Blockchain security!

JP Aumasson

TAURUS

<insert blockchain meme here>

Are blockchains secure?

- **No major issue** ever in Bitcoin nor Ethereum internals
- **But:** lot of new code, new protocols, complex logic, wide attack surface, unexperienced developers; a recipe for blockchain bugs

Blockchain bugs?

- **Design** bugs: functionality works as intended, but can be abused by an attacker
- **Software** bugs: software errors allow the program to enter an insecure state, unintended by the design

Such bugs can be found either in the blockchain itself, or on the applications running on top of it (smart contracts, etc.)

Multiple targets

A typical cryptocurrency needs several applications

- **Wallets**: desktop, mobile, where private keys are stored
- **Validation nodes**: which run a consensus mechanism to authorize transactions and ensure the blockchain's consistency—be it PoW or PoS

Attackers goals

Main goal: **free money**

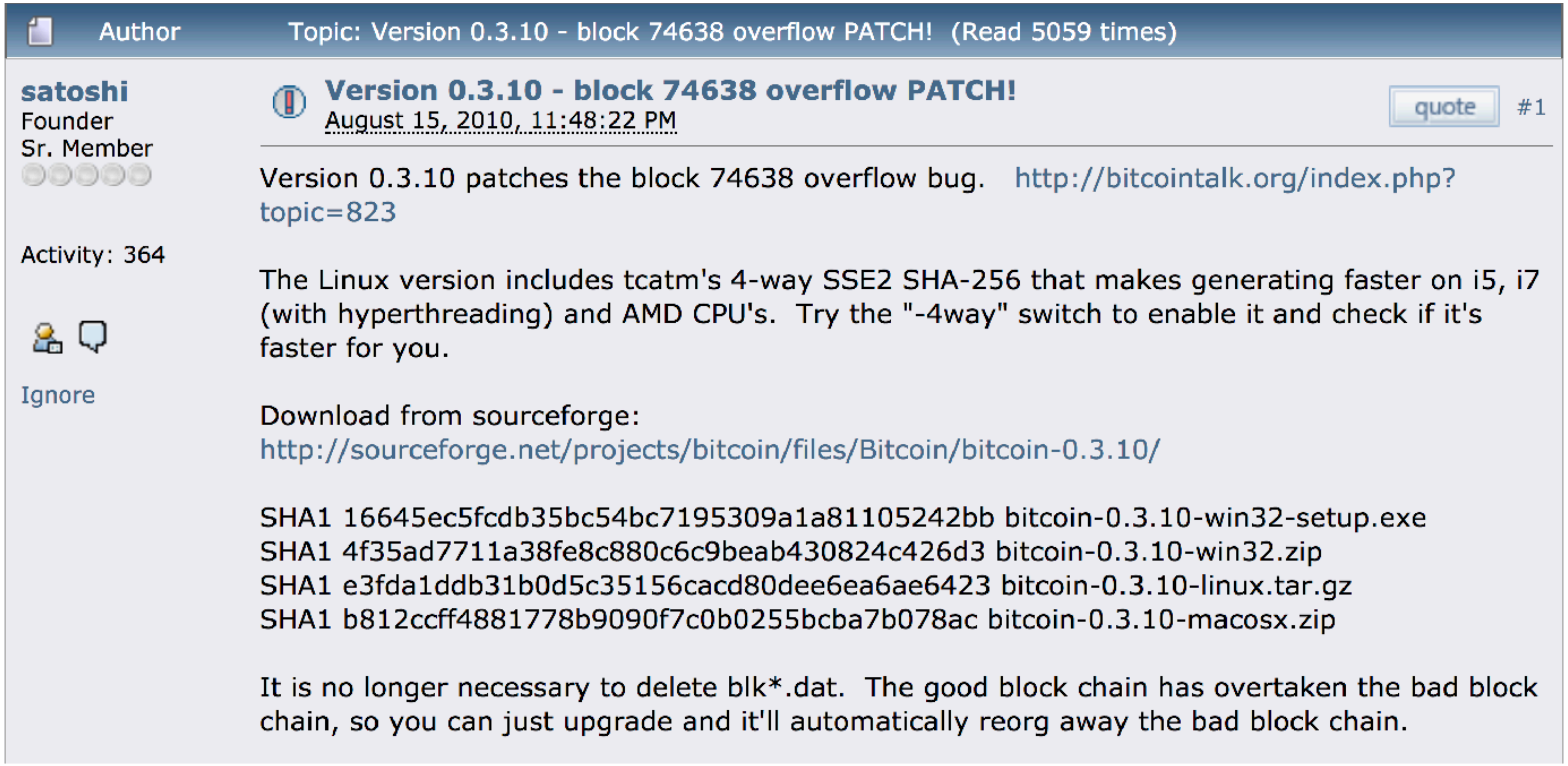
- Steal private keys/seeds/wallets
- Issue transactions on behalf of other clients
- Create coins/tokens out of thin air

Other goals: network denial-of-service, user lock out, etc.

1. Bitcoin overflow
2. Ethereum reentrancy
3. Zerocoin multi-spend
4. Lisk accounts hijack
5. Parity wallet bug
6. IOTA’s hash function
7. Bitgrail withdrawal
8. BatchOverflow
9. Verge consensus
10. BIP32 utilities

1. Bitcoin overflow (CVE-2010-5139)

The worst problem ever in Bitcoin



Author Topic: Version 0.3.10 - block 74638 overflow PATCH! (Read 5059 times)

satoshi
Founder
Sr. Member

Activity: 364

Ignore

Version 0.3.10 - block 74638 overflow PATCH!
August 15, 2010, 11:48:22 PM

quote #1

Version 0.3.10 patches the block 74638 overflow bug. http://bitcointalk.org/index.php?topic=823

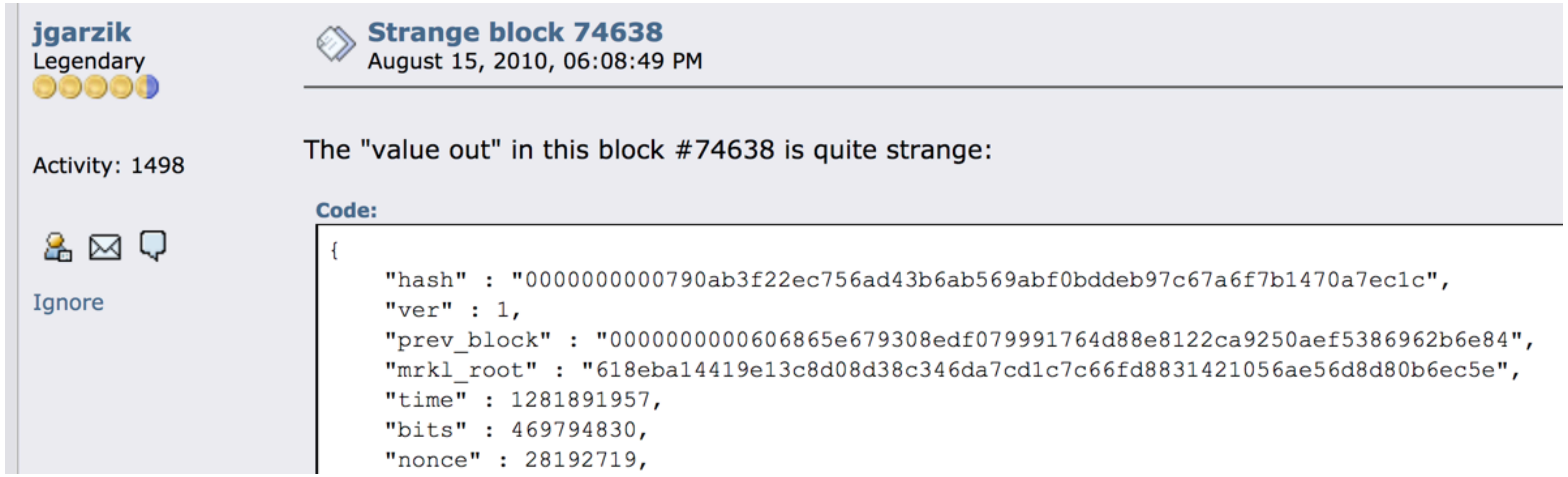
The Linux version includes tcatm's 4-way SSE2 SHA-256 that makes generating faster on i5, i7 (with hyperthreading) and AMD CPU's. Try the "-4way" switch to enable it and check if it's faster for you.

Download from sourceforge:
http://sourceforge.net/projects/bitcoin/files/Bitcoin/bitcoin-0.3.10/

```
SHA1 16645ec5fcdb35bc54bc7195309a1a81105242bb bitcoin-0.3.10-win32-setup.exe
SHA1 4f35ad7711a38fe8c880c6c9beab430824c426d3 bitcoin-0.3.10-win32.zip
SHA1 e3fda1ddb31b0d5c35156cacd80dee6ea6ae6423 bitcoin-0.3.10-linux.tar.gz
SHA1 b812ccff4881778b9090f7c0b0255bcba7b078ac bitcoin-0.3.10-macosx.zip
```

It is no longer necessary to delete blk*.dat. The good block chain has overtaken the bad block chain, so you can just upgrade and it'll automatically reorg away the bad block chain.

What happened?



jgarzik
Legendary

Activity: 1498

Ignore

Strange block 74638
August 15, 2010, 06:08:49 PM

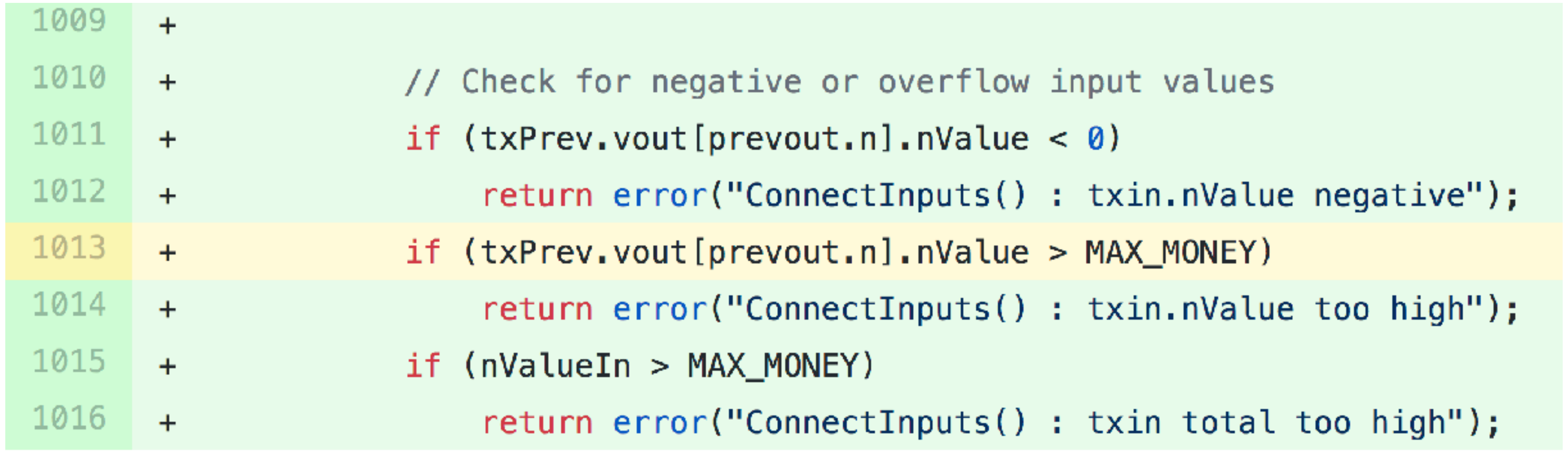
The "value out" in this block #74638 is quite strange:

Code:
```
{
    "hash" : "0000000000790ab3f22ec756ad43b6ab569abf0bddeb97c67a6f7b1470a7ec1c",
    "ver" : 1,
    "prev_block" : "000000000606865e679308edf079991764d88e8122ca9250aef5386962b6e84",
    "mrkl_root" : "618eba14419e13c8d08d38c346da7cd1c7c66fd8831421056ae56d8d80b6ec5e",
    "time" : 1281891957,
    "bits" : 469794830,
    "nonce" : 28192719,
```

- The sum of output values in a transaction overflowed the integer variable
- Transaction check validated the (negative) sum
- **184,467,440,737.09551616** bitcoins were created…

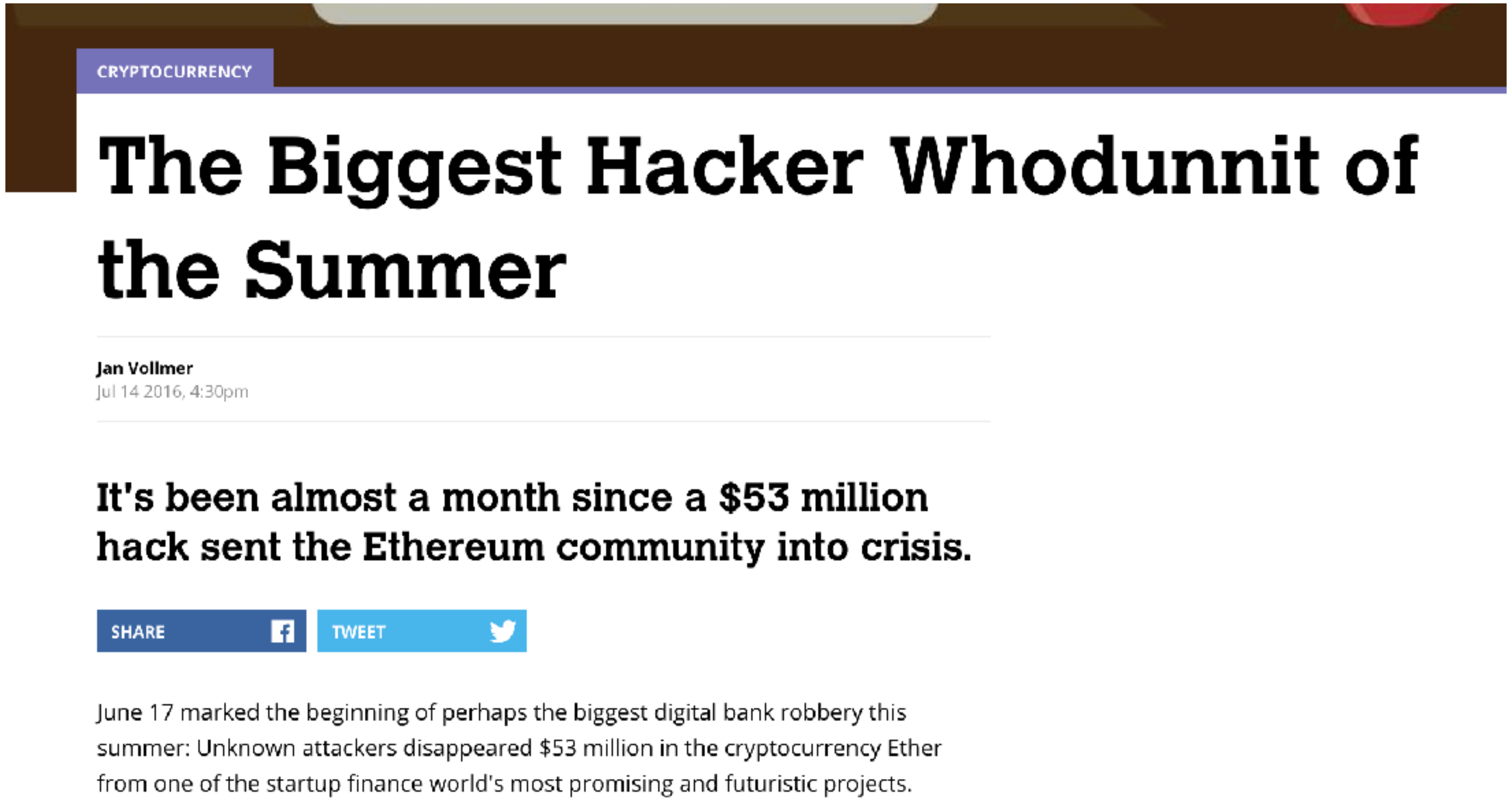
Lessons and solutions

- Simple input validation bug, would likely have been caught in a security audit of the code
- Soft fork de facto invalidated the transaction
- Fixed within 5h, patched in 0.3.10



```
+
+            // Check for negative or overflow input values
+            if (txPrev.vout[prevout.n].nValue < 0)
+                return error("ConnectInputs() : txin.nValue negative");
+            if (txPrev.vout[prevout.n].nValue > MAX_MONEY)
+                return error("ConnectInputs() : txin.nValue too high");
+            if (nValueIn > MAX_MONEY)
+                return error("ConnectInputs() : txin total too high");
```

2. Ethereum reentrancy (a.k.a. DAO bug)



CRYPTOCURRENCY

The Biggest Hacker Whodunnit of the Summer

Jan Vollmer
Jul 14 2016, 4:30pm

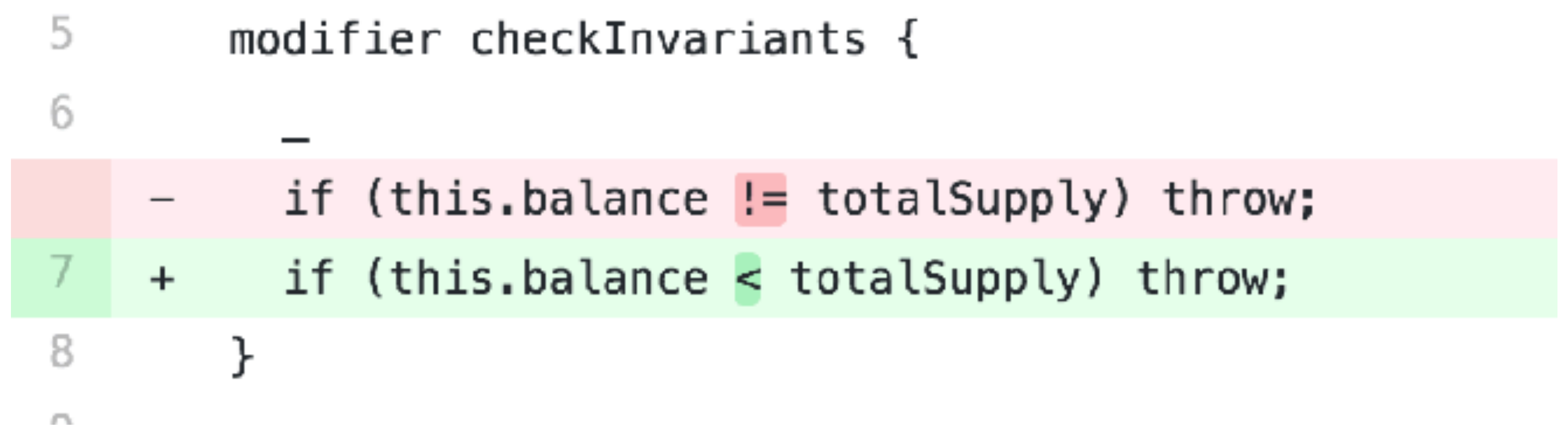
It's been almost a month since a $53 million hack sent the Ethereum community into crisis.

