OPS

BLOCKCHAIN PENETRATION TESTING: HACKING THE VULNERABLE FUMBLECHAIN

BLOG · AJAY CHANDHOK

In most cases, a vulnerable blockchain is nothing to brag about. However, Kudelski Security's FumbleChain is just that, and they're proud of it. FumbleChain is a purposefully insecure blockchain that encourages users to learn about blockchain security.

It contains a series of challenges that each demonstrates a common blockchain vulnerability. Once you solve the problem, you receive FumbleCoin rewards. Don't get too excited, though; these coins have no monetary value. The real rewards are the skills you learn along the way.

Never a team to back down from challenges, we took our blockchain penetration testing knowledge to FumbleChain, successfully finding every one of its vulnerabilities. The following are our results.

WARNING: *The rest of this article contains solutions to FumbleChain's vulnerability challenges. If you're planning on performing blockchain penetration testing on FumbleChain yourself, you may want to do so before finishing this article.*

.

.

.

.

.

.

.

OPS

.

.

.

.

Vulnerability #1: No Input Sanitation Enables Replay Attacks

The first vulnerability we discovered in our blockchain penetration testing efforts was the potential for replay attacks due to a lack of input sanitation, particularly in the Blockchain.py file.

What should happen: When you initiate a blockchain transaction, the network checks the *magic number* of the operation against the *magic number* associated with the blockchain. If those numbers don't match up, the transaction is invalid.

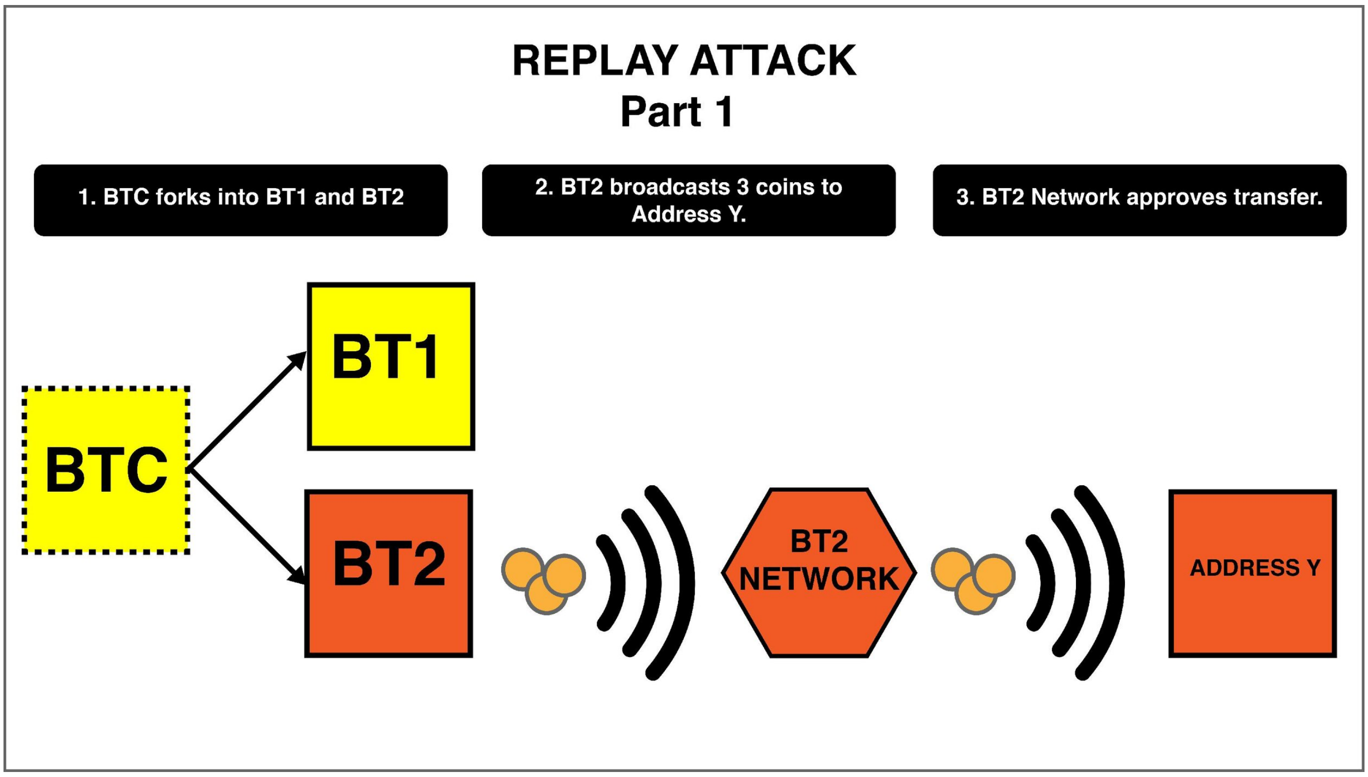
The FumbleChain vulnerability: FumbleChain isn't correctly performing the validation of the magic number. Therefore, you can create a transaction on any FumbleChain fork and effectively copy it to the main chain. If the wallet in the main chain contains enough funds, the operation will still execute.

