כתיבת חוזים חכמים ברמה גבוהה: מדריך ליצירה והפצה מאובטחת

מבוא

חוזים חכמים הם אבני היסוד של טכנולוגיית הבלוקצ'יין. הם מאפשרים לבצע פעולות אוטומטיות, בלתי ניתנות לשינוי, המנוהלות בצורה שקופה ואמינה. עם זאת, הפצה של חוזה חכם לבלוקצ'יין היא התחייבות כבדה – כל טעות בקוד עלולה לגרום להפסדים כספיים ולפגיעה באמון המשתמשים.

במאמר זה נציג שיטות עבודה מומלצות לכתיבה של חוזים חכמים ברמה גבוהה, המיועדים להפצה לבלוקצ'יין ללא חשש. נדון באבטחה, אופטימיזציה, מנגנוני שדרוג, וניהול נתונים.

עקרונות יסוד לכתיבת חוזים חכמים מאובטחים

1. השתמשו בספריות קוד מוכנות

- OpenZeppelin: ספריית קוד פתוח המספקת חוזים מאובטחים ומוכנים, כגון ERC-20 (מטבעות), ERC-721 (NFTs), ועוד, בקרת גישה.
- שימוש בספריות מוכנות חוסך זמן ומפחית סיכונים, מכיוון שהן נבדקות באופן מקצועי על ידי מומחי אבטחה.

2. הגנו מפני פרצות נפוצות

- Reentrancy (קריאות חוזרות): מנעו התקפות שבהן חוזה זדוני קורא שוב ושוב לחוזה שלכם.
- פתרון: השתמשו במודיפייר `noReentrancy`:

```
bool private locked;
modifier noReentrancy() {
    require(!locked, "Reentrancy not allowed");
    locked = true;
    _;
    locked = false;
```

```
}
```

- Overflow/Underflow: מנעו בעיות של חישובים שגויים במספרים.
- פתרון: השתמשו ב-`SafeMath` (נכלל בגרסאות מתקדמות של Solidity).

3. בקרת גישה (Access Control)
- הגבילו פונקציות רגישות למנהלים או לבעלי הרשאה.
- דוגמה:

```
import "@openzeppelin/contracts/access/Ownable.sol";

contract MyContract is Ownable {
    function sensitiveAction() public onlyOwner {
// רק הבעלים יכול לבצע פעולה זו
    }
}
```

4. שמרו על שקיפות
- השתמשו באירועים (Events) כדי לעדכן משתמשים על פעולות שמבוצעות בחוזה.
- דוגמה:

```
event FundsWithdrawn(address indexed user, uint256 amount);

function withdraw(uint256 amount) public {
    emit FundsWithdrawn(msg.sender, amount);
}
```

ניהול נתונים ושמירה לאורך זמן

1. השתמשו במבנים יעילים

- הימנעו מאחסון נתונים מיותר שעלול להוביל לעלויות גבוהות.
- לדוגמה, השתמשו במיפוי (Mapping) במקום מערכים כאשר יש צורך בגישה מהירה.

2. פתרון לשדרוג חוזים (Upgradeable Contracts)
- חוזים חכמים רגילים הם בלתי ניתנים לשינוי, ולכן יש צורך במנגנון שדרוג כדי להתאים אותם לצרכים עתידיים.
- Proxy Contracts: מבנה שבו חוזה פרוקסי מפנה קריאות לחוזה לוגי שניתן לעדכן.
- דוגמה:

```
contract Proxy {
    address public implementation;

    function upgradeTo(address newImplementation) public {
        implementation = newImplementation;
    }

    fallback() external {
        (bool success, ) = implementation.delegatecall(msg.data);
        require(success);
    }
}
```

שיטות עבודה מומלצות לניהול אבטחה

1. תכנון קפדני
- תכננו מראש את כל התרחישים האפשריים (כולל שגיאות ופעולות זדוניות).
- בדקו היטב את כל המקרים שבהם החוזה יכול להיכשל (כגון הפקדות כפולות או קריאות שגויות).

2. בדיקות יחידה ואינטגרציה
- בצעו בדיקות יחידה (Unit Tests) ואינטגרציה (Integration Tests) על

כל פונקציה.
- השתמשו בכלים כמו Hardhat או Truffle לבדיקות אוטומטיות.

