Laporan Audit Mandiri

Gold Indonesia Republic (GIDR)

Audit Dilakukan Oleh:

BRAIN IPB University

Daftar Isi

Versi Kontrol Dokumen

Versi	Penyusun	Organisasi	Perubahan	Tanggal
1.0	Shabrina Basyasyah, M. Iqbal	BRAIN IPB	Initial Draft	11 Desember 2024, 10.47 AM
2.0	M. Iqbal	BRAIN IPB	Penambahan Daftar Isi, Temuan Audit, Rangkuman, Ruang Lingkup Audit, dan Kesimpulan	26 Desember 2024, 2.39 PM

halaman pengesahan

1. Ringkasan Utama

1.1 Latar Belakang

Secara historis, emas telah memainkan peran penting penting dalam sistem keuangan global. Emas langka, stabil secara kimiawi, mudah dibentuk, dan berharga secara universal. Emas telah bertahan sebagai penyimpan nilai selama ribuan tahun dan tetap populer hingga saat ini dengan rata-rata perdagangannya senilai US$112,2 miliar yang terjadi setiap hari melalui ETF, derivatif, dan pasar fisik.

Emas bertahan sebagai aset safe haven. Direkomendasikan secara luas sebagai bagian dari portofolio investasi yang terdiversifikasi dan terus menjadi aset yang lebih disukai daripada mata uang lokal bagi miliaran individu di negara-negara dengan tender fiat yang tidak stabil.

Pasar emas saat ini menghadapi kontradiksi: tidak ada emas fisik yang mudah dimiliki dan diperdagangkan. Memiliki emas fisik dalam jumlah besar sangat dimungkinkan, namun mahal untuk disimpan, sulit dibagi, dan tidak praktis dipindahkan sehingga sulit diperdagangkan. Sebagai alternatif, investor bisa berdagang emas digital, dan ETF emas.

Gold Indonesia Republic (GIDR) hadir dengan misi utama men-digitalisasi dan me mobilisasi aset emas Indonesia menjadi tersedia ke seluruh penjuru dunia. Kami memiliki visi dimana aset emas dapat bergerak secara global, 24/7, serta tanpa gesekan dan batas. Kami melihat emas sebagai aset yang ideal, karena emas memiliki keunikan dimana permintaan pasar yang besar—nilai pasarnya lebih dari US$9 triliun— dan cenderung memiliki ketahanan terhadap inflasi. Meski demikian, emas secara fisik merupakan aset yang rumit dan mahal untuk dimiliki, disimpan, dan dipindahkan.

GIDR bertujuan untuk menghadirkan efisiensi, keamanan, kemudahan, dan kepercayaan yang belum pernah terjadi sebelumnya untuk memiliki dan memperdagangkan aset emas dalam bentuk aset digital. GIDR menawarkan solusi untuk dapat memperdagangkan emas secara fraksional di pedagang aset kripto (crypto exchange). Dalam hal ini, GIDR berfungsi sebagai tabungan emas digital dalam bentuk aset kripto (stable coin), dapat ditukarkan ke aset digital lain pada crypto exchange terkait, serta dapat ditukarkan menjadi emas fisik dengan syarat dan ketentuan yang ditentukan. GIDR, seperti halnya stable coin pada umumnya, berfungsi sebagai aset yang lebih safe haven, tidak fluktuatif secara ekstrim, dan cenderung stabil sehingga bagi sebagian investor lebih merasa aman menyimpan aset mereka dalam bentuk GIDR.

GIDR adalah aset kripto berbasis emas yang dikembangkan oleh PT Indonesia Blockchain Persada (Blocktogo) dan PT Pegadaian. Harga GIDR sebanding dengan harga satu gram emas yang mengacu pada harga jual emas fisik pada Galeri24. Informasi selengkapnya tentang GIDR dapat diakses di https://gidr.co.id.

