקי של "1" מתקיים. במידה ואגדיר כי רק הערך הזה מתקיים עבורי כערך של הספרה "אחד", פתרתי את הבעיה הזו.

מה קורה כאשר אני מקבל מחרוזת של מספר תווים שכולם הספרה "1"? האם יש הגדרה כמה פעמים מופיעה הספרה הזו? האם אני צריך להגדיר כי הופעה חד פעמית בלבד נדרשת?

לאחר הגדרות אלו, זה אומר כי אני צריך לכתוב עוד בדיקות, ולהבטיח כי כל הגדרה אשר אני רוצה כי לא תתקיים, בהכרח לא מתקיימת.

ומה אם פתאום גיליתי תו חדש בתקן Unicode שרק פורסם אשר מכיל את הספרה "1" בצורה אחרת, כיצד הקוד שלי יתמודד אז? במצב כזה, תמיד אני במירוץ אחר מציאת הגדרות מאוד מדויקות מה נכון ולהבטיח כי כל דבר אחר יחזיר לי "לא נכון".

והנה באג במערכת: מסתבר כי כתבתי את הפונקציה IsOne בצורה מסובכת אשר מרגישה כטובה. מה הכוונה?

```
func IsOne(input string) bool {
  return len(input) == 1 &&
         rune(input[0]) == 0x31
end
```

אני בודק האם האורך הוא "1", כי אם לא, אין טעם לבדוק את השאר, נכון?

כעת, על מנת להבטיח כי קיבלתי באמת את הספרה "1", אני בודק האם מה שנמצא לי בתא "0" במחרוזת זה בעצם הערך האסקי של "1". בשורה של rune, בעצם אני מחביא היתכנות לבאג. למי שאינו יודע, ב-Go, המחרוזת מכילה כברירת המחדל UTF-8, מה שאומר שתו אחד יכול להיות בין בית בודד עד לארבעה בתים. אני בודק אורך של תו, ולא אורך של בית. אני בודק עם rune האם התו (אשר יכול להיות להיות עד 4 בתים) הוא 0x31. עד כאן הכל תקין. אבל האם ידעתם כי Unicode כתקן מחזיק בתוכו מבנה ביטים מסוים אשר יכול להבטיח כי זה אינו בית שלם?

איננני יודע להגיד כי המצב בקוד מספק לי מענה לכך. כלומר אני סומך על המימוש של שפת Go כי כך זה הדבר, אך איננני בודק זאת בעצמי. כך שאני לא מכסה את כל האפשרויות, ואני יכול להחביא באג פוטנציאלי אשר אינו קשור ישירות לקוד שלי, אך כן יכול להתרחש.

קוד כל כך פשוט כביכול, יכול להחביא בתוכו בעיות אשר ללא ידע או ניסיון קודם יכולים לצוץ ללא הבנה מוקדמת.

התפקיד של הבדיקות הן לא להבטיח כי לא יהיו בעיות, אלא להבטיח כי מה שאני מכיר ויודע שיכול להתרחש לא יתרחש. כך שהבדיקות שומרות על קוד מפני בעיות ידועות ומוכרות, במידה ואינן יוצרות false positive או true negative, אך זה כל התפקיד שלהן.

סיכום

במאמר זה סיפקתי תיאוריה המסבירה מדוע מערך הבדיקות המוכר אינו אמצעי להבטיח כי אין באגים או בעיות בקוד קיים. ההדגמה הקצרה שלי נועדה לספק גישה נגדית לגישה הרווחת כי תפקיד הבדיקות הוא להבטיח כי אין באגים או בעיות אבטחה.

חשוב להדגיש כי אין מאמר זה בא להציע כי מערכי בדיקות הוא דבר מיותר או שניתן להתעלם מהצורך בו, אלא תפקיד המאמר הוא להסביר מה המקום של מערך בדיקות, וכי אין לבלבל בין השימוש במערך בדיקות לבין הרצון להבטיח כי אין באגים או בעיות.