Parity's Wallet Bug is not Alone

ethereum (http://hackingdistributed.com/tag/ethereum/) bitgo (http://hackingdistributed.com/tag/bitgo/) parity (http://hackingdistributed.com/tag/parity/) wallet (http://hackingdistributed.com/tag/wallet/) security (http://hackingdistributed.com/tag/security/)

Emin Gün Sirer (http://hackingdistributed.com/egs/) Thursday July 20, 2017 at 05:18 AM

← Older (http://hackingdistributed.com/2017/07/11/atomic-transfer/) Newer → ()

A bug was found in Ethereum's Parity Multisig Wallet yesterday, wherein anyone could take ownership of any multisig wallet and usurp the funds. And it looks like someone did just that, emptying out around $30M from three multisig addresses affecting three ICOs. The root cause of the bug was a missing "internal" identifier, where a function that should only have been called internally got exposed to everyone.

This is not the first instance of such a bug. The point of this blog post is to underscore some takeaways that might make sure that it's the last. I'll do this by describing a bug I found and fixed in BitGo's multisig wallet last year, and making the case for a general fix.

The Backstory

I took time away from this to look at BitGo code. I was

I was sitting on the beach at the summer house (which I had bought with my proceeds from the earlier DAO hack [1]) when I got a message from Benedict Chan and Ben Davenport at BitGo. They wanted me to take a look at their multisig wallet. I don't normally do code reviews -- they are hard work, carry too much reputational risk, and with some probability, people will misrepresent my limited involvement as some kind of endorsement of their entire business proposition. Further, they didn't offer any compensation and IC3 had no relationship with BitGo. But I liked BitGo at the time, mostly because they employed great people like Ben Chan and Jameson Lopp. Plus it was a lazy summer afternoon, I was sitting by the Med, with the crickets doing their thing,

and the wind wasn't up yet to kiteboard. So I decided to spend about 15 minutes to look, through the glare on my laptop screen, at their code.

younger and dumber.

This is a good time to look through their code yourself and see if you can spot any vulnerabilities. It should be a little easier given the context. Here it is (https://github.com/BitGo/eth-multisig-v2/blob/acc689e822d1acde412b19b9b33638f509f51283/contracts/WalletSimple.sol).

If you're feeling adventurous, see if you can spot the second attack. There isn't just one, but *two* related and distinct vulnerabilities.

In case you are feeling lost, like I initially felt, here's some orientation. This code is different from the usual multisig pattern one would build on Ethereum. BitGo evidently decided to replicate their Bitcoin workflow in Ethereum, so they pass around an object from one party to another, building the multisig transaction off-chain. In contrast, the natural approach in Ethereum would have been to keep track of the number of signatories inside the contract. It took me several minutes to figure out this design decision. Around minute 12, I spotted the bugs.

Do you see the big problem?

Spoilers follow.

The big flaw is that a compromised signer can call tryInsertSequenceId() with any value they like. And a natural thing to call it with is, of course, a bunch of numbers close to MAXINT. After that, the multisig wallet is stuck. It will not be able to issue any further transactions. Any funds in the contract will be stuck forever. Well, either forever, or until BitGo is able to lobby King Vitalik [2] and get him to agree to a hard fork. Needless to say, that's in jest; I've recently read one too many rants by Bitcoiners who are upset that Ethereum is gaining dominance. The truth is, Vitalik couldn't pull off a hard fork even if he wanted to. BitGo would have to convince the entire Ethereum community to swap the state of their contract to re-mobilize their funds. Chances of that happening are somewhere between an ice-skating vacation in hell, Bitcoin Core adopting something invented by someone who does not go to extra lengths to appease their fragile egos, and Craig Wright actually signing something with the genesis key -- squarely in the "not going to happen" category.

The bug would have been averted by marking tryInsertSequenceId() as **private**.

Not at all ironically, the bug in the BitGo multisig contract is *due exactly to the same root*

cause as in the Parity multisig smart contract.

A Second Vulnerability in the BitGo Contract

There is a second vulnerability in this contract. As I wrote to BitGo

Safe defaults would have led to less time on the laptop, more time on the water.

> I'm also worried that someone (eg equivalent of a hacked Bitfinex) could manipulate the infrastructure around this wallet contract so as to make the other party (BitGo) skip over most the sequence id space and sign 10 legitimate transactions with really large seqids, then be unable to sign any more transactions.

This is an attack on the BitGo API via a compromised client. Even if the **private** or **internal** modifier had been present, the contract would have been open to this kind of attack. These kinds of API design issues are fairly tricky to spot. It's one thing to verify the correctness of a contract against a formal spec, another thing to verify it against an implicitly assumed but not actually enunciated spec, and a different thing altogether to see if an implicitly assumed but not actually enunciated spec would have been safe given the operating conditions.

