דף תכנון הפרויקט במבוא לאבטחת המרחב המקוון

תוכן עניינים

כללי:

הפרוטוקול הוא בינארי וממומש מעל TCP.

השרת משתמש ב-selectors כמימוש לריבוי לקוחות.

מרגע שהלקוח התחבר בהצלחה לשרת, יוצג בפניו תפריט הודעות:

0 - שליחת הודעה. יהיה עליו להקליד מספר בן 10 ספרות שאליו ירצה לשלוח הודעה.

1 - קבלת הודעות. השרת ישלח ללקוח את כל ההודעות שהתקבלו עבורו אי פעם.

2 - התנתקות מהשרת.

הלקוח שומר את הפרטים שלו בקובץ client_info.txt בפורמט הבא:

<phone_number>:<sign_up>

<private_key>

למשל:

1231231233:1

—---- BEGIN PRIVATE KEY —-----

MIGHAgEAMBM…

….

—---- END PRIVATE KEY —-----

אם הקובץ לא תקין תבוצע קריאה של מספר הטלפון מה-terminal.

אם sign_up הוא 0 (הלקוח לא היה מחובר), אין מפתח פרטי או שהמפתח הפרטי לא היה תקין, יג'ונרטו מפתחות חדשים.

המידע הנ"ל שנוצר בשלב הרישום (מספר טלפון ומפתח פרטי) נרשם לתוך הקובץ בסיום שלב זה.

מה שומרים בשרת:

טבלת משתמשים: עבור כל משתמש שומרים מספר טלפון ומפתח ציבורי בפורמט PEM.

טבלת הודעות: מי שלח את ההודעה, ואת התוכן המוצפן שלה.

המידע נשמר בעזרת pickle לתוך הקובץ db.pkl בסיום ריצת השרת לצורכי recovery.

מה שומרים בלקוח:

מספר טלפון, האם הוא רשום לשרת (משתנה בוליאני) ומפתח פרטי נשמרים גם ב-RAM וגם בקובץ טקסט בסיום הריצה.

בנוסף להם, נשמרים ב-RAM גם המפתח הציבורי ורשימת אנשי קשר - אלה הם מספרים שכבר תקשרנו איתם בסשן הנוכחי, ושומרים עבורם את ה-shared secret, מפתח ה-AES וה-iv.

שיטת ההצפנה:

שליחת הודעה:

משתמשים שרוצים לשלוח אחד לשני הודעה צריכים להחליף ביניהם מפתח סימטרי.

מכיוון שזאת הצפנה מקצה לקצה, לא נרצה שהשרת ידע מה המפתח.

כדי לעשות זאת, כל אחד מהם יג'נרט זוג מפתחות אסימטריים (ציבורי ופרטי) של אלגוריתם החלפת מפתחות דיפי-הלמן מבוסס עקומים אליפטיים - ECDH.

בחרנו באלגוריתם הזה משום שהוא מספק אותה רמת אבטחת עבור גודל מפתחות קטן בהרבה בהשוואה ל-RSA.

למשל, מפתח RSA בגודל 2048 ביט מספק רמת אבטחה דומה למפתח ECDH בגודל 256 ביט בלבד.

האלגוריתם ECDH נשען על הבעיה המתמטית של למצוא את k (המפתח הפרטי) במשוואה P = G * k, כאשר G היא נקודה קבועה כלשהי ו-P הוא נקודה על העקום האליפטי (המפתח הציבורי).

עקום אליפטי הוא פונקציה מהצורה y^2 = x^3 + ax + b.

הקושי של הבעיה הזאת מספק לנו הצפנה מקצה לקצה🙂 (לפחות עד שימכרו מחשבים קוואנטים בKSP).

תהליך ההצפנה עצמו יקרה באופן הבא:

כל לקוח שפותח את האפליקציה בפעם הראשונה ייצר זוג מפתחות אסימטריים.

כשהוא ישלים את תהליך הרישום לשרת (עוד עליו בהמשך), הוא יכול לשלוח הודעות למספרי טלפון אחרים במערכת.

משתמשת שרוצה להתחיל את הצ'אט (נניח אליס) מבקשת מהשרת את המפתח הציבורי של מי שהיא רוצה לשלוח לו את ההודעה (נניח בוב).

היא מייצרת מפתח AES (מפתח סימטרי) ושומרת אותו אצלה.