SHARE TWEET

June 17 marked the beginning of perhaps the biggest digital bank robbery this summer: Unknown attackers disappeared $53 million in the cryptocurrency Ether from one of the startup finance world's most promising and futuristic projects.

What happened? (*simplified*)

- Smart contract DAO.sol vulnerable to reentrancy (~smart contract cousin of concurrency)
- Attacker creates a contract that interacts with the vulnerable contract, to fool the contract into “thinking” it has more money than it actually has, using nested function calls
- Exploit idea: attacker “pretends" to withdraw money so that the contract “thinks" it has less money than it has, then attacker can steal this extra money..
- Mitigation: checks that the contract “thinks” it has **at least as much** money as the total supply when entering deposit and withdraw functions:

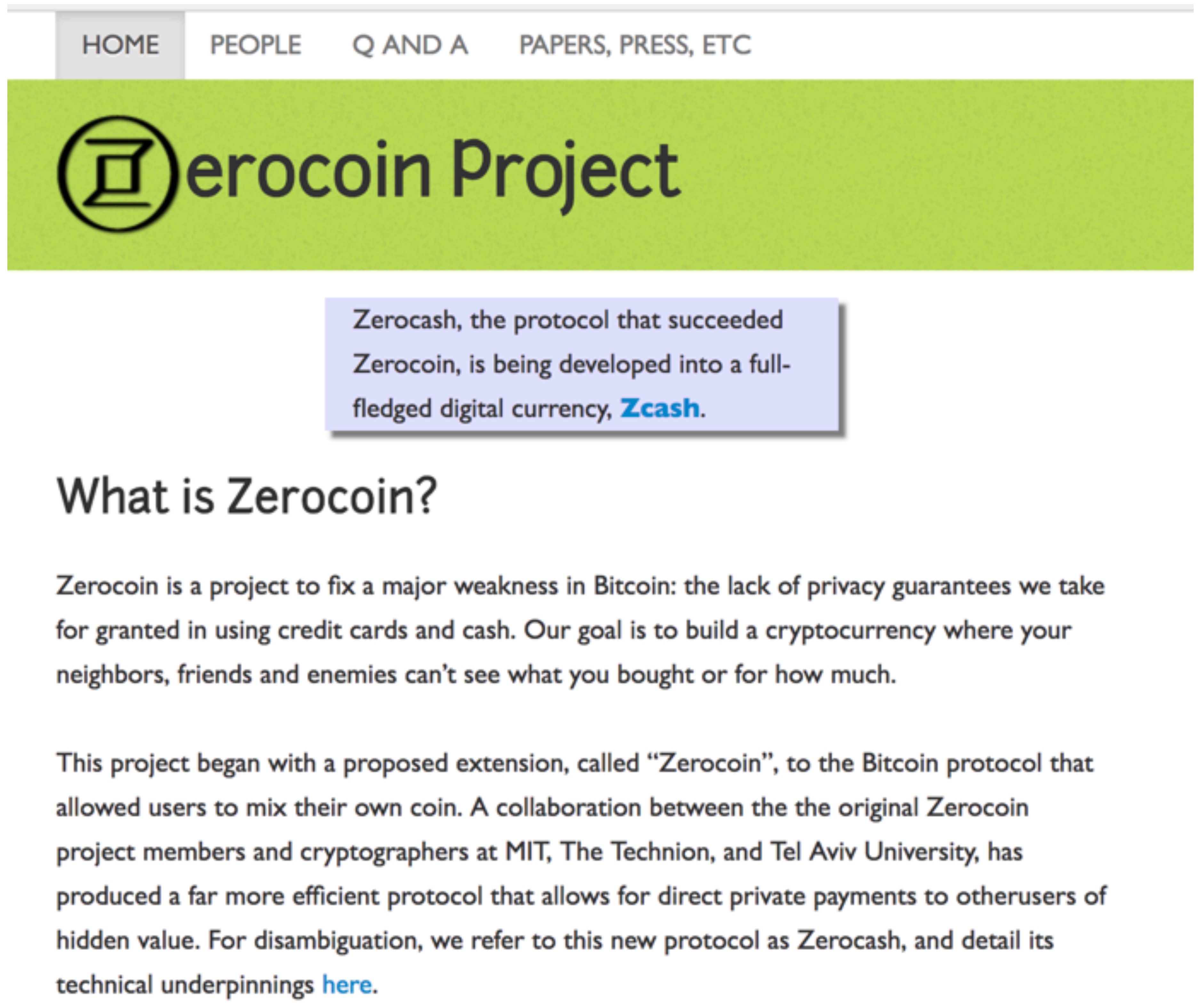


```
    modifier checkInvariants {
      _
-     if (this.balance != totalSupply) throw;
+     if (this.balance < totalSupply) throw;
    }
```

Lessons and solutions

- Smart contract programming and reasoning is hard, because based on an unconventional model
- Need for audits and best practices (safe arithmetic, avoid external calls, check invariants, etc.)
- Some platforms use simpler, non-Turing complete logic (yet richer than Bitcoin’s scripts) to reduce the risk

3. Zerocoin multi-spend



Zerocash, the protocol that succeeded Zerocoin, is being developed into a full-fledged digital currency, **Zcash**.

What is Zerocoin?

Zerocoin is a project to fix a major weakness in Bitcoin: the lack of privacy guarantees we take for granted in using credit cards and cash. Our goal is to build a cryptocurrency where your neighbors, friends and enemies can't see what you bought or for how much.

This project began with a proposed extension, called "Zerocoin", to the Bitcoin protocol that allowed users to mix their own coin. A collaboration between the the original Zerocoin project members and cryptographers at MIT, The Technion, and Tel Aviv University, has produced a far more efficient protocol that allows for direct private payments to otherusers of hidden value. For disambiguation, we refer to this new protocol as Zerocash, and detail its technical underpinnings here.

Experimental academic project, warned people not to use their code in production..

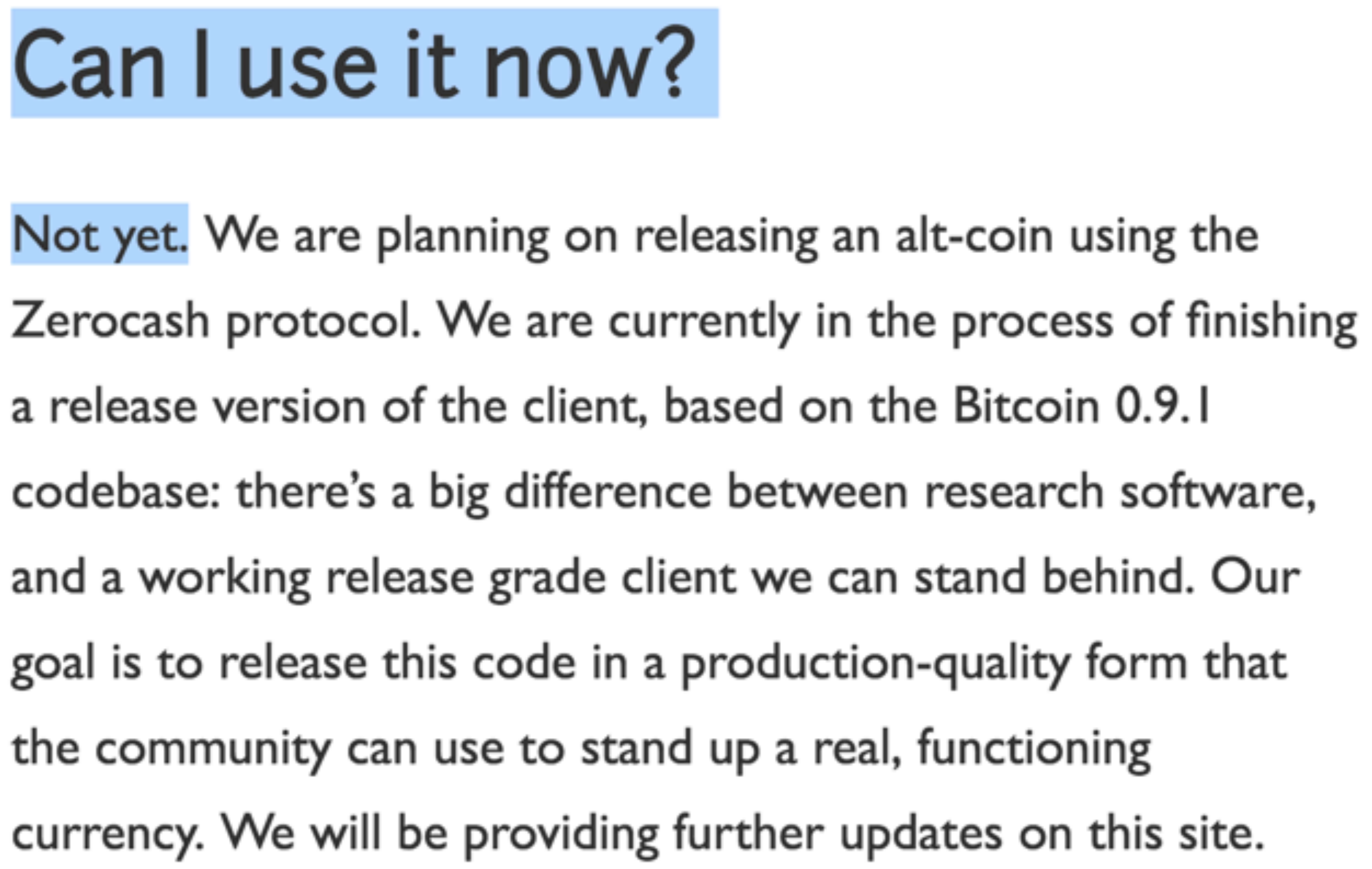
What happened?

zcoin
Private financial transactions, enabled by the Zerocoin Protocol
DOWNLOAD

- Zero-knowledge proof checking that a coin isn’t already spent before spending it..
- Checks the unicity of the “serial number”, an integer less than the group order..
- Bug: code actually using “serial mod q”'s value, without checking “serial < q"
- Consequence: coins with distinct serial numbers but same value mod q could all be spent, as if they were several copies of the same coin

Lessons and solutions

- Cryptography is fragile and complex to audit
- Don’t use experimental code for critical operations (especially if their authors warn you against it)



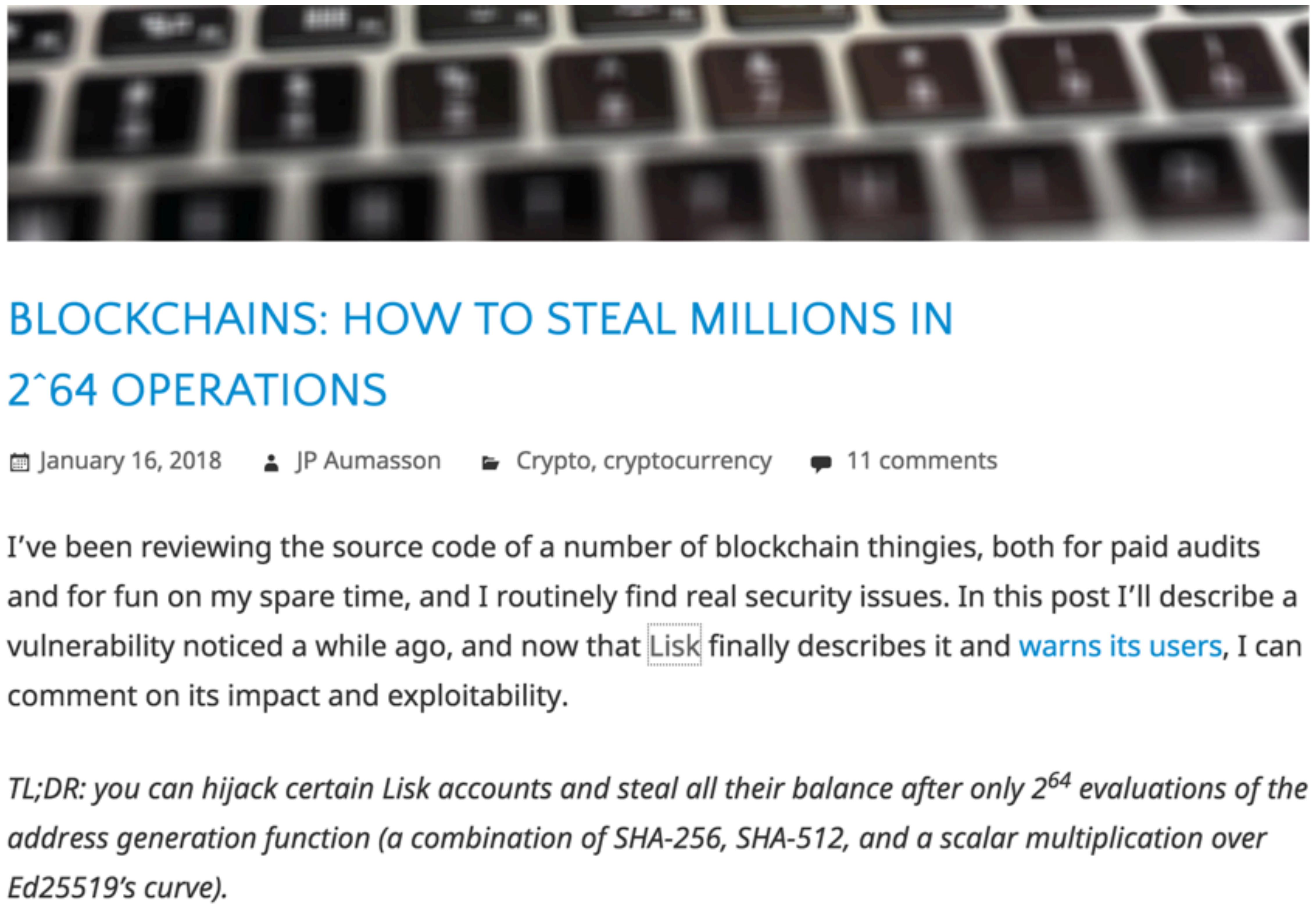
Can I use it now?

Not yet. We are planning on releasing an alt-coin using the Zerocash protocol. We are currently in the process of finishing a release version of the client, based on the Bitcoin 0.9.1 codebase: there’s a big difference between research software, and a working release grade client we can stand behind. Our goal is to release this code in a production-quality form that the community can use to stand up a real, functioning currency. We will be providing further updates on this site.

4. Lisk account hijack

Access the power of blockchain

Lisk makes it easy for developers to build and deploy blockchain applications in JavaScript. Join the leading ecosystem for world-changing dapps.



