ERC1155CONTRACT
SELLCONTRACT
INCELEMESI(RESEARCH)

SERCAN ÇELENK

12 Temmuz 2021

İÇİNDEKİLER

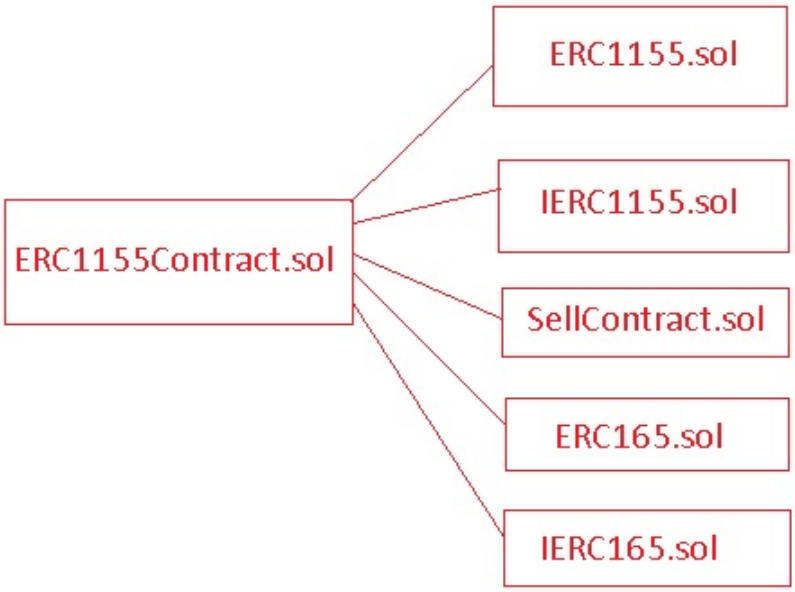
1

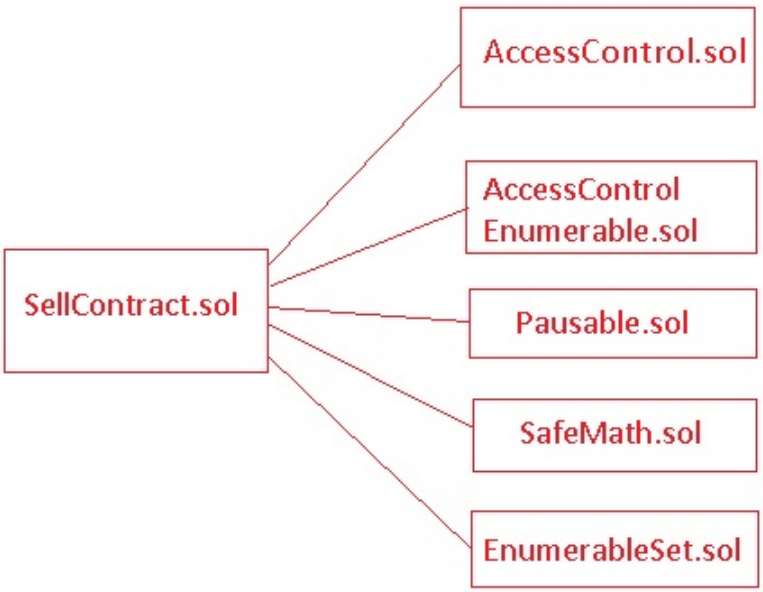
HEBYS, kullanıcıların dijital varlıkları keşfetmesini, oluşturmasını ve ticaretini yapmasnı saglayan bir NFT pazaryeridir.
Pazaryerinin asıl amacı kullanıcı deneyimini en üst seviyeye çıkartmak ve karmaşıklığı en aza indirmektedir.

