**תכנות מערכות דפנסיבי**

**ניתוח איומים לפרויקט MessageU – ממן 15**

**07.11.2021**

**נכתב על ידי דניאל בוזקוב**

**323237172**

רקע

הפרוייקט MessageU הינו רשת חברתית אשר מאפשרת למשתמשיה תקשורת מוצפנת מקצה לקצה, קרי E2EE.

התקשורת מבוססת ארכיטקטורת שרת-לקוח, בה המסרים בין הלקוחות עוברים תחילה בצד השרת לאחסון, ומאפשרת למשתמשים למשוך הודעות מיועדות להן בעת הצורך.

לטובת קיצור המסמך, לא נפרט את כלל התכולות הנדרשות שכן ניתן למצוא אותן במסמך פירוט המטלה.

מטרת מסמך זה היא לסקור את החולשות והאיומים האפשריים המוצעות בפרוטוקול, בארכיטקטורת ובתפעול הפרוייקט.

המסמך יציג קודם לכן הנחות מוצא עליהן הוא יתבסס בסקירת האיומים, משאבים עליהם ישנה חשיבות להגן, והצעות לתיקון החולשות והסרת האיומים.

המסמך יבנה לפי סיווג התוקפים לפי סבירות בהתאם להנחות המוצא אשר יוצגו בהמשך.

נכסים להגנה

המידע הבא יוגדר כחשוב לצורך המשך המסמך, ונקבע שיש להגן על נכסים אלה בעדיפות ראשונה.

* **הקובץ me.info (נתייחס כ'אשרה' מעתה והלאה)**

קובץ זה מכיל את הפרטים האישיים של כל משתמש:

* שם המשתמש
* המזהה היחודי שלו
* המפתח הפרטי שלו

איבוד קובץ זה מאפשרת לכל אחד להתחזות למשתמש שלו הקובץ הזה שייך.

נוסיף עוד כי מצד התוכנה אין אפשרות לזהות אם מדובר באשרה המקורית של המשתמש או באשרה גנובה. יתר על כן, שינוי תוכן האשרה ימנע ממשתמש להמשיך להשתמש במערכת בצורה תקינה עד למחיקת האשרה מחדש. פירוט נוסף בהמשך ניתוח האיומים.

* **המפתח הסימטרי של המשתמש**

המפתח הסימטרי משמש את הלקוח לתקשורת מוצפנת מקצה לקצה עם לקוח אחר.

דליפת המפתח תאפשר הצצה לתוכן התקשורת המוצפנת.

שינוי ערך המפתח הסימטרי תדרוש תיאום מפתח מחדש.

* **פרטיות המשתמש**

כאמור מדובר בפרוייקט התומך בהצפנת מידע מקצה לקצה ושימוש במזהים למשתמשים.

נרצה להגן על פרטיות המשתמשים.

* **השרת**

פעולת השרת השותפת הכרחית להמשך ריצת הפרוייקט.

* **מסד הנתונים**

כאן מוחזק מידע רגיש על כלל משתמשי המערכת. שינוי המידע יכול להטעות את כלל המשתמשים.

הנחות מוצא

* טווח בטוח בין המשתמש עצמו לבין המכונה המריצה את תוכנת הלקוח.

כלומר, אין צד מתווך בין המשתמש למכונה, והמגע ישיר ולא על גבי טווח תקשורת לא מאובטח.

* המכונה מריצה את תוכנת הלקוח על גבי מערכת הפעלה מודרנית עם שיקולים אבטחתיים, בפרט חלוקת זיכרון בין תהליכים ותהליכונים במערכת והגנה על גישה למערכת הקבצים.

השימוש במערכת הפעלה עם דגש אבטחתי (למשל Windows, Ubuntu) אינה מונעת איומים, אך מורידה את סבירותם לצורך המשך הניתוח.

* כתובת השרת הנמצאת בקובץ server.info מתואמת מול השרת האמיתי.
* למשתמש גישה ישירה למערכת הקבצים על המכונה שלו לצורך שמירת וקריאת האשרה.
* לתוקף יש גישה למבנה ההודעות והפרוטוקול (למשל על ידי השגת תוכנת הלקוח וניתוחה).
* לתוקף יש גישה והיכרות מלאה עם תוכנת צד הלקוח, אך לא עם תוכנת צד השרת.
* התוקף ישתדל לשמור על חסויותו, משמע לא יבצע פעולות רועשות כמו לחסום את התקשורת מצד הלקוח.