We were able to create a transaction involving a mainnet wallet on the FumbleChain testnet. Then, we copied information from the JSON response and created a raw transaction on the mainnet with that information. Because the magic number isn't included when signing a transaction, we were able to alter the new transaction's magic number to match the mainnet and replay the transaction.

After mining six blocks, we received our stolen funds.

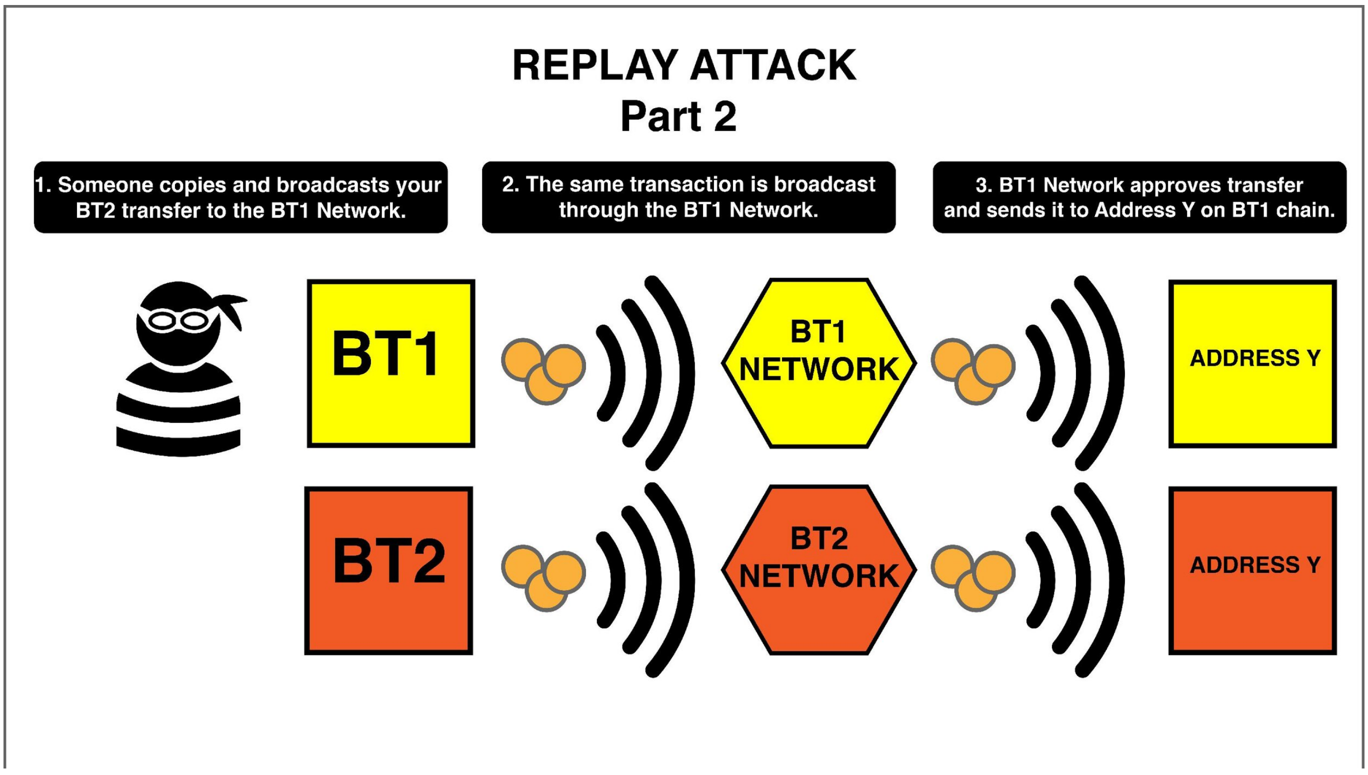
How to remediate: FumbleChain needs to include the magic number when creating transaction signatures. Doing so ensures that the signatures will be uniquely valid on a particular fork, and only that fork. Additionally, FumbleChain should add more logic in the Blockchain.py file to validate the magic number of transactions.

