30th Annual Network and Distributed System Security Symposium

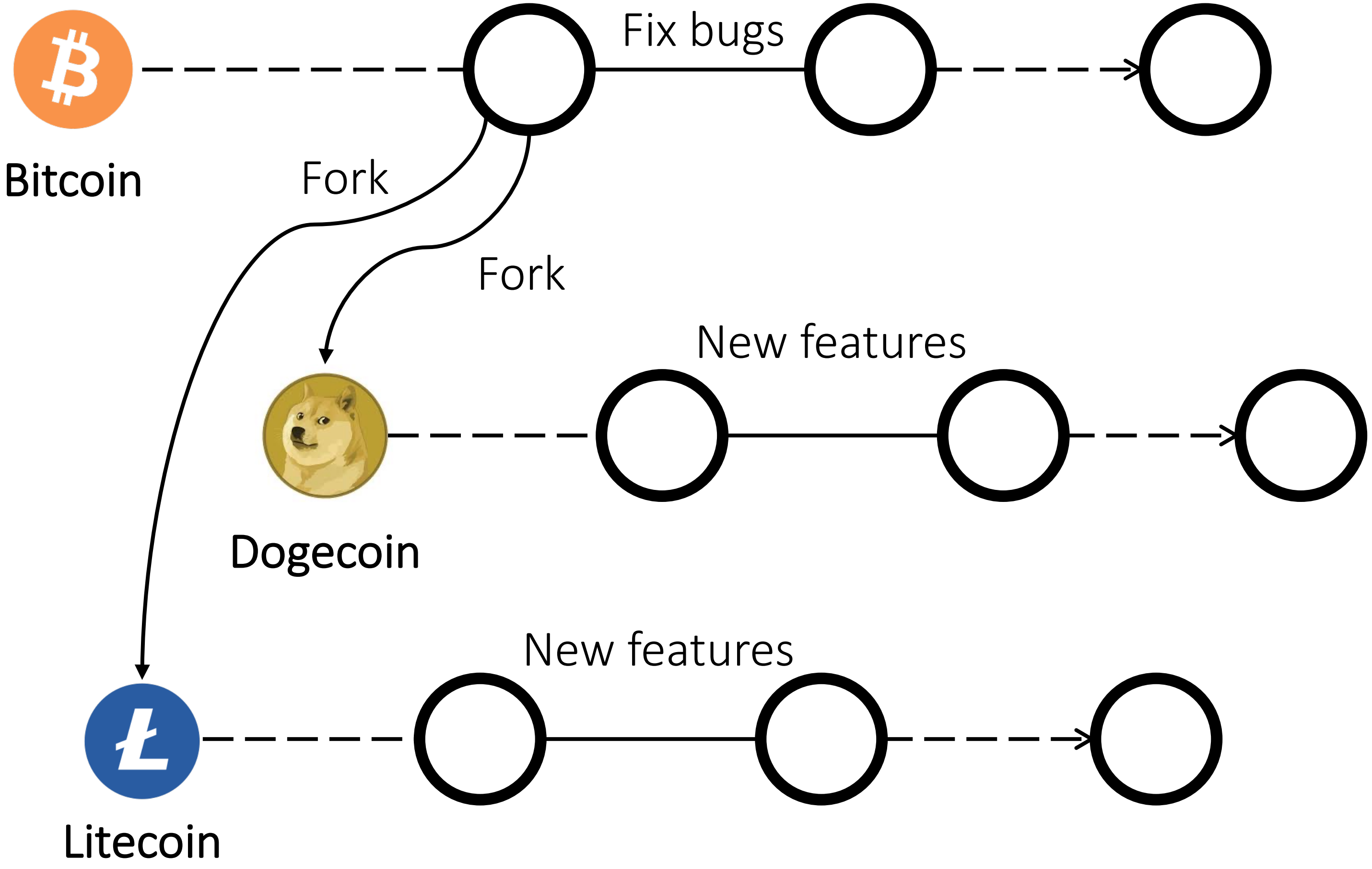
NDSS 2023

BlockScope: Detecting and Investigating Propagated Vulnerabilities in Forked Blockchain Projects

Xiao Yi[1], Yuzhou Fang[1], Daoyuan Wu[1]*, and Lingxiao Jiang[2]

[1]The Chinese University of Hong Kong
[2]Singapore Management University

Background



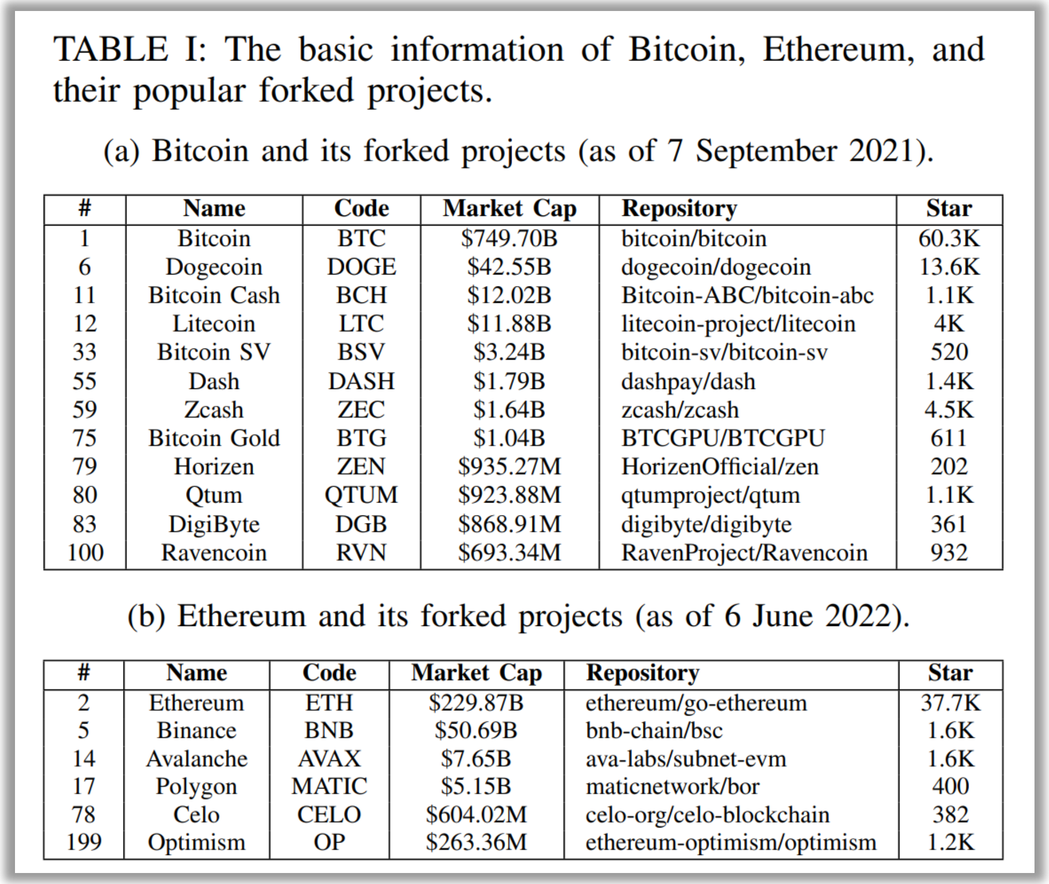


TABLE I: The basic information of Bitcoin, Ethereum, and their popular forked projects.

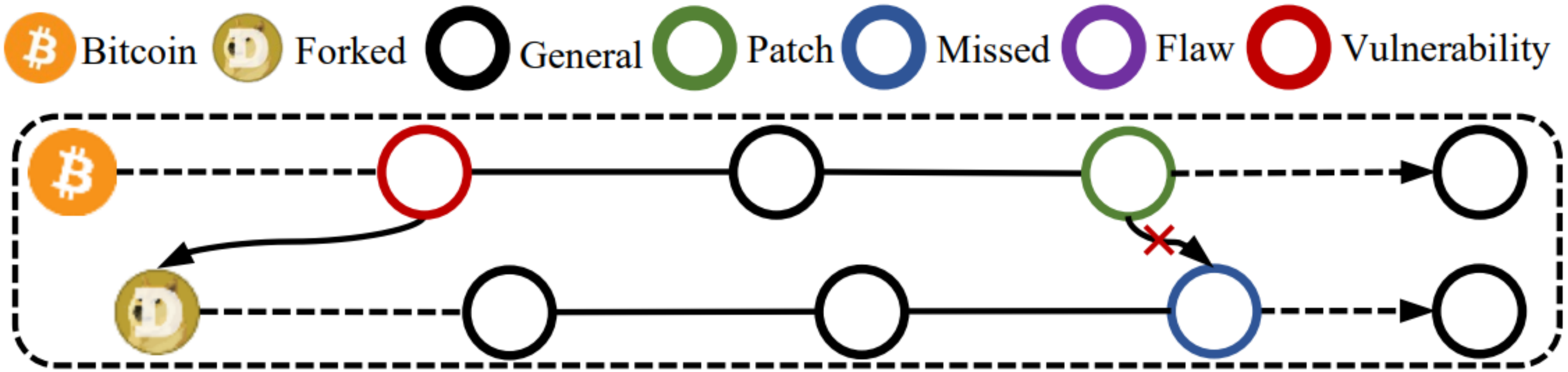
(a) Bitcoin and its forked projects (as of 7 September 2021).