ניתוח איומים בסבירות גבוהה

1. **פרוטוקול**

איומים הדורשים האזנה לתקשורת ללא כלים מפותחים נוספים.

* 1. **מידע גלוי**

כאמור התקשורת מול השרת אינה מוצפנת, כלומר המזהים חשופים לכל אחד.

חשיפת המזהה מאפשרת לבצע התאמה בין מזהים למשתמשים פוטנציאליים (שכן השם באשרה אינו בהכרח מצביע על האדם ישירות), למשל על פי שעות הפעילות, תדירותן, ומעקב אחרי הנמענים של ההודעות המוצפנות.

מעבר לעולם התקשורת, דליפת מידע זו מאפשרת הצמדה חד חד ערכית בין אדם משתמש לבין השימוש שלו בפרוייקט ובכך פוגעת בפרטיותו.

פתרון:

הצפנת התקשורת בין הלקוח לשרת בצמד מפתחות שונה על מנת לאפשר טווח תקשורת בין השרת ללקוח, ולהסתיר גם את המזהים.

* 1. **התקפת Anti-Replay**

לכל הודעה בודדת אין מזהה ייחודי.

על כן, תוקף אשר מאזין ברשת יכול להסניף הודעה ועל אף שאינו בהכרח מסוגל להצפינה הוא מסוגל לשמור אותה, ולשלוח אותה שוב ושוב, והצד המקבל יאלץ לאשר את ההודעות שכן הן תקינות ברמת הפרוטוקול ודורשות טיפול.

מכיוון שאין הלקוח מאזין לבקשות באופן קבוע, אינו מהווה מטרה עיקרית לתקיפה זו.

אך השרת מעצם היותו ה slave בתקשורת מאזין באופן קבוע ומצפה לטפל בהודעות.

הנספת הודעות תקינות מאפשרת להעמיס על השרת מעבר למצופה בפרוייקט עד לרמה שהעומס יורגש ברמת המשתמשים שכן לשרת יקח זמן להתפנות לטיפול בבקשות אמיתיות ולא העתקות.

תיאור תקיפה לצד השרת:

* + הסנפת בקשת שליפת לקוחות בשלב מתקדם במצב בו טבלת המשתמשים עמוסה בשרת.
  + שליחת הודעה זו כמות גדולה של פעמים.
  + השרת יאלץ לטפל בבקשה מעמיסה פעמים רבות ולא יתפנה לטיפול בהודעות של לקוחות.

תקיפה לצד הלקוח:

* + שליחת בקשת "הודעה מוצפנת למשתמש" פעמים רבות לשרת, ובכך למלא את תיבת הדואר של המשתמש ולהקשות בפענוח כל ההודעות.

פתרון:

הוספת מספור וחתימה להודעות הפרוטוקול ושימוש בחלון זז.

באופן זה, הודעות ישנות והודעות מטופלות לא יטופלו שוב על ידי השרת, ושינוי תוכן הודעה יזוהה בעת וידוא החתימה של ההודעה בשרת.

* 1. **DoS**

במסגרת הרישום, נשלחת בקשת הרשמה עם שם המשתמש ומפתח פומבי.

לפי הגדרת הפרוטוקול, על השרת להחזיר הודעת שגיאה במקרה בו מתקבל שם משתמש אשר כבר קיים בבסיס הנתונים.

תוקף ברשת יכול לאתר בקשת הרשמה, להבין את המבנה שלה, ולשלוח לשרת בקשת הרשמה יחידה לכל שם משתמש אפשרי בעזרת מאגרי שמות משתמשים נפוצים אשר נגישים ברשת (לפחות שמות אנושיים עד 100 תווים).

מהלך זה ימלא את העמודות בצד השרת עבור כל השמות האלה, ותמנע ממשתמשים חדשים להירשם בעזרת שם פשוט (או כל שם, אם לתוקף היה מספיק זמן לשלוח בקשות אך זה פחות סביר). בכך השרת לא יוכל לאפשר הרשמה לאף משתמש חדש.