Tujuan audit teknologi blockchain untuk GIDR adalah untuk memastikan bahwa sistem yang digunakan mampu menghadirkan efisiensi, keamanan, kemudahan, dan kepercayaan

dalam mendukung transaksi emas digital secara fraksional. Audit bertujuan mengevaluasi efisiensi teknologi blockchain yang diterapkan, termasuk kecepatan proses, biaya transaksi rendah, dan penggunaan sumber daya yang optimal. Selain itu, audit ini juga fokus pada peningkatan keamanan dengan mengidentifikasi serta menghilangkan potensi kerentanan untuk melindungi aset digital pengguna dari ancaman eksternal maupun risiko internal. Dalam hal aksesibilitas, audit akan memastikan platform GIDR menawarkan pengalaman pengguna yang sederhana dan intuitif, serta mendukung integrasi dengan berbagai crypto exchange untuk perdagangan aset digital yang lancar.

1.2 Rangkuman Temuan Audit

Selama proses audit smart contract GIDR, ditemukan dua temuan:

1) Fungsi _minting tidak ditentukan batasan maksimum (MAX_MINT)

2) Ditemukan kasus underflow error pada fungsi _transfer

1.3 Kesimpulan

Kontrak tidak aman untuk diluncurkan sampai temuan terkait batasan maksimum *minting* dan underflow error ditindaklanjuti.

2. Ruang Lingkup Audit

2.1 Deskripsi Proyek

Proyek ini adalah platform Web3 berbasis Polygon yang memungkinkan pengguna untuk melakukan staking token ERC-20 guna menghasilkan *yield*. Kontrak cerdas ini mendukung fungsi minting, transfer, dan burning token, memastikan efisiensi dan keamanan dalam ekosistemnya.

2.2 Tujuan Audit

Proyek ini adalah platform Web3 berbasis Polygon yang memungkinkan pengguna untuk melakukan staking token ERC-20 guna menghasilkan *yield*. Kontrak cerdas ini mendukung fungsi minting, transfer, dan burning token, memastikan efisiensi dan keamanan dalam ekosistemnya.

2.3 Batasan Audit

Audit hanya mencakup pengecekan vulnerability pada kode program baik menggunakan *tools* dan secara manual, unit testing fungsi - fungsi, dan keamanan kontrak cerdas.

3. Metodologi

3.1 Pendekatan Audit

Pendekatan audit smart contract GIDR mencakup penggunaan *tools* seperti MythX dan Slither untuk mendeteksi kerentanan umum, serta pengecekan kode secara manual untuk mengidentifikasi masalah logika dan kerentanan kode program kontrak cerdas.

3.2 Alat yang digunakan

Dalam proses audit smart contract GIDR, terdapat dua alat yang digunakan, yaitu MythX untuk analisis smart contract secara general dan bahasa pemrograman TypeScript untuk pengujian smart contract dengan berbagai skenario.

3.3 Cakupan Kode Program

Seluruh bagian smart contract GIDR telah diaudit, meliputi fungsi minting, burning, transfer, dan setFee.

4. Rincian Temuan

4.1 Kode Program

a) Fungsi minting

```
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply += amount;
    unchecked {
        // Overflow not possible: balance + amount is at most totalSupply + amount, which is checked above.
        _balances[account] += amount;
    }
    emit Transfer(address(0), account, amount);

    _afterTokenTransfer(address(0), account, amount);
}
```

Gambar 1 Kode Program Fungsi Minting

Description: Terdapat kerentanan pada fungsi minting, yaitu tidak ada batasan nilai maksimumnya. Hal ini memungkinkan pemilik kontrak melakukan *minting* melebihi banyaknya modal pembelian emas fisik yang dimiliki oleh Blocktogo.

Vulnerability Level: Medium.

Impact Analysis: Pengguna GIDR dapat melakukan *minting* untuk mencetak token GIDR dalam jumlah yang tidak wajar, melampaui logika proses bisnis yang ditetapkan (1 token GIDR = 1 gram emas). Sebagai contoh, penyerang dapat memanfaatkan akses tersebut

untuk mencetak 11.000.000 token, yang setara dengan 11 ton emas. Jumlah ini melebihi banyaknya modal pembelian emas fisik yang dimiliki oleh Blocktogo.