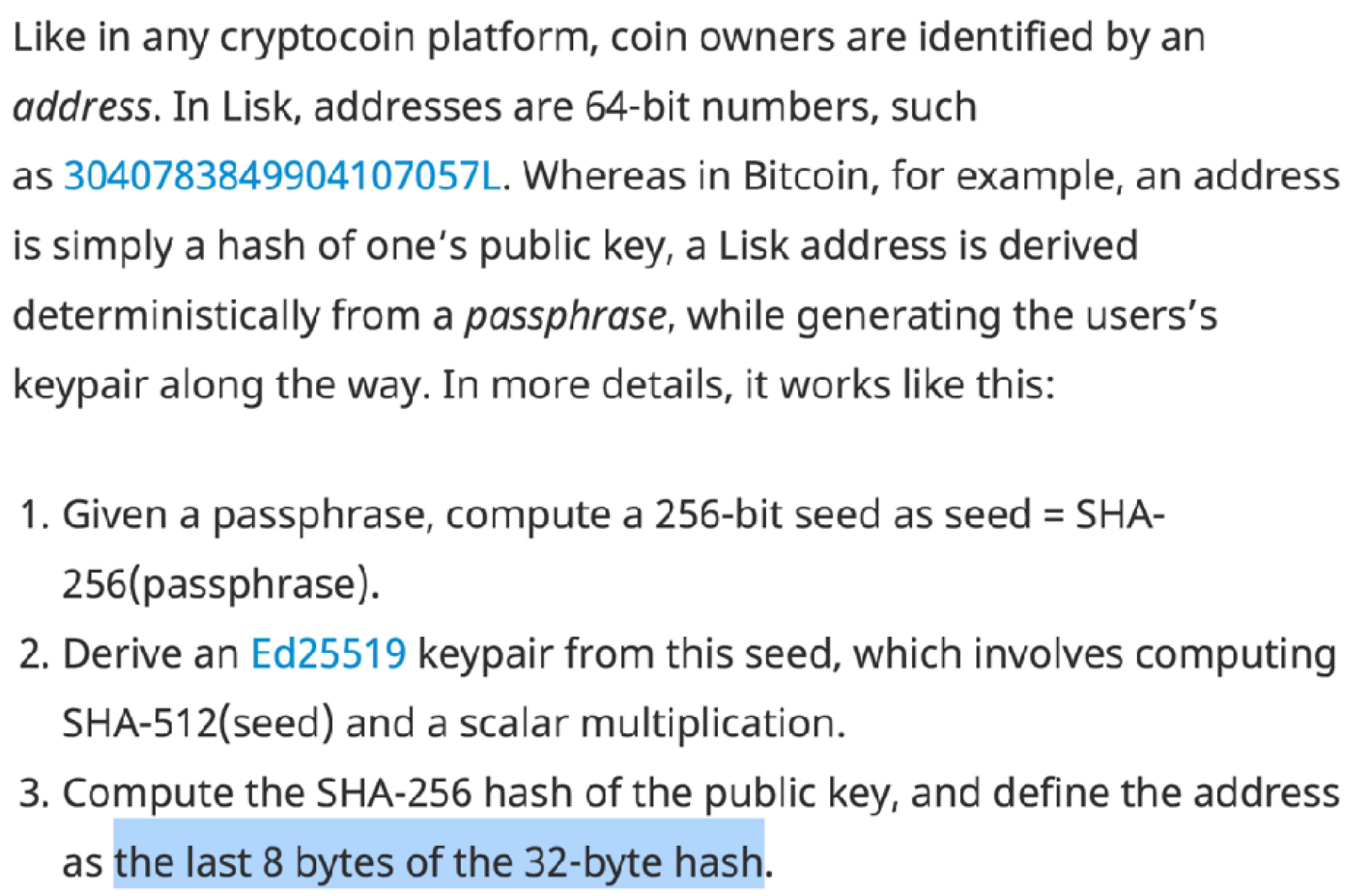
BLOCKCHAINS: HOW TO STEAL MILLIONS IN 2^64 OPERATIONS

January 16, 2018 · JP Aumasson · Crypto, cryptocurrency · 11 comments

I've been reviewing the source code of a number of blockchain thingies, both for paid audits and for fun on my spare time, and I routinely find real security issues. In this post I'll describe a vulnerability noticed a while ago, and now that Lisk finally describes it and warns its users, I can comment on its impact and exploitability.

TL;DR: you can hijack certain Lisk accounts and steal all their balance after only 2^{64} evaluations of the address generation function (a combination of SHA-256, SHA-512, and a scalar multiplication over Ed25519's curve).

What happened?

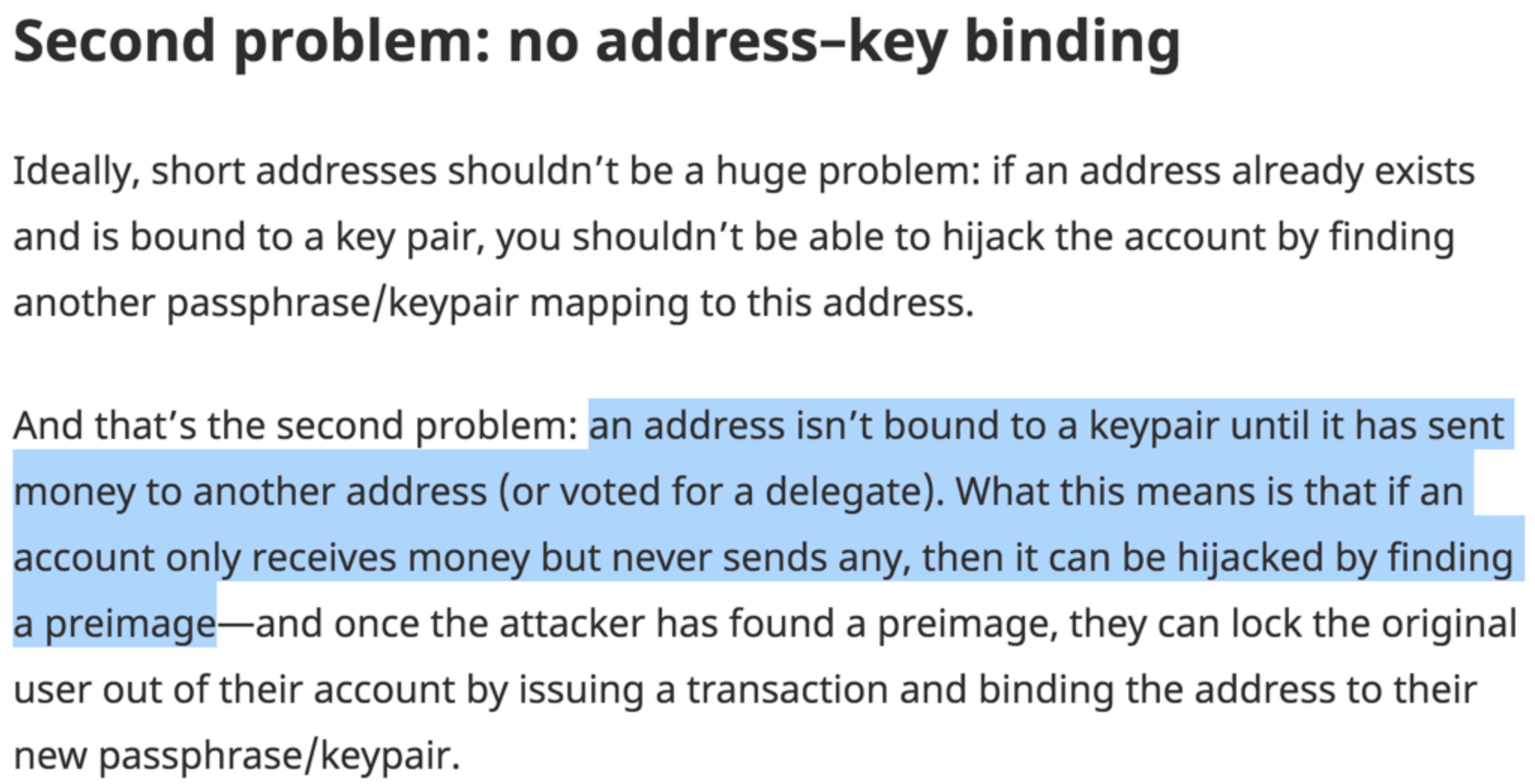


Like in any cryptocoin platform, coin owners are identified by an *address*. In Lisk, addresses are 64-bit numbers, such as 3040783849904107057L. Whereas in Bitcoin, for example, an address is simply a hash of one's public key, a Lisk address is derived deterministically from a *passphrase*, while generating the users's keypair along the way. In more details, it works like this:

1. Given a passphrase, compute a 256-bit seed as seed = SHA-256(passphrase).
2. Derive an Ed25519 keypair from this seed, which involves computing SHA-512(seed) and a scalar multiplication.
3. Compute the SHA-256 hash of the public key, and define the address as the last 8 bytes of the 32-byte hash.

- Trivial collisions Address(passphrase 1) = Address(passphrase 2)
- Preimage search for a valid passphrase of a given address in ~2^{64} trials

What happened?



Second problem: no address–key binding

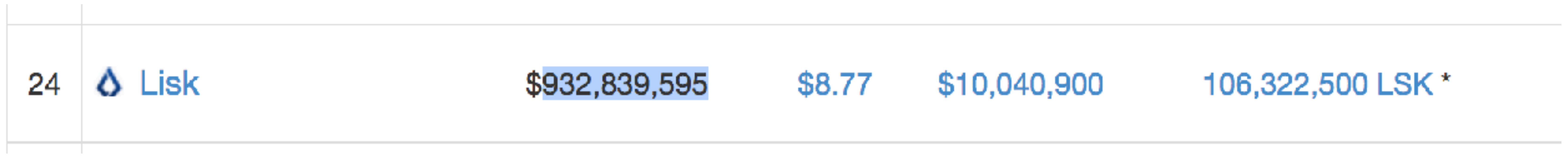
Ideally, short addresses shouldn't be a huge problem: if an address already exists and is bound to a key pair, you shouldn't be able to hijack the account by finding another passphrase/keypair mapping to this address.

And that's the second problem: an address isn't bound to a keypair until it has sent money to another address (or voted for a delegate). What this means is that if an account only receives money but never sends any, then it can be hijacked by finding a preimage—and once the attacker has found a preimage, they can lock the original user out of their account by issuing a transaction and binding the address to their new passphrase/keypair.

- Multi-target attack will do ~2^{49} operations to hit one-of-64 target addresses with 256 cores
- Lisk recommends a workaround: send a least one transaction from any new address in order to broadcast the public key to the network (thus binding the key to the address)

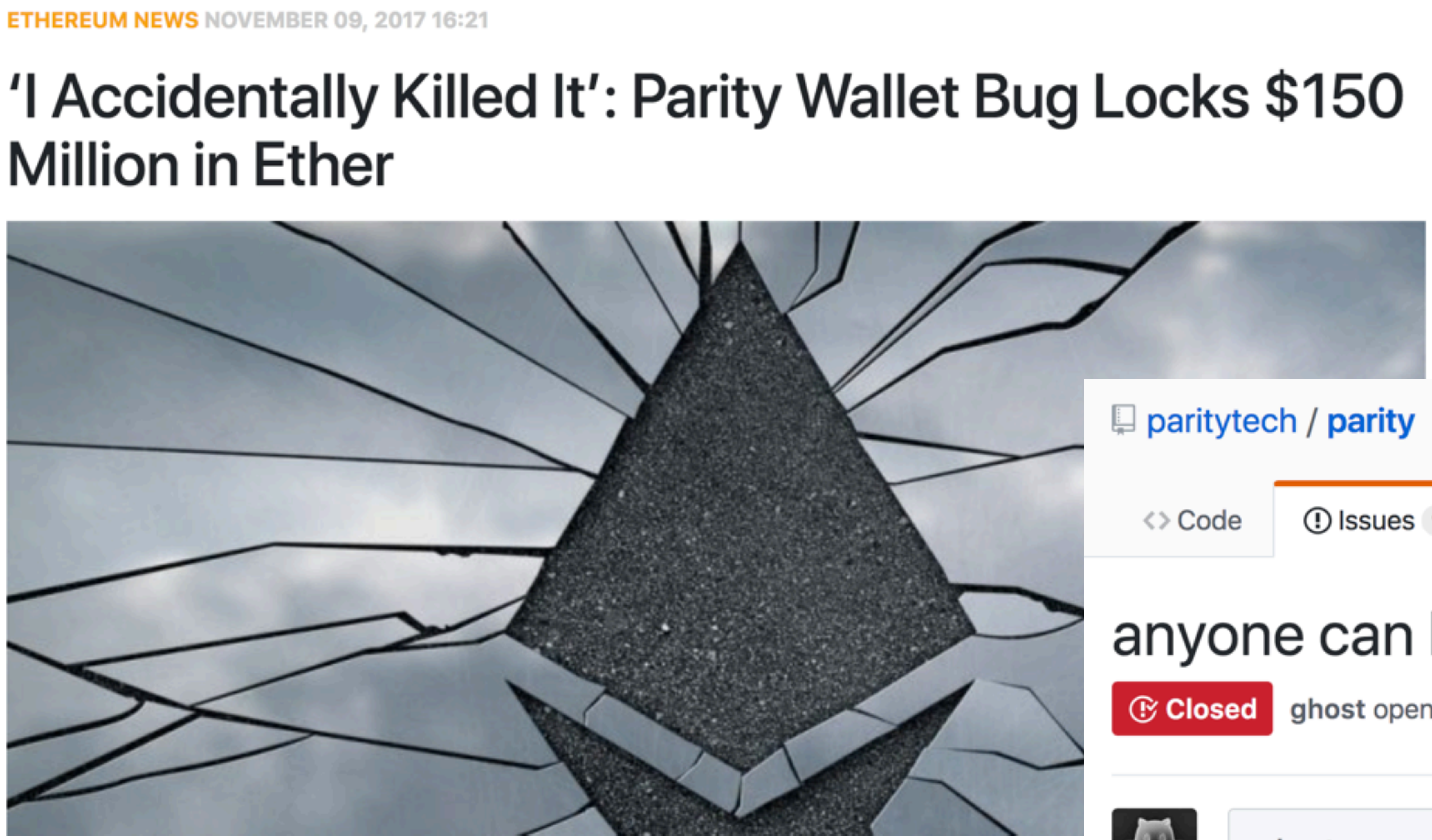
Lessons and solutions

- Flawed address derivation scheme, despite good choice of cryptographic algorithm
- Other embarrassing security issues in Lisk (secret sent in clear to validators, etc.), evidently wasn't designed nor reviewed by security people
- Lisk published an advisory, but cannot fix the problem



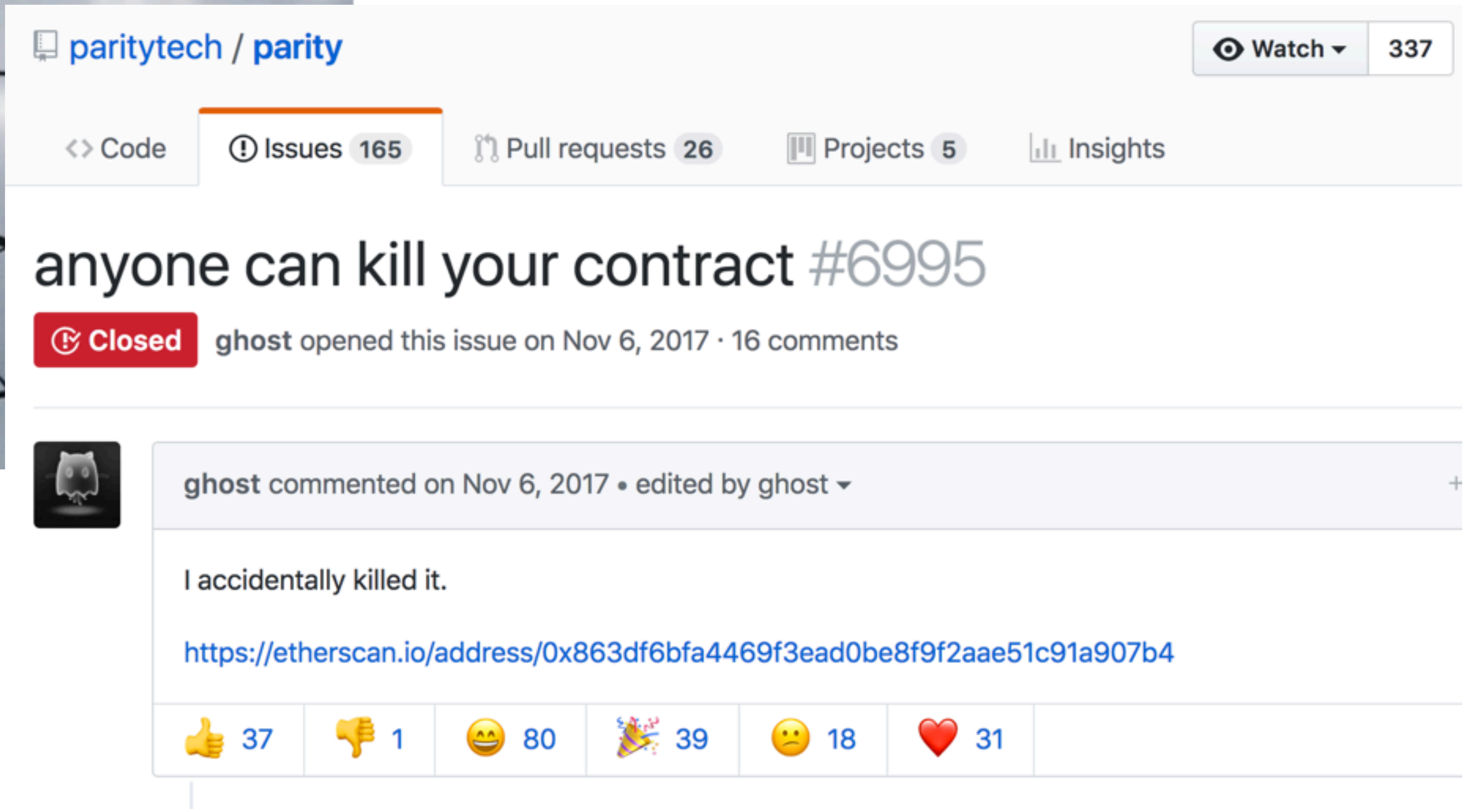
24	Lisk	$932,839,595	$8.77	$10,040,900	106,322,500 LSK *

5. Parity wallet bug



ETHEREUM NEWS NOVEMBER 09, 2017 16:21

'I Accidentally Killed It': Parity Wallet Bug Locks $150 Million in Ether



paritytech / parity

Watch 337

Code | Issues 165 | Pull requests 26 | Projects 5 | Insights

anyone can kill your contract #6995

Closed ghost opened this issue on Nov 6, 2017 · 16 comments

ghost commented on Nov 6, 2017 • edited by ghost

I accidentally killed it.

