BITCOIN CVE-2018-17144

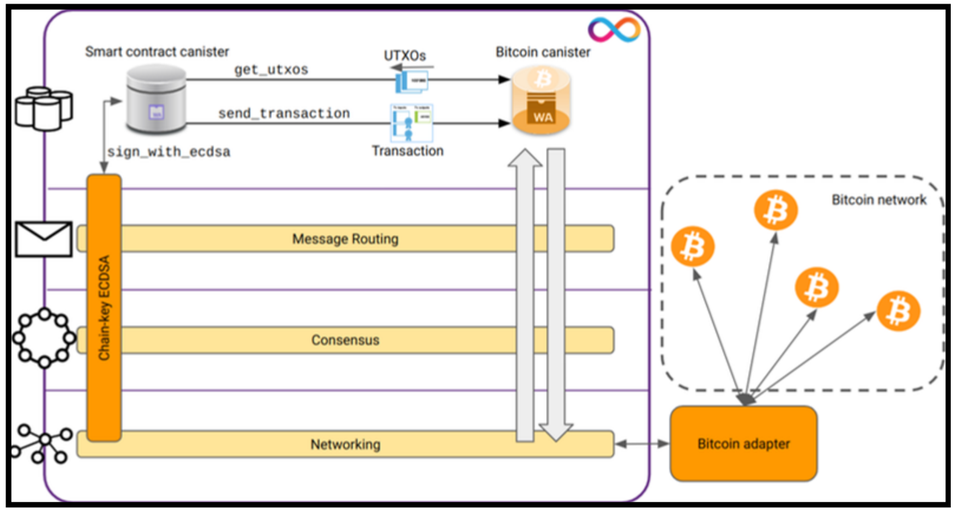
PRESENTED BY MEGHA HEGDE, JOSIANE UWUMUKIZA, ABBY CHOU

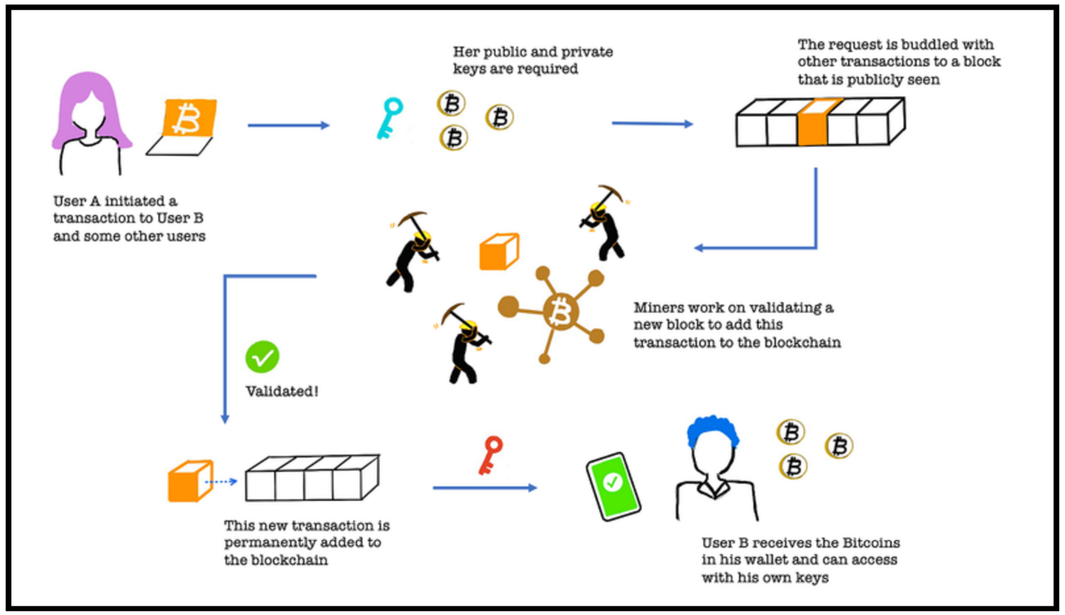
```
bool CTransaction::ConnectInputs(CTxDB& txdb, map<uint256, CTx
ol, CDiskTxPos posThisTx, int nHeight, int64& nFees, bool fBlock, bool fMiner,
int64 nMinFee)
{
    // Take over previous transactions' spent pointers
    if (!IsCoinBase())
    {
        int64 nValueIn = 0;
        for (int i = 0; i < vin.size(); i++)
        {
            ...
            // Check for conflicts
            if (!txindex.vSpent[prevout.n].IsNull())
                return fMiner ? false : error("ConnectInputs() : %s prev tx alr
eady used at %s", GetHash().ToString().substr(0,6).c_str(), txindex.vSpent[prev
out.n].ToString().c_str());

            // Mark outpoints as spent
            txindex.vSpent[prevout.n] = posThisTx;
            ...
        }
        ...
    }
    ...
}
```