את ההודעה הסודית שלה היא מצפינה איתו.

היא מייצרת גם shared secret שמבוסס על המפתח הפרטי שלה ועל המפתח הציבורי של בוב.

את המפתח הסימטרי היא מצפינה עם ה-shared secret, ושולחת לשרת את המפתח המוצפן יחד עם הiv וההודעה המוצפנת.

השרת שולח לבוב את ההודעה הזאת ובוב יכול לבקש את המפתח הציבורי של אליס מהשרת, ולבצע דבר דומה מאוד למה שאליס עשתה.

הוא יכול לייצר את ה- shared secret עם המפתח הפרטי שלו והמפתח הציבורי של אליס.

ככה הוא יכול לפענח את מפתח ה-AES וכתוצאה מכך גם את ההודעה.

כל הודעה שנשלחת בין אליס לבוב ובחזרה מוצפנת בעזרת המפתח הסימטרי, שהשרת לא יודע.

לכן זוהי הצפנת E2EE.

<u>וידוא זהות השרת:</u>

המפתח הציבורי של השרת (מפתח EC) מוטבע באפליקציית הלקוח.

הלקוח מג'נרט challenge רנדומלי ושולח אותו לשרת.

השרת צריך לייצר message digest עם האלגוריתם **sha256**, לחתום אותו עם המפתח EC הפרטי שלו ולשלוח את החתימה ללקוח.

הלקוח בודק את החתימה עם המפתח הציבורי של השרת, ואם אכן החתימה תקינה, זה אומר שאכן השרת הוא זה שמתקשר איתנו.

<u>תהליך הרישום הראשוני:</u>

הלקוח מוודא את זהות השרת בעזרת המפתח הציבורי שמוטבע באפליקציית הלקוח.

הלקוח שולח לשרת את מספר הטלפון שלו.

השרת בודק האם מספר הטלפון קיים כבר, אם כן הוא יחזיר שגיאה.

אחרת, הוא ישלח ללקוח קוד בעל 6 ספרות **בערוץ המאובטח** (בדומה לSMS).

הלקוח ישלח לשרת את הקוד שקיבל ב**ערוץ המאובטח**.

השרת יבדוק האם הקוד נכון.

אם כן, השרת יאשר את הרישום.

אחרת השרת יחזיר שגיאה.

הלקוח ינסה שוב 3 פעמים ואז השרת יחסום אותו (יסגור את ערוץ התקשורת).

תהליך ההתחברות ללקוח קיים:

הלקוח מוודא את זהות השרת בעזרת המפתח הציבורי שמוטבע באפליקציית הלקוח.

הלוקח שולח לשרת את מספר הטלפון שלו ומבקש להתחבר.

השרת עונה ללקוח עם challenge.

הלקוח מחזיר לשרת את החתימה המתאימה ל-challenge הזה.

אופן יצירת והחלפת המפתחות:

כל לקוח יג'נרט מפתח ציבורי ומפתח פרטי על בסיס ECDH (דיפי הלמן מבוסס עקומים אליפטיים), המפתח הציבורי יישלח לשרת והמפתח הפרטי יישאר אצל הלקוח.

לשרת יש מפתח ציבורי גם כן, שהוא קבוע ומובנה באפליקציית הלקוח, ומפתח פרטי שנשמר אצל השרת.

לצורך שליחת הודעות ישתמשו במפתח סימטרי AES-CBC-256.

הצד שרוצה להתחיל את האינטראקציה, יג'נרט מפתח AES.

אם המקבל **לא** נמצא אצלו ברשימת אנשי קשר, השולח ישלח לשרת בקשה לקבל את המפתח הציבורי של המקבל

לאחר קבלת המפתח הציבורי, השולח ייצר את הshared secret מהמפתח הפרטי שלו והמפתח הציבורי של המקבל

השולח ייצר מפתח AES ויצפין את ההודעה שהוא רוצה לשלוח עם ה-AES.

השולח יצפין את מפתח ה-AES עצמו עם ה-shared secret.

השולח ישלח לשרת בפורמט json את מפתח ה-AES המוצפן, ה-iv ואת ההודעה המוצפנת.

השרת יעביר ללקוח היעד את ההודעה.

אם ללקוח היעד אין כבר את מפתח ה-AES מאינטראקציה קודמת עם השולח, הוא יבקש את המפתח הציבורי של השולח.