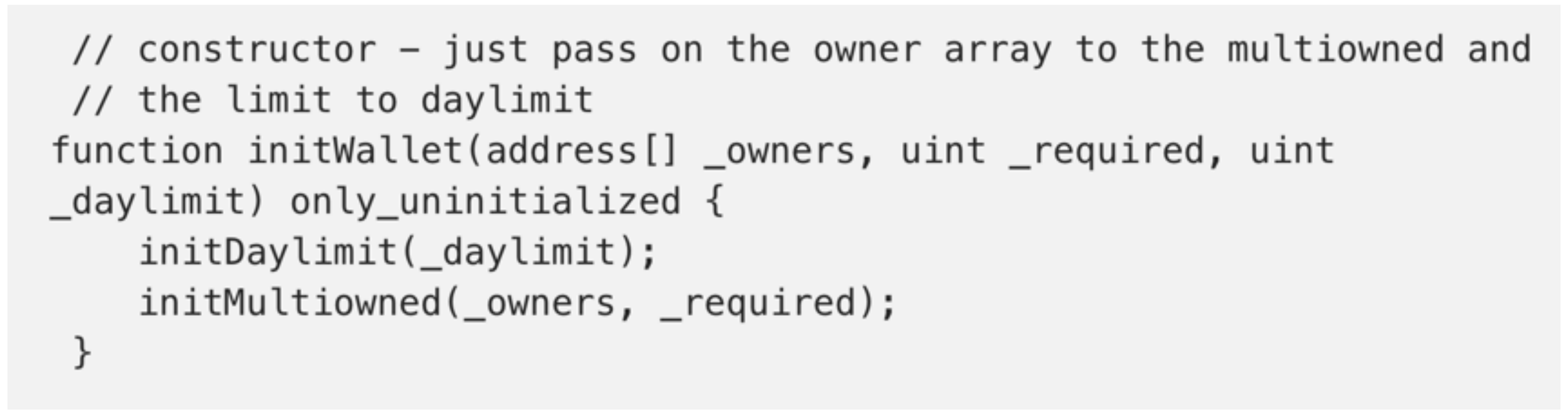
https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4

👍 37 👎 1 😄 80 🎉 39 😕 18 ❤️ 31

Whoops!

What happened?

- Library contract with **uninitialized owner**
- Attacker took over the contract using the `initWallet()` function

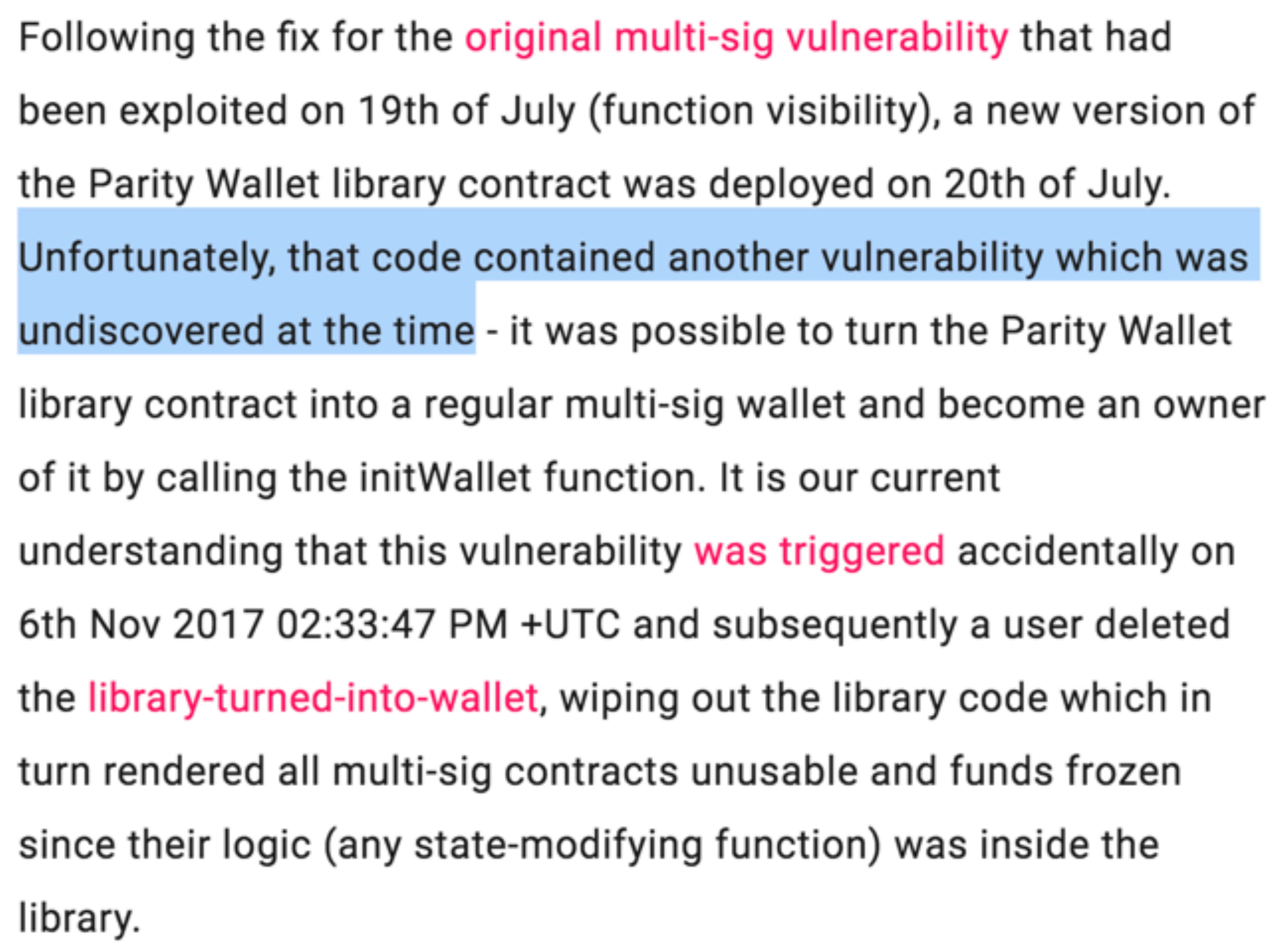


```
 // constructor - just pass on the owner array to the multiowned and
 // the limit to daylimit
function initWallet(address[] _owners, uint _required, uint
_daylimit) only_uninitialized {
    initDaylimit(_daylimit);
    initMultiowned(_owners, _required);
 }
```

- Then he killed the contract, thereby **freezing all wallets** that were dependent on this library

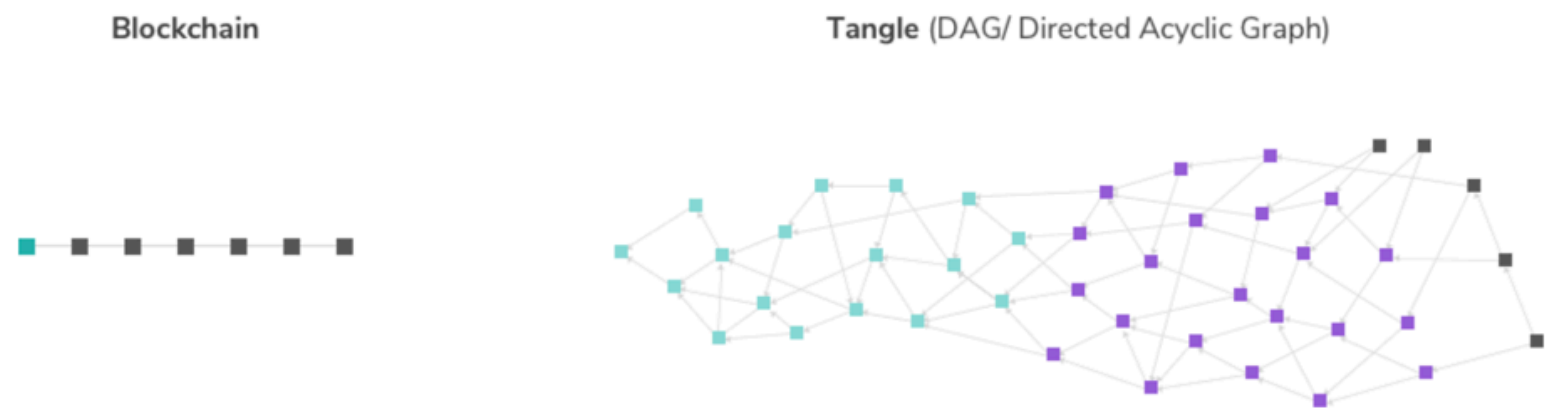
Lessons and solutions

- Smart contracts are hard, embarrassing issue for Parity
- Issue actually in the patch of a previous security issue (!)

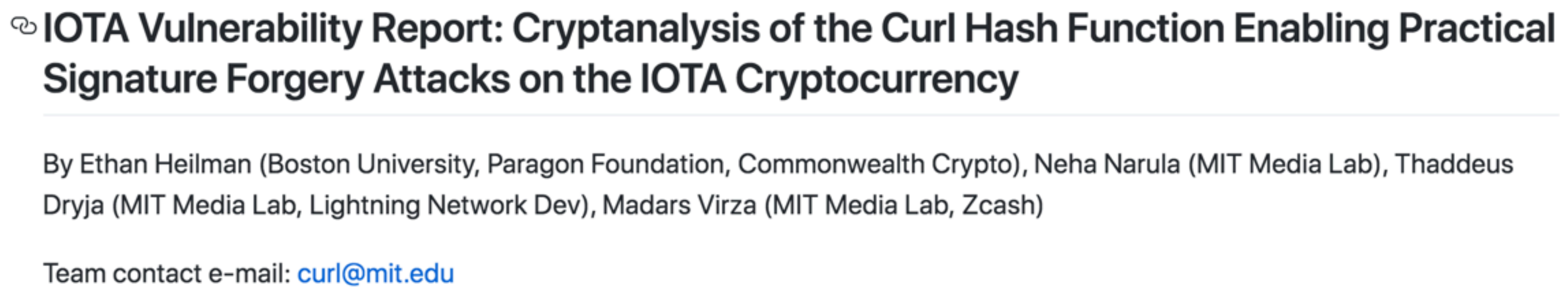


Following the fix for the original multi-sig vulnerability that had been exploited on 19th of July (function visibility), a new version of the Parity Wallet library contract was deployed on 20th of July. Unfortunately, that code contained another vulnerability which was undiscovered at the time - it was possible to turn the Parity Wallet library contract into a regular multi-sig wallet and become an owner of it by calling the initWallet function. It is our current understanding that this vulnerability was triggered accidentally on 6th Nov 2017 02:33:47 PM +UTC and subsequently a user deleted the library-turned-into-wallet, wiping out the library code which in turn rendered all multi-sig contracts unusable and funds frozen since their logic (any state-modifying function) was inside the library.

6. IOTA's hash function



IOTA = blockchain-less blockchain, using "trinary" rather than binary arithmetic, and… custom crypto!



IOTA Vulnerability Report: Cryptanalysis of the Curl Hash Function Enabling Practical Signature Forgery Attacks on the IOTA Cryptocurrency

By Ethan Heilman (Boston University, Paragon Foundation, Commonwealth Crypto), Neha Narula (MIT Media Lab), Thaddeus Dryja (MIT Media Lab, Lightning Network Dev), Madars Virza (MIT Media Lab, Zcash)

Team contact e-mail: curl@mit.edu

What happened?

- IOTA’s transaction signatures are not standard public key signatures..
- But one-time hash-based signatures, using key = f(seed, index), with incremented index..
- Instead using a standard hash function, IOTA designed **a custom hash** (“curl”)…
- Which turned out to be completely insecure (collisions easy to find)

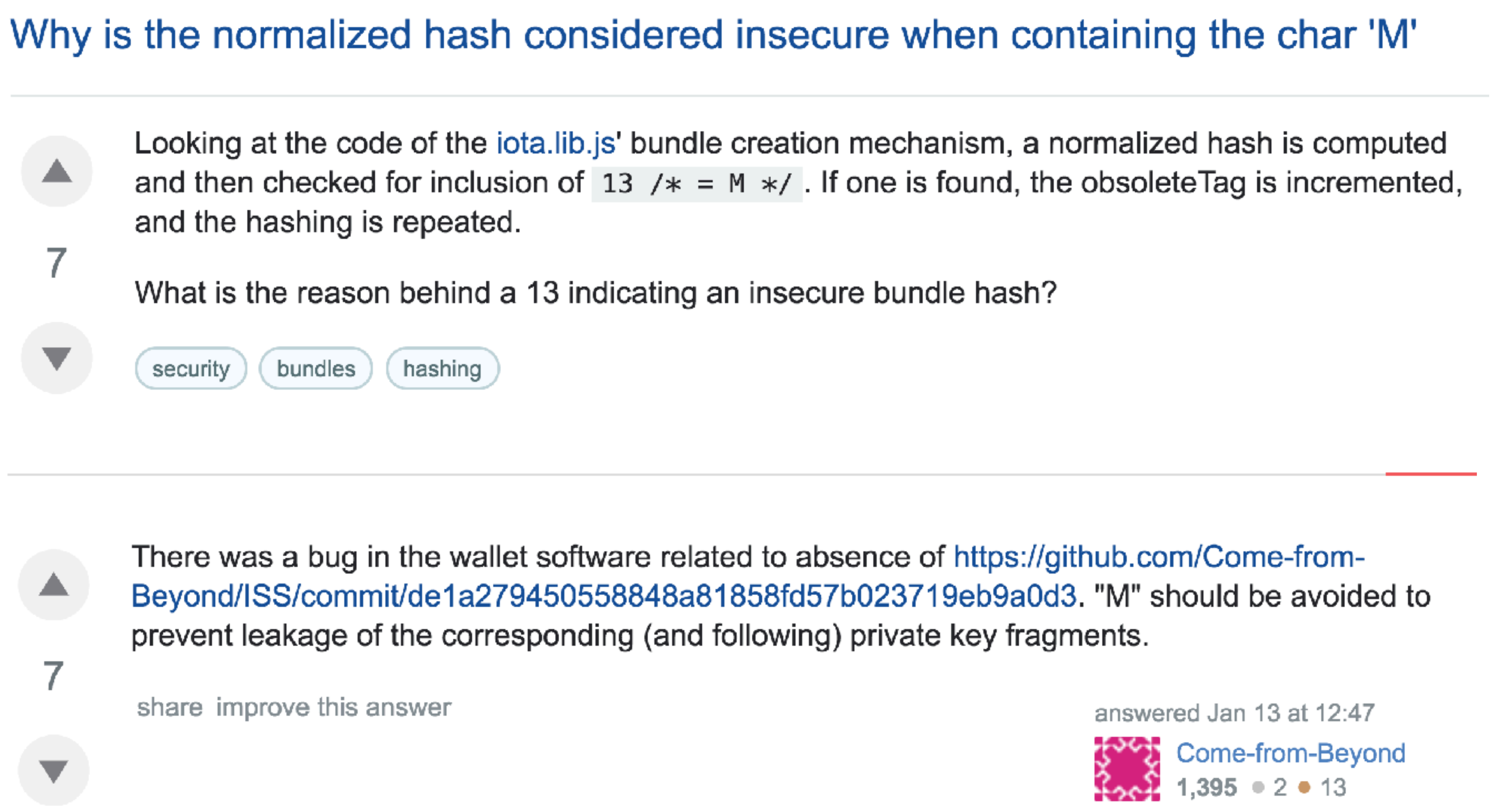
What happened?

- IOTA’s transaction signatures are not standard public key signatures..
- But one-time hash-based signatures, using key = f(seed, index), with incremented index..
- Instead using a standard hash function, IOTA designed **a custom hash** (“curl”)…
- Which turned out to be completely insecure (collisions easy to find)
- But… exploitation not trivial because each key is used once.. still a security issue

Lessons and solutions

- Crypto by non-cryptographers is often a disaster
- Innovating is good, but frankenstein experiments with $Bs at stake can be scary, should be done responsibly
- PR disaster for IOTA, but token value didn’t suffer much
- IOTA needs more analysis!

6 bis. IOTA’s “M"



Why is the normalized hash considered insecure when containing the char 'M'

7

Looking at the code of the iota.lib.js' bundle creation mechanism, a normalized hash is computed and then checked for inclusion of `13 /* = M */`. If one is found, the obsoleteTag is incremented, and the hashing is repeated.