פתרונות:

* + להמנע מיוחודיות השם ולהיעזר במזהה משתמש.
  + להצפין את התקשורת בין הלקוח לשרת.
  + למספר את ההודעות ולחתום עליהן בדומה לסעיף הקודם.
  1. **שגיאות לוגיות**

בעת משיכת ההודעות של הלקוח מן השרת, ישנה חלוקה למספר קטגוריות של הודעות:

* + בקשת מפתח סימטרי
  + שליחת מפתח סימטרי
  + הודעת טקסט

האכיפה על תוכן ההודעה לא מוגדר ברמת הפרוטוקול, מה שמאפשר ללקוח לשלוח הודעת טקסט המכילה את התווים “Request for symmetric key” או “Symmetric key received”, וללקוח השני לא תהיה דרך לזהות אם מדובר בתוכן של הודעת טקסט של לקוח נוסף או בהדפסת מידע המובנית בתוכנת הלקוח, ובכך עלולה לגרום ללקוח לפעול שלא בהתאם

פתרון:

להוסיף לתחילת ההודעה (בנוסף לכותרת המוען וכתורת התוכן) כותרת לסוג ההודעה.

* 1. **תיאום מפתח כפוי**

בהתאם לפרוטוקול המוגדר, אין דבר המונע מלקוח לשלוח מפתח סימטרי ללקוח אחר אשר כלל לא ביקש לתאם את המפתח.

אי אכיפה זו מאפשרת לכל אחד לשלוח הודעות מוצפנות במפתח סימטרי לכל שאר לקוחות הרשת (שליחת הודעת מפתח סימטרי ולאחר מכן תוכן נוסף המוצפן באותו מפתח אשר תוכנת הלקוח השני תפענח ותציג לפי האפיון).

אומנם לא מדובר בחלושה אבטחתית, אך ישנו מקום להספמה (מלשון spam) של כלל הלקוחות.

פתרון:

* + תיאום המפתח בשיטה המערבת אקטיבית את שני הצדדים (למשל דיפי-הלמן), כך שקשר בין שני צדדים יתבצע רק בהסכמה של שני הצדדים.
  + אכיפה על שיחות בצד הלקוח, כך שהודעות תוכן המתקבלות על ידי לקוח שטרם תיאם מפתח לא יתקבלו בכלל (ולא יבוזבזו משאבים על ניסיון פענוח), ושהודעת שליחת מפתח סימטרי לא תתקבל מלקוח שלא נשלחה לו בקשה לקבלת מפתח סימטרי.

1. **צופן**

איומים הדורשים האזנה לתקשורת, הבנה קריפטולוגית עמוקה ויכולות עיבוד חזקות.

* 1. **הצפנה כפולה**

נבחין כי אף מפתח צופן אינו מוחלף.

אם כן, על ידי הצפנת מידע אשר ניתן לחיזוי או שימוש בהצפנות רבות, תוקף יכול לשבור את הצופן על ידי brute force מושכל.

* + **מפתח סימטרי**

אורך חייו של המפתח הסימטרי אומנם קצר יותר יחסית למפתח הפרטי, אך ניתן לצפות להצפנת תוכן אשר ניתן לחיזוי, למשל תחילת הודעה ב"שלום" או ההדר של מבנה הקובץ.

ההצפנה הכפולה מקלה על שבירת הצופן ומעמידה את המפתח בסכנה.

פתרון:

החלפת המפתחות לאחר פרק זמן קבוע או לאחר מספר שימושים אשר קבוע מראש.

פעולה זו תלווה בעדכון השרת בחלופת המפתח הפומבי, ועדכון הלקוח השני בעדכון המפתח הסימטרי או תיאום מפתח מחדש.

ניתוח איומי המשך – לאחר השמשת חולשות מערכת

1. **תוקף בצד לקוח**

איומים האפשריים עקב גישה של תוקף למכונה של הלקוח.

1. **שינוי האשרה**

במידה ותוקף הצליח לגשת למערכת הקבצים של המשתמש ולשנות את האשרה (למשל על ידי DirtyCow או תקיפות Privilege Escalation אחרות), הלקוח לא יוכל להשתמש בה שנית לצורך התחברות ויהיה עליו למחוק את האשרה התקולה ולהירשם מחדש.

פתרונות:

* + שימוש במערכת הפעלה מרודדת ולצמצם את משטח התקיפה במערכת עצמה.
  + במידה ואפשר, לנעול את הזיכרון בו נשמרת האשרה במנגנון חומרתי לקריאה בלבד.

1. **הדלפת המפתח הסימטרי והפרטי**

על פי ההגדרה, כלל המפתחות נטענים אצל המשתמש בזמן ריצה ועל כן שמורים בערימה.