The version	0.14.x	0.15.x-0.16.2
The attack space	• Transaction spends **same output twice** within one block • **Node crashes** with assertion failure **(DoS)** • Takes down the node but doesn't break consensus	• **Transaction spends output twice** (output from previous block) • **Node accepts** it as valid • Miner can spend same Bitcoin twice = **INFLATION** • Silent exploitation possible

What is CVE-2018-17144

- **The Threat:** Critical vulnerability allowing unlimited Bitcoin creation
- **Duration:** Existed undetected for 2+ years (2016-2018)
- **Potential Impact:** Could have destroyed Bitcoin's 21M supply guarantee
- **The Savings:** Optimized away 600 microseconds of validation time
- **The Cost:** Nearly collapsed a $100+ billion network
- **Outcome:** Patched within 48 hours, never exploited on mainnet





Bitcoin: an overview

1. **UTXO Model:** Bitcoin tracks "unspent transaction outputs" - like digital checks that haven't been cashed
2. **Validation Pipeline:**
   - *checkTransaction*: Validates individual transactions
   - *CheckBlock*: Validates blocks containing transactions
   - *UpdateCoins:* Updates the UTXO database
3. **Consensus Rules:** Every node must agree on what's valid. ***Why This Matters:*** Consensus bugs can permanently split the network

Timeline of events

The attack evolution

Version	Case 1A (Multi tx, mempool)	Case 1B (Multi tx, block)	Case 2A (Single tx, mempool)	Case 2B (Single tx, block)
Pre-2013	✅ Caught	✅ Caught	✅ Caught	✅ Caught (twice!)
2013-2016 (PR #2224)	✅ Caught	✅ Caught	✅ Caught	✅ Caught (twice!)
0.14.x (PR #9049)	✅ Caught	✅ Caught	✅ Caught	💥 **DoS** (crash)
0.15.x-0.16.2 (+ PR #10537)	✅ Caught	✅ Caught	✅ Caught	💥 **DoS** (FRESH) or 💰 **Inflation** (DIRTY)
0.16.3+ (Fixed)	✅ Caught	✅ Caught	✅ Caught	✅ Caught

***The attack space**

1A: *Multiple mempool transactions*
1B: *Multiple block transactions*
2A: *Single mempool transaction*
2B: *Single block transaction*

→ **All attacks spend the same UTXO multiple times**

The Phase	Time/PR	Event
The Setup Phase	PR 443 (2011)	Added duplicate input check
	PR 2224(2013)	Improve error handling during validation
The vulnerable phase	PR 9049 (2016)	Remove duplicatable duplicate-input check
	PR 10195 + 10537 (2017)	UTXO redesign + assertion changes
The Crisis phase	September 17, 2018	Matt Corallo identifies inflation bug
	September 18, 2018	Patch available - 33.5 hours after initial report
The longtail phase	May 2019	60,000+ vulnerable nodes still exist

PR 443

Added a check for duplicate inputs within the **same** transaction to CheckTransaction, a suite of sanity checks to reject any obviously invalid transactions from inclusion to the mempool.