Any Forkable Blockchain Is Susceptible


REPLAY ATTACK
Part 1
1. BTC forks into BT1 and BT2
2. BT2 broadcasts 3 coins to Address Y.
3. BT2 Network approves transfer.
BTC
BT1
BT2
BT2 NETWORK
ADDRESS Y

A replay attack starts with a successful transaction on a blockchain fork.| Source: Exodus

Protection against replay attacks is especially vital in blockchain forks. When Bitcoin Cash forked away from the Bitcoin main chain, the development team needed to add additional information to transactional data as a form of replay protection.


REPLAY ATTACK
Part 2
1. Someone copies and broadcasts your BT2 transfer to the BT1 Network.
2. The same transaction is broadcast through the BT1 Network.
3. BT1 Network approves transfer and sends it to Address Y on BT1 chain.
BT1
BT1 NETWORK
ADDRESS Y
BT2
BT2 NETWORK
ADDRESS Y

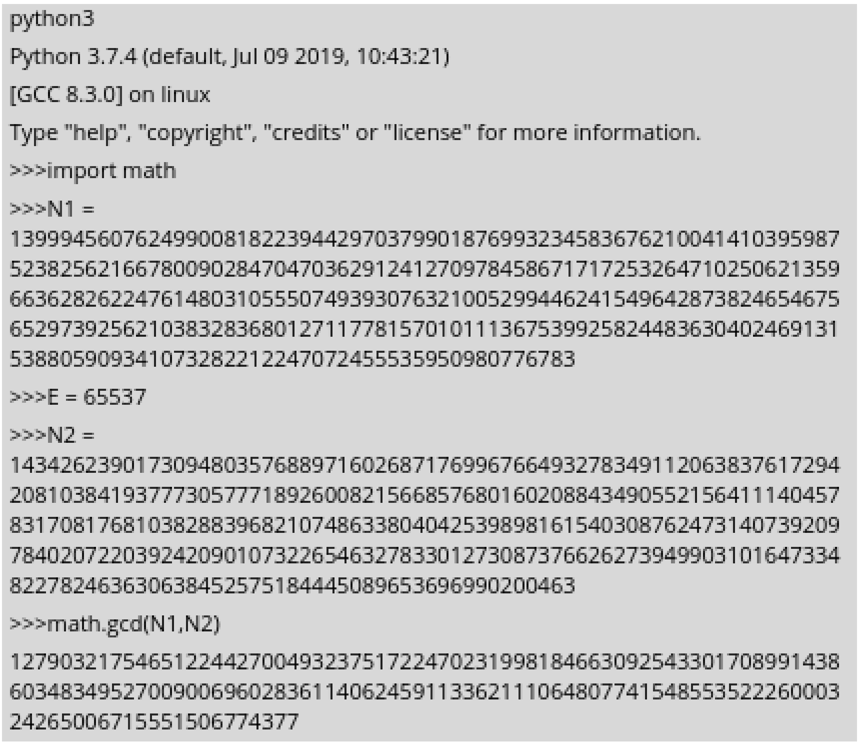
Vulnerability #2: Improper Key Pair Generation Exposes Private Keys