תוקף מתוחכם יכול למפות את כלל הזכרונות הממופים בגדלים הנדרשים למפתחות ובכך לאפשר לעצמו גישה ישירה למפתחות אותם מקצה המשתמש בזמן הריצה.

בכך יכול התוקף להמשיך ולהסניף את התעבורה אשר תהא גלויה לחלוטין.

פתרונות:

* + שימוש במערכת הפעלה מרודדת ולאפשר הקצאות דינמיות אך ורק לתוכנת הלקוח.
  + שינוי הארכיטקטורה לאפשרת שמירת המפתחות בצורה סטטית.

1. **שינוי server.info להתחזות או MITM**

במידה ולתוקף ישנה אפשרות לשנות קבצי מערכת, אין דבר המונע ממנו לנתב את כלל התעבורה דרך מכונה אישית, בה הוא מסוגל להחזיר כל תשובה שירצה.

דוגמה למימוש התקפה עם התחזות:

* + רקע: קיימת ידיעה שאיש A אמור לשלוח מידע חשוב וסודי לאיש B בעזרת MessageU והם לא מתקשרים במציאות.
  + התוקף משנה את פרטי Server.info במכונה של איש A, ומנתב את הבקשות של A למחשבו האישי של התוקף.
  + התוקף מזייף רשומה של איש B כתשובה לבקשת רשימת המשתמשים, ומוסיף מפתח פומבי שהוא יצר.
  + איש A יתאם מפתח סימטרי מול התוקף במחשבה שהוא מתאם מול B וישלח לו את המידע הסודי.

דוגמה למימוש התקפה עם MITM:

* + רקע: קיימת ידיעה שאיש A אמור לשלוח מידע חשוב וסודי לאיש B בעזרת MessageU והם מתקשרים במציאות.
  + התוקף משנה את פרטי Server.info במכונה של איש A, ומנתב את הבקשות של A למחשבו האישי של התוקף.
  + התוקף משנה את פרטי Server.info במכונה של איש B, ומנתב את הבקשות של B למחשבו האישי של התוקף.
  + כאשר A מבקש את רשימת המשתמשים, התוקף מזייף רשומה של איש B כתשובה ומוסיף מפתח פומבי שהוא יצר.
  + כאשר B מבקש את רשימת המשתמשים, התוקף מזייף רשומה של איש A כתשובה ומוסיף מפתח פומבי שהוא יצר.
  + איש A ואיש B יתאמו מפתח סימטרי מול התוקף במחשבה שהם מתאם אחד מול השני.
  + איש A ישלח את המידע הסודי המפתח הסימטרי שלו מול התוקף, והתוקף יפענח את המידע עם המפתח הסימטרי ויקרא את התוכן.
  + התוקף יצפין את המידע עם המפתח הסימטרי שתיאם מול B וישלח לו.

פתרון:

* + טעינת התוכנה עם המפתח הפומבי של השרת.
  + בקשת אשרה מהשרת אשר ניתן לוודא רק בעזרת מפתח וידוא אשר טעון מראש בתוכנה.

1. **הזרקות SQL**

הפירוט על ההתקפה מוצג בחומר הלימוד ביחידה 7.

פתרון:

להמנע מהחדרת נתונים של הלקוח ישירות לשאילתות ה SQL. למשל על ידי שימוש מראש ב Django, אשר מבצע הפרדה בין בניית השאילתא לבין קליטת הפרמטרים לשאילתא.

1. **תוקף בצד השרת**

איומים מצד תוקף בצד השרת מסוכנים מאוד וקשה להתגבר עליהם, ולכן נניח כי הם פחות סבירים וישנה בקרה מעמיקה על הפעילות על חומרת השרת.

1. **הדלפת מידע ללקוח יעד בצורה "שקטה"**

תוקף בצד השרת עלול לרצות להדליף מידע על גבי המערכת למשל לצד לקוח אשר משתף פעולה.

דבר זה נוח להפליא בזכות הפרוטוקול המאפשר שליחת הודעות מוצפנות בגודל דינמי, כך שלפי הגדרה אין באפשרות השרת לסנן את תוכן ההודעה היוצאת עד כדי תאימות בגודל.

מידע זה יכול לכלול כל דבר בין רשימת קריאות המערכת אותן מבצע השרת (בעזרת קישור להשמשת חולשה אחרת) ועד לניטור תעבורה כללי.