Proof of Concept: Pengguna GIDR dapat memanfaatkan kelemahan ini untuk mencetak token dalam jumlah tidak wajar. Sebagai contoh, penyerang dapat memanggil fungsi minting secara eksternal dengan memasukkan nilai sebesar 11.000.000, yang akan meningkatkan saldo pengguna tertentu sebesar 11.000.000 token GIDR. Aksi ini dapat dilakukan tanpa adanya batasan jumlah maksimum minting yang sesuai dengan logika proses bisnis.

Recommendation:

- Tambahkan batasan nilai minting (MAX_MINT) mengikuti banyaknya total modal pembelian emas fisik yang dimiliki oleh Blocktogo.
- Mengimplementasikan akun multi signature sehingga setiap transaksi yang dilakukan harus disetujui sebelum dieksekusi.

Fixed Status: Tidak perlu perbaikan. Kontrak cerdas GIDR telah menerapkan akun multisignature dan GIDR merupakan stable coin sehingga proses minting tidak mempengaruhi harga emas.

b) Fungsi transfer

```
function _transfer(address from, address to, uint256 amount) internal override {
  uint256 amountReceived = amount;
  if (fee > 0) {
    amountReceived -= fee;
    super._transfer(from, feeReceived, fee);
    emit Fee(from, feeReceived, fee);
  }
  super._transfer(from, to, amountReceived);
}
```

Gambar 2 Kode Program Fungsi Transfer

Description: Terdapat kerentanan pada fungsi transfer, yaitu tidak ada validasi pada nilai parameter amount dan fee yang memastikan bahwa nilai fee tidak lebih dari amount. Hal ini menyebabkan terjadinya underflow error pada variabel amountReceived.

Vulnerability Level: Low.

Impact Analysis: Tidak adanya validasi amount > fee, menyebabkan terjadinya underflow pada variabel amountReceived. Hal ini dapat menyebabkan transfer gagal dan rentan dieksploitasi oleh penyerang.

Proof of Concept: Dilakukan transfer dengan nilai gas fee melebihi nilai transfer (fee = 15 dan amount = 10) menggunakan fungsi transfer sehingga terjadi underflow pada variabel amountReceived (amountReceived = 10 - 15 = -5; tipe data amountReceived uint256).

Recommendation: Tambahkan logika untuk memeriksa apakah nilai fee melebihi nilai transfer. Jika kondisi terpenuhi, kembalikan pesan pengecualian.

Fixed Status: Sudah diperbaiki (24 Desember 2024).

4.2 Keamanan Kontrak

a) Akun Multisignature

Kontrak cerdas GIDR telah menerapkan akun multisignature (2 akun dari Blocktogo, 1 akun dari Pegadaian) sehingga setiap transaksi yang masuk ke dalam antrian harus mendapatkan persetujuan sebelum dieksekusi. Hal ini dapat mencegah terjadinya transaksi yang tidak memenuhi kontrak cerdas dan proses bisnis GIDR.

b) Akses Kontrol

```
function _authorizeUpgrade(address) internal override onlyOwner {
    versionCode += 1;
}

function setFee(address _feeReceived, uint256 _fee) external onlyOwner {
    require(_feeReceived != address(0), "Address cannot be null");
    feeInfo = FeeInfo(_feeReceived, _fee);
    emit SetFee(_feeReceived, _fee);
}

function mint(address _to, uint256 _amount) external onlyOwner {
    _mint(_to, _amount);
}
```

Gambar 3 Kode program yang menerapkan akses kontrol

Kontrak cerdas GIDR menerapkan Access Control berupa onlyOwner untuk memastikan bahwa hanya pemilik kontrak (address yang memiliki hak kepemilikan) yang dapat menjalankan fungsi kontrak sehingga mencegah akses tidak sah.