Next, during our blockchain penetration testing process, we found that some FumbleChain wallets are cryptographically weak due to a key generation flaw. This vulnerability enables a malicious party to generate some wallets' private keys from their public keys. With that information, you gain control of the wallet and its funds.

What should happen: There should be no possible way for someone to figure out a wallet's private key solely from public key information.

The FumbleChain vulnerability: Certain FumbleChain wallets share a private number with other vulnerable wallets. We were able to capitalize on this mistake and figure out the private keys of wallets with which we don't have ownership.

We first extracted the public numbers from two available wallets (N1, N2, and E) using the OpenSSL utility. Then, we found the greatest common divisor between N1 and N2 (p). Dividing N1 and N2 by p gave us the private number for each wallet (q).



```
python3
Python 3.7.4 (default, Jul 09 2019, 10:43:21)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>import math
>>>N1 =
139994560762499008182239442970379901876993234583676210041410395987
523825621667800902847047036291241270978458671717253264710250621359
663628262247614803105550749393076321005299446241549642873824654675
652973925621038328368012711778157010111367539925824483630402469131
538805909341073282212247072455535950980776783
>>>E = 65537
>>>N2 =
143426239017309480357688971602687176996766493278349112063837617294
208103841937773057771892600821566857680160208843490552156411140457
831708176810382883968210748633804042539898161540308762473140739209
784020722039242090107322654632783301273087376626273949903101647334
822782463630638452575184445089653696990200463
>>>math.gcd(N1,N2)
127903217546512244270049323751722470231998184663092543301708991438
603483495270090069602836114062459113362111064807741548553522260003
24265006715551506774377
```

Discovering N1, N2, E, and p is as simple as running a Python script.

How to remediate: The FumbleChain team should review its source code regarding wallet generation. Ideally, a public key modulus should never share a private number with another wallet. To prevent issues moving forward, FumbleChain could generate two wallets at a time, verifying that they don't share a private number (i.e., the gcd(N1,N2) == 1). Then, provide one of those wallets to the user and discard the other one.

Airtight Cryptography Is a Never-Ending Battle

Cryptography is complex, making it a never-ending battle to protect it from ever-evolving cybercriminals. When implementing cryptography into your blockchain security system, it's best to use tried and true methods and avoid rolling your own crypto (i.e., creating your own cryptography).

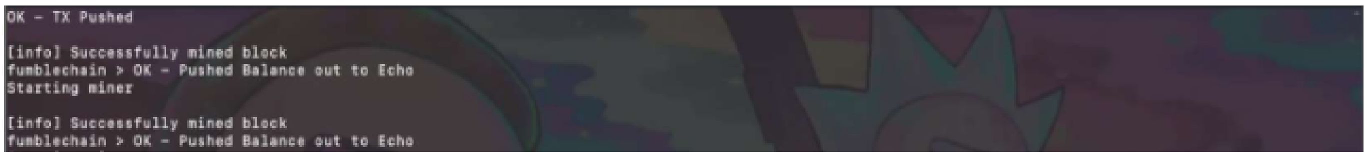
Additionally, you need to continually review your code to ensure that new mechanisms won't have a detrimental effect on legacy code. Robust testing (both manual and automated), as well as sanity checks, go a long way in ensuring security as your product evolves.

Vulnerability #3: Transaction Retransmission Allows Infinite Transactions

Finally, our blockchain penetration testing revealed that you could retransmit an already successful FumbleChain transaction. Because the signature of a completed transaction is valid, the network automatically accepts the repeat transaction and adjusts the wallet balances accordingly.