#	Name	Code	Market Cap	Repository	Star
1	Bitcoin	BTC	$749.70B	bitcoin/bitcoin	60.3K
6	Dogecoin	DOGE	$42.55B	dogecoin/dogecoin	13.6K
11	Bitcoin Cash	BCH	$12.02B	Bitcoin-ABC/bitcoin-abc	1.1K
12	Litecoin	LTC	$11.88B	litecoin-project/litecoin	4K
33	Bitcoin SV	BSV	$3.24B	bitcoin-sv/bitcoin-sv	520
55	Dash	DASH	$1.79B	dashpay/dash	1.4K
59	Zcash	ZEC	$1.64B	zcash/zcash	4.5K
75	Bitcoin Gold	BTG	$1.04B	BTCGPU/BTCGPU	611
79	Horizen	ZEN	$935.27M	HorizenOfficial/zen	202
80	Qtum	QTUM	$923.88M	qtumproject/qtum	1.1K
83	DigiByte	DGB	$868.91M	digibyte/digibyte	361
100	Ravencoin	RVN	$693.34M	RavenProject/Ravencoin	932

(b) Ethereum and its forked projects (as of 6 June 2022).

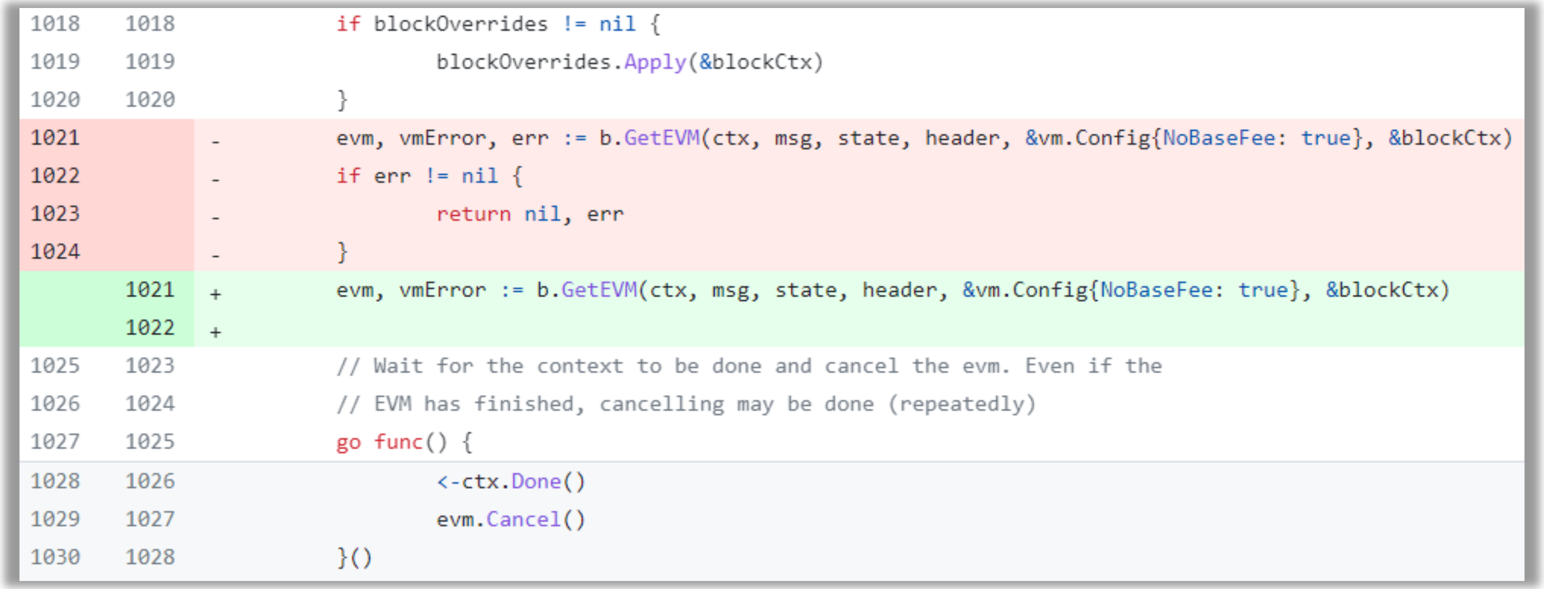
#	Name	Code	Market Cap	Repository	Star
2	Ethereum	ETH	$229.87B	ethereum/go-ethereum	37.7K
5	Binance	BNB	$50.69B	bnb-chain/bsc	1.6K
14	Avalanche	AVAX	$7.65B	ava-labs/subnet-evm	1.6K
17	Polygon	MATIC	$5.15B	maticnetwork/bor	400
78	Celo	CELO	$604.02M	celo-org/celo-blockchain	382
199	Optimism	OP	$263.36M	ethereum-optimism/optimism	1.2K

Problem



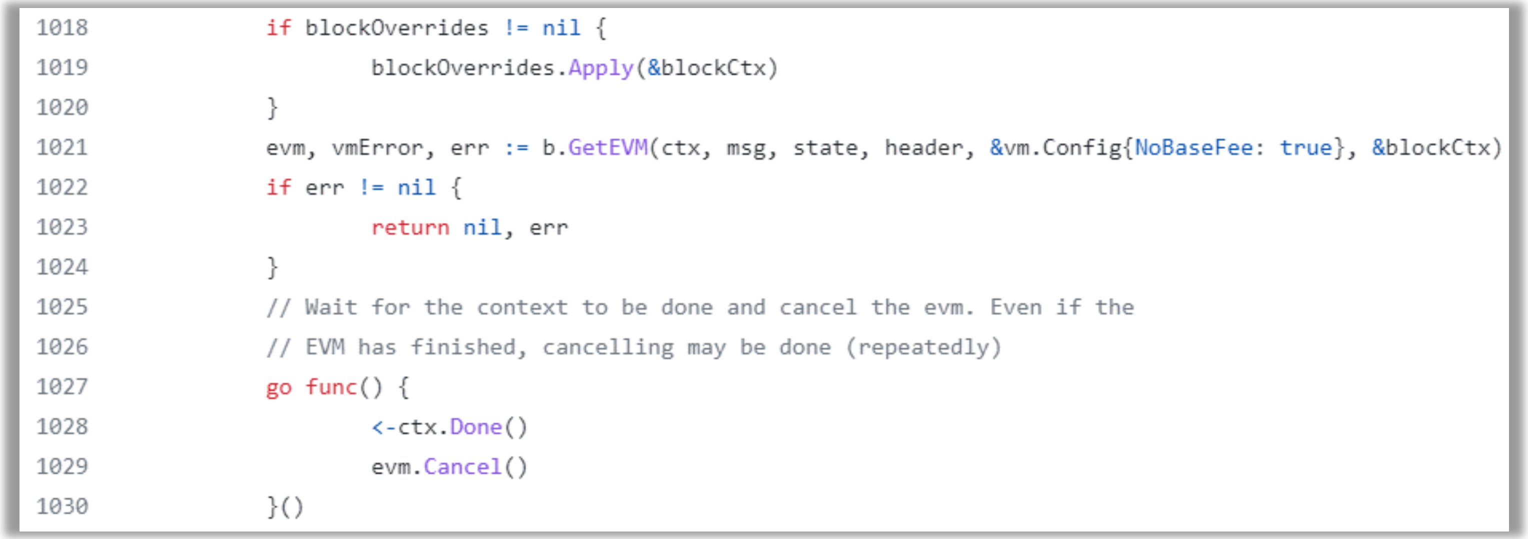
(a) The `fork` type: vulnerabilities directly forked in the beginning.