c) Validasi Input

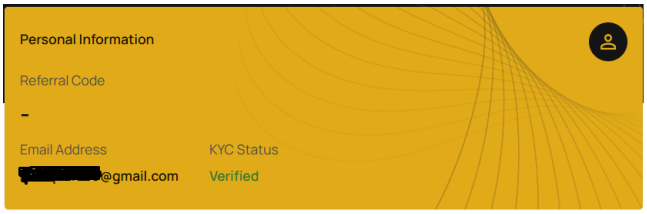
```
require(account != address(0), "ERC20: mint to the zero address");
```

Gambar 4 Baris kode program yang menerapkan validasi input address

Kontrak cerdas GIDR telah menerapkan validasi untuk memastikan bahwa input alamat (address) tidak boleh kosong, sehingga dapat mencegah kesalahan atau data yang tidak valid dalam proses transaksi.

d) Know Your Customer (KYC)

Pembelian dan perdangangan GIDR menggunakan platform Gudang Kripto yang telah menerapkan KYC mengikuti peraturan Bappebti nomor 8 tahun 2021 tentang pentingnya penerapan KYC secara komprehensif oleh pedagang aset kripto.



Gambar 5 Informasi pengguna dengan status KYC terverifikasi

5. Kualitas Kode

5.1 Struktur Kode

Kode program kontrak cerdas GIDR menggunakan modul-modul bawaan dari library OpenZeppelin, seperti UPSUpgradeable, OwnableUpgradeable, dan ERC20Upgradeable yang memudahkan pengembangan kontrak cerdas tanpa perlu mendefinisikan logika fungsi dari awal.

```
import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol";
```

Gambar 6 Kode program library OpenZeppelin

Kode program ditulis mengikuti konsep OOP (Object Oriented Programming) yang sudah menjadi standar umum dalam pengembangan kontrak cerdas. Selain itu, kode ditulis secara jelas dan penamaan fungsi yang sederhana sehingga mudah dipahami. Hal ini memudahkan pemeliharaan dan meningkatkan keterbacaan kode program.

```
contract GIDR is UUPSUpgradeable, OwnableUpgradeable, ERC20Upgradeable {
  uint256 public versionCode;

  address public feeReceived;
  uint256 public fee;

  event SetFee(address indexed feeReceived, uint256 indexed fee);
  event Fee(address indexed from, address indexed feeReceived, uint256 indexed amount);

  function initialize() public initializer {
    __Ownable_init();
    __ERC20_init("Gold Indonesia Republic", "GIDR");
  }

  function _authorizeUpgrade(address) internal override onlyOwner {
    versionCode += 1;
  }

  function setFee(address _feeReceived, uint256 _fee) external onlyOwner {
    require(_feeReceived != address(0), "Address cannot be null");
    feeReceived = _feeReceived;
```

Gambar 7 Kode program yang menerapkan OOP

5.2 Optimalisasi

Berikut rekomendasi optimalisasi kode program agar lebih efisien:

a) Efisiensi Gas

Pada fungsi _transfer sebaiknya nilai amountReceived langsung dihitung (amount - fee) tanpa perlu diinisiasi terlebih dahulu (amountReceived = amount) untuk mengefisiensikan penggunaan gas. Selain itu, ditambahkan pengecekan nilai amount > fee.

Fungsi transfer (saat ini):

```
function _transfer(address from, address to, uint256 amount) internal override {
    uint256 amountReceived = amount;
    if (fee > 0) {
        amountReceived -= fee
        super._transfer(from, feeReceived, fee);
        emit Fee(from, feeReceived, fee);
    }
    super._transfer(from, to, amountReceived);
}
```

Gambar 8 Kode program fungsi transfer

Fungsi transfer (rekomendasi):

```
function _transfer(address from, address to, uint256 amount) internal override {
  require(amount >= fee, "Fee cannot be higher than amount");
  uint256 amountReceived = amount - fee;
  if (fee > 0) {
    super._transfer(from, feeReceived, fee);
    emit Fee(from, feeReceived, fee);
  }
  super._transfer(from, to, amountReceived);
}
```

Gambar 9 Kode program fungsi transfer (rekomendasi)

b) Optimalisasi Variabel

Variabel feeReceived dan fee pada smart contract GIDR didefinisikan secara terpisah. Kedua variabel tersebut dapat digabung dalam satu variabel bertipe struct untuk mengefisiensikan biaya penyimpanan.