3. ביקורת צד שלישי (Audit)
- לפני העלאת החוזה לבלוקצ'יין, העבירו אותו ביקורת על ידי חברת אבטחה המתמחה בבלוקצ'יין.

אופטימיזציה לעלויות גז (Gas)

1. הימנעו מפונקציות יקרות
- פונקציות שמבצעות לולאות או כותבות הרבה נתונים לבלוקצ'יין יכולות לעלות מאוד ביוקר.
- השתמשו במבנים יעילים כמו מיפויים במקום מערכים.

2. אחסון נתונים מחוץ לבלוקצ'יין
- אם יש מידע שאינו קריטי, שמרו אותו מחוץ לבלוקצ'יין (למשל, ב-IPFS או בשרתים מבוזרים אחרים).

3. שימוש מחושב בשיטות
- הימנעו מכתיבה חוזרת של נתונים שאפשר לחשבם בכל פעם מחדש.

דוגמה לחוזה חכם מאובטח ומוכן להעלאה
```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/math/SafeMath.sol";

contract SecureToken is Ownable {
    using SafeMath for uint256;

    string public name = "SecureToken";
```

```
    string public symbol = "STK";
    uint8 public decimals = 18;
    uint256 public totalSupply;

    mapping(address => uint256) private balances;
    mapping(address => mapping(address => uint256)) private
allowances;

    event Transfer(address indexed from, address indexed to,
uint256 value);
    event Approval(address indexed owner, address indexed spender,
uint256 value);

    constructor(uint256 _initialSupply) {
        totalSupply = _initialSupply * 10decimals;
        balances[msg.sender] = totalSupply;
    }

    function balanceOf(address account) public view returns
(uint256) {
        return balances[account];
    }

    function transfer(address to, uint256 amount) public returns
(bool) {
        require(balances[msg.sender] >= amount, "Insufficient
balance");
        balances[msg.sender] = balances[msg.sender].sub(amount);
        balances[to] = balances[to].add(amount);
        emit Transfer(msg.sender, to, amount);
        return true;
    }

    function approve(address spender, uint256 amount) public returns
(bool) {
        allowances[msg.sender][spender] = amount;
        emit Approval(msg.sender, spender, amount);
```

```
        return true;
    }

    function transferFrom(address from, address to, uint256 amount)
public returns (bool) {
        require(balances[from] >= amount, "Insufficient balance");
        require(allowances[from][msg.sender] >= amount, "Allowance
exceeded");
        balances[from] = balances[from].sub(amount);
        balances[to] = balances[to].add(amount);
        allowances[from][msg.sender] = allowances[from]
[msg.sender].sub(amount);
        emit Transfer(from, to, amount);
        return true;
    }
}
```

הסבר החוזה SecureToken

החוזה הזה נועד ליצור מטבע מבוזר מותאם אישית שמבוסס על בלוקצ'יין של Ethereum, עם פונקציות סטנדרטיות לניהול העברות מטבעות, אישור שימוש (approval), והעברות על ידי צד שלישי (transferFrom). החוזה כולל גם מנגנוני אבטחה בסיסיים באמצעות ספריות קוד פתוח של OpenZeppelin.

פירוט חלקי החוזה

1. כותרת וייבוא ספריות

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/math/SafeMath.sol";
```

- SPDX-License-Identifier:
- מגדיר את רישיון הקוד (MIT במקרה הזה).

- pragma solidity ^0.8.0:
- מציין שהחוזה נכתב בגרסה 0.8.0 ומעלה של Solidity.

- ייבוא ספריות:
  - `Ownable` : מספק פונקציות לניהול חוזה על ידי בעלים יחיד (Owner).
  - `SafeMath` : מונע בעיות חישוב כמו Overflow או Underflow במספרים שלמים.

2. הצהרת משתנים

```
string public name = "SecureToken";
string public symbol = "STK";
uint8 public decimals = 18;
uint256 public totalSupply;

mapping(address => uint256) private balances;
mapping(address => mapping(address => uint256)) private allowances;
```

- name, symbol, decimals:
- פרטי המטבע:
  - `name` : שם המטבע (SecureToken).
  - `symbol` : קיצור שם המטבע (STK).

- `decimals`: מספר הספרות אחרי הנקודה (הסטנדרט עבור 18 – ERC-20).

- totalSupply:
- סך כל המטבעות שנוצרו.

- balances:
- מיפוי שמאחסן את היתרה של כל כתובת.

- allowances:
- מיפוי שמאחסן את כמות המטבעות שכתובת מסוימת אישרה לשימוש על ידי כתובת אחרת.

3. אירועים (Events)