- CheckTransaction: Context-independent checks, not consensus checks

```
317 +    // Check for duplicate inputs
318 +    set<COutPoint> vInOutPoints;
319 +    BOOST_FOREACH(const CTxIn& txin, vin)
320 +    {
321 +        if (vInOutPoints.count(txin.prevout))
322 +            return false;
323 +        vInOutPoints.insert(txin.prevout);
324 +    }
325 +
```

PR 2224

Made sweeping changes to error handling system.
Made the following small change in the UpdateCoins method:

```
 {
     // mark inputs spent
     if (!IsCoinBase()) {
         BOOST_FOREACH(const CTxIn &txin, vin) {
             CCoins &coins = inputs.GetCoins(txin.prevout.hash);
             CTxInUndo undo;
-            if (!coins.Spend(txin.prevout, undo))
-                return error("UpdateCoins() : cannot spend input");
+            assert(coins.Spend(txin.prevout, undo));
             txundo.vprevout.push_back(undo);
         }
     }
```

If the assert failed, the node would crash. Assuming the txn passed all the other consensus checks, failing the assert would imply some sort of memory corruption.
Unintentionally made the check from 443 no longer redundant but critical.

PR 9049

```
1107        - bool CheckTransaction(const CTransaction& tx, CValidationState &state)
      1107  + bool CheckTransaction(const CTransaction& tx, CValidationState &state, bool fCheckDuplicateInputs)
```

PR 9049

```
1131  -     // Check for duplicate inputs
1132  -     set<COutPoint> vInOutPoints;
1133  -     for (const auto& txin : tx.vin)
1134  -     {
1135  -         if (vInOutPoints.count(txin.prevout))
1136  -             return state.DoS(100, false, REJECT_INVALID, "bad-txns-inputs-duplicate");
1137  -         vInOutPoints.insert(txin.prevout);
      1131  +     // Check for duplicate inputs - note that this check is slow so we skip it in CheckBlock
```

gmaxwell on Nov 2, 2016 — Contributor

point out that it's also redundant there please!

dexX7 on Sep 19, 2018 — Contributor

How is it redundant? Can you point to the case where it's checked a second time?

👍 7 😄 2 😶 1

```
      1132  +     if (fCheckDuplicateInputs) {
      1133  +         set<COutPoint> vInOutPoints;
      1134  +         for (const auto& txin : tx.vin)
      1135  +         {
      1136  +             if (!vInOutPoints.insert(txin.prevout).second)
      1137  +                 return state.DoS(100, false, REJECT_INVALID, "bad-txns-inputs-duplicate");
      1138  +         }
```

PR 9049

17:37 < sipa> do we even need that check?
17:37 < BlueMatt> which? the transaction-length one? no, it is 100% redundant
17:38 < BlueMatt> the duplicate-inputs one? unclear, probably not but we added it for a reason
17:38 < BlueMatt> dont recall what it was
17:38 < BlueMatt> i do remember that we had a reason, however
17:38 < gmaxwell> was it related to bip30?
17:39 -!- AaronvanW [~ewout@unaffiliated/aaronvanw] has quit [Read error: Connection reset by peer]
17:40 < BlueMatt> hum...i dont see why? :/
17:40 < BlueMatt> lol, all these half-empty blocks are fucking with my benchmarking :/
17:42 < sipa> https://github.com/bitcoin/bitcoin/pull/443
17:42 < sipa> you added this.
17:42 < BlueMatt> yes, and i recall having a reason :(

PR 9049

```
17:37 < sipa> do we even need that check?
17:37 < BlueMatt> which? the transaction-length one? no, it is 100% redundant
17:38 < BlueMatt> the duplicate-inputs one? unclear, probably not but we added it for a reason
17:38 < BlueMatt> dont recall what it was
17:38 < BlueMatt> i do remember that we had a reason, however
17:38 < gmaxwell> was it related to bip30?
17:39 -!- AaronvanW [~ewout@unaffiliated/aaronvanw] has quit [Read error: Connection reset by peer]
17:40 < BlueMatt> hum...i dont see why? :/
17:40 < BlueMatt> lol, all these half-empty blocks are fucking with my benchmarking :/
17:42 < sipa> https://github.com/bitcoin/bitcoin/pull/443
17:42 < sipa> you added this.
17:42 < BlueMatt> yes, and i recall having a reason :(
```

Result: Can crash nodes by submitting transactions that doublespend the same input.