Idea



```diff
 if blockOverrides != nil {
     blockOverrides.Apply(&blockCtx)
 }
-evm, vmError, err := b.GetEVM(ctx, msg, state, header, &vm.Config{NoBaseFee: true}, &blockCtx)
-if err != nil {
-    return nil, err
-}
+evm, vmError := b.GetEVM(ctx, msg, state, header, &vm.Config{NoBaseFee: true}, &blockCtx)
+
 // Wait for the context to be done and cancel the evm. Even if the
 // EVM has finished, cancelling may be done (repeatedly)
 go func() {
     <-ctx.Done()
     evm.Cancel()
 }()
```

One submission of Ethereum



```go
if blockOverrides != nil {
    blockOverrides.Apply(&blockCtx)
}
evm, vmError, err := b.GetEVM(ctx, msg, state, header, &vm.Config{NoBaseFee: true}, &blockCtx)
if err != nil {
    return nil, err
}
// Wait for the context to be done and cancel the evm. Even if the
// EVM has finished, cancelling may be done (repeatedly)
go func() {
    <-ctx.Done()
    evm.Cancel()
}()
```

Forked projects

Challenges & Research Gap

A) 3 types of code clones:

➢ Type-1 clones refer to two identical code fragments with variations in whitespaces, layouts, and comments
➢ Type-2 clones include Type-1 clones and extend the variations to identifiers, literals, and types, e.g., variable renaming
➢ Type-3 clones further extend these variations to syntactically similar code with inserted, deleted, or updated statements

B) Huge number of lines of code (LOC):

➢ Bitcoin: 4.2M C/C++ LOC
➢ Ethereum: 3.5M Go LOC

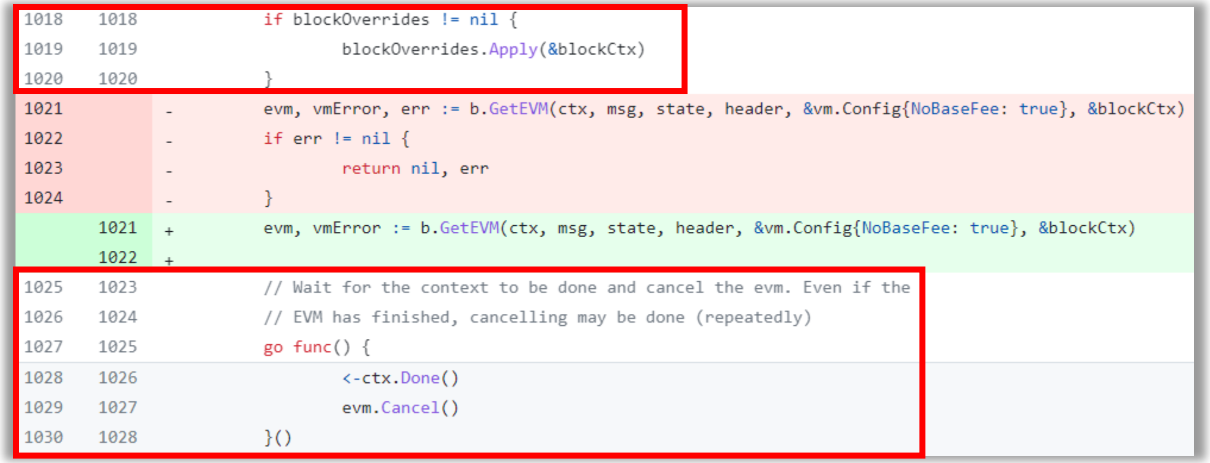
Methodology

A) 3 types of code clones:

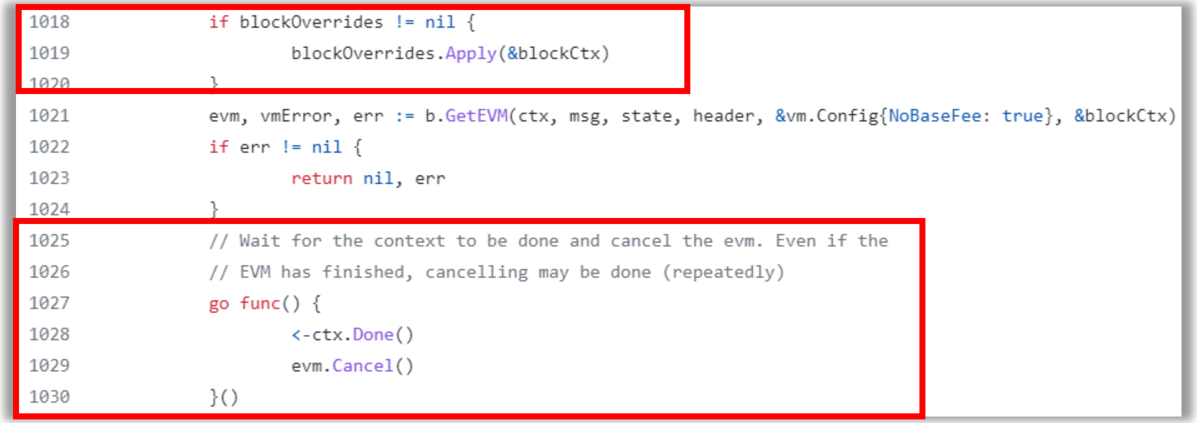
➢ Adopting similarity-based code match for being more tolerant to variant code clones

B) Huge number of lines of code (LOC):

➢ Leveraging patch code contexts to search and locate only potentially relevant code

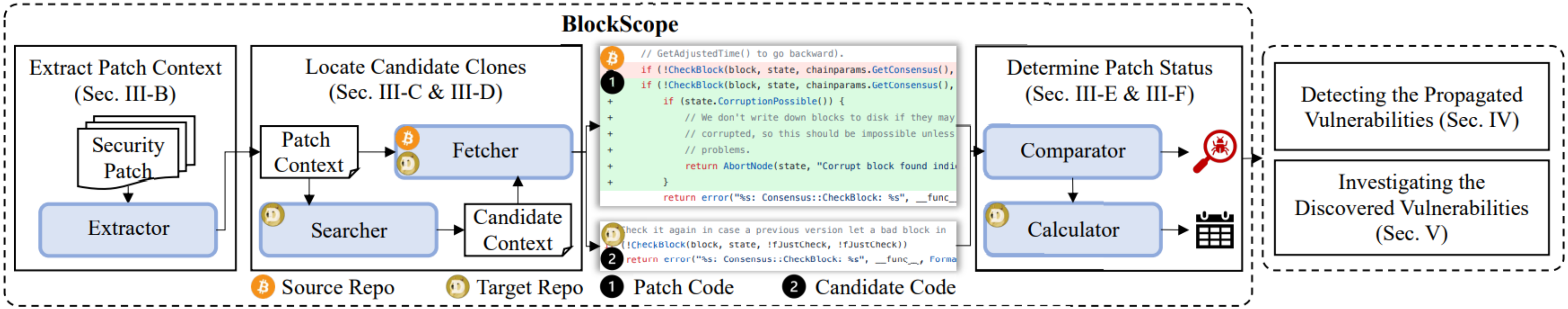


```
      if blockOverrides != nil {
              blockOverrides.Apply(&blockCtx)
      }
-     evm, vmError, err := b.GetEVM(ctx, msg, state, header, &vm.Config{NoBaseFee: true}, &blockCtx)
-     if err != nil {
-             return nil, err
-     }
+     evm, vmError := b.GetEVM(ctx, msg, state, header, &vm.Config{NoBaseFee: true}, &blockCtx)
+
      // Wait for the context to be done and cancel the evm. Even if the
      // EVM has finished, cancelling may be done (repeatedly)
      go func() {
              <-ctx.Done()
              evm.Cancel()
      }()
```



```
      if blockOverrides != nil {
              blockOverrides.Apply(&blockCtx)
      }
      evm, vmError, err := b.GetEVM(ctx, msg, state, header, &vm.Config{NoBaseFee: true}, &blockCtx)
      if err != nil {
              return nil, err
      }
      // Wait for the context to be done and cancel the evm. Even if the
      // EVM has finished, cancelling may be done (repeatedly)
      go func() {
              <-ctx.Done()
              evm.Cancel()
      }()
```