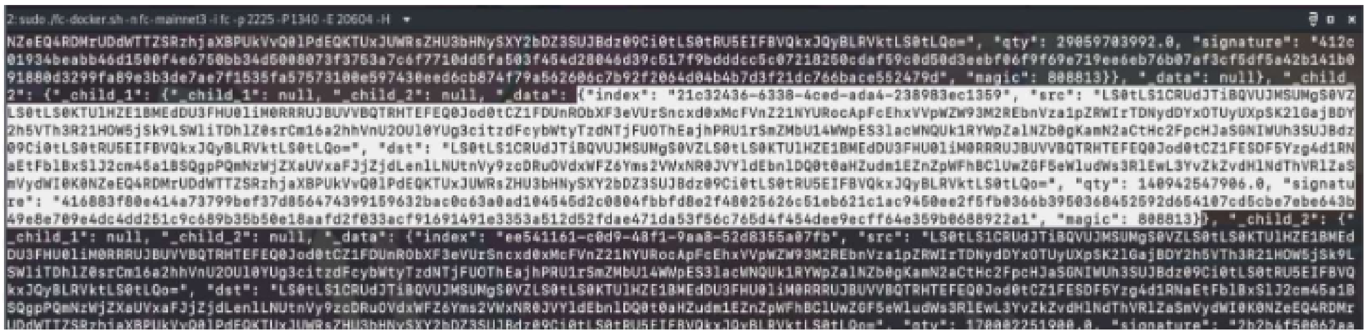
What should happen: You should not be able to resubmit a transaction that had already been executed.

The FumbleChain vulnerability: In our blockchain penetration testing, we edited FumbleChain's CLI.py script to create a custom function. The function targeted FumbleChain's Echo wallet, a wallet that receives tokens and responds by sending back an equivalent amount.


```
OK - TX Pushed

[info] Successfully mined block
fumblechain > OK - Pushed Balance out to Echo
Starting miner

[info] Successfully mined block
fumblechain > OK - Pushed Balance out to Echo
```