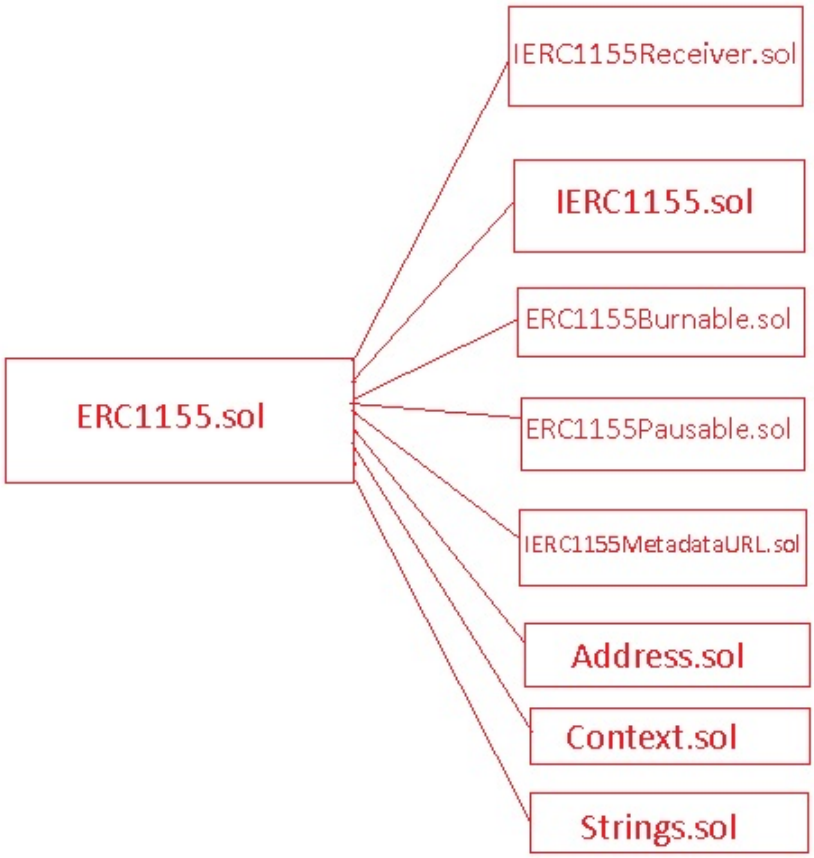
Bu rapor;
- Projenin sürekli olarak geliştirilip iyileştirilmesi
- Yazılım Geliştirme Yaşam Döngüsü(SDLC)
- Açıklar ve güvenlik zaafiyetleri
gibi pek çok konuda bilgi sahibi olunması amacıyla hazırlanmıştır.

İşbu derlenen bu rapor ile akıllı kontratın geleceği için çerçeve çizilmiş, düşük seviyeli(low level) etkenler belirlenmiştir.

2 Dosya Bağımlılığı(File Dependency Graph)






ERC1155.sol
IERC1155Receiver.sol
IERC1155.sol
ERC1155Burnable.sol
ERC1155Pausable.sol
IERC1155MetadataURL.sol
Address.sol
Context.sol
Strings.sol

3 Bulgular(Findings)

3.1 Tam Sayı Bölmeyi Engelleme(Integer Division)

Tam sayı bölme gerçekleştiğinde, sayı bir tam sayıya en yakın değere yuvarlanır. Daha fazla doğruluk gerekiyorsa bir çarpan kullanmak veya pay ve payda değerlerini sağlamak daha iyidir. Bu saklanan değerler, zincir dışı modda pay-payda sonucunun hesaplanmasına yardımcı olur.

SellContract.sol adlı kontratı incelediğimde, SafeMath kütüphanesi direkt olarak openzeppelin'den çağırılmıştır. Bu sözleşme içinde sözleşme çağırmaya benzer ve içinde exploit sızması gibi çeşitli riskleri barındırır. Ana kontratımızın içinde çarpanları kendimiz vererek SafeMath'ı yazmalıyız. Sonrasında istediğimiz yerden tekrar çağırabiliriz. Örnek olarak;

```
1  library SafeMathKutuphanesi {
2      function add(uint a, uint b) internal pure returns (uint c) {
3          c = a + b;
4          require(c >= a);
5      }
6      function sub(uint a, uint b) internal pure returns (uint c) {
7          require(b <= a);
8          c = a - b;
9      }
10     //sercancelenk
11     function mul(uint a, uint b) internal pure returns (uint c) {
12         c = a * b;
13         require(a == 0 || c / a == b);
14     }
15     function div(uint a, uint b) internal pure returns (uint c) {
16         require(b > 0);
17         c = a / b;
18     }
19 }
```

3.2 Yönlendirme(Forwarding)

2021'in 4. çeyreğinde planlanan IDO için; IDO başlangıcında fonlar multisig contract'ta tutulmalı. Bu sayede IDO sonuna kadar fonlar taşınamaz. Taşınacağı zaman tek bir imza değil, bir den fazla kişinin imzası gerekir.

En kötü senaryo da, elimizdeki smart contract ile IDO yapıldığını düşünelim. Bu IDO'da fonlar sözleşmede toplandıktan 3 gün sonra sözleşme sahibinin private key'i çeşitli hack yöntemleri ile veya fiziksel şekilde ele geçirilirse; Tüm fon başka bir cüzdana aktarılabilir veya yıkıcı fonksiyon çağırılarak sözleşmeye geri dönülemez zararlar verebilir.