Workflow



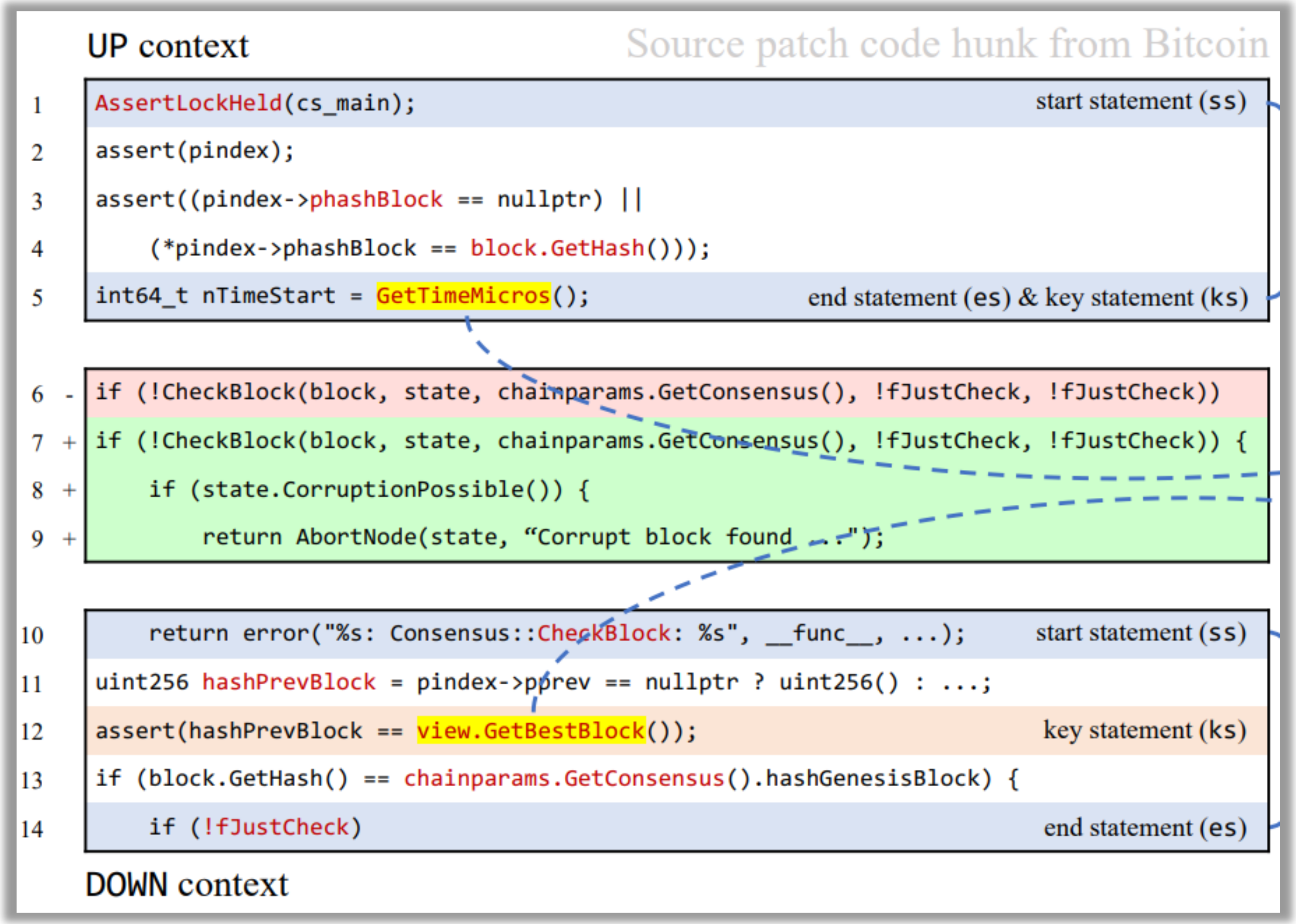
```
    // GetAdjustedTime() to go backward).
    if (!CheckBlock(block, state, chainparams.GetConsensus(),
    if (!CheckBlock(block, state, chainparams.GetConsensus(),
+       if (state.CorruptionPossible()) {
+           // We don't write down blocks to disk if they may
+           // corrupted, so this should be impossible unless
+           // problems.
+           return AbortNode(state, "Corrupt block found indi
+       }
        return error("%s: Consensus::CheckBlock: %s", __func_
```

```
heck it again in case a previous version let a bad block in
(!CheckBlock(block, state, !fJustCheck, !fJustCheck))
 return error("%s: Consensus::CheckBlock: %s", __func__, Forma
```

Fig. 2: The overall workflow of BlockScope and our study.

Extractor

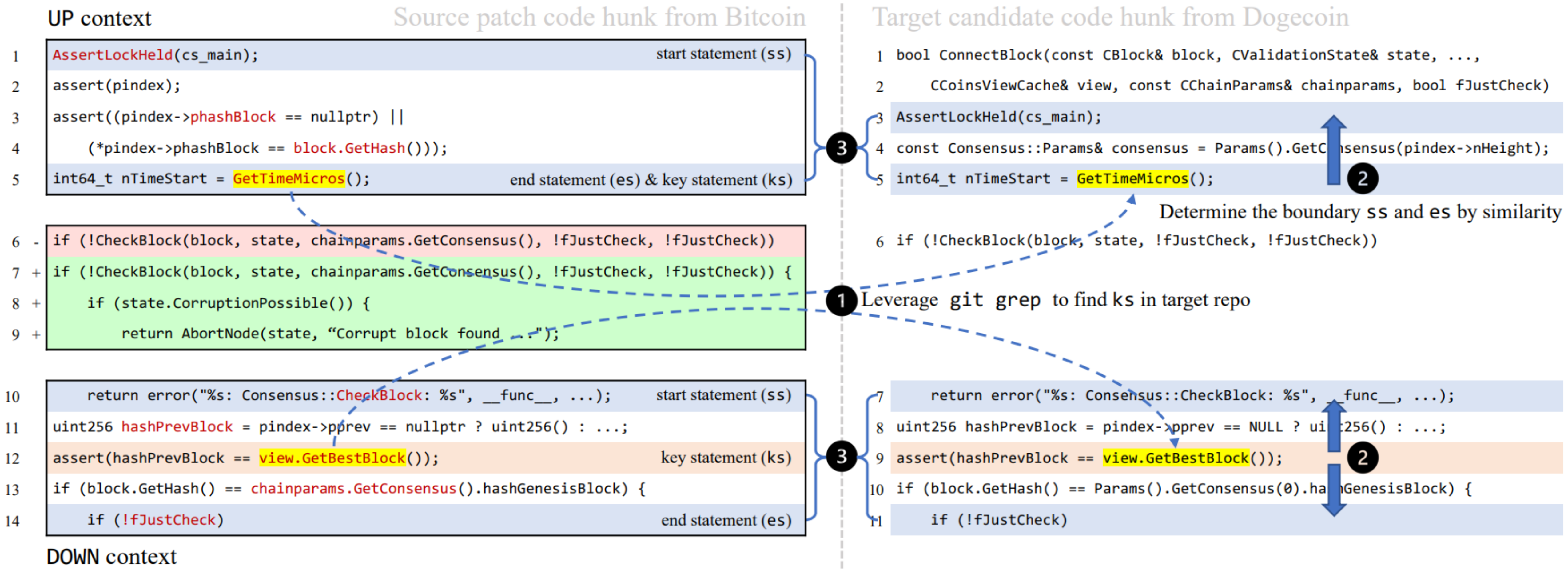
Extracting Patch Contexts from the Source Repositories



```
  AssertLockHeld(cs_main);
  assert(pindex);
  assert((pindex->phashBlock == nullptr) ||
      (*pindex->phashBlock == block.GetHash()));
  int64_t nTimeStart = GetTimeMicros();

- if (!CheckBlock(block, state, chainparams.GetConsensus(), !fJustCheck, !fJustCheck))
+ if (!CheckBlock(block, state, chainparams.GetConsensus(), !fJustCheck, !fJustCheck)) {
+     if (state.CorruptionPossible()) {
+         return AbortNode(state, "Corrupt block found ...");

      return error("%s: Consensus::CheckBlock: %s", __func__, ...);
  uint256 hashPrevBlock = pindex->pprev == nullptr ? uint256() : ...;
  assert(hashPrevBlock == view.GetBestBlock());
  if (block.GetHash() == chainparams.GetConsensus().hashGenesisBlock) {
      if (!fJustCheck)
```

Searcher

Searching for Candidate Contexts in the Target Repositories


UP context
Source patch code hunk from Bitcoin
start statement (ss)
end statement (es) & key statement (ks)
start statement (ss)
key statement (ks)
end statement (es)
DOWN context
Target candidate code hunk from Dogecoin
Determine the boundary ss and es by similarity
1 Leverage git grep to find ks in target repo
2
2
3
3

```
AssertLockHeld(cs_main);
assert(pindex);
assert((pindex->phashBlock == nullptr) ||
    (*pindex->phashBlock == block.GetHash()));
int64_t nTimeStart = GetTimeMicros();
```

```
- if (!CheckBlock(block, state, chainparams.GetConsensus(), !fJustCheck, !fJustCheck))
+ if (!CheckBlock(block, state, chainparams.GetConsensus(), !fJustCheck, !fJustCheck)) {
+     if (state.CorruptionPossible()) {
+         return AbortNode(state, “Corrupt block found ...”);
```

```
    return error("%s: Consensus::CheckBlock: %s", __func__, ...);
uint256 hashPrevBlock = pindex->pprev == nullptr ? uint256() : ...;
assert(hashPrevBlock == view.GetBestBlock());
if (block.GetHash() == chainparams.GetConsensus().hashGenesisBlock) {
    if (!fJustCheck)
```