What is the reason behind a 13 indicating an insecure bundle hash?

security bundles hashing

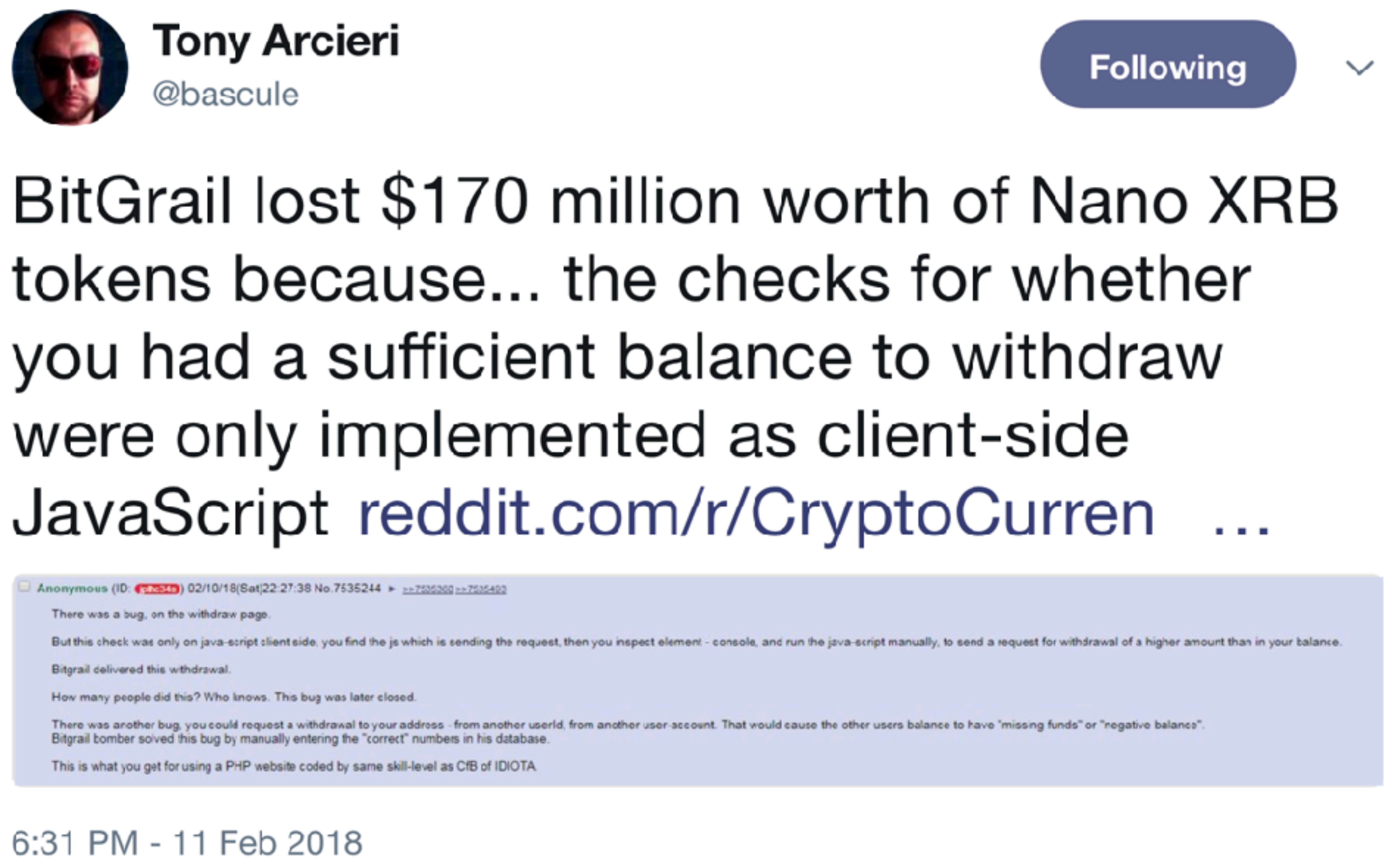
7

There was a bug in the wallet software related to absence of https://github.com/Come-from-Beyond/ISS/commit/de1a279450558848a81858fd57b023719eb9a0d3. "M" should be avoided to prevent leakage of the corresponding (and following) private key fragments.

share improve this answer

answered Jan 13 at 12:47
Come-from-Beyond
1,395 ● 2 ● 13

7. Bitgrail withdrawals



Tony Arcieri
@bascule

Following

BitGrail lost $170 million worth of Nano XRB tokens because... the checks for whether you had a sufficient balance to withdraw were only implemented as client-side JavaScript reddit.com/r/CryptoCurren …

Anonymous (ID: [illegible]) 02/10/18(Sat)22:27:38 No.7535244 ▸ >>7535360 >>7535493

There was a bug, on the withdraw page.

But this check was only on java-script client side, you find the js which is sending the request, then you inspect element - console, and run the java-script manually, to send a request for withdrawal of a higher amount than in your balance.

Bitgrail delivered this withdrawal.

How many people did this? Who knows. This bug was later closed.

There was another bug, you could request a withdrawal to your address - from another userId, from another user-account. That would cause the other users balance to have "missing funds" or "negative balance".
Bitgrail bomber solved this bug by manually entering the "correct" numbers in his database.

This is what you get for using a PHP website coded by same skill-level as CfB of IDIOTA.

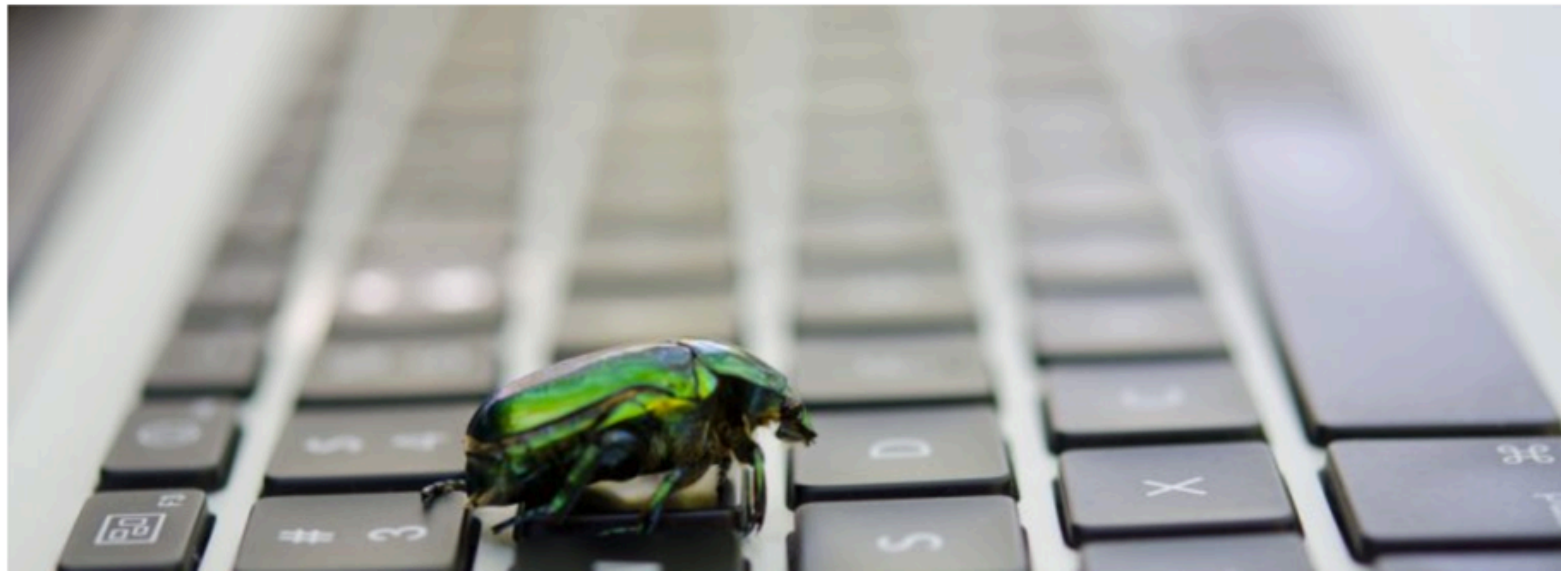
6:31 PM - 11 Feb 2018

Reminder: client-side validation can't be trusted

8. BatchOverflow

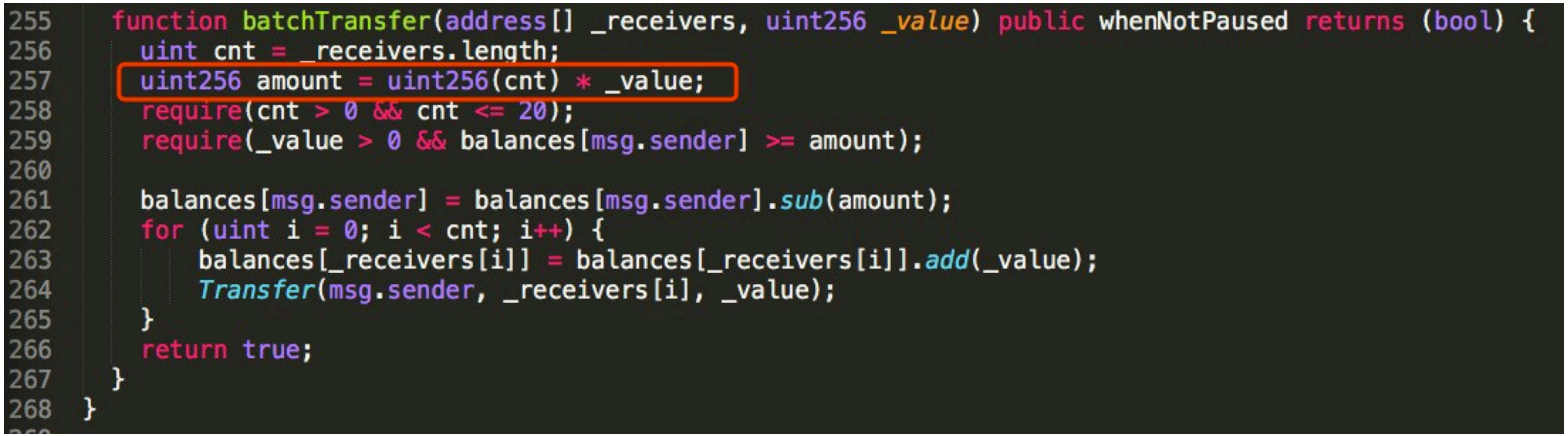
ETHEREUM NEWS APRIL 25, 2018 14:42

OKEx Suspends ERC20 Deposits on Discovery of Critical Ethereum Smart Contract Bug



What happened?

- Several alt coins added a “batch transfer” functionality to ERC20 contracts..
- Sends a same amount `_value` of tokens to a list of receivers:

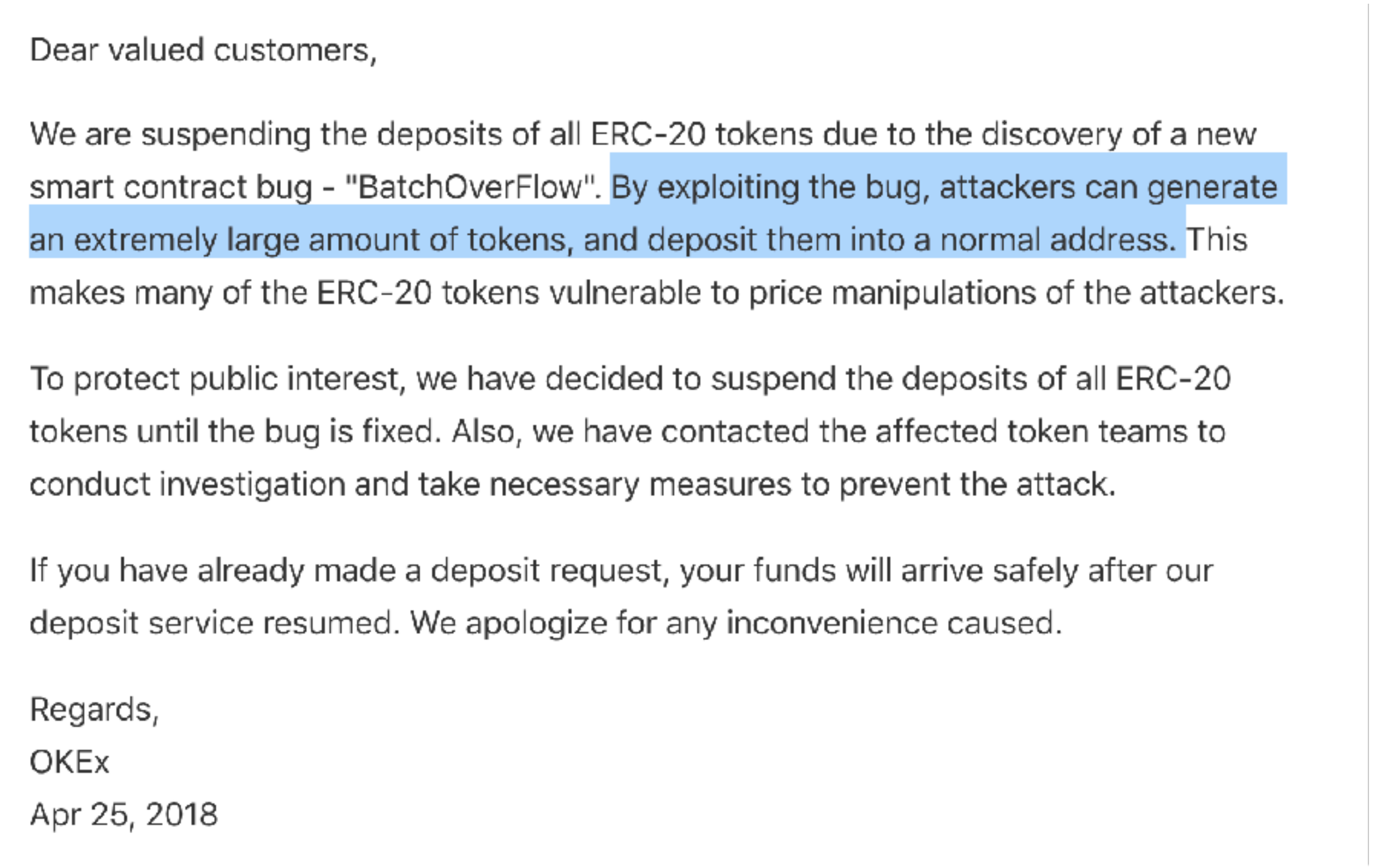


```
255   function batchTransfer(address[] _receivers, uint256 _value) public whenNotPaused returns (bool) {
256     uint cnt = _receivers.length;
257     uint256 amount = uint256(cnt) * _value;
258     require(cnt > 0 && cnt <= 20);
259     require(_value > 0 && balances[msg.sender] >= amount);
260
261     balances[msg.sender] = balances[msg.sender].sub(amount);
262     for (uint i = 0; i < cnt; i++) {
263         balances[_receivers[i]] = balances[_receivers[i]].add(_value);
264         Transfer(msg.sender, _receivers[i], _value);
265     }
266     return true;
267   }
268 }
```

- Integer overflow in total amount computation: attacker can set `amount` to a low (or even zero) value, passing the check line 259, yet sending high amount `_value` to receivers

Lessons and solutions

- Beginner error in simple smart contract function
- Would have been spotted in 5min in a security audit



Dear valued customers,

We are suspending the deposits of all ERC-20 tokens due to the discovery of a new smart contract bug - "BatchOverFlow". By exploiting the bug, attackers can generate an extremely large amount of tokens, and deposit them into a normal address. This makes many of the ERC-20 tokens vulnerable to price manipulations of the attackers.

To protect public interest, we have decided to suspend the deposits of all ERC-20 tokens until the bug is fixed. Also, we have contacted the affected token teams to conduct investigation and take necessary measures to prevent the attack.

If you have already made a deposit request, your funds will arrive safely after our deposit service resumed. We apologize for any inconvenience caused.

Regards,
OKEx
Apr 25, 2018

9. Verge

The Verge Hack, Explained

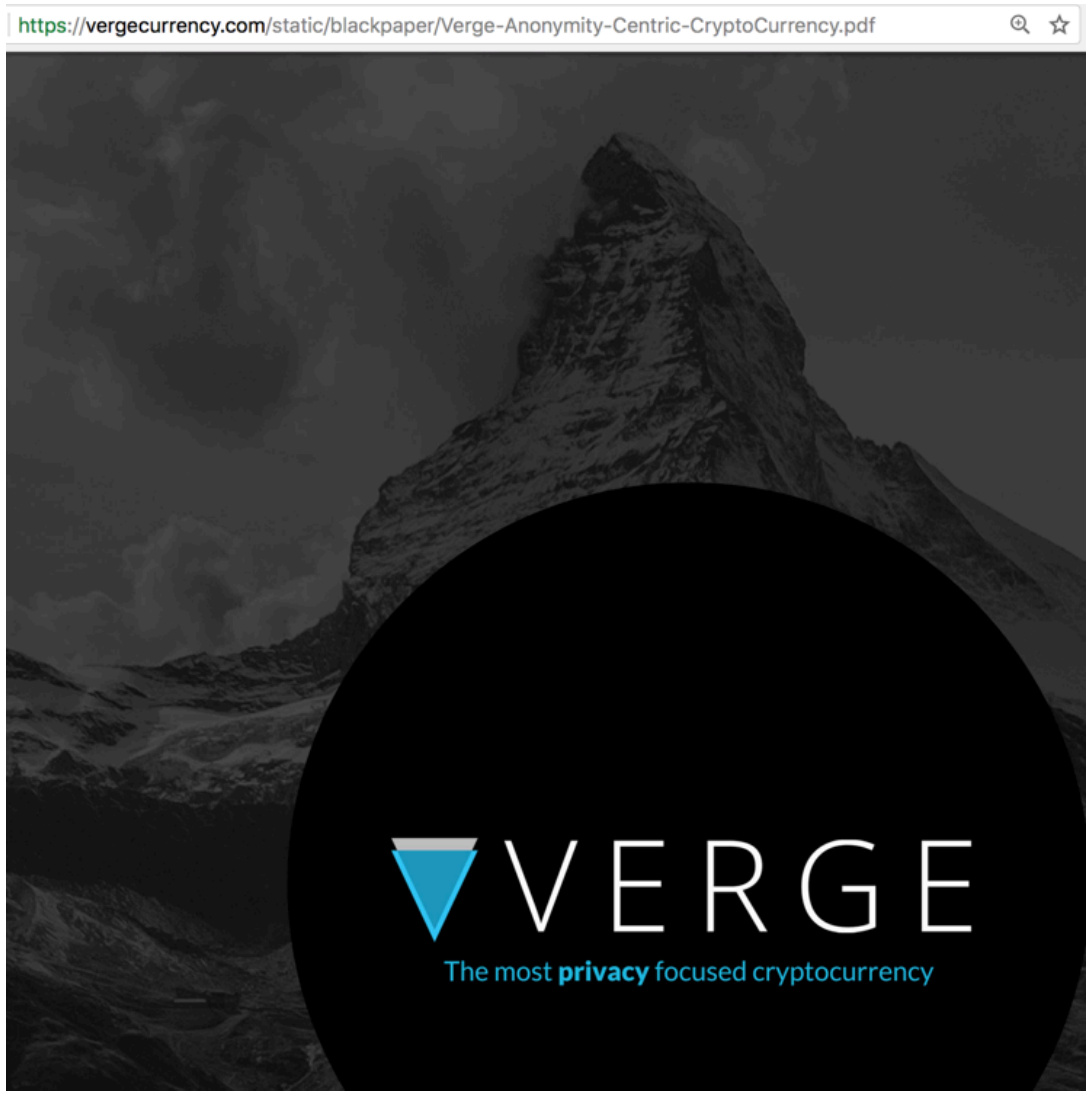
Time Warps, Mining Exploits, Denial of Service, and More!