בהנחה ונמצאה גם יכולת הזרקת קוד מרחוק, הדפלה זו מאפשרת ערוץ שלם של תקשורת בין התוקף המרוחק לבין צד השרת ואף הרצת shell נוחה עם חיווי לתוקף (העברת פלט הפקודות חזרה דרך הערוץ ה"מוצפן" הדינמי).

פתרון:

* + לצמצם עד כמה שניתן שימוש בגדלים דינמיים, או להגדיר גודל הודעה מוצפנת מקסימלי.
  + הפרדת תפקידי השרת ברמת חומרה או לפחות ברמת פרוססים בין ניהול מידע לבין סינון וניטור המידע הנכנס והיוצא.
  + הקמת צד ניטור מאובטח שיעקוב אחר תעבורת השרת לזיהוי אנומליות כגון זו, או מעקב אחר כניסת והוצאות ההודעות ללקוחות (כלומר לוודא כי לא ייתכן כי לקוח ישלוף יותר הודעות ממספר ההודעות שנשלחו לו).

ניתוח איומים בסבירות נמוכה

1. **חולשות specter, meltdown, וגישה ישירה אחרות**

השמשת חולשות מהמשפחה הזו תאפשר לתוקף בעל גישה ישירה למכונה לגשת לאשרת המשתמש ואף למפתח הסימטרי במידה והספיק ליצור אחד.

בצד השרת התקיפה תאפשר לתוקף לגשת לכלל מסד הנתונים המקומי אותו השרת מתפעל.

חולשות אלה דורשות מהתוקף גישה ישירה למכונות ולכן פחות סבירות.

פתרון:

להגן על החומרה, ולהרחיקה מזרים ואנשים שלא אוהבים אותך במיוחד.

1. **חולשות מימוש אשר האפיון אינו מתייחס אליהן**
2. **RCE via pickle deserialization**

בצד השרת נדרשים לבצע Deserialization בפייתון של מידע הנשלח מהלקוח.

לא הוגדר מראש מהו מנגנון הדי-סרלוז הרצוי לפרוייקט.

על כן, ניתן בקלות דעת לבחור ב Pickle, אך ברמת המימוש של המודול נבחין כי ישנו פתח לתקיפה.

מתוך התיעוד של המודול:

Whenever an object is pickled, the \_\_reduce\_\_ method defined by it gets called. This method returns either a string, which may represent the name of a Python global, or a tuple describing how to reconstruct this object when unpickling.

אם כן, דיו לתוקף להמיר לבתים בעזרת Pickle מחלקה זדונית הממשת את \_\_reduce\_\_ להרצת קוד בצד השרת.

פתרון:

לא להשתמש ב Pickle.

1. **הגדרת פרק ה Timeout לשמירה על משאבים**

לא הוגדר זמן Timeout בהמתנה לבקשה מלקוח.

מצד אחד, הפרוייקט לא שואף לספק שירותים ללקוחות מרוחקים ולא לוקח אחראיות על בעיות קליטה או תקשורת אפשריות, אבל על פי הגדרת הפרוייקט מתאפשרת הגדרת Timeout ארוכה אשר פותחת חלון לשלל תקיפות משאבים כגון Slow Loris.

פתרון:

להשתמש בזמן המוקצב הדיפולטי, או להגדיר פרק זמן קצר המאפשר תקשורת מהירה.

פתרונות והקשחות כלליות

מעבר לכל המאמצים למניעת התקפות, נרצה גם להקשות על ביצוע התקפות אשר אין ביכולתנו למנוע. נציג מנגנונים ושיקולים לצורך קידום מטרה זו:

1. **רוורסינג**

תוקף בעל גישה לקובץ ההרצה של צד הלקוח יכול ללמוד המון על התנהלותו על ידי רוורסינג פשוט. על מנת להקשות עליו כמה שיותר ולהגן על קבצי ההרצה אשר יפורסמו, נרצה להוסיף שלבי Obfuscation בשלב ההידור כנלמד במהלך השיעורים בסמסטר.

דרך פעולה אחת היא להגדיר זאת בשלב ההידור. זאת ניתן לבצע בקלות בעזרת LLVM, או התקנת תוספות ל GCC (למרות שהקומפיילר לא הוגדר באפיון).