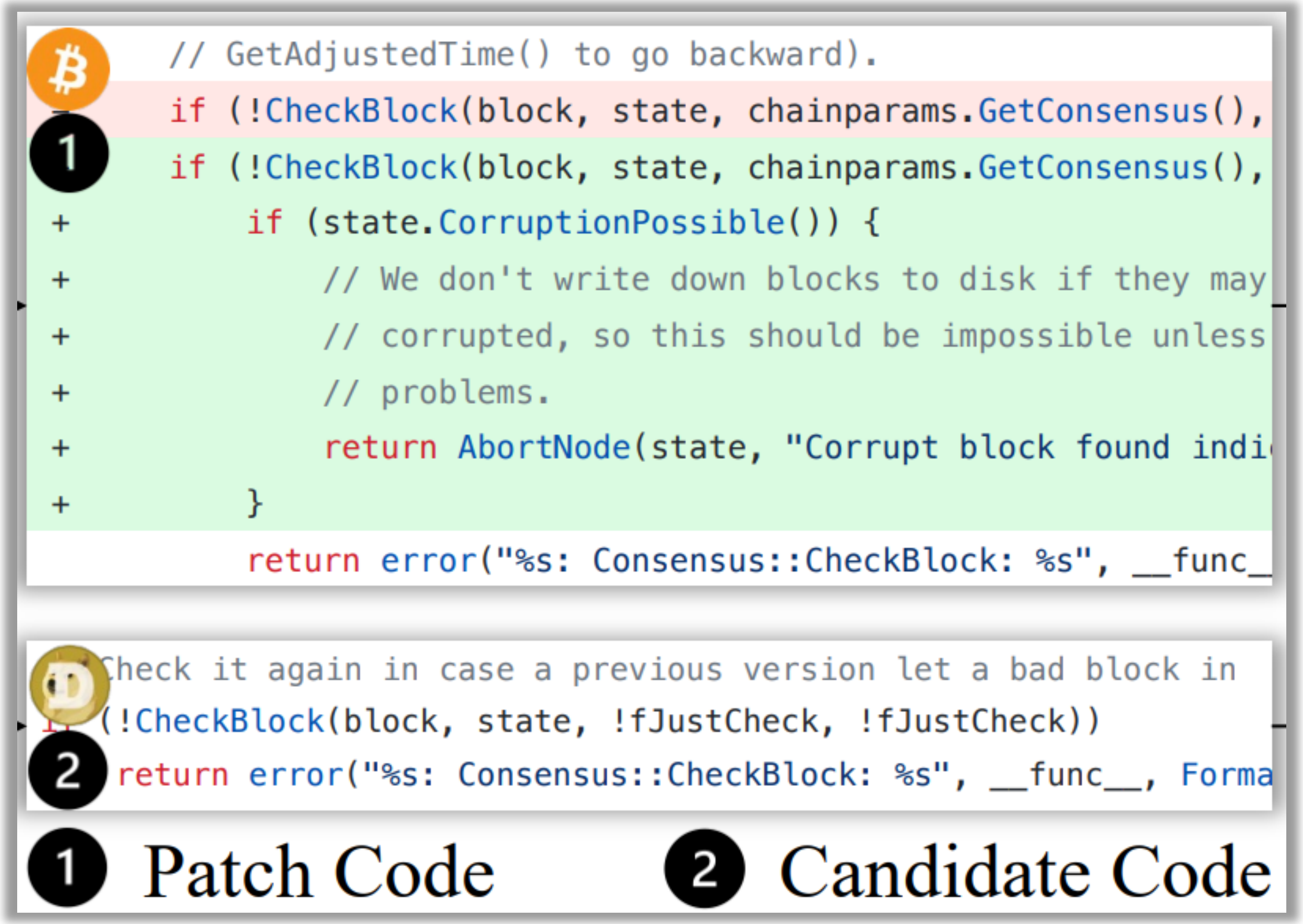
```
bool ConnectBlock(const CBlock& block, CValidationState& state, ...,
    CCoinsViewCache& view, const CChainParams& chainparams, bool fJustCheck)
AssertLockHeld(cs_main);
const Consensus::Params& consensus = Params().GetConsensus(pindex->nHeight);
int64_t nTimeStart = GetTimeMicros();

if (!CheckBlock(block, state, !fJustCheck, !fJustCheck))
```

```
    return error("%s: Consensus::CheckBlock: %s", __func__, ...);
uint256 hashPrevBlock = pindex->pprev == NULL ? uint256() : ...;
assert(hashPrevBlock == view.GetBestBlock());
if (block.GetHash() == Params().GetConsensus(0).hashGenesisBlock) {
    if (!fJustCheck)
```

Fig. 3: Illustrating BlockScope’s context-based search process for finding candidate contexts in a target repository.

Fetcher

Fetching Patch and Candidate Code Hunks from the Source and Target Repositories



```
    // GetAdjustedTime() to go backward).
    if (!CheckBlock(block, state, chainparams.GetConsensus(),
    if (!CheckBlock(block, state, chainparams.GetConsensus(),
+       if (state.CorruptionPossible()) {
+           // We don't write down blocks to disk if they may
+           // corrupted, so this should be impossible unless
+           // problems.
+           return AbortNode(state, "Corrupt block found indi
+       }
        return error("%s: Consensus::CheckBlock: %s", __func_
```

```
Check it again in case a previous version let a bad block in
(!CheckBlock(block, state, !fJustCheck, !fJustCheck))
 return error("%s: Consensus::CheckBlock: %s", __func__, Forma
```

1 Patch Code 2 Candidate Code

Comparator

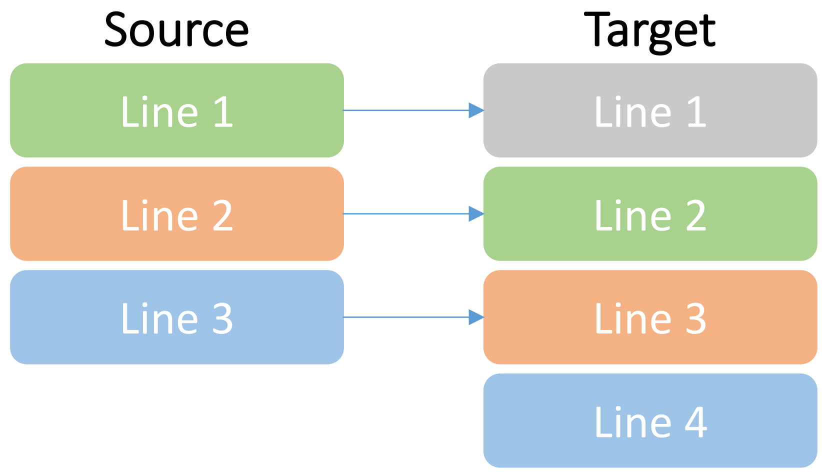
Measuring the Similarity between Patch and Candidate Code

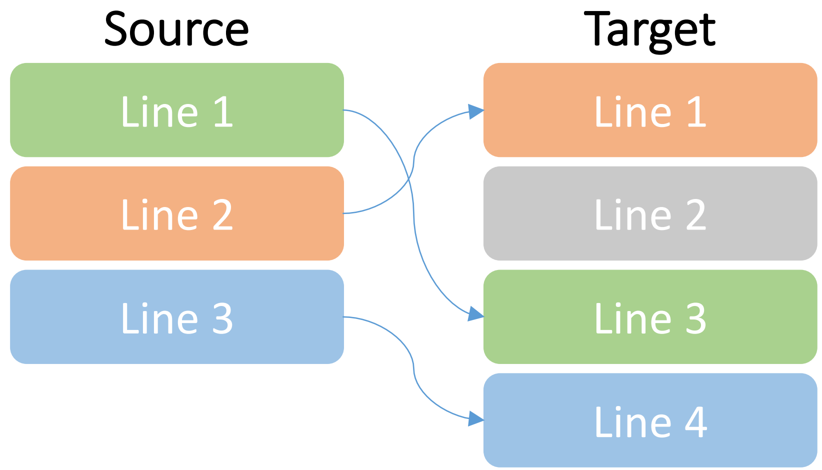
- given a source code fragment *S* with *p* code statements and a target code fragment *T* with *q* code statements

$$\frac{1}{p}\sum_{i=1}^{p} \mathrm{strsim}(S_i, T_i)$$

$$\mathrm{SIMILARITY}(S,T) = \frac{1}{p}\sum_{i=1}^{p} \mathrm{strsim}(S_i, T_j)\, r^{|i-j|}$$