I'm fairly paranoid, so I would have hardened the API by ensuring that the smart contract does not allow one to advance through the identifier space non-sequentially. That is, I'd rule out skipping around. If performance and concurrency is an issue, I might allow the contract to process a sequential window of W requests. Counting one by one, or even by W, for W<1,000,000, all the way up to 2^256 would not constitute a feasible attack -- that DoS attempt would be preceded by the heat death of the universe. That would prohibit exhaustion of sequence ids by misbehaving clients.

Takeaways

Every big contract is simultaneously

I personally had a good run finding critical bugs, but this portends a dangerous trend. Specifically, with my co-authors Dino Mark and Vlad Zamfir, I found 9 bugs in the DAO (http://hackingdistributed.com/2016/05/27/dao-call-for-moratorium/), one of which was used by the DAO hacker. And I found-yet-incorrectly-dismissed the other bug that the DAO

hacker used (https://www.bloomberg.com/features/2017-the-ether-thief/). We seem to be living through a phase particularly rich in critical bugs.

a bug bounty and a lambo.

There are some depressing takeaways here.

Independent contracts suffer from bugs around the same pain points.

Two separate multisig wallet contracts, which have been scrutinized heavily, were vulnerable to the same kind of bug. This reifies the finding from the 1970's that N-version programming tends to suffer from bugs in the same critical points. Even if one spends the effort to implement N independent versions of the same functionality, they might fail at the same time or around the same breaking points. True independence of failures is difficult to achieve.

Open source scrutiny isn't sufficient to render a contract secure.

Worse, many eyeballs don't make code safe. Just about every ICO, trust and company used the Parity multisig wallet, and that code was considered well-tested. One could perhaps try to make the case that Parity is a volunteer-run, open source project. We have seen, with OpenSSH and Heartbleed, that that is no guarantee of close scrutiny. So one could claim that professional software houses have their act together better. But that'd contradict the fact that BitGo got bit by the same exact bug. So, clearly, the problem here is systemic and needs a systemic solution.

But there are some positive lessons here, as well as a path forward that can put an end to these kinds of problems.

Programmers need safe defaults.

As the database community painstakingly discovered, most programmers cannot handle power. No matter what kind of noise developers make about how they understand their application and its unique properties, how they can tailor the underlying systems just right for their requirements, the fact is, they can't. Just look at the sorry state of NoSQL and consistency models. Regular programmers need sane defaults and mechanisms that fail safe when the programmer does not exert extra special effort.

Default-private is a safe, sane default.

And there is also a positive lesson for language design. These kinds of bugs would be greatly reduced if the default policy in Solidity were "default-private." Almost every other language uses default-private, forcing a smart contract programmer to take extra steps to expose a procedure to outsiders. The tradeoff in Solidity is, frankly, wrong. Default-public is too error prone.

Adding this feature to Solidity would not be extraordinarily difficult.

I'd urge the Solidity project to change this behavior. Make fields and functions in a contract be **default-private**. Such a change does not affect already-deployed smart contracts, does not require a fork, and can even be made optional with the addition of a compiler flag. It's relatively trivial to affect compared to the impossibility of recovering lost funds.

There are other features that one could add to Solidity.

Chief among the new features to be added to Solidity, besides default-private, are new types for fixed-point arithmetic, a common requirement for financial software. As we saw with Bancor (http://hackingdistributed.com/2017/06/19/bancor-is-flawed/), smart contracts rolling their own arithmetic is not a good proposition. Second on the list would be overflow-protected arithmetic types. That would address the features that everyone currently implements with SafeMath. Finally, I'd suggest better type inference. This post (https://news.ycombinator.com/item?id=14691212) provides a good list of gripes to start out with. All of these new features and types would have little to no impact on existing contracts.

It might be time to start thinking about refining Solidity.

By every metric, Solidity has been a huge success. It has introduced large numbers of people to smart contract programming. It has anticipated and built in provisions, such as pre- and post-conditions, that are useful for maintaining invariants, a critical feature for smart contracts. And it will continue to serve as a great introductory language for at least a few years to come. This provides ample time to start thinking about refinements to the language that address sources of common errors.

The BitGo After-Story

Some of you may be wondering how BitGo handled this episode.

I notified BitGo of this issue and they promptly fixed their code (https://github.com/BitGo/eth-multisig-v2/commit/8042188f08c879e06f097ae55c140e0aa7baaff8#diff-b498cc6fd64f83803c260abd8de0a8f5). They had apparently removed the "private" modifier to facilitate unit testing.