Variable feeReceived dan fee (saat ini):

```
address public feeReceived;
uint256 public fee;

function setFee(address _feeReceived, uint256 _fee) external onlyOwner {
  require(_feeReceived != address(0), "Address cannot be null");
  feeReceived = _feeReceived;
  fee = _fee;
  emit SetFee(_feeReceived, _fee);
}
```

Gambar 10 Kode program fungsi setFee

Variable feeReceived dan fee (rekomendasi):

```
struct FeeInfo {
  address feeReceived;
  uint256 fee;
}
FeeInfo public feeInfo;

function setFee(address _feeReceived, uint256 _fee) external onlyOwner {
  require(_feeReceived != address(0), "Address cannot be null");
  feeInfo = FeeInfo(_feeReceived, _fee);
  emit SetFee(_feeReceived, _fee);
}
```

Gambar 11 Kode program fungsi setFee (rekomendasi)

6. Testing

Dilakukan pengujian fungsi kontrak cerdas GIDR pada *environment* localhost menggunakan kode editor visual studio code.

Terdapat 4 fungsi smart contract yang akan diuji:

a) Minting
b) Transfer Token
c) Transfer Fee
d) Burning

Selama pengujian kontrak, dua akun public wallet berbeda digunakan dengan rincian sebagai berikut:

Account #1: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8

Private Key: 0x59c6995e998f97a5a0044966f0945389dc9e86dae88c7a8412f4603b6b78690d

Account #2: 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC

Private Key: 0x5de4111afa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a

1. Minting

   Terdapat tiga skenario pengujian yang dilakukan:

   a) Pengguna sukses melakukan minting (GIDR-TEST-1A)

      Dilakukan pengujian fungsi minting dengan banyaknya token sejumlah 1000 menggunakan wallet address 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266.

```
it("1. Minting : User successfully mints token amount 1000", async () => {
    const amountMint = await parseEther("1000");
    const initialBalance = await instance_gidr.balanceOf(accounts[1].address);
    await expect(instance_gidr.mint(accounts[1].address, amountMint))
        .to.emit(instance_gidr, "Transfer")
        .withArgs(addressNull, accounts[1].address, amountMint);

    const finalBalance = await instance_gidr.balanceOf(accounts[1].address);
    expect(finalBalance).to.equal(initialBalance.add(amountMint));
});
```

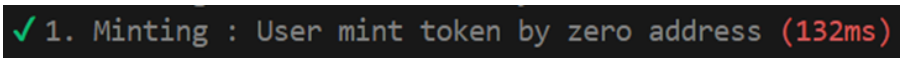
Gambar 12 Hasil pengujian GIDR-TEST-1A

Setelah dilakukan pengujian, banyaknya token pada wallet dengan address 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266 dari saldo awal 0 token meningkat menjadi 1000 token. Transaksi tersebut memenuhi ekspektasi pengujian (Gambar 12).

b) Pengguna melakukan minting dengan address *null* (GIDR-TEST-1B)

Dilakukan pengujian fungsi minting tanpa menginputkan wallet address (address *null*).

```
it("1. Minting : User mint token by zero address", async () => {
    const amountMint = await parseEther("1000");
    await expect(
        instance_gidr.mint(addressNull, amountMint)
    ).to.be.revertedWith("ERC20: mint to the zero address");
});
```



Gambar 13 Hasil pengujian GIDR-TEST-1B

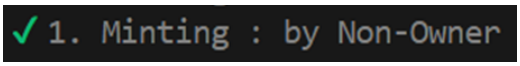
Setelah dilakukan pengujian, transaksi tersebut memenuhi expected result, yaitu menghasilkan output **ERC20: mint to the zero address** dan pengujian berhasil (Gambar 13).