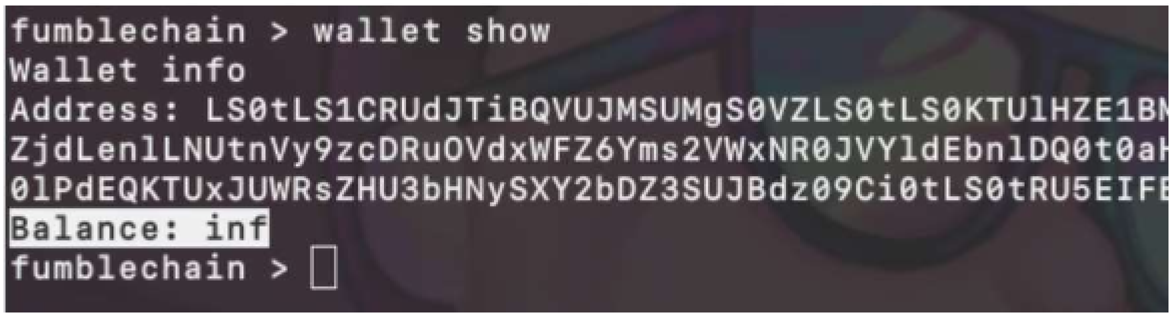


```
2: sudo ./fc-docker.sh -n fc-mainnet3 -i fc -p 2225 -P 1340 -E 20604 -H
```

```
NZeEQ4RDMrUDdWTTZSRzhjaXBPUkVvQ01PdEQKTUxJUWRsZHU3bHNySXY2bDZ3SUJBdz09Ci0tLS0tRU5EIFBVQkxJQyBLRVktLS0tLQo=", "qty": 29059703992.0, "signature": "412c
01934beabb46d1500f4e6750bb34d6008073f3753a7c6f7710dd5fa503f454d28046d39c517f9bdddcc5c07218250cdaf59c0d50d3eebf06f9f69e719ee6eb76b07af3cf5df5a42b141b0
91880d3299fa89e3b3de7ae7f1535fa57573100e597430eed6cb874f79a562606c7b92f2064d04b4b7d3f21dc766bace552479d", "magic": 888813}}, "_data": null}, "_child_
2": {"_child_1": {"_child_1": null, "_child_2": null, "_data": {"index": "21c32436-6338-4ced-ada4-238983ec1359", "src": "LS0tLS1CRUdJTiBQVUJMSUMgS0VZ
LS0tLS0KTUlHZE1BMEdDU3FHU0liM0RRRUJBUVVBQTRHTEFEQ0Jod0tCZ1FDUnRObXF3eVUrSncxd0xMcFVnZ21NYURocApFcEhxVVpWZW93M2REbnVza1pZRWIrTDNydDYxOTUyUXpSK21GajBDY
2h5VTh3R21HOW5jSk9LSWliTDhlZ0srCm16a2hhVnU2OUl0YUg3citzdFcybWtyTzdNTjFUOThEajhPRU1rSmZMbU14WWpES3lacWNQUk1RYWpZalNZb0gKamN2aCtHc2FpcHJaSGNIWUh3SUJBdz
09Ci0tLS0tRU5EIFBVQkxJQyBLRVktLS0tLQo=", "dst": "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlHZE1BMEdDU3FHU0liM0RRRUJBUVVBQTRHTEFEQ0Jod0tCZ1FESDF5Yzg4d1RN
aEtFblBxS1J2cm45a1BSQgpPQmNzWjZXaUVxaFJjZjdLenlLNUtnVy9zcDRuOVdxWFZ6Yms2VWxNR0JVYldEbnlDQ0t0aHZudm1EZnZpWFhBC1UwZGF5eWludWs3RlEwL3YvZkZvdHlNdThVRlZaS
mVydWI0K0NZeEQ4RDMrUDdWTTZSRzhjaXBPUkVvQ01PdEQKTUxJUWRsZHU3bHNySXY2bDZ3SUJBdz09Ci0tLS0tRU5EIFBVQkxJQyBLRVktLS0tLQo=", "qty": 140942547906.0, "signatu
re": "416883f80e414a73799bef37d856474399159632bac0c63a0ad104545d2c0804fbbfd8e2f48025626c51eb621c1ac9450ee2f5fb0366b3950368452592d654107cd5cbe7ebe643b
49e8e709e4dc4dd251c9c689b35b50e18aafd2f033acf91691491e3353a512d52fdae471da53f56c765d4f454dee9ecff64e359b0688922a1", "magic": 888813}}, "_child_2": {"
_child_1": null, "_child_2": null, "_data": {"index": "ee541161-c0d9-48f1-9aa8-52d8355a07fb", "src": "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlHZE1BMEd
DU3FHU0liM0RRRUJBUVVBQTRHTEFEQ0Jod0tCZ1FDUnRObXF3eVUrSncxd0xMcFVnZ21NYURocApFcEhxVVpWZW93M2REbnVza1pZRWIrTDNydDYxOTUyUXpSK21GajBDY2h5VTh3R21HOW5jSk9L
SWliTDhlZ0srCm16a2hhVnU2OUl0YUg3citzdFcybWtyTzdNTjFUOThEajhPRU1rSmZMbU14WWpES3lacWNQUk1RYWpZalNZb0gKamN2aCtHc2FpcHJaSGNIWUh3SUJBdz09Ci0tLS0tRU5EIFBVQ
kxJQyBLRVktLS0tLQo=", "dst": "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlHZE1BMEdDU3FHU0liM0RRRUJBUVVBQTRHTEFEQ0Jod0tCZ1FESDF5Yzg4d1RNaEtFblBxS1J2cm45a1B
SQgpPQmNzWjZXaUVxaFJjZjdLenlLNUtnVy9zcDRuOVdxWFZ6Yms2VWxNR0JVYldEbnlDQ0t0aHZudm1EZnZpWFhBC1UwZGF5eWludWs3RlEwL3YvZkZvdHlNdThVRlZaSmVydWI0K0NZeEQ4RDMr
UDdWTTZSRzhjaXBPUkVvQ01PdEQKTUxJUWRsZHU3bHNySXY2bDZ3SUJBdz09Ci0tLS0tRU5EIFBVQkxJQyBLRVktLS0tLQo=", "qty": 170002251900.0, "signature": "2b2b4f50062aa
```