Privacy as a *choice*.
A secure and anonymous cryptocurrency.

Built with a focus on privacy.

Read the *Blackpaper* of Verge Currency

9. Verge

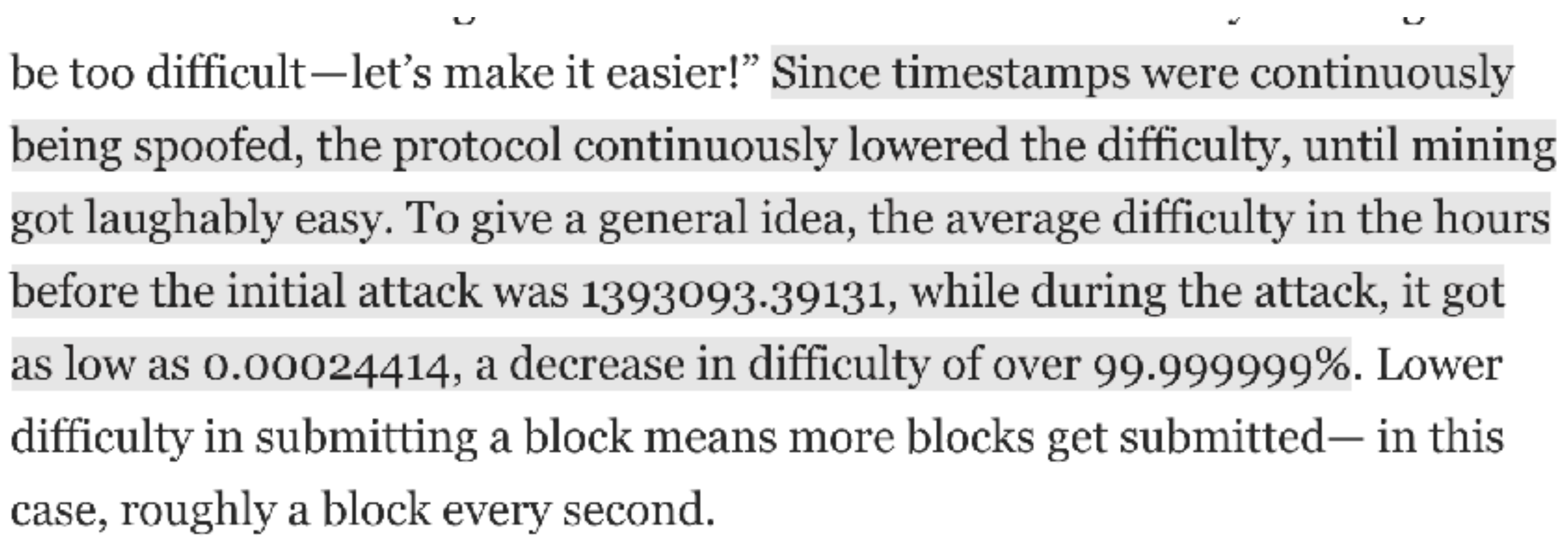

https://vergecurrency.com/static/blackpaper/Verge-Anonymity-Centric-CryptoCurrency.pdf

VERGE

The most **privacy** focused cryptocurrency

What happened?

- Consensus protocol adapts the proof-of-work difficulty based on the number of blocks confirmed in the **last two hours**..
- Attacker kept sending blocks with spoofed (earlier) timestamps to fool the difficulty calculation algorithm..



be too difficult—let's make it easier!" Since timestamps were continuously being spoofed, the protocol continuously lowered the difficulty, until mining got laughably easy. To give a general idea, the average difficulty in the hours before the initial attack was 1393093.39131, while during the attack, it got as low as 0.00024414, a decrease in difficulty of over 99.999999%. Lower difficulty in submitting a block means more blocks get submitted— in this case, roughly a block every second.

- Exploit facilitated by Verge's (nonsensical) use of 5 hashing algorithms: attacker only lowered the difficulty for Scrypt, thereby only competing with Scrypt miners

Lessons and solutions

- Complexity tends to bring insecurity
- Secure-sounding features (5 algorithms!) can backfire
- Consensus parameters depending on untrusted input is a recipe for disasters
- Many coins vulnerable to time manipulation (e.g. via forged NTP responses that could fool the OS)
- A black paper instead of a white paper doesn’t help

10. BIP32 utilities

Derivation of BIP32 accounts (key pair, address) from a seed and a path
https://github.com/prusnak/bip32utils (Python)

```
$ bip32gen
usage: bip32gen [-h] [-x] [-X] -i {entropy,xprv,xpub} [-n AMOUNT]
                [-f FROM_FILE] [-F TO_FILE] -o OUTPUT_TYPE [-v] [-d]
                chain [chain ...]
bip32gen: error: the following arguments are required:
-i/--input-type, -o/--output-type, chain
```

What happened? (1/2)

Generate an address from a 32-byte seed:

```
$ echo bc0ef283f57fd5e4f36657053228eae8d2d5b0e4d87c6ee069a9cade39411d63 |
bip32gen -x -i entropy  -o addr m
1Jzuo5xm62i8gFQLQb58f2F5a7nTK3o8bD
```

What happened? (1/2)

Generate an address from a 32-byte seed:

```
$ echo bc0ef283f57fd5e4f36657053228eae8d2d5b0e4d87c6ee069a9cade39411d63 |
bip32gen -x -i entropy  -o addr m
1Jzuo5xm62i8gFQLQb58f2F5a7nTK3o8bD
```

Generate an address from the 16-byte truncated seed:

```
$ echo bc0ef283f57fd5e4f36657053228eae8 | bip32gen -x -i entropy -o addr m
1Jzuo5xm62i8gFQLQb58f2F5a7nTK3o8bD
```

What happened? (1/2)

Generate an address from a 32-byte seed:

```
$ echo bc0ef283f57fd5e4f36657053228eae8d2d5b0e4d87c6ee069a9cade39411d63 |
bip32gen -x -i entropy  -o addr m
1Jzuo5xm62i8gFQLQb58f2F5a7nTK3o8bD
```

Generate an address from the 16-byte truncated seed:

```
$ echo bc0ef283f57fd5e4f36657053228eae8 | bip32gen -x -i entropy -o addr m
1Jzuo5xm62i8gFQLQb58f2F5a7nTK3o8bD
```

WTF?!

What happened? (1/2)

Generate an address from a 32-byte seed:

```
$ echo bc0ef283f57fd5e4f36657053228eae8d2d5b0e4d87c6ee069a9cade39411d63 |
bip32gen -x -i entropy  -o addr m
1Jzuo5xm62i8gFQLQb58f2F5a7nTK3o8bD
```

Generate an address from the 16-byte truncated seed:

```
$ echo bc0ef283f57fd5e4f36657053228eae8 | bip32gen -x -i entropy -o addr m
1Jzuo5xm62i8gFQLQb58f2F5a7nTK3o8bD
```

WTF?!

- Length of the seed is a parameter `-n AMOUNT`, default to 16 bytes
- Longer seeds will be silently truncated to 16 bytes without warning/error
- Did a PR to fix this, now merged..

What happened? (2/2)

bip32gen keeps deriving incorrect xpub/xpriv values 🤦‍♂️

For certain BIP32 paths only…

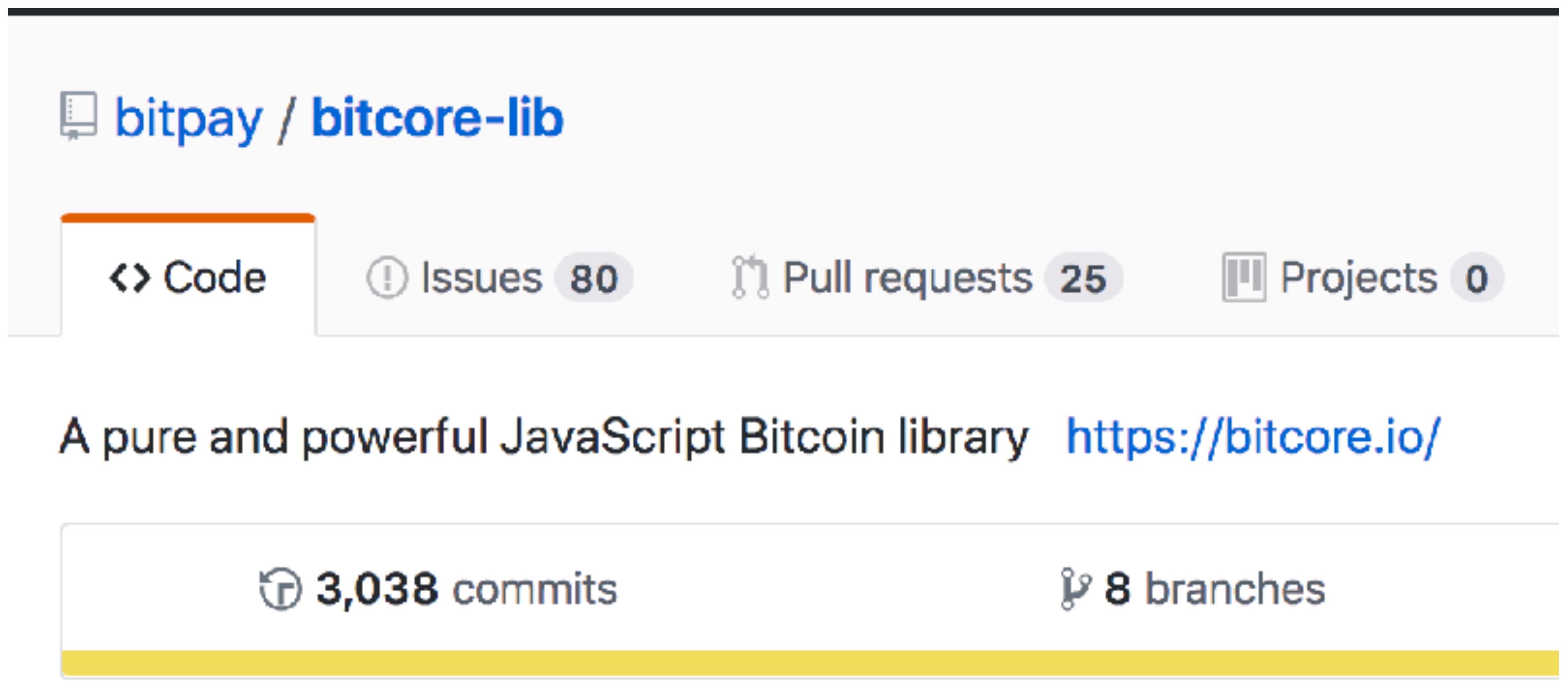
Gave up investigating, enough hours wasted,
switched to a more established (but bigger) tool in JS…

What happened? (2/2)

bip32gen keeps deriving incorrect xpub/xpriv/address 🤦‍♂️

Risk: get an address for which you don't have the key

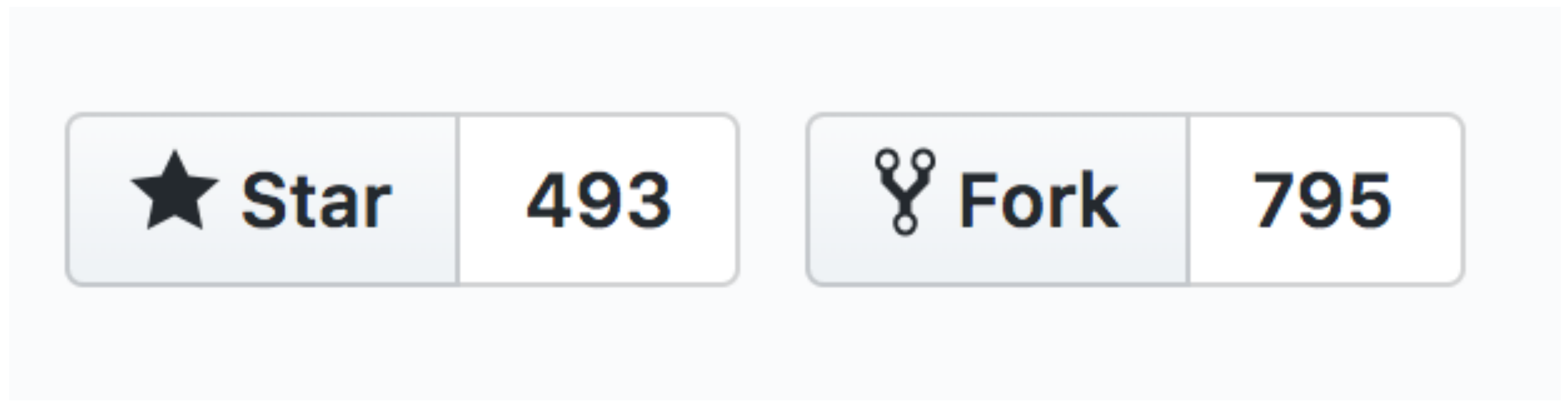
Gave up investigating, enough hours wasted,
switched to a more established (but bigger) tool in JS…



bitpay / bitcore-lib

A pure and powerful JavaScript Bitcoin library https://bitcore.io/

3,038 commits 8 branches



What happened? (3/2)

BIP32 key derivation failed on my macbook..

What happened? (3/2)

BIP32 key derivation failed on my macbook..

Pinpointed the problem to… **SHA-512**

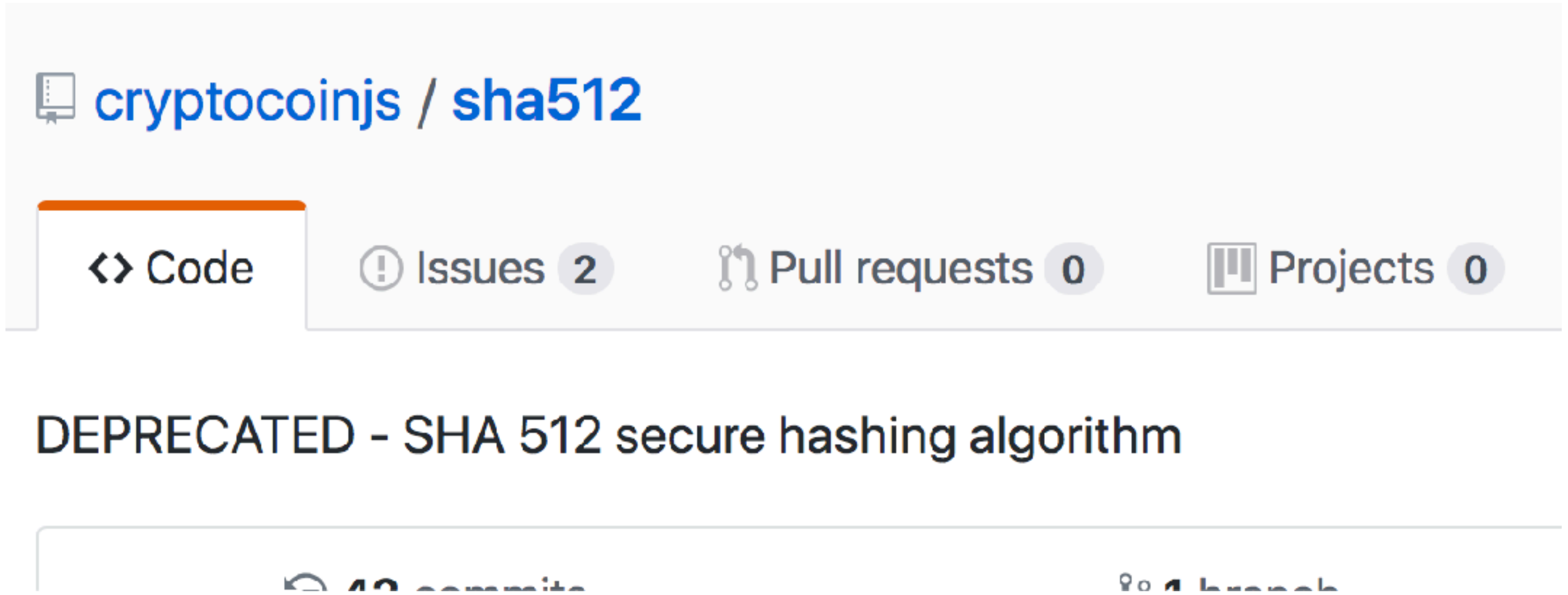
Which SHA-512 implementation? 🤔

What happened? (3/2)

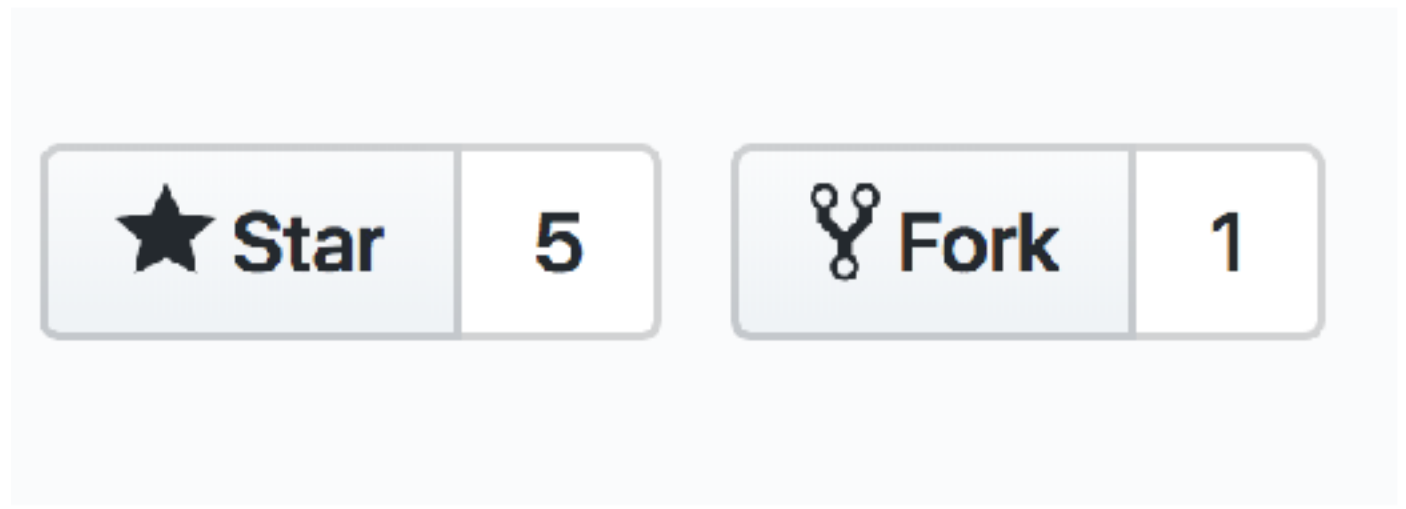
BIP32 key derivation failed on my macbook..