אחר כך הוא ייצר את ה-shared secret מהמפתח הפרטי שלו ומהמפתח הציבורי של השולח.

הוא יפענח את מפתח ה-AES וישמור אצלו באנשי קשר.

הוא יפענח את ההודעה ויענה (או יסנן).

כפי שאפשר לראות, בטיחות החלפת המפתחות מדהימה.

שלמות במובן הרחב:

מקור ההודעה: וידוא זהות השרת נעשית בתחילת התחברות/הרשמה בעזרת המפתח הציבורי של השרת, שמוטבע באפליקציה. הלקוח מצפין איזשהו challenge שהשרת צריך לפענח ובכך להוכיח שהוא באמת השרת.

וידוא זהות הלקוח נעשית בפעם הראשונה בעזרת קוד בן 6 ספרות, שנוצר אצל השרת ונשלח ב"ערוץ מאובטח" (בדומה לSMS).

כדי שהלקוח יוכל להירשם עם המספר טלפון הזה, הוא צריך להקליד את הקוד לשרת וככה השרת מאמת שבאמת מספר הטלפון הזה שייך ללקוח.

בפעמים הבאות (בקשות התחברות) הלקוח מזדהה בעזרת מפתח אסימטרי. השרת שולח ללקוח איזשהו challenge, ומבקש ממנו להחזיר לו את החתימה של ה-challenge הזה. הלקוח מחזיר חתימה שהוא מייצר בעזרת המפתח הפרטי שלו, והשרת מוודא שהחתימה הזו נכונה בעזרת המפתח הפומבי של הלקוח. אם החתימה נכונה אז הלקוח מצליח להתחבר.

תוכן ההודעה: ההודעה מוצפנת מקצה לקצה ולכן כל התעסקות בהודעה תגרור טעות שתמנע פענוח של ההודעה, ולא יהיה ניתן להתערב בתוכן בצורה שתטעה את אחד מהצדדים.

שימוש ב-KDF :

השימוש ב-KDF נועד כדי להגדיל את האנטרופיה של סיסמאות/passphrases וכדי להפוך dictionary attacks לארוכות ולא כדאיות לתוקפים. זה נעשה על ידי הוספת salt לסיסמה לפני ה-hash במספר איטרציות גבוה. זמן החישוב לא משמעותי למשתמש הרגיל, אבל לתוקף הוא יגדיל מאוד את הזמן הנדרש לביצוע מתקפת dictionary. מאוד מומלץ לבצע KDF על סיסמאות ו-passphrases, אבל מכיוון שאנחנו לא השתמשנו לא באלה ולא באלה בפרויקט שלנו אלא במפתחות קריפטוגרפיים שמבוססים על הרנדומליות של מערכת ההפעלה, לא הוספנו KDF.

פרטי הפרוטוקול:

לכל הבקשות והתשובות יש את המבנה הכללי הזה:

	שדה	גודל
header	קוד בקשה	1 בתים
	גודל תוכן הבקשה (payload)	4 בתים
payload	תוכן הבקשה	משתנה

בקשה לשרת

קוד בקשה 0

הלקוח ניתק את החיבור לשרת.

קוד בקשה 1

הלקוח מבקש מהשרת לאמת את הזהות שלו:

שדה	**גודל**
challenge	32 בתים

קוד בקשה 31

תחילת תהליך רישום:

שדה	**גודל**
מספר טלפון	10 בתים

קוד בקשה 32

סיום תהליך רישום (אימות קוד חד פעמי + שליחת מפתח ציבורי):

שדה	גודל
קוד חד פעמי	6 בתים
מפתח ציבורי	178 בתים

קוד בקשה 41

תחילת תהליך הזדהות לקוח קיים:

שדה	גודל
מספר טלפון	10 בתים

קוד בקשה 42

סיום תהליך הזדהות לקוח קיים:

שדה	גודל
חתימה של ה-challenge שקיבל מהשרת	משתנה (70-72 בתים)

קוד בקשה 61

בקשת התחלת שיחה עם לקוח אחר:

שדה	גודל
מספר טלפון של הלקוח האחר	10 בתים
אורך החתימה של המידע הנ"ל	בית
חתימה של המידע הנ"ל	משתנה (70-72 בתים)

קוד בקשה 62

שליחת הודעה מוצפנת ללקוח אחר:

שדה	גודל
מספר טלפון של הלקוח האחר	10 בתים
אורך החתימה של המידע הנ"ל	בית
מפתח AES מוצפן	48 בתים
ערך ה-IV	16 בתים
ההודעה המוצפנת	משתנה
חתימה של המידע הנ"ל	משתנה (70-72 בתים)

קוד בקשה 63

קבלת כל ההודעות שלי.