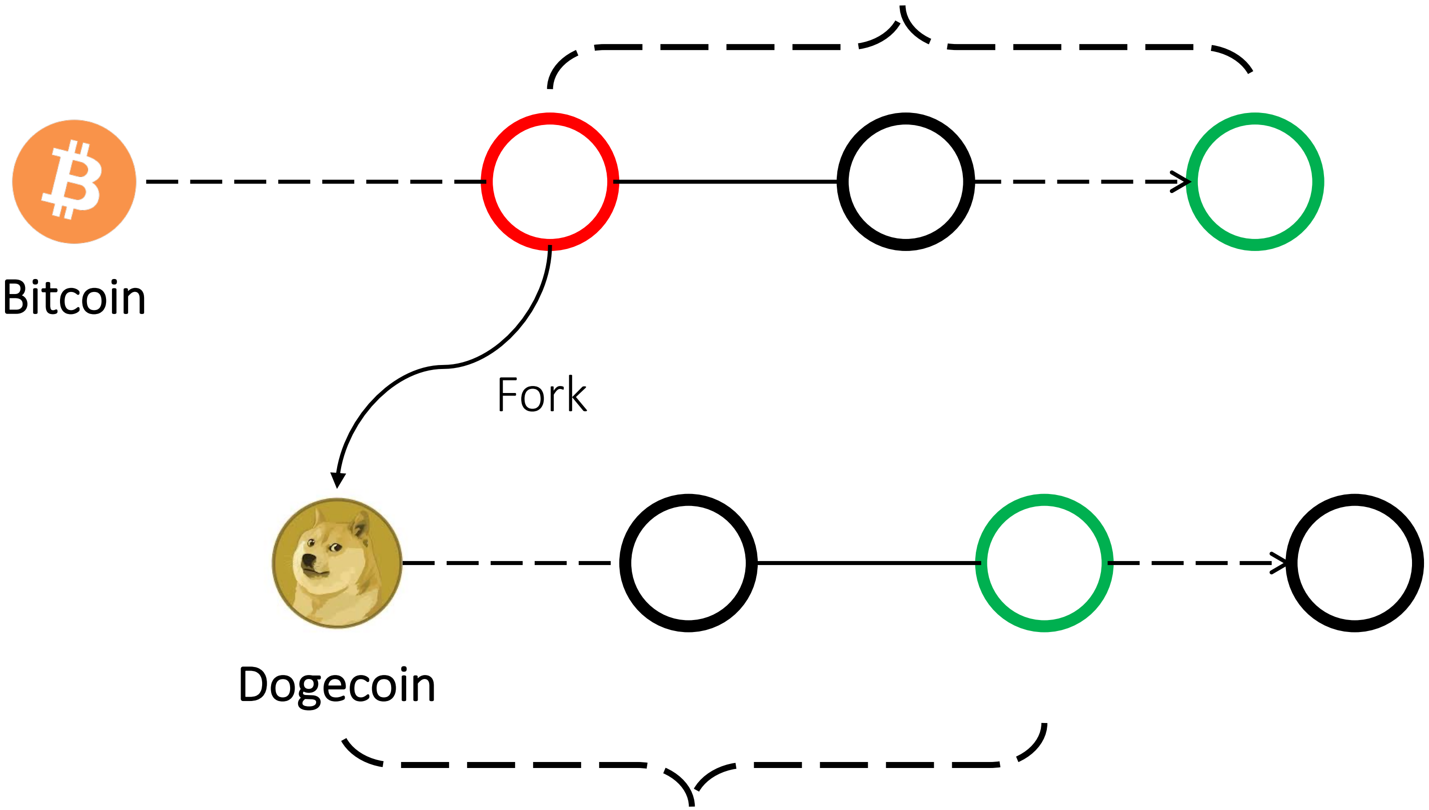
$$\text{s.t., } j = \operatorname*{arg\,max}_{1\le k\le q} \mathrm{strsim}(S_i, T_k)$$





Calculator

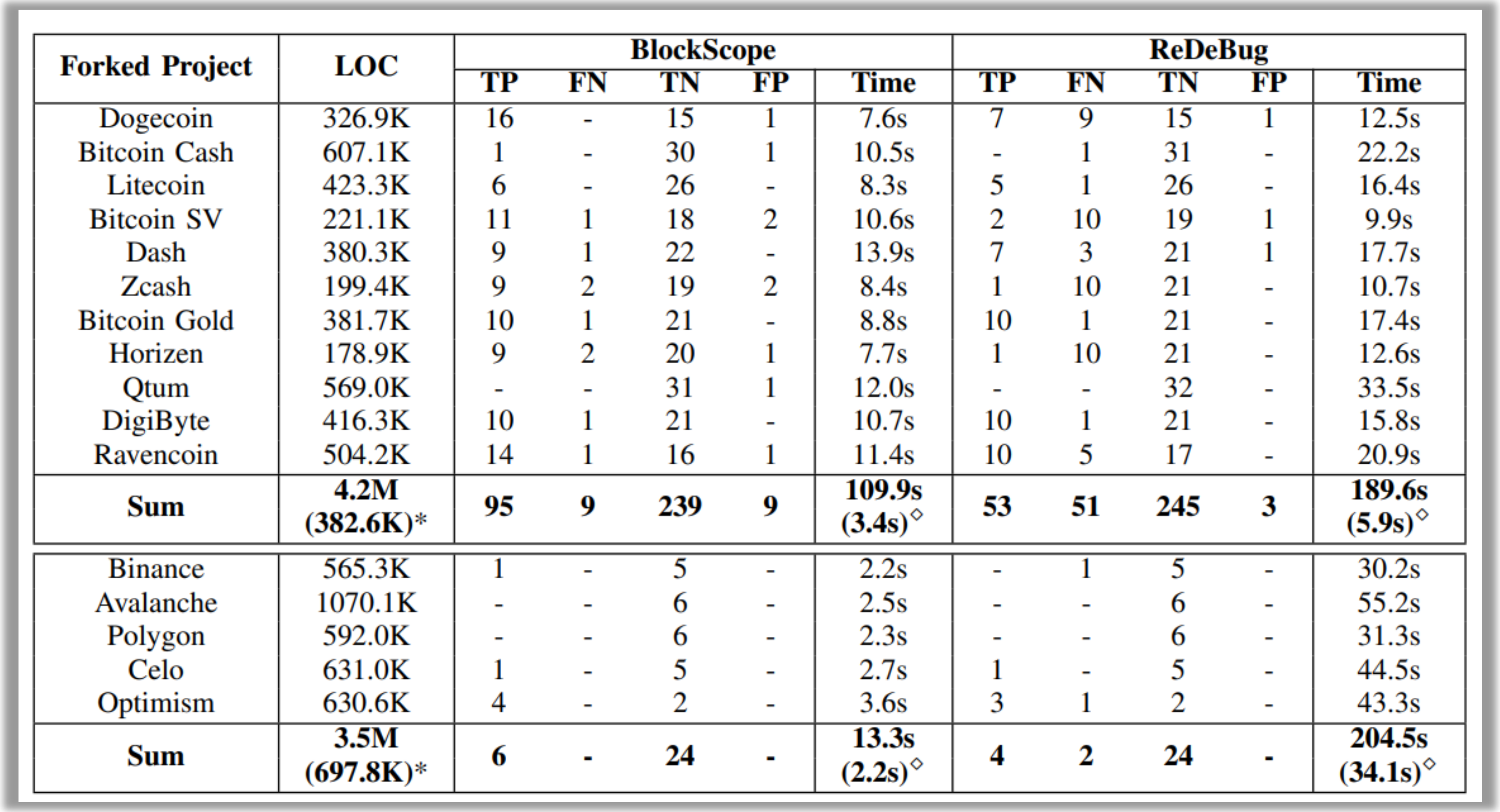
Determining Patch Delays for the Vulnerabilities Already Patched in the Target Repositories



Experiments

Dataset:

- Bitcoin —— 32 patches
- Ethereum —— 6 patches



Forked Project	LOC	BlockScope					ReDeBug				
		TP	FN	TN	FP	Time	TP	FN	TN	FP	Time
Dogecoin	326.9K	16	-	15	1	7.6s	7	9	15	1	12.5s
Bitcoin Cash	607.1K	1	-	30	1	10.5s	-	1	31	-	22.2s
Litecoin	423.3K	6	-	26	-	8.3s	5	1	26	-	16.4s
Bitcoin SV	221.1K	11	1	18	2	10.6s	2	10	19	1	9.9s
Dash	380.3K	9	1	22	-	13.9s	7	3	21	1	17.7s
Zcash	199.4K	9	2	19	2	8.4s	1	10	21	-	10.7s
Bitcoin Gold	381.7K	10	1	21	-	8.8s	10	1	21	-	17.4s
Horizen	178.9K	9	2	20	1	7.7s	1	10	21	-	12.6s
Qtum	569.0K	-	-	31	1	12.0s	-	-	32	-	33.5s
DigiByte	416.3K	10	1	21	-	10.7s	10	1	21	-	15.8s
Ravencoin	504.2K	14	1	16	1	11.4s	10	5	17	-	20.9s
Sum	**4.2M** **(382.6K)***	**95**	**9**	**239**	**9**	**109.9s** **(3.4s)**$^{\diamond}$	**53**	**51**	**245**	**3**	**189.6s** **(5.9s)**$^{\diamond}$
Binance	565.3K	1	-	5	-	2.2s	-	1	5	-	30.2s
Avalanche	1070.1K	-	-	6	-	2.5s	-	-	6	-	55.2s
Polygon	592.0K	-	-	6	-	2.3s	-	-	6	-	31.3s
Celo	631.0K	1	-	5	-	2.7s	1	-	5	-	44.5s
Optimism	630.6K	4	-	2	-	3.6s	3	1	2	-	43.3s
Sum	**3.5M** **(697.8K)***	**6**	**-**	**24**	**-**	**13.3s** **(2.2s)**$^{\diamond}$	**4**	**2**	**24**	**-**	**204.5s** **(34.1s)**$^{\diamond}$