```
event Transfer(address indexed from, address indexed to, uint256 value);
event Approval(address indexed owner, address indexed spender, uint256 value);
```

- Transfer:
- משדר אירוע כאשר מתבצעת העברת מטבעות בין כתובות.
- Approval:
- משדר אירוע כאשר בעל מטבעות מאשר לכתובת אחרת להעביר מטבעות בשמו.

4. פונקציית הבנאי (constructor)

```
constructor(uint256 _initialSupply) {
    totalSupply = _initialSupply * 10decimals;
    balances[msg.sender] = totalSupply;
}
```

- _initialSupply:
- המשתמש שמעלה את החוזה מזין את כמות המטבעות שייווצרו בתחילת הדרך.
- הכמות מוכפלת ב-(decimals) 10^{18} כדי להתאים ליחידות הקטנות של Ethereum.

- balances[msg.sender]:
- כל המטבעות המונפקים בתחילת הדרך מוקצים לבעל החוזה (המעלה).

5. פונקציות קריאה (View Functions)

balanceOf

```
function balanceOf(address account) public view returns (uint256) {
    return balances[account];
}
```

- מחזירה את היתרה של כתובת מסוימת.

6. פונקציות לוגיות

transfer

```
function transfer(address to, uint256 amount) public returns (bool) {
    require(balances[msg.sender] >= amount, "Insufficient balance");
    balances[msg.sender] = balances[msg.sender].sub(amount);
    balances[to] = balances[to].add(amount);
    emit Transfer(msg.sender, to, amount);
    return true;
}
```

- תפקיד:
- מעבירה מטבעות מהמשתמש שמבצע את הקריאה (msg.sender)

לכתובת אחרת.

- שלבי הפעולה:
1. בודקת שיש למשתמש מספיק יתרה.
2. מפחיתה את הסכום מהיתרה של `msg.sender`.
3. מוסיפה את הסכום לכתובת היעד (`to`).
4. משדרת אירוע `Transfer`.

approve

```
function approve(address spender, uint256 amount) public returns
(bool) {
    allowances[msg.sender][spender] = amount;
    emit Approval(msg.sender, spender, amount);
    return true;
}
```

- תפקיד:
- מאפשרת לבעל מטבעות לאשר לכתובת אחרת (`spender`) להשתמש בכמות מסוימת של מטבעות בשמו.

- שלבי הפעולה:
1. מעדכנת את המיפוי `allowances` עבור `msg.sender` והמאשר (`spender`).
2. משדרת אירוע `Approval`.

transferFrom

```
function transferFrom(address from, address to, uint256 amount)
public returns (bool) {
    require(balances[from] >= amount, "Insufficient balance");
    require(allowances[from][msg.sender] >= amount, "Allowance
exceeded");
```

