מבוא למטבעות קריפטוגרפים – תרגיל בית 3

תאריך הגשה: יום חמישי, 16 ינואר, 23:59

מטרת התרגיל:

בתרגיל זה תלמדו כיצד לתכנת בשפת התכנות solidity. התרגיל יורכב משני חלקים, כאשר בחלק הראשון אתם תבצעו deploy לחוזה של ארנק ותבצעו עליו מתקפת re-entrancy. בחלק השני, אתם תממשו משחק אבן-נייר-מספריים.

הגדרות:

אתם תצטרכו:

- Remix IDE
- Hardhat
  - במחשב שלכם תוכלו להתקין את hardhat בעזרת npm:

    npm init

    npm install –save-dev hardhat

    להגדיר פרויקט hardhat חדש (תיצרו קובץ hardhat.config.js ריק):

    npx hardhat init

    להריץ רשת לוקאלית בעזרת:

    npx hardhat node
- מקורות טובים כדי ללמוד לעומק את הסינטקס של שפת solidity:
  - solidity-by-example
  - Solidity documentation
  - בנוסף אתם יכולים לראות את המצגות מהכיתה ולינקים נוספים שקיימים בהם.

חלק 0 (חימום):

1. בצעו Deploy לחוזה החכם Wallet שנמצא תחת הקובץ VulnerableWallet.sol בעזרת Remix. תבצעו את זה ברשת hardhat לוקאלית שחיברתם ל-Remix.
2. תפקידו כסף אל הארנק. תנסו לעשות את זה מכמה חשבונות שונים ש-Hardhat מספק.
3. תבדקו אם אתם יכולים להעביר כסף מ-Wallet בחזרה אל אחת הכתובות.

אין צורך להגיש דבר בחלק זה.

חלק 1 – מתקפת Reentrancy:

החוזה החכם VulnerableWallet.sol מכיל בתוכו באג מסוג reentrancy ולכן ניתן לתקוף אותו. התפקיד שלכם היא לבצע מתקפה שכזו.

- בתוך WalletAttack.sol, תממשו חוזה שתוקף ומנצל את החולשה הזו. ניצול החולשה צריך לרוץ ישר לאחר ה-deploy של החוזה WalletAttack, והפונקציה שאיתה מבצעים את התקיפה נקראת עם 1 ether.
- כדי שהמתקפה תצליח, אתם חייבים למשוך לפחות 3 ether מהחוזה החכם הקורבן. כלומר, אתם תצטרכו לבדוק זאת עם ארנק קורבן שיש לו לפחות 3 eth שהוא הפקיד כדי שיהיה מספיק כסף לגנוב. שלושת ה-ether היא בנוסף ל-1 ether שמועבר עם הקריאה לפונקציה שמנצלת את החולשה של החוזה החכם.
- אין צורך לשנות את הסכום שנגנב בהתבסס על הסכום שקיים בתוך הארנק. רק תהיו בטוחים שאתם גונבים לפחות 3 ether. את הסכום שאתם גונבים תשלחו אל הכתובת של התוקף (מי שקרא לפונקציה שמנצלת את החולשה).

אוכל למחשבה: הארנק Wallet2.sol חסין מפני מתקפת reentrancy. למה?

חומר עזר:

- כדי לקרוא לחוזה חכם (כמו החוזה VulnerableWallet) מתוך חוזה חכם אחר (כמו למשל החוזה החכם שתוקף), מאוד מומלץ להשתמש ב-interface. ה-interface של ה-Wallet הוגדר בקובץ WalletAttack.sol (נקרא WalletI). אתם יכולים להמיר את הכתובת ל-WalletI ואז לקרוא לפונקציה באופן הבא:

```
Wallet(wallet_address).some_function();
```

- [החלק הזו הוא אופציונלי]: אתם יכולים להשתמש ב-Remix כדי לבדוק את הקוד שלכם, או שאתם יכולים להשתמש בסקריפט python על מנת לקמפל, לבצע deployment ואז להריץ את הפונקציות של החוזה החכם. אתם יכולים לראות דוגמה בקובץ deploy.py. תצטרכו להתקין את החבילות הבאות:
  - py-solc-x: ספריית python שעוטפת את הקומפיילר של solidity.
  - web3: ספריית python שבעזרת אפשר לתקשר עם הבלוקצ'יין (קודקודי Ethereum). ניתן לראות את הדוקומנטציה פה.

בתרגיל הבא אנחנו נשתמש בכלים אלו, לכן אני ממליץ לנסות להכיר ולעבוד איתם. במקרה זה, מומלץ לעבוד עם VSCode במקום Remix בתור ה-IDE ולהוריד תוסף של solidity.

חלק 2 – אבן-נייר-מספריים:

בחלק זה, אתם תממשו מספר אבן-נייר-מספריים בתור חוזה חכם. החוזה יתמוך בכמה משחקים (ולא משחק בודד).
ה-API של החוזה החכם ניתן בתוך הקובץ RPS.sol. תוודאו שאתם מממשים באופן מלא את הפונקציונליות שלו. אל תשנו את ה-API, אבל אתם מוזמנים להוסיף עוד משתנים, פונקציות ועוד, כולל כאלו שהן פומביות, אבל אל תשנו את החתימות של הפונקציות external הקיימות, או את ה-constructor של החוזה החכם.