Pinpointed the problem to… **SHA-512**

Which SHA-512 implementation? 🤔


cryptocoinjs / sha512

DEPRECATED - SHA 512 secure hashing algorithm



What happened? (4/2)

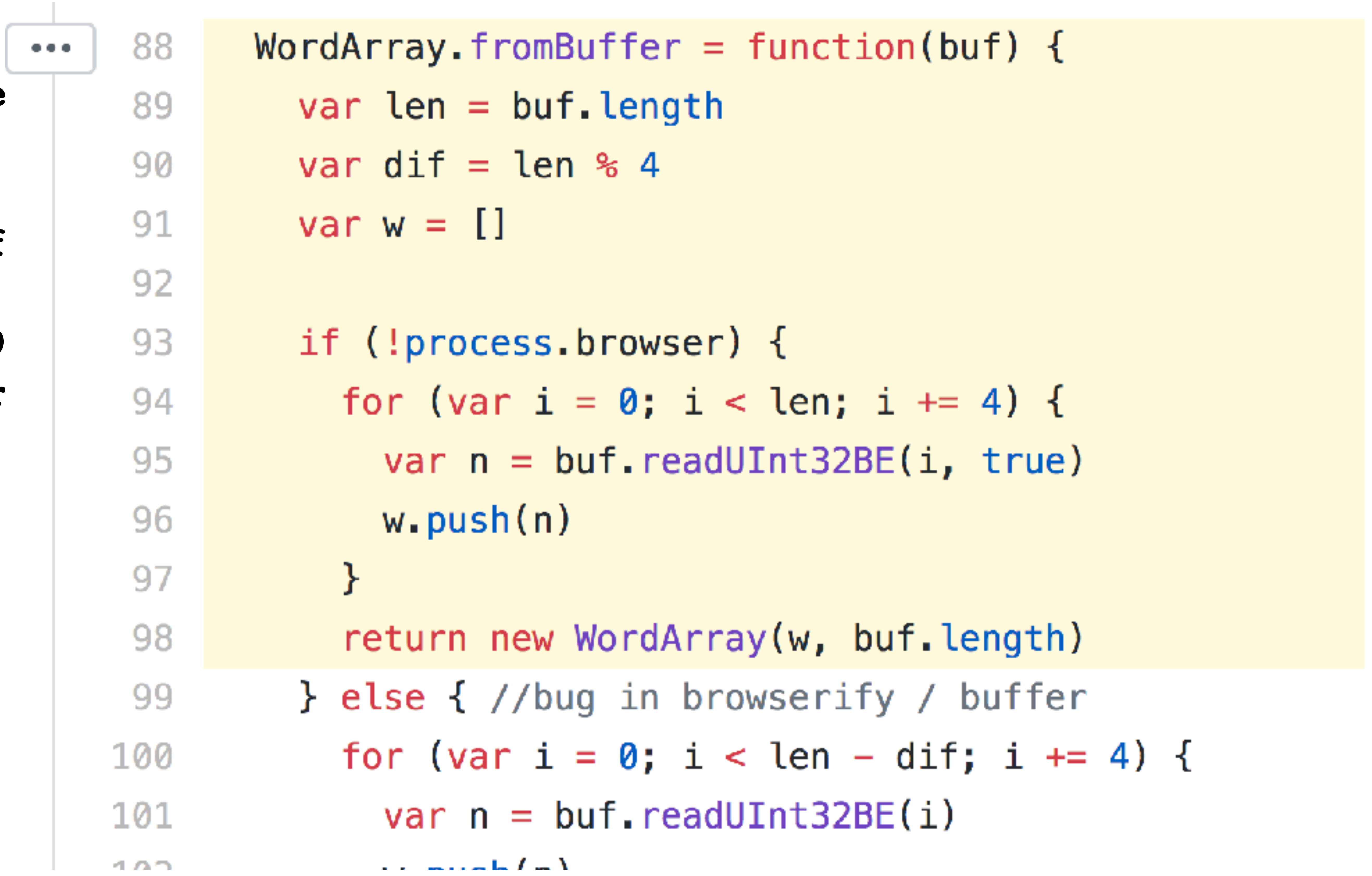
Root cause was a basic oob read

```
internal/buffer.js:55
  throw new ERR_OUT_OF_RANGE(type || 'offset',
  ^

RangeError [ERR_OUT_OF_RANGE]: The value of "offset" is out of range. It must be >= 0 and <= 161.
Received 164
    at boundsError (internal/buffer.js:55:9)
    at Buffer.readUInt32BE (internal/buffer.js:198:5)
```

What happened? (4/2)

Root cause was a basic oob read

```
internal/buffer.js:55
  throw new ERR_OUT_OF_RANGE(type || 'offse
  ^

RangeError [ERR_OUT_OF_RANGE]: The value of
Received 164
    at boundsError (internal/buffer.js:55:9
    at Buffer.readUInt32BE (internal/buffer
```

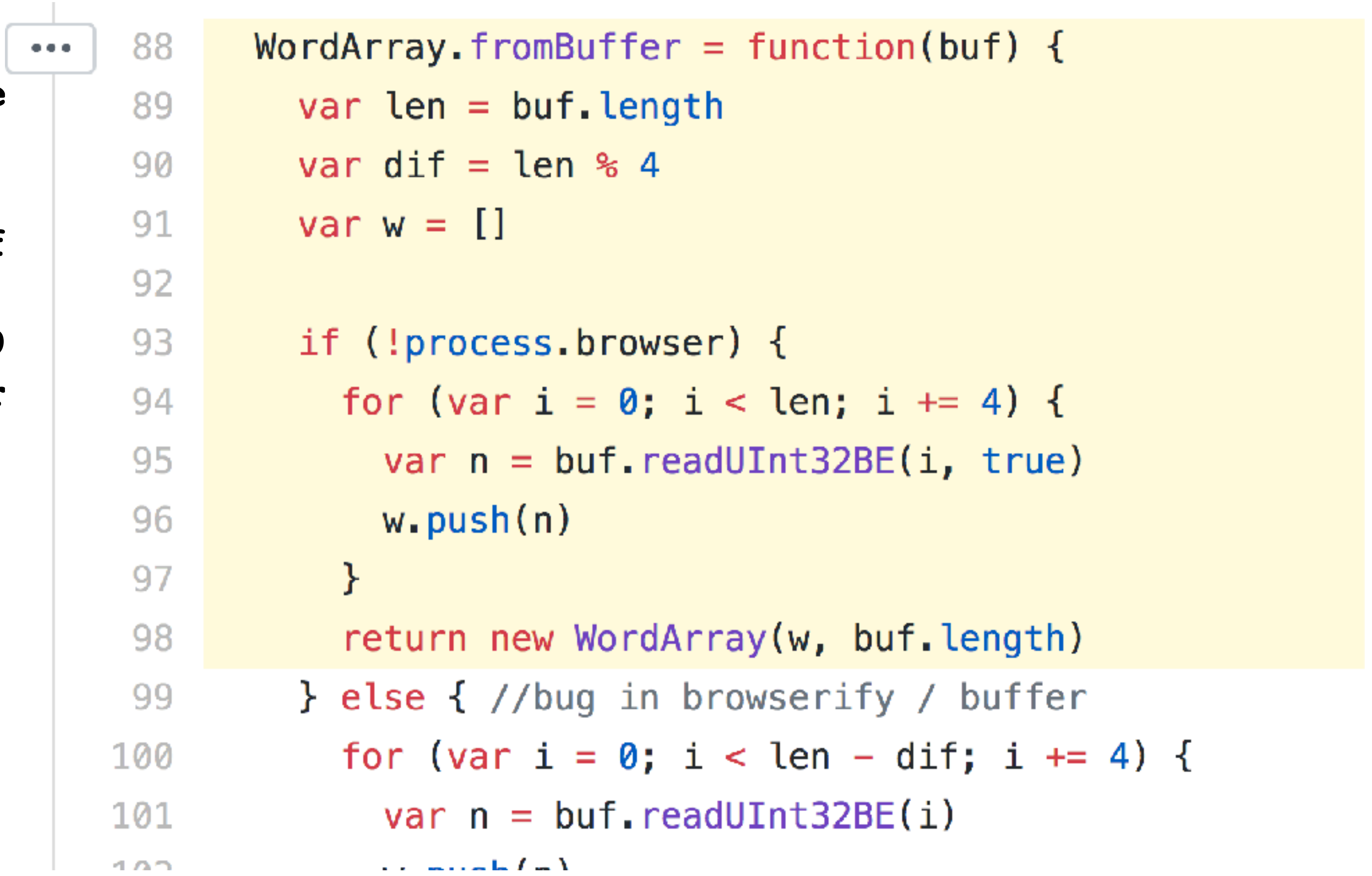


```
WordArray.fromBuffer = function(buf) {
  var len = buf.length
  var dif = len % 4
  var w = []

  if (!process.browser) {
    for (var i = 0; i < len; i += 4) {
      var n = buf.readUInt32BE(i, true)
      w.push(n)
    }
    return new WordArray(w, buf.length)
  } else { //bug in browserify / buffer
    for (var i = 0; i < len - dif; i += 4) {
      var n = buf.readUInt32BE(i)
```

What happened? (4/2)

Root cause was a basic oob read

```
internal/buffer.js:55
  throw new ERR_OUT_OF_RANGE(type || 'offse
  ^

RangeError [ERR_OUT_OF_RANGE]: The value of
Received 164
    at boundsError (internal/buffer.js:55:9
    at Buffer.readUInt32BE (internal/buffer
```



```
WordArray.fromBuffer = function(buf) {
  var len = buf.length
  var dif = len % 4
  var w = []

  if (!process.browser) {
    for (var i = 0; i < len; i += 4) {
      var n = buf.readUInt32BE(i, true)
      w.push(n)
    }
    return new WordArray(w, buf.length)
  } else { //bug in browserify / buffer
    for (var i = 0; i < len - dif; i += 4) {
      var n = buf.readUInt32BE(i)
```

Unexploitable :-/

Lessons and solutions

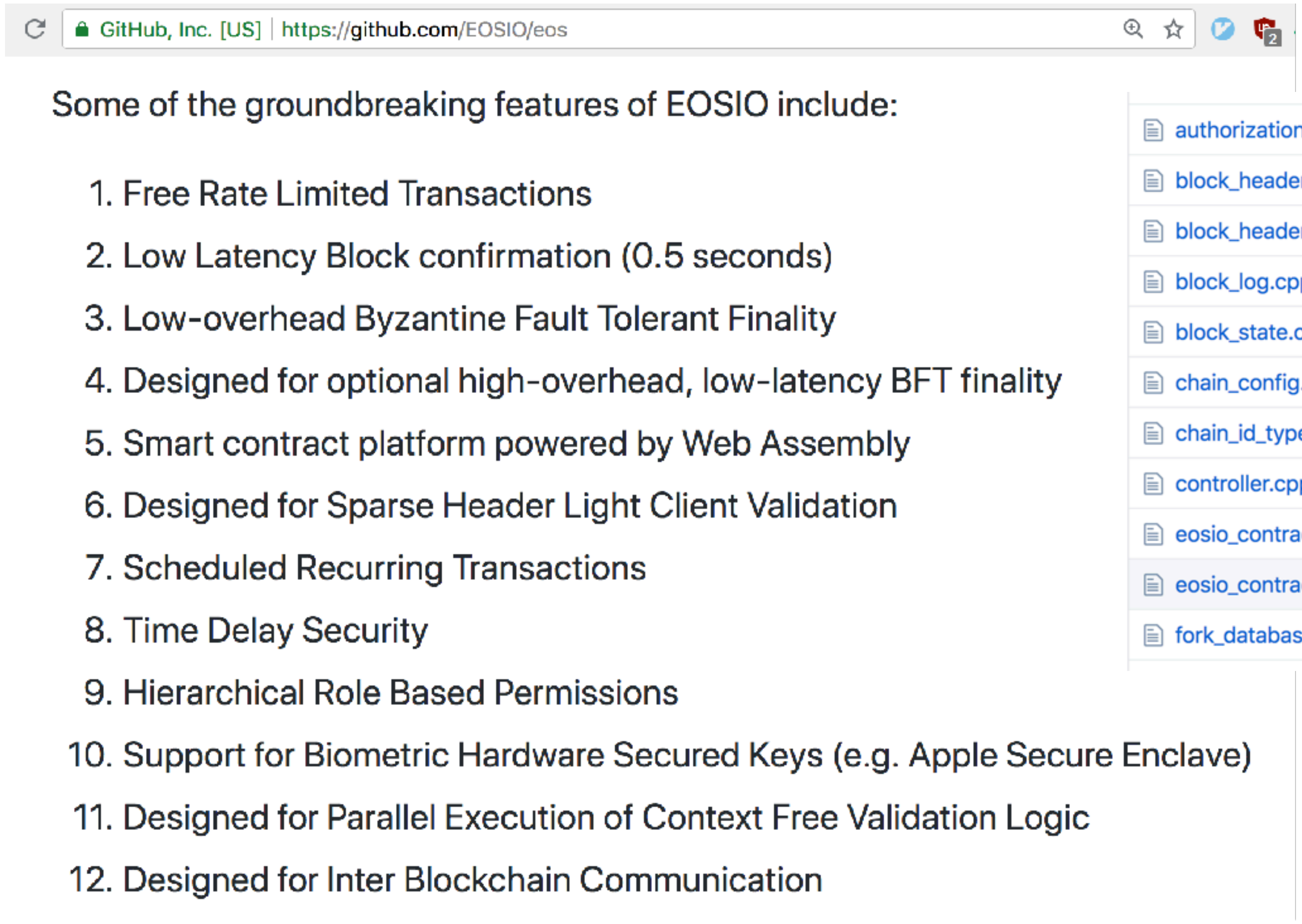
- Don’t use any open-source utility in production!
- Node/JS dependency trees can hide insecure||deprecated code, not always reported by tools like nsp or npm-dview
- A crypto API should default to the most secure behavior, be “misuse resistant" / fail-safe

Conclusions

- Many more bugs to be found
- Usual suspects: complex logic, unsafe language, rushed deployment, lack of testing, third-party dependencies