c) Minting dilakukan oleh bukan pemilik wallet (GIDR-TEST-1C)

Dilakukan pengujian fungsi minting dimana minting dilakukan oleh non owner, pengujian ini bertujuan memastikan Smart Contract GIDR menegakkan otorisasi yang benar.

```
it("1. Minting : by Non-Owner", async () => {
  const mintAmount = parseEther("1000");

  await expect(
    instance_gidr.connect(accounts[2]).mint(accounts[1].address, mintAmount)
  ).to.be.revertedWith("Ownable: caller is not the owner");
});
```



Gambar 14 Hasil pengujian GIDR-TEST-1C

Setelah dilakukan pengujian, smart contract GIDR sukses menegakkan otorisasi sehingga minting yang diinisiasi oleh akun wallet 2, tetapi dilakukan oleh akun wallet 1 dianggap tidak sah dengan menampilkan pesan error “Ownable: caller is not the owner”. Skenario pengujian sesuai dengan hasil yang diharapkan (Gambar 14).

2. Transfer Token

Terdapat tiga skenario pengujian yang dilakukan:

a) Pengguna berhasil melakukan transfer (GIDR-TEST-2A)

Dilakukan pengujian transfer dari akun wallet 1 ke akun wallet 2 sejumlah 400 token. Masing-masing akun wallet memiliki saldo awal sebanyak 1000 token.

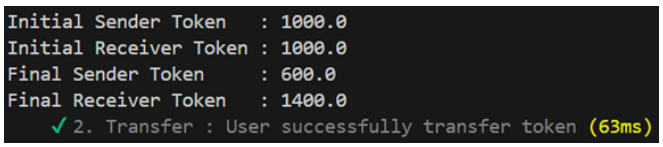
```
it("2. Transfer : User successfully transfer token", async () => {
  const amountTransfer = await parseEther("400");
  await instance_gidr.mint(accounts[1].address, parseEther("1000"));
  await instance_gidr.mint(accounts[2].address, parseEther("1000"));

  const initialSenderBalance = await instance_gidr.balanceOf(accounts[0].address);
  const initialReceiverBalance = await instance_gidr.balanceOf(accounts[1].address);

  await expect(instance_gidr.transfer(accounts[2].address, amountTransfer))
    .to.emit(instance_gidr, "Transfer")
    .withArgs(accounts[1].address, accounts[2].address, amountTransfer);
  const finalSenderBalance = await instance_gidr.balanceOf(accounts[1].address);
  const finalReceiverBalance = await instance_gidr.balanceOf(accounts[2].address);
```

```
    console.log("Initial Sender Token :", ethers.utils.formatEther(initialSenderBalance));
    console.log("Initial Receiver Token :", ethers.utils.formatEther(initialReceiverBalance));
    console.log("Final Sender Token :",ethers.utils.formatEther(finalSenderBalance));
    console.log("Final Receiver Token :", ethers.utils.formatEther(finalReceiverBalance));

    expect(finalSenderBalance).to.equal(
      initialSenderBalance.sub(amountTransfer)
    }
    expect(finalReceiverBalance).to.equal(
      initialReceiverBalance.add(amountTransfer)
    );
});
```



Gambar 15 Hasil pengujian GIDR-TEST-2A

Setelah diujikan, Banyaknya saldo akun sender (wallet 1) berkurang 400 token menjadi 600 dari awalnya 1000 token dan akun receiver (wallet 2) bertambah 400 token dari awalnya 1000 menjadi 1400 token. Pengujian memenuhi output yang diharapkan (Gambar 15).

b) Pengguna melakukan transfer dengan saldo kurang dari banyaknya token yang ditransfer (GIDR-TEST-2B)