Experiments

Developers' response

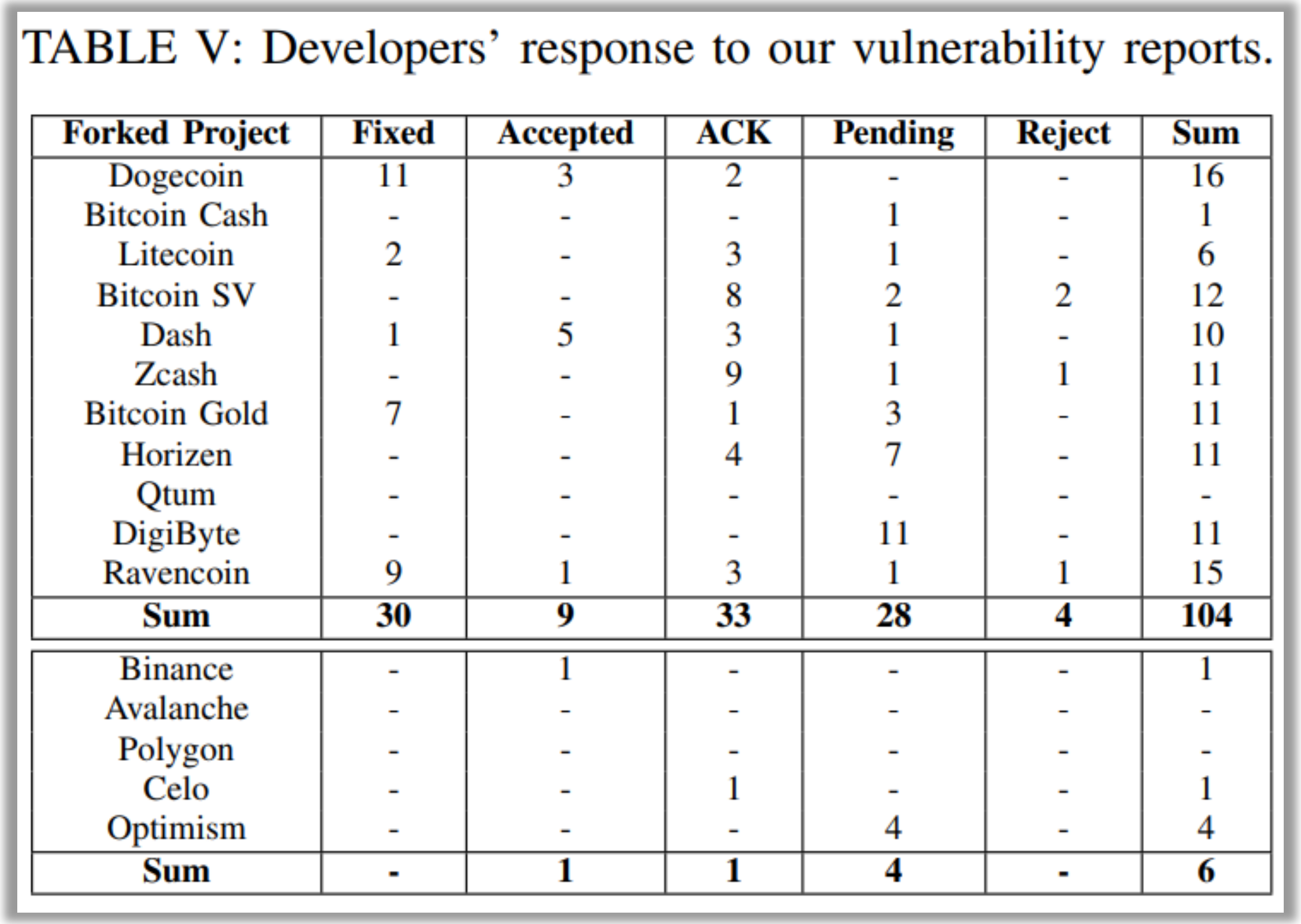


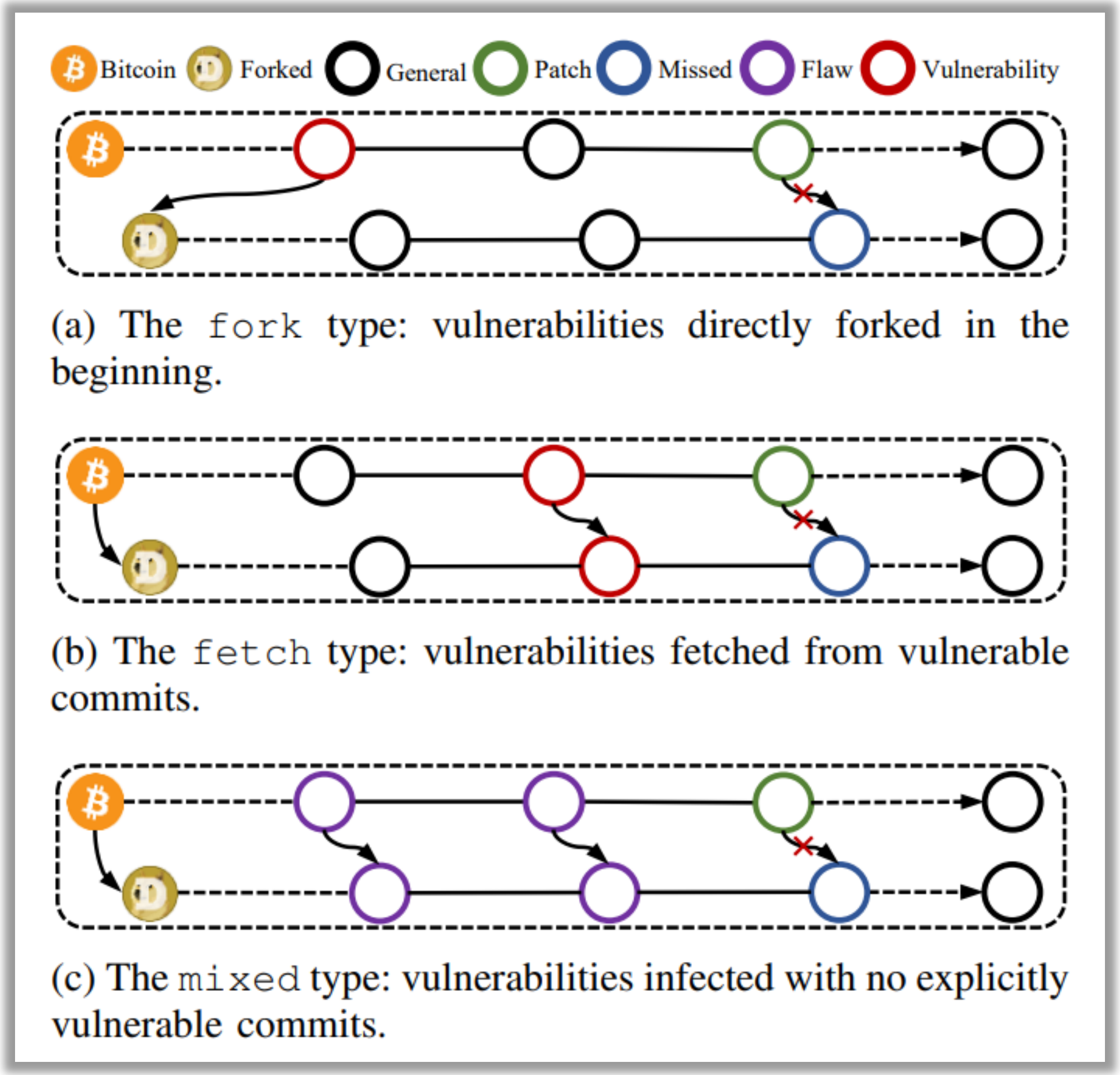
TABLE V: Developers' response to our vulnerability reports.

Forked Project	Fixed	Accepted	ACK	Pending	Reject	Sum
Dogecoin	11	3	2	-	-	16
Bitcoin Cash	-	-	-	1	-	1
Litecoin	2	-	3	1	-	6
Bitcoin SV	-	-	8	2	2	12
Dash	1	5	3	1	-	10
Zcash	-	-	9	1	1	11
Bitcoin Gold	7	-	1	3	-	11
Horizen	-	-	4	7	-	11
Qtum	-	-	-	-	-	-
DigiByte	-	-	-	11	-	11
Ravencoin	9	1	3	1	1	15
Sum	**30**	**9**	**33**	**28**	**4**	**104**
Binance	-	1	-	-	-	1
Avalanche	-	-	-	-	-	-
Polygon	-	-	-	-	-	-
Celo	-	-	1	-	-	1
Optimism	-	-	-	4	-	4
Sum	**-**	**1**	**1**	**4**	**-**	**6**

Experiments

Three types of the vulnerability propagation

- ➢ Fork
- ➢ Fetch
- ➢ Mixed



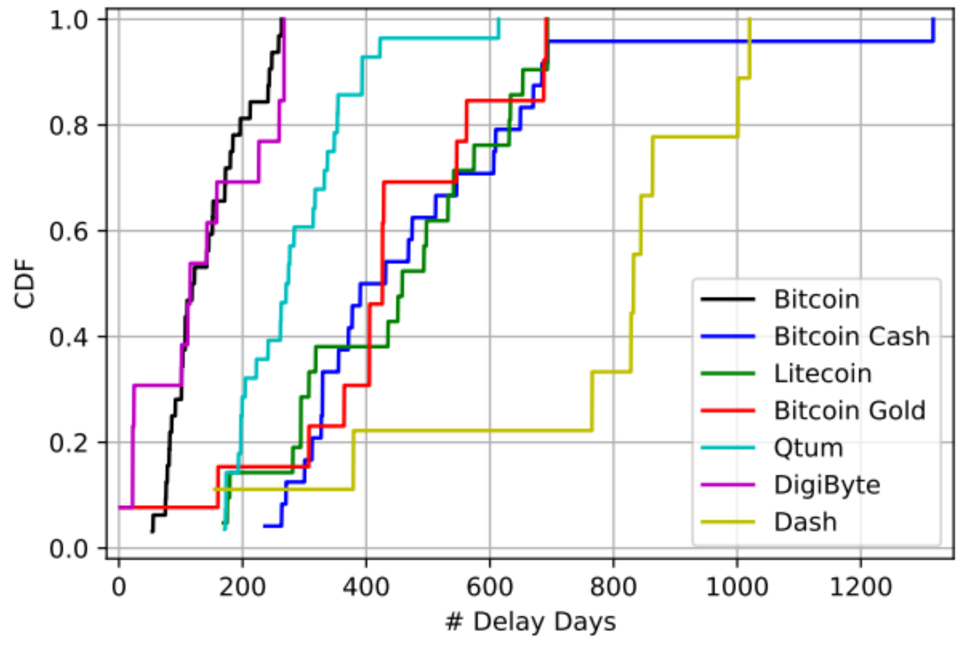
(a) The `fork` type: vulnerabilities directly forked in the beginning.