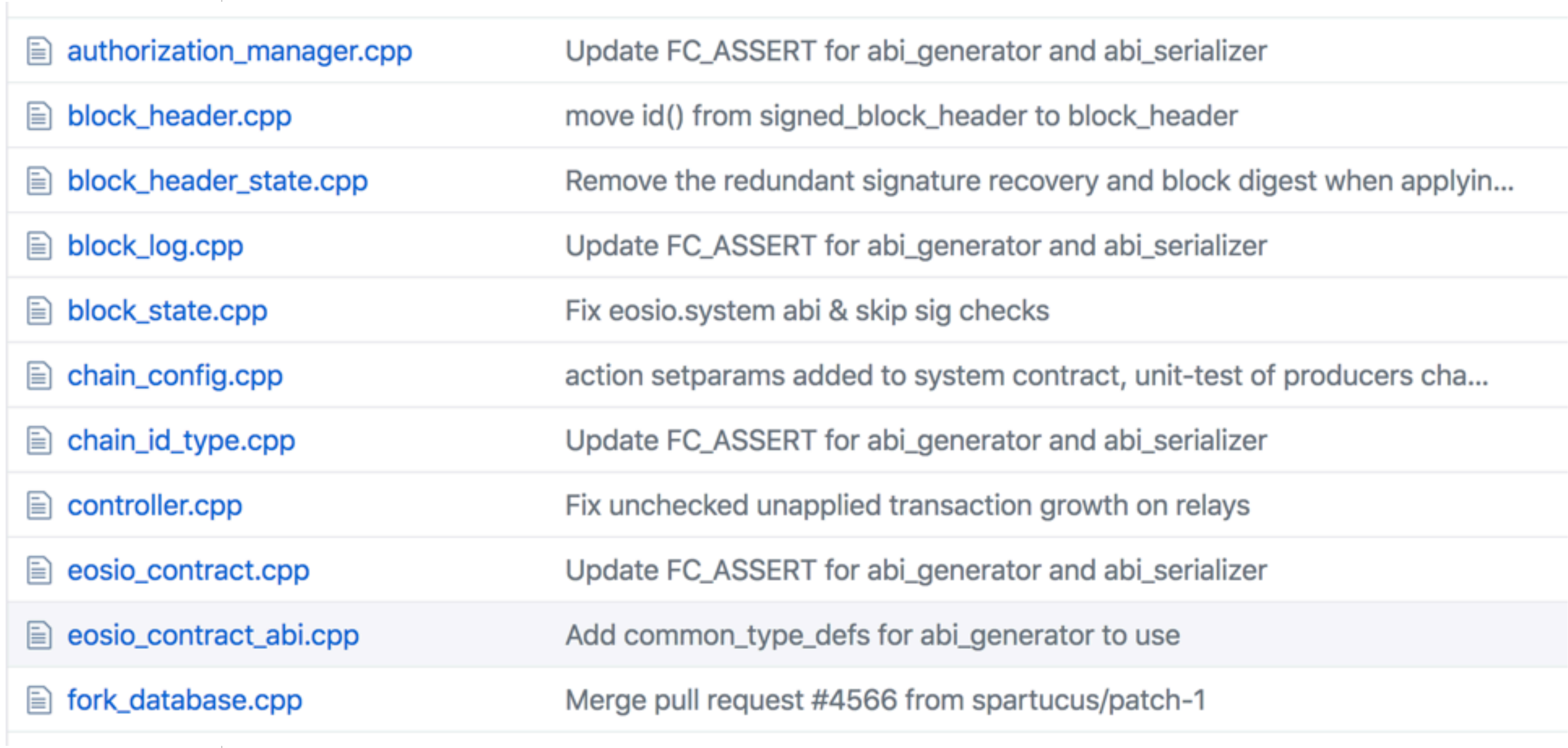
Conclusions

- Many more bugs to be found
- Usual suspects: complex logic, unsafe language, rushed deployment, lack of testing, third-party dependencies
- Recent example: EOS

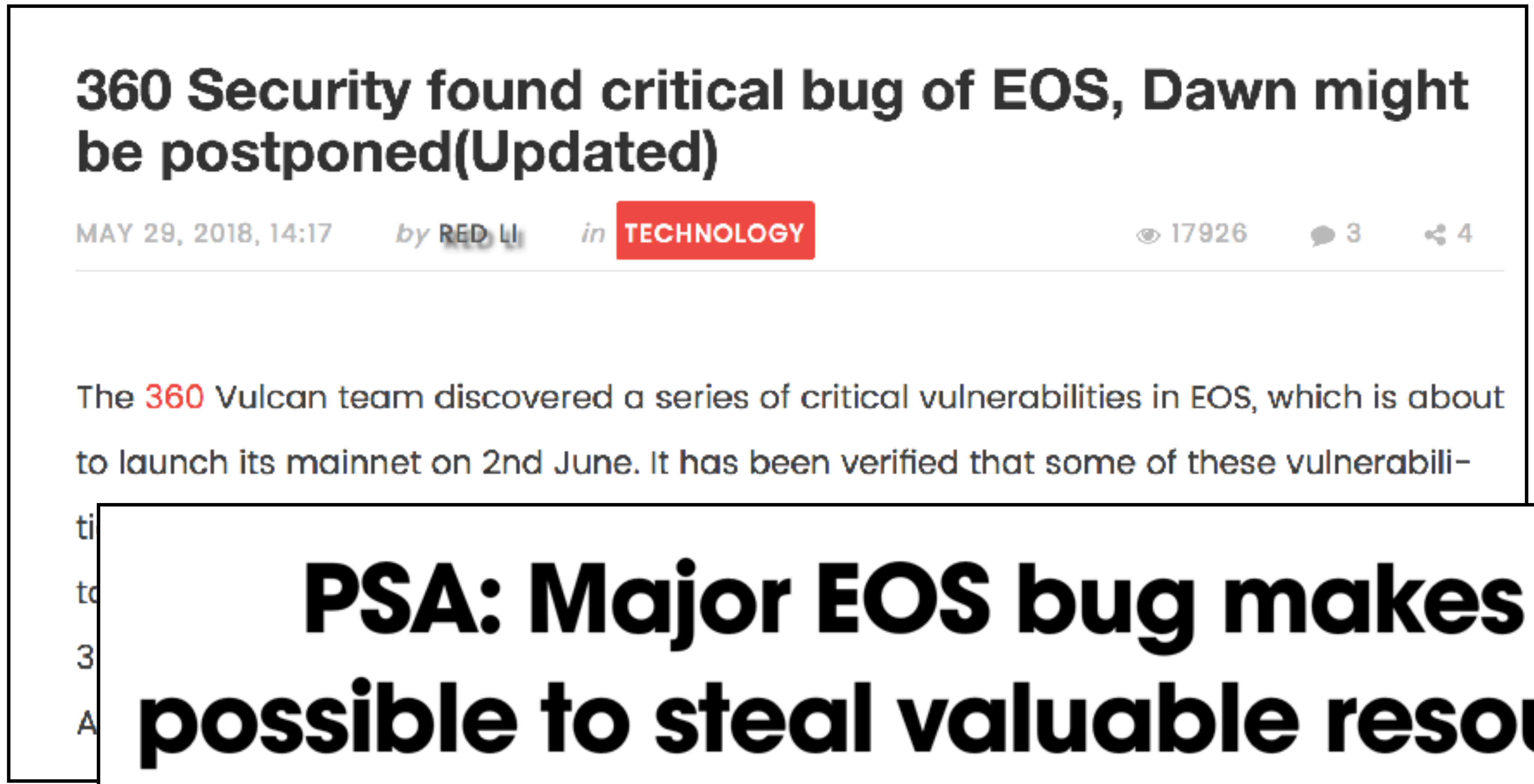

GitHub, Inc. [US] | https://github.com/EOSIO/eos

Some of the groundbreaking features of EOSIO include:

1. Free Rate Limited Transactions
2. Low Latency Block confirmation (0.5 seconds)
3. Low-overhead Byzantine Fault Tolerant Finality
4. Designed for optional high-overhead, low-latency BFT finality
5. Smart contract platform powered by Web Assembly
6. Designed for Sparse Header Light Client Validation
7. Scheduled Recurring Transactions
8. Time Delay Security
9. Hierarchical Role Based Permissions
10. Support for Biometric Hardware Secured Keys (e.g. Apple Secure Enclave)
11. Designed for Parallel Execution of Context Free Validation Logic
12. Designed for Inter Blockchain Communication



authorization_manager.cpp	Update FC_ASSERT for abi_generator and abi_serializer
block_header.cpp	move id() from signed_block_header to block_header
block_header_state.cpp	Remove the redundant signature recovery and block digest when applyin...
block_log.cpp	Update FC_ASSERT for abi_generator and abi_serializer
block_state.cpp	Fix eosio.system abi & skip sig checks
chain_config.cpp	action setparams added to system contract, unit-test of producers cha...
chain_id_type.cpp	Update FC_ASSERT for abi_generator and abi_serializer
controller.cpp	Fix unchecked unapplied transaction growth on relays
eosio_contract.cpp	Update FC_ASSERT for abi_generator and abi_serializer
eosio_contract_abi.cpp	Add common_type_defs for abi_generator to use
fork_database.cpp	Merge pull request #4566 from spartucus/patch-1



360 Security found critical bug of EOS, Dawn might be postponed(Updated)

MAY 29, 2018, 14:17 by RED LI in TECHNOLOGY 17926 3 4

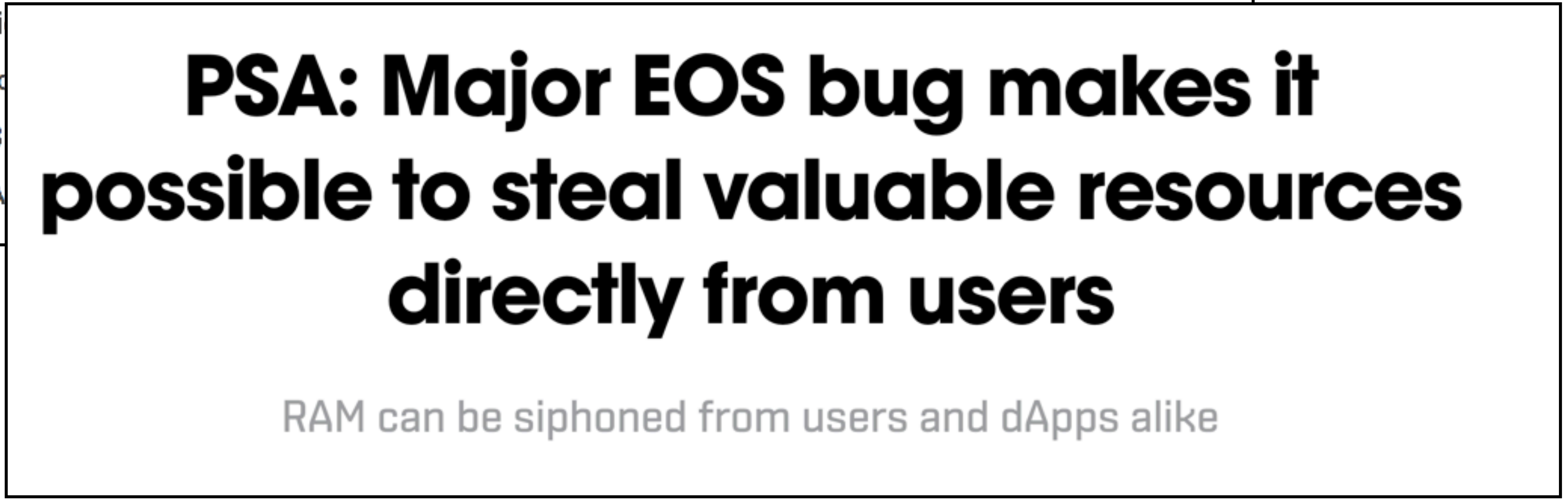
The 360 Vulcan team discovered a series of critical vulnerabilities in EOS, which is about to launch its mainnet on 2nd June. It has been verified that some of these vulnerabili-

ti

tc

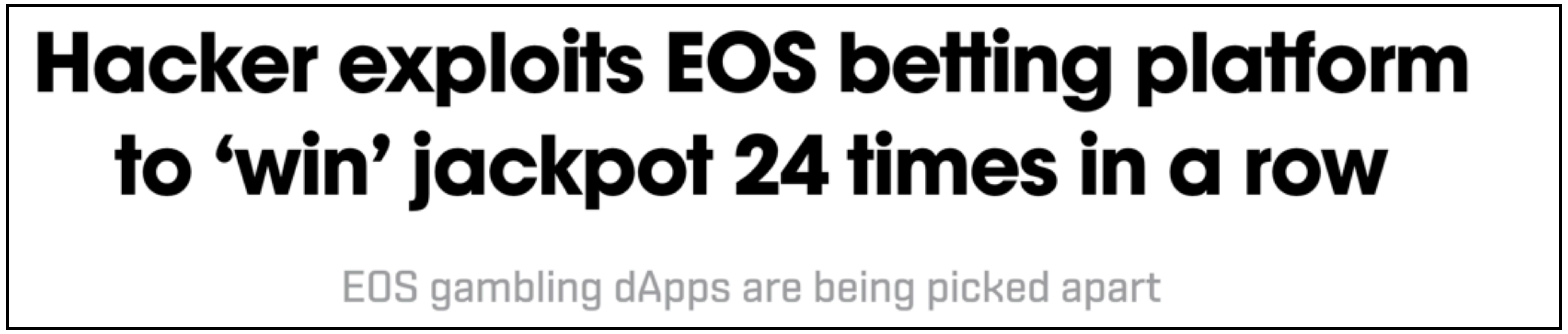
3

A



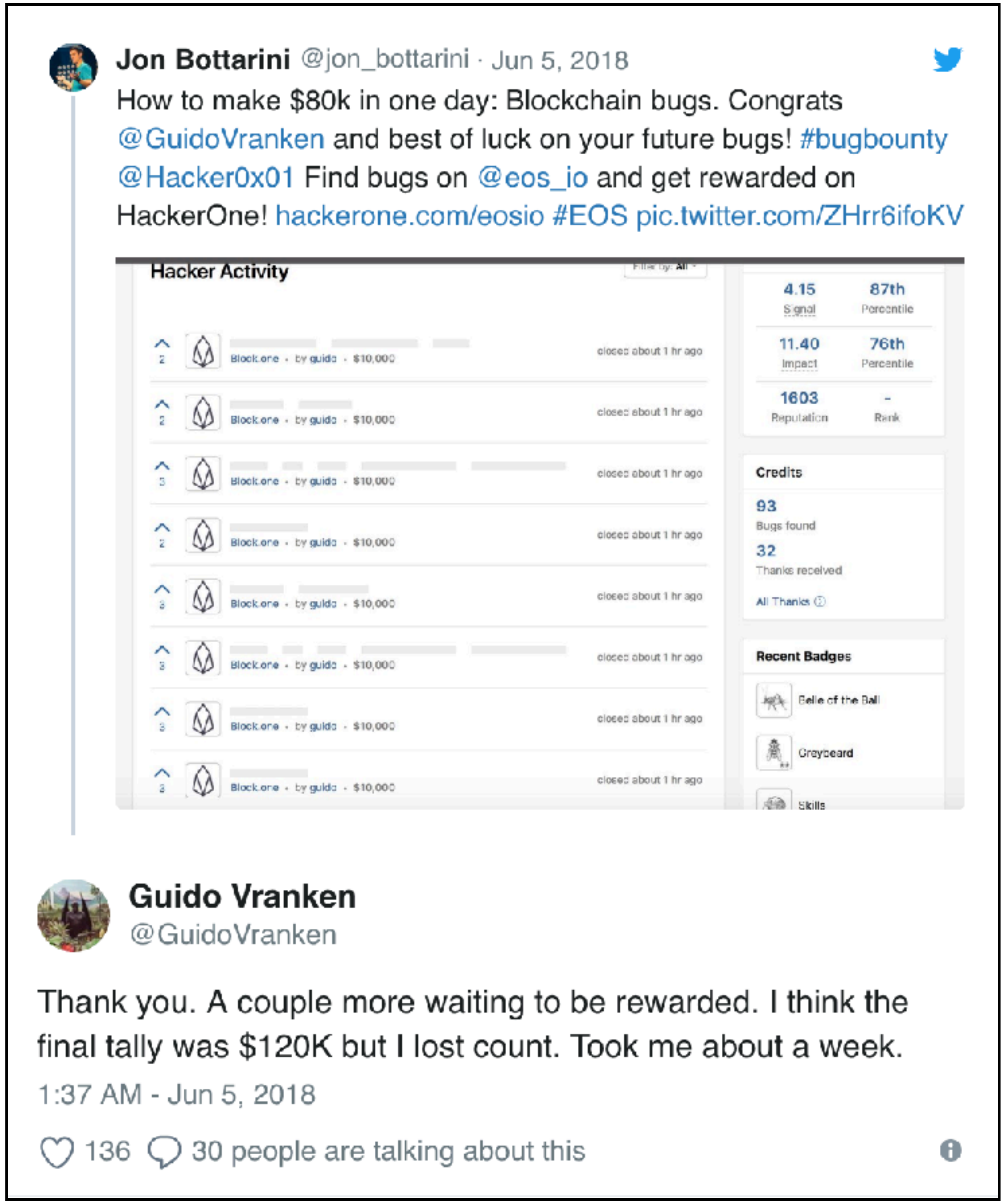
PSA: Major EOS bug makes it possible to steal valuable resources directly from users

RAM can be siphoned from users and dApps alike



Hacker exploits EOS betting platform to 'win' jackpot 24 times in a row

EOS gambling dApps are being picked apart



Jon Bottarini @jon_bottarini · Jun 5, 2018

How to make $80k in one day: Blockchain bugs. Congrats @GuidoVranken and best of luck on your future bugs! #bugbounty @Hacker0x01 Find bugs on @eos_io and get rewarded on HackerOne! hackerone.com/eosio #EOS pic.twitter.com/ZHrr6ifoKV

Hacker Activity

Filter by: All

2 Block.one · by guido · $10,000 closed about 1 hr ago

2 Block.one · by guido · $10,000 closed about 1 hr ago

3 Block.one · by guido · $10,000 closed about 1 hr ago

2 Block.one · by guido · $10,000 closed about 1 hr ago

3 Block.one · by guido · $10,000 closed about 1 hr ago

3 Block.one · by guido · $10,000 closed about 1 hr ago

3 Block.one · by guido · $10,000 closed about 1 hr ago

3 Block.one · by guido · $10,000 closed about 1 hr ago

4.15 Signal	87th Percentile
11.40 Impact	76th Percentile
1603 Reputation	- Rank

Credits

93 Bugs found

32 Thanks received

All Thanks

Recent Badges

Belle of the Ball

Greybeard

Skills

Guido Vranken
@GuidoVranken

Thank you. A couple more waiting to be rewarded. I think the final tally was $120K but I lost count. Took me about a week.

1:37 AM - Jun 5, 2018

136 30 people are talking about this

Conclusions

- Many more bugs to be found
- Usual suspects: complex logic, unsafe language, rushed deployment, lack of testing, third-party dependencies
- Recent example: EOS
- Security audits help, but won’t find all the bugs
- Nice way to make money for bug hunters :-)

Thank you!

@veorq https://aumasson.jp https://taurusgroup.ch