תשובה ללקוח

קוד תשובה 0

השרת ניתק את החיבור ללקוח.

קוד תשובה 1

השרת מאמת את זהותו מול הלקוח:

שדה	גודל
signed challenge	512 בתים

קוד תשובה 31

תהליך רישום החל בהצלחה - נשלח OTP בערוץ המאובטח.

קוד תשובה 32

תהליך רישום הסתיים בהצלחה.

קוד תשובה 41

תהליך התחברות החל בהצלחה, נשלח challenge.

שדה	גודל
challenge	32 בתים

קוד תשובה 42

תהליך התחברות הסתיים בהצלחה.

קוד תשובה 61

שולח מפתח ציבורי של הלקוח שביקשת:

שדה	גודל
מפתח ציבורי	178 בתים

קוד תשובה 62

הודעה נשלחה בהצלחה.

קוד תשובה 63

הנה ההודעות שלך:

שדה	גודל
הודעות (בפורמט JSON)	אינו ידוע מראש
חתימה של המידע	512 בתים

<u>קודי שגיאה:</u>

קוד תשובה 128

סוג הבקשה אינו תקין.

קוד תשובה 131

מספר הטלפון איתו אתה מנסה להירשם למערכת כבר רשום למערכת.

קוד תשובה 132

הקוד החד-פעמי שהזנת הינו שגוי.

קוד תשובה 133

הזנת קוד חד-פעמי שגוי יותר מדי פעמים. הקישור לשרת נותק.

קוד תשובה 134

ניסיון הרישום נכשל מכיוון שהמפתח הפומבי שניתן לשרת אינו תקין.

קוד תשובה 141

ניסיון ההתחברות נכשל מכיוון שהמס' טלפון שהתקבל אינו רשום במערכת.

קוד תשובה 161

מספר הטלפון של הלקוח שביקשת לא קיים במערכת.

קוד תשובה 251

חתימה לא תקינה.

קוד תשובה 255

קוד הבקשה אינו מזוהה.

הבטחת זמינות - Availability

נתחיל מזה שאם תוקף יחליט שהשרת שלנו הוא מטרה ראויה, לא תהיה לו בעיה להקריס אותו עם סקריפט פשוט בפייתון, פשוט מעצם העובדה שהשתמשנו בselect שמוגבל במספר הסוקטים שהוא יכול להחזיק (1024).

בחרנו ב-select כי מדובר במערכת קטנה, אם רוצים לאפשר scale יותר גבוה אז אפשר להשתמש ב-epoll, שמוגבל רק על ידי משאבי המערכת. אבל גם זאת הגבלה!

במקרה שרוצים להבטיח scale גבוה וזמינות לכל המשתמשים, בשלב מסוים לא תהיה ברירה אלא להגדיל את מספר השרתים והמשאבים בהם.

אפשר לנסות לעצור תקיפת DOS על ידי rate limiting - כלומר הגבלת מספר הבקשות שמגיעות מלקוח בפרק זמן נתון, וניתן גם להגביל את הזמן שאנחנו ממתינים למשתמש שיזין את הקוד לפני סגירת החיבור.

תוקף יוכל לנסות במקרה כזה תקיפת DDOS, כלומר להשתמש בbotnet כדי לתקוף את האפליקציה שלנו (תרחיש סביר, אנחנו יודעים). ניתן לממש הגנות שונות כמו rate limiting על השרת עצמו (להגביל את מספר הבקשות שהשרת עצמו מטפל בהן בטווח זמן מסוים), להוסיף load balancers ולחסום כתובות IP מאיזורים גיאוגרפיים שלא אמורים להשתמש באפליקציה שלנו.

אז האם אפשר להבטיח זמינות? להבטיח אפשר, לקיים פחות (גם סתם באגים יכולים למנוע זמינות XD).

יש פתרונות שונים שאפשר לממש כדי להתמודד עם תקיפות מניעת שירות, וזה הדרך הכי טובה לנסות להבטיח זמינות.