Dilakukan pengujian dengan skenario pengguna dengan saldo akun wallet 1 sebanyak 600 token mentransfer token sebanyak 1000 token.

```
it("2. Transfer : User transfer more than balance", async () => {
    const amountTransfer = await parseEther("1000");
    await expect(
      instance_gidr.transfer(accounts[1].address, amountTransfer)
    ).to.be.revertedWith("ERC20: transfer amount exceeds balance");
});
```

Gambar 16 Hasil pengujian GIDR-TEST-2B

Setalah pengujian, transaksi tersebut dianggap gagal karena saldo yang dimiliki (600 token) kurang dari banyaknya token yang akan ditransfer (1000 token)

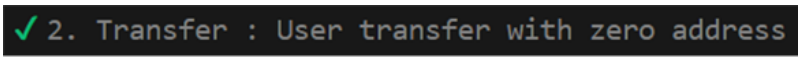
sehingga menampilkan pesan “ERC20: transfer amount exceeds balance”. Pengujian tersebut sesuai dengan output yang diharapkan (Gambar 16).

c) Pengguna melakukan transfer ke address *null* (GIDR-TEST-2C)

Dilakukan pengujian fungsi transfer dengan address *null*.

```
it("2. Transfer : User transfer with zero address sender", async () => {
    const amountTransfer = parseEther("100");
    await instance_gidr.mint(accounts[1].address, amountTransfer);

    await expect(
        instance_gidr.transfer(addressNull, amountTransfer)
    ).to.be.revertedWith("ERC20: transfer to the zero address");
});
```



Gambar 17 Hasil pengujian GIDR-TEST-2C

Setelah dilakukan pengujian, transaksi transfer gagal yang ditandai dengan munculnya pesan “ERC20: transfer to the zero address” karena transfer dilakukan dengan address *null* (Gambar 17).

3. Transfer Fee

Ada dua skenario pengujian yang dilakukan:

a) Pengguna melakukan transfer dengan fee yang lebih besar dari saldo yang ditransfer (GIDR-TEST-3A)

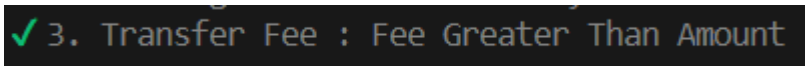
Dilakukan pengujian transfer fee dengan nilai transfer 250 token dan fee sebesar 400 token.

```
it("3. Transfer Fee : Fee Greater Than Amount", async () => {
  const transferAmount = parseEther("250"); // Transfer amount
  const excessiveFee = parseEther("400"); // Fee greater than transfer amount

  // Set an excessive fee
  await instance_gidr.setFee(accounts[1].address, excessiveFee);

  // Attempt the transfer and expect it to revert
  await expect(
    instance_gidr.transfer(accounts[0].address, transferAmount)
  ).to.be.revertedWith("ERC20: transfer amount is less than fee");
```

```
});
```



Gambar 18 Hasil pengujian GIDR-TEST-3A

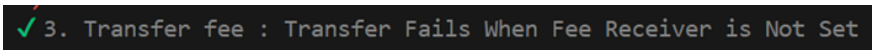
Setelah dilakukan pengujian, kontrak cerdas GIDR dapat mengatasi permasalahan underflow error saat fee lebih besar dari saldo yang ditransfer sehingga pengujian berhasil (Gambar 18).

b) Transfer dilakukan jika address fee = *null* (GIDR-TEST-3B)

Dilakukan pengujian dengan skenario jika address fee berupa *zero address*.

```
it("3. Transfer fee : Transfer Fails When Fee Receiver is Not Set", async () => {
   const transferAmount = parseEther("100"); // Total transfer amount
   const fee = parseEther("20"); // Fee to be deducted

   await expect(
     instance_gidr.setFee(ethers.constants.AddressZero, fee)
   ).to.be.revertedWith("Address cannot be null");
});
```



Gambar 19 Hasil pengujian GIDR-TEST-3B

Setelah dilakukan pengujian, transaksi tersebut gagal dikarenakan address fee nya bernilai null dan saat melakukan transfer address fee tidak boleh null sehingga muncul pesan error “Address cannot be null”. Hal ini sudah memenuhi skenario pengujian yang seharusnya (Gambar 19).