Bu yüzden standart smart contract yerine IDO gibi toplu fonlamalarda multisig wallet contract kullanılması gerekir. Bu sayede seçilecek iki sözleşme sahibinin de imzası(private key) olmadan hiçbir işlem yapılamaz.

3.3 Olay Günlükleri(Contract Activity)

Akıllı sözleşmeleri izlemek onları güvence altına almanın önemli bir yoludur. Bu sayede tüm sözleşme işlemlerini izlemek mümkündür fakat mesaj çağrıları blok zincirine kayıt edilmez. Bundan dolayı yapılan gerçek değişiklik yerine, yalnızca giriş parametreleri görünür durumda kalır. 'Olay' sözleşmede meydana gelen bir şeyi kayıt altına tutmanın yöntemlerinden biridir. Kayıt edilen olaylar, diğer sözleşme verileriyle birlikte blok zincirinde kalır ve gelecekteki denetim işlemleri için kullanılabilir hale gelir.

```
3  contract TEMAvakfi{
4      mapping(address=>uint) balances;
5      function eventMessage() payable public{
6          balances[msg.sender]+=msg.value;
7      }
8  }
9
10 contract tarimBakanligi{
11     function fidanAl() payable public{
12         tarimBakanligi.eventMessage.value(msg.value/20)();
13     }
14 }
15
16 TEMAvakfi.eventMessage();
```

3.4 Soyut Sözleşmeler ve Arayüzler Arasındaki Ödünleşimler(Tradeoff)

Arayüzler ve soyut sözleşmeler, akıllı sözleşmeler için özelleştirilebilir ve yeniden kullanılabilir bir yaklaşım sağlamada etkilidir. Arayüzler, soyut sözleşmelere benzeseler de bazı işlevlerden yoksundurlar. Örnek verecek olursam depolamaya erişemezler veya sözleşme devralamazlar.

ERC1155Contract.sol kontratımıza baktığım da yalnızca IERC165 adında bir arayüz fonksiyonu bulunmakta.

3.5 Killswitch

Testnette gözden kaçabilen bir sorun, Mainnet'e geçildiğinde geri dönülemez sonuçlara sebep olabilir. Varlıkların çalınması, Flaş Kredi, DoS saldırıları, Gas Price Impact vs.

Bu gibi durumlarda yapılacak ilk şey sözleşmenin dondurulması yani killswitch'tir. Bu işlem Pauseable kütüphanesi ile de yapılabilse de yine sözleşmeye dışarıdan başka bir sözleşme çağırmak yerine ana kontrat içine gömülü olarak fonksiyon yazmak daha kullanışlı olacaktır.

```
1
2  void passive(){
3      require_auth(_self);
4      setStatusDelegate(0);
5  }
6
7  void active(){
8      require_auth(_self);
9      setStatusDelegate(1);
10 }
11 //sercancelenk
12 void process(){
13     assert(setStatusDelegate().passive == 0, "Contract is passive.!");)
14 }
```

3.6 Sayısal Taşma(Numerical Overflow)

Aritmetik işlemler yapılırken sınır koşulları yanlış kontrol edilirse değerler taşabilir. Bu da kullanıcıların varlıklarının kaybına neden olur. Aritmetik işlemlerde uint64t kullanmak yerine kendi yazacağımız SafeMath kütüphanesini direkt ana kontrata yerleştirmeliyiz.

3.7 Düşük Seviyeli Çağrılar(Low Level Call)

Düşük seviyeli çağrılarda dönüş değerlerinde problemler çıkmaktadır. Örneğin kullanıcının sayı girmesi gereken yere harf girmesi, veya sözleşme adreslerinin yanlış değer döndürmesi.

```
//Operator authorization inquiry for sales contract from creator contract
(bool success, bytes memory result) =
    creatorContractAddress.call(
        abi.encodeWithSignature(
            "isApprovedForAll(address,address)",
            _ownerAddress,
            contractAddress
        )
    );
```

Altını çizdiğim kısım düşük seviyeli bir çağrıdır. Çözüm olarak; çağırılan sözleşmeler ayrı dosyalardaysa, bunlar içe aktarılabilir veya çağırılan sözleşme ile, çağıran sözleşme ayrı bir soyut sözleşmeye aktarılabilir. Eğer içe aktarma yapılırsa bytecode artar bu da performansı düşürebilir.