```
    balances[from] = balances[from].sub(amount);
    balances[to] = balances[to].add(amount);
    allowances[from][msg.sender] = allowances[from]
[msg.sender].sub(amount);
    emit Transfer(from, to, amount);
    return true;
}
```

- תפקיד:
- מאפשרת להעביר מטבעות מחשבון אחד (`from`) לחשבון אחר (`to`) על ידי צד שלישי.

- שלבי הפעולה:
1. בודקת שיש לכתובת `from` מספיק מטבעות.
2. בודקת ש-`msg.sender` אושר לבצע את ההעברה באמצעות פונקציית `approve`.
3. מעדכנת את היתרה של `from` ואת היתרה של `to`.
4. מפחיתה את ההרשאה עבור `msg.sender` ב-`allowances`.
5. משדרת אירוע `Transfer`.

אבטחה ואופטימיזציה

1. שימוש ב-SafeMath:
- מונע בעיות Overflow ו-Underflow בעת חישובים.

2. בקרת גישה (Ownable):
- מספק פונקציות שניתן להפעיל רק על ידי הבעלים (owner).

3. אירועים:
- כל פעולה חשובה מתועדת דרך אירועים, מה שמסייע לניטור ושקיפות.

שימושים אפשריים
1. מטבע מבוזר: ניתן להשתמש בחוזה זה להנפקת מטבע ייחודי לפרויקט.

2. מערכות תשלומים: מאפשר תשלומים בין משתמשים.
3. הטמעה במערכות מורכבות: מתאים לפרויקטים כמו DEX, משחקים או מערכות תגמולים.

החבילות והשירותים שבהם אנו משתמשים בחוזה החכם

החוזה שכתבנו עושה שימוש בשתי חבילות עיקריות: OpenZeppelin ו-SafeMath. חבילות אלו הן כלים חזקים וסטנדרטיים בקהילת Solidity שמספקות שירותים מוכנים לכתיבת חוזים חכמים בצורה מאובטחת ויעילה.

1. חבילת OpenZeppelin

OpenZeppelin היא ספריית קוד פתוח מוכרת ומובילה בעולם הבלוקצ'יין. היא מספקת חוזים חכמים מאובטחים ומודולים לשימוש חוזר, שעברו ביקורת מקצועית והם הבסיס לפרויקטים רבים.

המודולים שבהם השתמשנו:

1. Ownable:
- מאפשר להגדיר בעלות על החוזה (Ownership) וליצור פונקציות שרק הבעלים של החוזה יכולים להפעיל.
- מתאים במיוחד למקרים שבהם החוזה צריך לנהל הרשאות, כמו עדכון פרמטרים או משיכה של כספים.

פונקציות מרכזיות:
- `owner`: מחזירה את כתובת הבעלים הנוכחי של החוזה.
- `onlyOwner`: מודיפייר שמאפשר גישה לפונקציות מסוימות רק לבעלים.

יתרונות:
- ניהול פשוט של הרשאות.
- מפחית סיכון שכתובות לא מורשות יפעלו על החוזה.

דוגמה לשימוש ב-Ownable:

```
import "@openzeppelin/contracts/access/Ownable.sol";

contract MyContract is Ownable {
    function sensitiveAction() public onlyOwner {
// פעולה שרק הבעלים יכול לבצע
    }
}
```

2. ERC-20 (לא נכלל בחוזה זה):
- OpenZeppelin מספקת גם מימוש מאובטח ומוכן לסטנדרט ERC-20.
- אם היינו משתמשים בו, הוא היה חוסך כתיבה של פונקציות כמו `approve`, `transfer`, ו-`transferFrom`.

2. חבילת SafeMath

היא ספריית עזר שמספקת פונקציות מתמטיות בסיסיות (כמו SafeMath חיבור, חיסור, כפל וחילוק) תוך מניעת בעיות Overflow (גלישה מספרית) ו-Underflow.

למה להשתמש ב-SafeMath?
- Overflow ו-Underflow:
- בעיות Overflow מתרחשות כשמוסיפים מספרים וכתוצאה מכך חורגים מהמקסימום של המשתנה.
- בעיות Underflow מתרחשות כשמנסים לחסר יותר ממה שיש, והתוצאה היא ערך שגוי.
- דוגמה לבעיה:
- אם `uint256` מגיע למקסימום (2^256-1), כל חיבור נוסף "יגלוש" חזרה ל-0.
- SafeMath מונעת זאת על ידי הוספת בדיקות במתמטיקה.

פונקציות עיקריות ב-SafeMath:
1. `add`: מחברת שני מספרים ומוודאת שאין Overflow.
2. `sub`: מחסרת שני מספרים ומוודאת שאין Underflow.

3. `mul`: כופלת שני מספרים בבטחה.
4. `div`: מבצעת חילוק עם טיפול בשגיאות.

דוגמה לשימוש ב-SafeMath:

```
import "@openzeppelin/contracts/utils/math/SafeMath.sol";

contract MySafeContract {
   using SafeMath for uint256;

   uint256 public totalSupply;

   function increaseSupply(uint256 amount) public {
      totalSupply = totalSupply.add(amount);
   }