האתגר העיקרי בתכנון המשחק הוא לוודא ששחקן לא יכול לרמות. מכיוון שבמשחק אבן-נייר-מספריים על שני השחקנים לפעול באותו הזמן, בעוד שבלוקצ'יין מאפשר לבצע טרנזקציה אחת כל פעם, אז אנחנו נשתמש ב-commitments (התחייבויות) על מנת לאפשר לשחקנים לבצע מהלכים כך שהם חבויים מהצד השני. (למידע נוסף תקראו על Bit commitment במודל ה-random oracle פה). השחקנים יגלו את ההתחייבות שלכם ואז יוגדר המנצח.

הפקדה ומשיכת כספים: החוזה RPS יכול לקבל כסף מכל מקור בעזרת העברה ישירה (מבלי לקרוא לפונקציה כלשהי, אלא פשוט העברה כמו בין 2 חשבונות), שאותו כסף מאוחר יותר יהיה בשימוש של סכום ההתערבות על תוצאת המשחק.
שחקנים עם כסף בחוזה RPS יכול למשוך אותו, כל עוד אותם כספים אינם נעולים במשחקים שעדיין מתקיימים.

חלקי המשחק: לאחר שהשחקנים הפקידו כספים בחוזה, כל משחק ממשיך בהתאם לחלקים הבאים:

- כל שחקן מתחייב למהלך בעזרת הפונקציה make_move(). חלק זה כולל בתוכו נעילת חלק מהיתרה של השחקן בתור סכום התערבות (השחקן הראשון הוא זה שמגדיר את סכום ההתערבות, והשחקן השני צריך שהיתרה שלו תהיה מספיקה כדי שיוכל לשחק מולו).
- ברגע ששני המהלכים שוחקו, כל שחקן מגלה את המהלך שהוא ביצע בעזרת הפונקציה reveal_move().
- ברגע ששני המהלכים גלויים, אז המשחק הסתיים והמנצח מקבל את הפרס. במקרה של שוויון, הכספים חוזרים אל השחקנים.

התהליך שתואר פה למעלה מתאר תהליך תקין של המשחק, במקרה בו שני השחקנים הם אמינים. עם זאת, אחד מהשחקנים יכול לנסות לרמות, או להפסיק להגיב. במקרה כזה, השחקן השני צריך שיהיה לו את האמצעים לקבל בחזרה את הכסף שהוא הימר עליו בסוף המשחק:

- אם שחקן התחייב למהלך, והשחקן השני לא הגיב, אז השחקן הראשון יוכלו לבטל את המשחק. השחקן הראשון לא יוכל לבטל את המשחק כל עוד השחקן השני הגיב. (אוכל למחשבה – מה הבעיה עם ביטול אחרי שהשחקן השני התחייב למהלך, וזה לפני שהוא גילה אותו?)

- אם אחד השחקנים גילה את המהלך שלו, והשחקן השני לא גילה את המהלך שלו בתוך זמן גילוי המהלכים (שמוגדר מראש), אז השחקן שביצע את גילוי המהלך שלו יכול לבטל את המשחק. במקרה כזה הוא זוכה בכל הכסף, כולל הכסף של השחקן שלא ביצע את גילוי המהלך שלו. (אוכל למחשבה – למה למעשה צריך את המקרה הזה? למה שלא נבטל את המשחק ופשוט נחזיר את הכספים לשני השחקנים?)

לכל משחק יש gameID משלו – שזה מספר שרירותי שהשחקנים בוחרים. ניתן לקבל את המצב שבו המשחק נמצא בעזרת הפונקציה getGameState().

התחייבות למהלך: על מנת לממש סכימת התחייבות פשוטה, ניתן לשחקנים לשלוח את ה-hash של המהלך שהם בחרו לבצע ביחד עם מפתח רנדומלי בתור ההתחייבות. אותה התחייבות לאחר מכן תיבדק ע"י שליחת המהלך ואותו מפתח כדי לוודא שה-hashes אכן תואמים. בחוזה RPS יש פונקציה שמאפשרת לכם לחשב את ה-hashes האלו כדי לקבל בחזרה את ההתחייבויות בעזרת הפונקציה Keccack256. בנוסף סיפקתי לכם קוד python (commit.py) שנותן לכם ליצור התחייבויות בצורה שהוא דומה לאיך שהן נוצרות ב-solidity. כדי להריץ את הקוד יש להתקין את הספריות web3 ו-hexbytes בעזרת pip.

<u>הגשה:</u>

יש להגיש את הקבצים בתוך zip שנקרא ex3.zip. קובץ ה-zip יכיל את הקבצים WalletAttack.sol ו-RPS.sol ביחד עם README. אל תשלחו קבצים אחרים כמו למשל VulnerableWallet.sol.
הקוד שלכם ייבדק בצורה אוטומטית. אתם יכולים לשנות את הקבצים כל עוד אתם לא שוברים את ה-API שהוגדר (כלומר, אתם יכולים להוסיף עוד פונקציות, משתנים, מבנים ועוד).

שימו לב שרק שותף אחד צריך להגיש את התרגיל!

בהצלחה!