Can one blame Romus or Romulus for their upbringing?

This brings up yet another takeaway -- there ought to be some way of dropping private modifiers during testing, and only during testing. Mutating one's code to perform testing is not good practice.

BitGo did not believe that the second issue would be a problem in practice, as they thought that the surrounding client code would not issue non-consecutive sequence numbers. It's their call and their own risk assessment. Recall that I wasn't getting paid for any of this and had no skin in the game, so there was no point in contesting their assessment. I'm told that BitGo ultimately did not launch a product on Ethereum.

I wrote a nice and friendly email to BitGo:

> Great to hear that that was useful. Much better to catch potential issues now, than to try to deal with the fallout from a multi-million contract that is stuck. I don't know if BitGo has a Technical Advisory Board, but if it does, I'd consider it. That'd create an open channel for interactions like this. Otherwise, the chances of catching me on vacation with cycles to spare are, regrettably, fairly low.

I never heard a word from BitGo for the next 4-5 months or so. I chalked that up to most early CTOs in the crypto space having been abandoned by their families at a young age and raised by wolves. I was surprised, however, when Ben Davenport publicly adopted a rabid small block stance and supported the small troll squad known as Dragon's Den (https://twitter.com/bendavenport/status/852052428941778945). When I emailed him about the impact of unprofessional conduct on the whole space, he sent me the following lovely message:

> I understand you are currently on sabbatical from Cornell. Are your fellow faculty members aware of the attention that's being drawn to the Cornell CS department due to your public statements? Hope your activity during your sabbatical year doesn't influence your tenure committee too much.

You all, who realize that I received tenure many years prior, can imagine the bemused look on my face at the implied, toothless threat on my employment. And what for? For criticizing trolls and their professional sponsors, be it by covertly paying them under the table, overtly encouraging them, or by turning a blind eye to their destructive activities. Tenure is intended precisely to stem these kinds of threats on academic freedom, and it worked beautifully.

Two months ago, I had the pleasure of meeting Mike Belshe, CEO of BitGo, in person. He issued an apology for the lack of response and a thank you for this episode, almost a year after the fact. Belshe is a grown-up and BitGo continues to employ excellent engineers. Regardless, I've decided to have absolutely nothing to do with BitGo in a professional capacity. Given how easy it was to find bugs in their code, coupled with the Bitfinex hack for which I consider BitGo partly responsible, I've decided to steer clear of their products.

--

[1] Just kidding.

[2] Terminology provided by Bitcoin maximalists.

← Older (http://hackingdistributed.com/2017/07/11/atomic-transfer/) Newer → ()

alone/&title=Parity's Wallet Bug is not Alone&summary=The bug in the Parity multisig wallet that caused the loss of $30M has the same root cause as a bug in the BitGo multisig wallet that I found a year

ago.&source=http%3A%2F%2Fhackingdistributed.com) Share on Reddit

(https://reddit.com/submit/?url=http://hackingdistributed.com/2017/07/20/parity-

wallet-not-alone/&title=Parity's Wallet Bug is not Alone) Share on Tumblr

(https://www.tumblr.com/widgets/share/tool?posttype=link&title=Parity's Wallet Bug is not Alone&caption=Parity's Wallet Bug is not Alone&content=http://hackingdistributed.com/2017/07/20/parity-wallet-not-alone/&canonicalUrl=http://hackingdistributed.com/2017/07/20/parity-wallet-not-

alone/&shareSource=tumblr_share_button) Share on E-Mail (mailto:?

subject=Parity's Wallet Bug is not Alone&body=http://hackingdistributed.com/2017/07/20/parity-wallet-not-alone/)

Emin Gün Sirer

Hacker and professor at Cornell, with interests that span distributed systems, OSes and networking. Current projects include HyperDex, OpenReplica and the Nexus OS. more... (http://hackingdistributed.com/egs/)

(http://hackingdistributed.com/egs/)

Follow @el33th4xor

Subscribe

(http://hackingdistributed.com/hackingdistributed.atom) (https://www.facebook.com/egsirer) (http://twitter.com/el33th4xor/)

Projects

- HyperDex (http://hyperdex.org)

- Weaver (http://weaver.systems)
- OpenReplica (http://openreplica.org)
- Nexus (http://www.cs.cornell.edu/people/egs/nexus/)
- Merlin (http://frenetic-lang.org/merlin/)

Recent Posts

Archive By Date (/calendar/)