אפשרות נוספת היא קימפול סטטי של תוכנית הלקוח, הצפנת קובץ ההרצה המקורי, וקישורו לקובץ הרצה אחר אשר יכיל אותו, יפענח אותו ויפרוס אותו בזמן הריצה. פעולה זו מונעת ניתוח סטטי של קבצי ההרצה ומקשה על תוקף אפשרי אשר ירצה לדלות מידע לגבי פעולת הלקוח.

1. **צמצום משטח התקיפה במערכת צד השרת**

כמובן שאין באפשרותנו להגביל את סביבת הריצה בצד הלקוח, אך יש באפשרותנו להגביל את סביבת הריצה של צד השרת.

במידה ואפשר נדאג כי השרת ירוץ על גבי מערכת הפעלה מצומצמת אשר תכיל אך ורק את מה שהכרחי לריצת השרת.

בכך נחסך קיום של דרייברים או ממשקים מסוכנים פוטנציאלית ומשטח התקיפה על השרת קטן.

דוגמה אחת למערכת כזו הינה hardened-kernel.

1. **מנגנוני הגנה למחסנית מכל הרמות**

מכיוון שיש באפשרותנו להגדיר ולהשפיע על סביבת הריצה של השרת, נרצה להשתמש במנגנונים חומרתיים להגנת המחסנית כגון CET ( Control-flow Enforcement Technology ) מעבר למנגנונים תוכנתיים בלבד כגון קנריות.

בצד הלקוח נוכל לאפשר אך ורק מנגנונים תוכנתיים אשר אפשריים ברמת ההידור כגון קנריות בלבד, שכן אין באפשרותנו להגדיר את סביבת הריצה של הלקוח.

1. **הפרדת מסד הנתונים מן השרת**

מסד הנתונים ופעולת השרת הוגדרו כנסכים שיש להגן עליהם. אם כן, צימודם מהווה סכנה מהותית שכן בעת התקפת השרת, מסד הנתונים כבר חשוף לתקיפה.

אם כן, יש להפרידם לגופים שונים, שכן שני הנכסים אינם דרושים את אותה סביבת הריצה.

כל סביבה תהיה מותאמת לנכס אותה היא מכילה, למשל נאפשר מודלים של פייתון אך ורק בסביבה בה רץ השרת, ובסביבת מסד הנתונים נדאג להרצת תהליכון יחיד עם הרשאות בלבד.

הערה: כל הדוגמאות לרידוד סביבת הריצה אפשריים ופותחו כבר כלים לניהולם.

1. **CI/CD**

שילוב של תהליך פיתוח תקין ומבוקר, המשלב אנליזה סטטית ודינמית מסוגל למנוע או לפחות לצמצם שגיאות אנוש בזמן פיתוח הפרוייקט, ובכך להקטין את משטח התקיפה האפשרי.

שיקולים פיתוחיים אבטחתיים

* על מנת להבטיח התנהגות צפויה ככל הניתן, בצד הלקוח נעשה שימוש מקיף בטמפלייטים במקום מחקלות ובכך להגדיר את ההתנהגות הרצויה עוד בזמן ההידור.
* בנוסף, השימוש המקיף מאפשרה הקצאות במחסנית בגדלים ידועים מראש הניתנים לאכיפה ומימוש בדיקות בזמן הידור, ונמנע משימוש בוירטואליזציה אשר משאירה מקום לתקיפות מחסנית מגוונות.
* לו הייתי כותב את הפרוייקט בתעדוף אבטחתי, הייתי מוסיף בדיקה בכל פונקציה בצד הלקוח שאכן מדובר בלקוח רשום ולא רק בעת קבלת הקלט מהמשתמש על מנת למנוע הרצת פונקציות במצב לא מוכן ממרחב הכתובות. אך על מנת לא להסתכן בהורדת נקודות על כפילות קוד ושמירה על אסטטיות בחרתי לרכז את האכיפה בפונקציית קבלת הקלט מהמשתמש. אעיר: מראש על מנת להריץ את הפונקציות לכל הבקשות האפשריות יש להשמיש חולשה אחרת ולהריץ קוד ממרחב הכתובות. אכפתי בכובד ראש את הקצאות הזיכרון בתוכנה ואני יוצא מנקודת הנחה שהקוד רץ במערכת הפעלה אשר מפרידה זכרונות של תהליכים, ולכן הרשתי לעצמי לוותר על הבדיקה הכפולה, ומה עוד ששליחת בקשות נוספות לא יתקבלו בצד השרת לו המשתמש אינו רשום.