   function decreaseSupply(uint256 amount) public {
      totalSupply = totalSupply.sub(amount, "Underflow: cannot
decrease below 0");
   }
}
```

הערה על Solidity 0.8.0 ומעלה:
- בגרסה זו של Solidity, יש טיפול מובנה ב-Overflow וב-Underflow, ולכן SafeMath פחות נחוצה.
- עם זאת, היא עדיין שימושית לשיפור קריאות הקוד ותחזוקה.

שירותים נוספים שיכולים להיות רלוונטיים

1. Oracles
- אם החוזה היה זקוק לנתונים חיצוניים (למשל, מחירי שוק), היינו משתמשים ב-Oracles כמו Chainlink.
- Oracles מספקים דרך אמינה למשוך מידע חיצוני לתוך בלוקצ'יין.

2. Proxy Contracts
- לשדרוג חוזים בעתיד, ניתן לשלב מנגנון Proxy.
- OpenZeppelin מספקת תבניות מוכנות לניהול חוזים שדרוגיים.

למה להשתמש בשירותים אלו?

יתרונות מרכזיים:
1. אבטחה:
- קוד שנבדק לעומק מפחית סיכונים של פרצות.
2. חיסכון בזמן:
- לא צריך לכתוב פונקציות בסיסיות מאפס.
3. שיפור קריאות ותחזוקה:
- הקוד ברור יותר, ומקל על מתכנתים אחרים להבין ולתחזק אותו.
4. תמיכה בקהילה:
- שירותים כמו OpenZeppelin נתמכים על ידי קהילה רחבה של מתכנתים ומומחים.

מתי להשתמש בספריות אלו?
- תמיד, אלא אם כן יש לך סיבה ברורה שלא. שימוש בספריות מאובטחות הוא סטנדרט מקצועי ומפחית משמעותית את הסיכון לשגיאות.

הרחבה מפורטת על החבילות והשירותים שאנחנו משתמשים בהם

1. חבילת OpenZeppelin

OpenZeppelin היא ספריית קוד פתוח שנחשבת לסטנדרט דה-פקטו לכתיבת חוזים חכמים מאובטחים. היא מספקת כלים מוכנים שנבדקו לעומק על ידי קהילה מקצועית ונרחבת.

רכיבים מרכזיים ב-OpenZeppelin

1.1 Ownable
-מטרה:
- מאפשר ניהול בעלות על החוזה. רק הבעלים של החוזה יכול לבצע פעולות מסוימות כמו שדרוגים או משיכה של כספים.

-שימושים נפוצים:
- הגבלת גישה לפונקציות קריטיות.
- העברת בעלות במקרה הצורך.

-פונקציות עיקריות:
- `owner`: מחזירה את הכתובת של הבעלים הנוכחי.
- `onlyOwner`: מודיפייר שמגביל גישה לפונקציות מסוימות כך שרק הבעלים יכול להפעילן.
- `transferOwnership`: מאפשר להעביר את הבעלות לכתובת אחרת.

-דוגמה לשימוש:

```
import "@openzeppelin/contracts/access/Ownable.sol";

contract MyContract is Ownable {
    function restrictedAction() public onlyOwner {
// פעולה שרק הבעלים יכול לבצע
    }

    function transferOwnershipToNewOwner(address newOwner)
public onlyOwner {
        transferOwnership(newOwner);
    }
}
```

1.2 ERC-20
-מטרה:

- מימוש סטנדרטי של מטבעות מבוזרים (Tokens). OpenZeppelin מספקת מימוש מלא לסטנדרט ERC-20 עם אבטחה ובקרת גישה.

-שימושים נפוצים:
- יצירת מטבעות מבוזרים לפרויקטים פיננסיים, מערכות תגמולים, או משחקים.

-פונקציות מוכנות:
- `transfer`: מעביר מטבעות מכתובת אחת לאחרת.
- `approve`: מאשר לכתובת אחרת (spender) להשתמש במטבעות שלך.
- `transferFrom`: מאפשר לצד שלישי להעביר מטבעות בשמך.
- `balanceOf`: מחזירה את היתרה של כתובת מסוימת.

-יתרון גדול:
- במקום לכתוב את כל הלוגיקה מאפס, אפשר להשתמש בקוד מוכן ובטוח שכולל את כל הפונקציות הדרושות.

-דוגמה לשימוש:

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MyToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("MyToken", "MTK") {
        _mint(msg.sender, initialSupply);
    }
}
```

1.3 AccessControl

-מטרה:
- ניהול גישה מורכב יותר מעבר לבעלות פשוטה (Ownable). מאפשר להגדיר רמות הרשאה שונות, כמו מנהלים, משתמשים רגילים, או תפקידים מותאמים אישית.