UTXO Refactoring

And then v0.15 broke it even more...

- UpdateCoins relied on invariant that **if a UTXO exists, it will be in memory, and marked as either spent or unspent.**

UTXO Refactoring

And then v0.15 broke it even more...

- UpdateCoins relied on invariant that **if a UTXO exists, it will be in memory, and marked as either spent or unspent.**
- 0.15 broke this by allowing UTXOs to be dropped from memory if they are fresh (recently loaded into cache) and then immediately spent.
    - As if the UTXO never existed...

UTXO Refactoring

And then v0.15 broke it even more...

- UpdateCoins relied on invariant that **if a UTXO exists, it will be in memory, and marked as either spent or unspent.**
- 0.15 broke this by allowing UTXOs to be dropped from memory if they are fresh (recently loaded into cache) and then immediately spent.
  - As if the UTXO never existed...
- Malicious block: spending the same input twice in the same transaction
  - Miners would never create a block like this (CheckTransaction doesn't allow such txns into the mempool), but validators don't catch it due to PR 9049.

The response

Discovery

September 17, 2018 - 14:57 UTC
Anonymous researcher reports crash bug to Bitcoin Core developers

Initial Assessment:
Denial of Service vulnerability - nodes crash when processing certain blocks

Identification

September 17, 2018 - 17:47 UTC(2 hours 50 minutes later)
Matt Corallo realizes the true severity

The critical insight:
Different versions behave differently (FRESH vs DIRTY coins)

Coordination

September 17, 2018 - 19:15 to 23:21 UTC
Immediate Actions:
Emergency outreach to major mining pools begins

Disclosure Strategy:
Public: Announce serious DoS bug requiring immediate patch
Private: Alert miners/exchanges about critical inflation risk

Patch

September 17-18, 2018
Timeline:
21:57 UTC (Sept 17): PR #14247 published with patch
20:44 UTC: Release binaries available

The Fix: Restored duplicate input check in CheckTransaction - always runs during block validation

Economic Impact

Denial of Service vulnerability

- Allowed a miner to construct a block with the same UTXO
- If this block was validated, affected nodes would crash
- An attack would broadcast a block with a double spent UTXO

- However, since it required PoW, they would have to sacrifice the block reward which was 12.5 BTC at the time for temporary network disruption.

- Therefore there was no strong economic incentive to exploit it.

- The only potential benefit arises from network disruption, not financial gain. Example: political gains.

Inflation Vulnerability

- Introduced the possibility of violating Bitcoin's fixed monetary supply.
- Allowed a transaction to spend the same UTXO without crashing
- Any acceptance of such a block would create valid-looking, yet economically illegitimate bitcoin.

- The inflated block would cause a visible anomaly on public ledgers and other miners would mine on top of other blocks or social consensus would rollback or orphaning the inflated block.
- Therefore, the attacker pays for the block's proof-of-work and risks losing the entire reward with no guarantee of retaining illicit coins.

- If not caught early could have caused market crash for Bitcoin since it's value is based on the fixed supply.

Key Takeaways

Lessons Learned

- Never optimize consensus-critical code **without extensive testing**
- Mandatory **test coverage** for all consensus changes
- Multiple **layers of review** for consensus changes

Recommendations

- **Technical:** Mandatory test coverage - no exceptions
- **Process:** Require historical analysis in code reviews
- **Development:** Correctness first, performance second

Why It Still Matters?

- Most discussed security incident in Bitcoin history
- Demonstrates unique risks in decentralized systems
- Cautionary tale for all cryptocurrency development

Thank you very much!