4. Burning

Ada dua skenario pengujian yang dilakukan:

a) Pengguna berhasil melakukan burning token (GIDR-TEST-4A)

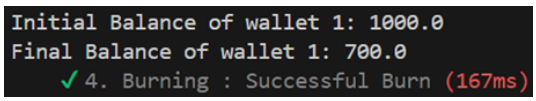
Dilakukan pengujian dengan skenario pengguna melakukan burning token pada akun wallet 1 sebanyak 300 token. Pengguna memiliki saldo wallet sebanyak 1000 token.

```
it("4. Burning : Successful Burn", async () => {
   const burnAmount = parseEther("300");
```

```
    const initialBalance = await instance_gidr.balanceOf(accounts[0].address);
    console.log("Initial Balance of wallet 1:", ethers.utils.formatEther(initialBalance));

    await expect(instance_gidr.burn(burnAmount))
        .to.emit(instance_gidr, "Transfer")
        .withArgs(accounts[0].address, addressNull, burnAmount);
    const finalBalance = await instance_gidr.balanceOf(accounts[0].address);
    console.log("Final Balance of wallet 1:",ethers.utils.formatEther(finalBalance));

    expect(finalBalance).to.equal(initialBalance.sub(burnAmount));
});
```



Gambar 20 Hasil pengujian GIDR-TEST-4A

Setelah pengujian, jumlah token yang dimiliki berkurang dari awalnya 1000 jadi 700 token dan hal ini sudah memenuhi skenario uji yang telah ditentukan (Gambar 20).

b) Pengguna melakukan burning melebihi saldo (GIDR-TEST-4B)

Dilakukan pengujian dengan skenario pengguna melakukan burning sebanyak 1000 token yang melebihi saldo yang tersimpan di dalam wallet (700 token).

```
it("4. Burning : Burn More Than Balance", async () => {
    const burnAmount = parseEther("1000");
    const walletSaldo = await instance_gidr.balanceOf(accounts[0].address);
    console.log("Wallet 1 Saldo :", ethers.utils.formatEther(walletSaldo));
    await expect(instance_gidr.burn(burnAmount)).to.be.revertedWith(
      "ERC20: burn amount exceeds balance"
    );
});
```

Gambar 21 Hasil pengujian GIDR-TEST-4B

Setelah dilakukan pengujian, jika banyaknya token yang di burning melebihi saldo yang dimiliki, maka menampilkan pesan error standar ERC20: burn amount exceeds balance. Hal ini sudah memenuhi skenario testing yang telah ditentukan (Gambar 21).

Fungsi	Minting			Transfer Token			Transfer Fee		Burning	
Kode Uji	GIDR -TES T-1A	GIDR -TES T-1B	GIDR -TES T-1C	GIDR -TES T-2A	GIDR -TES T-2B	GIDR -TES T-2C	GIDR -TES T-3A	GIDR -TES T-3B	GIDR -TES T-4A	GIDR -TES T-4B
Status	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

Tabel 1 Tabel status skenario pengujian fungsi kontrak cerdas GIDR

Dari 10 skenario pengujian yang dilakukan (Tabel 1), semuanya memenuhi output yang diharapkan sehingga semua pengujian tersebut berhasil.

7. Kesimpulan

Setelah dilakukan audit menyeluruh pada kontrak cerdas GIDR, semua temuan kerentanan yang teridentifikasi telah diperbaiki secara efektif. Dengan demikian, kontrak cerdas ini kini telah siap untuk diluncurkan. Namun, untuk menjaga integritas dan keamanan sistem secara berkelanjutan, pemantauan lanjutan tetap diperlukan. Proses pemantauan ini bertujuan untuk memastikan bahwa tidak ada kerentanan baru yang muncul serta untuk mengidentifikasi potensi masalah sejak dini, guna menghindari risiko yang dapat mempengaruhi kinerja dan kepercayaan pengguna terhadap GIDR.