USDT虚假转账安全风险

近日，2345新科技研究院区块链实验室观测到一起疑似盗取交易所USDT代币的攻击行为，攻击者利用交易所对USDT交易转账的判断逻辑缺陷，恶意构造虚假转账盗取交易所代币，如下图：

背景知识

1. USDT是基于Bitcoin区块的OMNI协议[1]资产类型，利用Bitcoin的OP_RETURN承载相关交易数据；
2. Bitcoin本身并不会校验OP_RETURN数据的“合法性”，可以是任意数据；
3. Bitcoin交易当区块确认数达到6的时候，就会被Bitcoin节点承认；
4. 那么问题来了，USDT的交易在OMNI的节点上如何被确认？我们继续来看。

实验室的同事们扫描了Bitcoin区块中的USDT交易数据，找到了一笔无效的交易：

> 1b5c80f487d2bf8b69e1bbba2b1979aacb1aca7a094c00bcb9abd85f9af738ea

根据omniexplorer网站提供的 api ，我们可以看到这笔交易的具体信息如下


```
{
  "amount": "28.59995822",
  "block": 502358,
  "blockhash": "00000000000000000005968fa48c49d7c4fb2363369d59db82897853fd937c71a",
  "blocktime": 1514985094,
  "confirmations": 27488,
```

```
  "divisible": true,
  "fee": "0.00200000",
  "invalidreason": "Sender has insufficient balance",
  "ismine": false,
  "positioninblock": 301,
  "propertyid": 31,
  "propertyname": "TetherUS",
  "referenceaddress": "1Po1oWkD2LmodfkBYiAktwh76vkF93LKnh",
  "sendingaddress": "18DmsHjHU6YM2ckFzub4pBneD8QXCXRTLR",
  "txid": "1b5c80f487d2bf8b69e1bbba2b1979aacb1aca7a094c00bcb9abd85f9af738ea",
  "type": "Simple Send",
  "type_int": 0,
  "valid": false, // 注意这里
  "version": 0
}
```

https://github.com/OmniLayer/omnicore/blob/v0.3.0/src/omnicore/tx.cpp#L1012

```
1012 int CMPTransaction::logicMath_SimpleSend()
1013 {
1014     if (!IsTransactionTypeAllowed(block, property, type, version)) {
1015         PrintToLog("%s(): rejected: type %d or version %d not permitted for property %d at block %d\n",
1016                 __func__,
1017                 type,
1018                 version,
1019                 property,
1020                 block);
1021         return (PKT_ERROR_SEND -22);
1022     }
1023
1024     if (nValue <= 0 || MAX_INT_8_BYTES < nValue) {
1025         PrintToLog("%s(): rejected: value out of range or zero: %d", __func__, nValue);
1026         return (PKT_ERROR_SEND -23);
1027     }
1028
1029     if (!IsPropertyIdValid(property)) {
1030         PrintToLog("%s(): rejected: property %d does not exist\n", __func__, property);
1031         return (PKT_ERROR_SEND -24);
1032     }
1033
1034     int64_t nBalance = getMPbalance(sender, property, BALANCE);
1035     if (nBalance < (int64_t) nValue) {
1036         PrintToLog("%s(): rejected: sender %s has insufficient balance of property %d [%s < %s]\n",
1037                 __func__,
1038                 sender,
1039                 property,
1040                 FormatMP(property, nBalance),
1041                 FormatMP(property, nValue));
1042         return (PKT_ERROR_SEND -25);
1043     }
1044
1045     // ------------------------------------------
1046
1047     // Special case: if can't find the receiver -- assume send to self!
1048     if (receiver.empty()) {
1049         receiver = sender;
1050     }
1051
1052     // Move the tokens
1053     assert(update_tally_map(sender, property, -nValue, BALANCE));
1054     assert(update_tally_map(receiver, property, nValue, BALANCE));
1055
1056     // Is there an active crowdsale running from this recepient?
1057     logicHelper_CrowdsaleParticipation();
```

上面 `CMPTransaction::logicMath_SimpleSend` 这个方法，会是 `Simple Send` 交易类型最终会调用的逻辑。通过上面标注的来看，OMNI内部有自己的一套基于地址的记账模型，通过地址可以获取地址的余额，它会把发送者当前的余额和发送的金额做一个对比，如果余额小于要发送的的金额，就会报余额不足，那么这笔交易就会因为余额不足而无效，就会给 `valid` 字段设置成 `false`。

总结一下，如果一笔 USDT 的交易合法的，要至少满足以下2个条件：

1. 要通过比特币的交易来构造，要符合比特币的余额验证（BTC）及交易规则验证
2. 要通过 USDT 自己的余额（USDT）验证

另外，根据实验室其他同事的研究结果来看，被USDT冻结的地址发起的交易也会被标记为无效交易。

> https://omniexplorer.info/address/3MbYQMMmSkC3AgWkj9FMo5LsPTW1zBTwXL

查看该冻结交易

> 203e932783961d986bccb29deed2b4084fa6fb9d1e3cda8c58c705f2a5fb4c68

攻击过程分析

综合钱包转账行为以及相关技术资料，推测攻击行为可能为：

1. 向交易所钱包构造并发起无效（虚假）转账交易；
2. 由于逻辑判断缺陷交易所将无效交易入账并计入到用户在交易所的资金账户；
3. 用户发起提币；
4. 交易所处理用户提币将币打到用户自己钱包地址；
5. 用户自己充币环节USDT没有任何损失，提币环节交易所把自己真实的USDT币打给用户，造成交易所损失。

影响态势

实验室的同事们跟踪一例疑似恶意攻击者的钱包地址：

> 16k5MgZHm2yxiKzrdeaY2vmn13xSSu5xg6

通过查询该钱包地址的交易记录发现该攻击者仍然在持续的构造虚假交易，推测该攻击者很有可能已经从某些交易所非法获利。

随着区块扫描的深入更多的疑似恶意钱包地址被发现。

防御建议

1. 交易所自查USDT处理逻辑，立即安排功能下线修正并且排查历史USDT交易记录；
2. 引入专业代码审计，提升代码的健壮性；
3. 提升开发人员对于区块链技术的基本认知，避免错误的认知导致错误的结果；
4. 建议各交易所将该钱包地址列入钱包地址黑名单；

5. 提升交易所整体风险控制流程，对于疑似风险交易予以拦截。

进一步思考

数字币交易所如雨后春笋，但是代码在上线前可能并没有做过专业审计，或者相关开发人员没有经过专业培训。千里之堤毁于蚁穴，一个小小的问题或者一时的疏忽大意可能造成巨大的损失，这个问题足以引以为戒。

引用链接：

1. https://github.com/OmniLayer/spec