Our script continuously mined blocks and re-transmitted transactions to the Echo wallet, growing our wallet balance.

Our malicious function first sent a transaction to the Echo wallet. It then replayed the Echo wallet's reply transaction (sending tokens back to our wallet) in the next block. Because FumbleChain only checks for transaction uniqueness within a block, we were able to continuously execute the same operation with each new block until our wallet balance reached an infinite amount.



```
fumblechain > wallet show
Wallet info
Address: LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlHZE1BN
ZjdLenlLNUtnVy9zcDRuOVdxWFZ6Yms2VWxNR0JVYldEbnlDQ0t0aH
01PdEQKTUxJUWRsZHU3bHNySXY2bDZ3SUJBdz09Ci0tLS0tRU5EIFE
Balance: inf
fumblechain > 
```

Eventually, our FumbleChain wallet reached an infinite amount of funds.

How to remediate: FumbleChain needs to add logic to its code that checks the uniqueness of transaction IDs against the entire blockchain, not just the block of the transaction.

Wallet Functions Are Just as Vulnerable as the Blockchain

Even if your blockchain is secure from a network, mining, and node perspective, the mechanisms with which those components interact still provide attack vectors.

Other than the preventative measures we've mentioned above, you should also implement stopgaps to limit the damage from an unforeseen attack. For example, FumbleChain could add an upper limit to the amount a wallet can hold.

OPS

Our Blockchain Penetration Testing Discovers Even More

In our blockchain penetration testing process, we found two additional vulnerabilities that the FumbleChain team hadn't created purposely.

Vulnerability #4

Considering the looming threat of nation-state level actors with ample resources, we determined that the RSA 1024bit Asymmetric Key Encryption that FumbleChain currently uses is insufficient. Continuing to use that encryption method for private/public key generation is likely to expose wallets over time.

Instead, we suggest that FumbleChain switch to an RSA 2048bit encryption scheme, at minimum, migrating previous wallets to the new scheme as well.

Vulnerability #5

Equally as important, we discovered that FumbleChain contains at least one denial of service (DOS) vulnerability. A malicious actor could overwhelm the system by mining blocks at such a high speed that it would overwhelm the other nodes. Testing this vulnerability was outside our scope, so we're unsure of the impact that this type of attack could have.

A simple solution to this DOS vulnerability is to implement some single-origin block and transaction rate-limiting, although we would need to perform additional testing to see how appropriate this solution would be.

Think Like a Criminal

We were able to discover every FumbleChain vulnerability (plus two extras) because the blockchain penetration testing process forces you to analyze the network as if you're the person attacking it. Just as with traditional penetration testing, it places you in the shoes of potential cybercriminals, providing you with the most effective way to keep your organization protected. Think like your attackers, and start protecting your blockchain today.

OPS

Sign up for our weekly newsletter to receive up-to-date cybersecurity news, tips, and additional resources.

Email Address SIGN UP

We respect your privacy.

PREVIOUS

LAST WEEK IN CYBERSECURITY NEWS - OCTOBER 22, 2019

NEXT

WHAT ARE THE DIFFERENT TYPES OF PENETRATION TESTING?

Solutions

Blockchain Security
Security Consulting

Actions

Get a Free Cybersecurity Consultation
Become a Partner
Get the Latest News

Partners

Follow Us

OPS