-פונקציות עיקריות:

- `grantRole` : מעניקה תפקיד לכתובת מסוימת.
- `revokeRole` : מסירה תפקיד מכתובת מסוימת.
- `hasRole` : בודקת אם כתובת מסוימת מחזיקה תפקיד.

-שימושים נפוצים:
- הגבלת גישה לפונקציות לפי תפקידים, כמו:
- מנהל (Admin): יכול לשדרג חוזים.
- משתמש (User): יכול להפעיל פונקציות רגילות.

-דוגמה לשימוש:

```
import "@openzeppelin/contracts/access/AccessControl.sol";

contract RoleBasedAccess is AccessControl {
    bytes32 public constant MANAGER_ROLE =
keccak256("MANAGER_ROLE");

    constructor() {
        _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _setupRole(MANAGER_ROLE, msg.sender);
    }

    function restrictedAction() public onlyRole(MANAGER_ROLE) {
// פעולה שמוגבלת למנהלים בלבד
    }
}
```

2. חבילת SafeMath

מטרה:
- מניעת בעיות חישוביות כמו Overflow ו-Underflow במספרים שלמים (Unsigned Integers). לדוגמה:
- Overflow: כשמוסיפים יותר מדי למספר והערך מתגלגל ל-0.
- Underflow: כשמחסרים מספר גדול מדי והערך מתגלגל לערך

המקסימלי.

פונקציות עיקריות:
1. `add` : מבצעת חיבור עם בדיקת Overflow.
2. `sub` : מבצעת חיסור עם בדיקת Underflow.
3. `mul` : מבצעת כפל בצורה בטוחה.
4. `div` : מבצעת חילוק עם טיפול בשגיאות.

דוגמה לבעיה ולפתרון:

קוד ללא SafeMath (בעייתי):

```
uint256 public totalSupply;

function addTokens(uint256 amount) public {
    totalSupply += amount; // אם הערך חורג מהמקסימום, זה יתגלגל ל-0
}
```

קוד עם SafeMath:

```
import "@openzeppelin/contracts/utils/math/SafeMath.sol";

contract SafeContract {
    using SafeMath for uint256;

    uint256 public totalSupply;

    function addTokens(uint256 amount) public {
        totalSupply = totalSupply.add(amount); // מונע Overflow
    }
}
```

3. אורקלים (Oracles)

מטרה:
- חוזים חכמים לא יכולים לגשת למידע חיצוני בעצמם. Oracles הם שירותים שמביאים נתונים חיצוניים (כמו מחירי שוק או מזג אוויר) לבלוקצ'יין.

שירותי אורקלים נפוצים:
1.Chainlink:
- מאפשר משיכת נתונים חיצוניים כמו מחירי מטבעות, מניות, או תוצאות ספורט.
2.Witnet:
- מערכת מבוזרת לאורקלים שמתמקדת בנתונים ציבוריים ומבוזרים.

שימוש לדוגמה ב-Chainlink:

```
import "@chainlink/contracts/src/v0.8/interfaces/
AggregatorV3Interface.sol";

contract PriceConsumer {
    AggregatorV3Interface internal priceFeed;

    constructor() {
        priceFeed = AggregatorV3Interface(0x...); // כתובת אורקל
ספציפית
    }

    function getLatestPrice() public view returns (int) {
        (,int price,,,) = priceFeed.latestRoundData();
        return price;
    }
}
```

4. Proxy Contracts

מטרה:
- מאפשרים לשדרג חוזים חכמים מבלי לאבד נתונים.

כיצד זה עובד:
- משתמשים בחוזה פרוקסי שמפנה קריאות לחוזה לוגי.
- במקרה של שדרוג, מחליפים רק את הכתובת של החוזה הלוגי.

דוגמה:

```
contract Proxy {
    address public implementation;

    function upgradeTo(address newImplementation) public {
        implementation = newImplementation;
    }

    fallback() external {
        (bool success, ) = implementation.delegatecall(msg.data);
        require(success);
    }
}
```

למה להשתמש בשירותים האלו?

יתרונות מרכזיים:
1.אבטחה:
- שירותים כמו OpenZeppelin ו-SafeMath מונעים פרצות נפוצות.
2.חיסכון בזמן:

- לא צריך לכתוב קוד מאפס.
3.שיפור תחזוקה:
- הקוד ברור ומאורגן.
4.תמיכה בקהילה:
- שירותים אלו נתמכים על ידי קהילה גדולה ומקצועית.