- Parity's Wallet Bug is not Alone (http://hackingdistributed.com/2017/07/20/parity-wallet-not-alone/)
- Atomically Trading with Roger: Gambling on the success of a hardfork (http://hackingdistributed.com/2017/07/11/atomic-transfer/)
- Twitter, Reddit and 4chan: The Web's Fake News Centipede (http://hackingdistributed.com/2017/07/11/web-centipede/)
- Bancor Is Flawed (http://hackingdistributed.com/2017/06/19/bancor-is-flawed/)
- Announcing The Town Crier Service (http://hackingdistributed.com/2017/06/15/town-crier/)
- Levels of Techie Enlightenment (http://hackingdistributed.com/2017/05/04/stages-of-enlightenment/)
- Hijacking Bitcoin: Routing Attacks on Cryptocurrencies (http://hackingdistributed.com/2017/05/01/bgp-attacks-on-btc/)
- Revealing the hidden links in the Monero blockchain (http://hackingdistributed.com/2017/04/19/monero-linkability/)
- A Thinking Person's Guide to the Latest Bitcoin Drama (http://hackingdistributed.com/2017/04/05/bitcoin-drama-response/)
- BitFury's Smallest-First Mining Is Bad For Bitcoin (http://hackingdistributed.com/2017/02/27/smallest-first-bad-for-bitcoin/)

more... (/page/2/)

Popular

- Introducing Weaver (/2014/12/16/introducing-weaver/)
- How to Disincentivize Large Bitcoin Mining Pools (/2014/06/18/how-to-disincentivize-large-bitcoin-mining-pools/)

- How A Mining Monopoly Can Attack Bitcoin (/2014/06/16/how-a-mining-monopoly-can-attack-bitcoin/)
- What Did Not Happen At Mt. Gox (/2014/03/01/what-did-not-happen-at-mtgox/)
- Bitcoin is Broken (/2013/11/04/bitcoin-is-broken/)
- Stack Ranking Is Not The Cause of Microsoft's Problems (/2013/08/24/stack-ranking-did-not-kill-microsoft/)
- How the Snowden Saga Will End (/2013/08/01/framework-for-surveillance/)
- What's Actually Wrong with Yahoo's Purchase of Summly (/2013/03/26/summly/)
- Broken By Design: MongoDB Fault Tolerance (/2013/01/29/mongo-ft/)
- Introducing Virtual Notary (/2013/06/20/virtual-notary-intro/)
- The Principled Documentation Manifesto (/2013/02/11/principled-documentation/)
- Introducing HyperDex Warp: ACID Transactions for NoSQL (/2013/02/05/hyperdex-warp/)

Blog Tags

bitcoin (http://hackingdistributed.com/tag/bitcoin/) / security (http://hackingdistributed.com/tag/security/) / hyperdex (http://hackingdistributed.com/tag/hyperdex/) / release (http://hackingdistributed.com/tag/release/) / ethereum (http://hackingdistributed.com/tag/ethereum/) / nosql (http://hackingdistributed.com/tag/nosql/) / selfish-mining (http://hackingdistributed.com/tag/selfish-mining/) / blocksize (http://hackingdistributed.com/tag/blocksize/) / dao (http://hackingdistributed.com/tag/dao/) / surveillance (http://hackingdistributed.com/tag/surveillance/) / privacy (http://hackingdistributed.com/tag/privacy/) / mongo (http://hackingdistributed.com/tag/mongo/) / broken (http://hackingdistributed.com/tag/broken/) / weaver (http://hackingdistributed.com/tag/weaver/) / nsa (http://hackingdistributed.com/tag/nsa/) / meta (http://hackingdistributed.com/tag/meta/) / leveldb (http://hackingdistributed.com/tag/leveldb/) / 51% (http://hackingdistributed.com/tag/51%25/) / smart contracts

(http://hackingdistributed.com/tag/smart%20contracts/) / graph stores
(http://hackingdistributed.com/tag/graph%20stores/) / bitcoin-ng
(http://hackingdistributed.com/tag/bitcoin-ng/) / voting
(http://hackingdistributed.com/tag/voting/) / vaults
(http://hackingdistributed.com/tag/vaults/) / snowden
(http://hackingdistributed.com/tag/snowden/) / satoshi
(http://hackingdistributed.com/tag/satoshi/) / philosophy
(http://hackingdistributed.com/tag/philosophy/) / mt. gox
(http://hackingdistributed.com/tag/mt.%20gox/) / mining pools
(http://hackingdistributed.com/tag/mining%20pools/) / micropayments
(http://hackingdistributed.com/tag/micropayments/) / hyperleveldb
(http://hackingdistributed.com/tag/hyperleveldb/)