(b) The `fetch` type: vulnerabilities fetched from vulnerable commits.

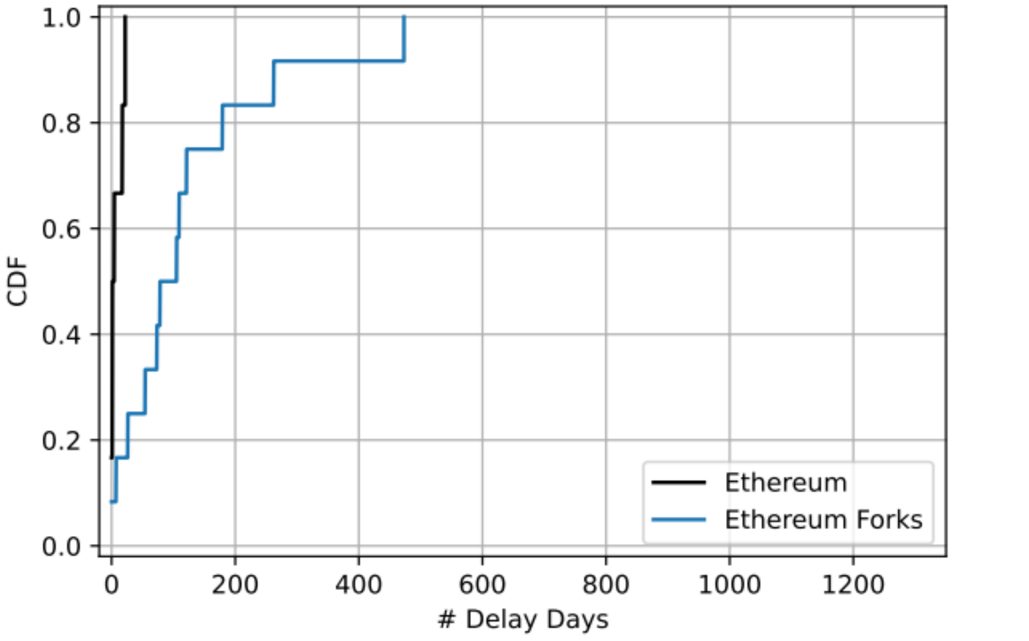
(c) The `mixed` type: vulnerabilities infected with no explicitly vulnerable commits.

Experiments

Patch Delay Analysis



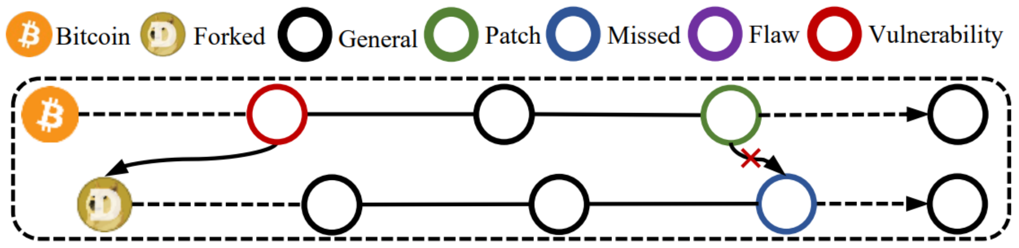
(a) For Bitcoin and its forked projects with enough patched cases.



(b) For Ethereum and its forked projects as a whole.

Summary

Problem

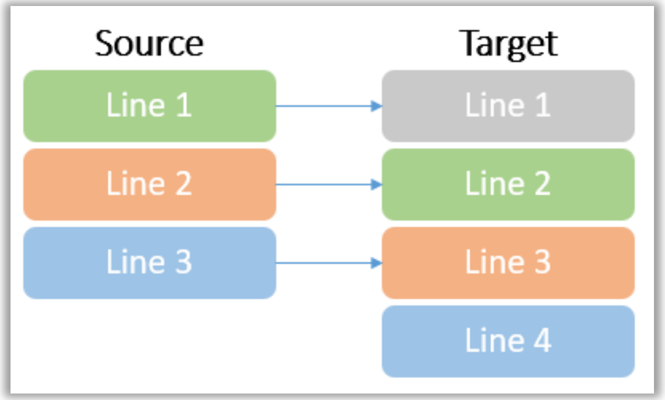


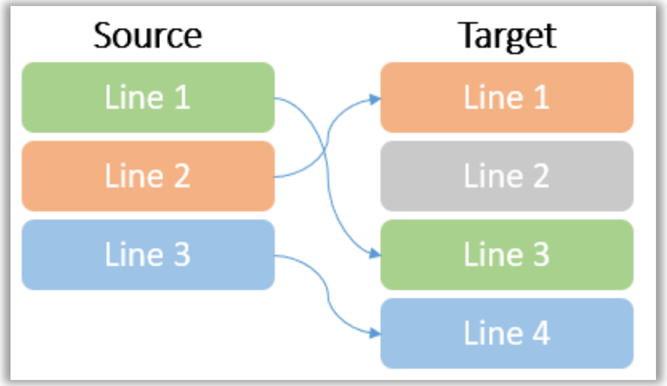
(a) The `fork` type: vulnerabilities directly forked in the beginning.

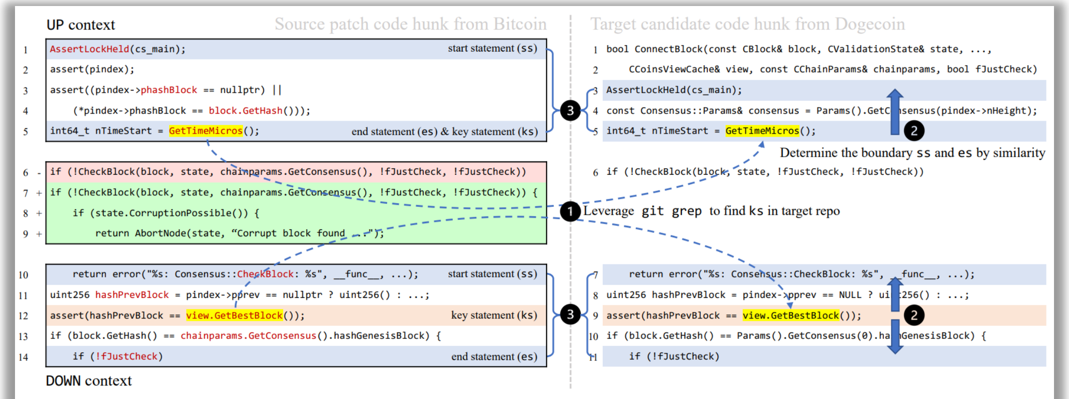
Challenges & Research Gap

- ➢ 3 types of code clones
- ➢ Huge number of lines of code (LOC)

Methodology







UP context
Source patch code hunk from Bitcoin
start statement (ss)
end statement (es) & key statement (ks)
start statement (ss)
key statement (ks)
end statement (es)
DOWN context
Target candidate code hunk from Dogecoin
1 Leverage git grep to find ks in target repo
2 Determine the boundary ss and es by similarity
3

```
  AssertLockHeld(cs_main);
  assert(pindex);
  assert((pindex->phashBlock == nullptr) ||
      (*pindex->phashBlock == block.GetHash()));
  int64_t nTimeStart = GetTimeMicros();

- if (!CheckBlock(block, state, chainparams.GetConsensus(), !fJustCheck, !fJustCheck))
+ if (!CheckBlock(block, state, chainparams.GetConsensus(), !fJustCheck, !fJustCheck)) {
+     if (state.CorruptionPossible()) {
+         return AbortNode(state, “Corrupt block found ...");

      return error("%s: Consensus::CheckBlock: %s", __func__, ...);
  uint256 hashPrevBlock = pindex->pprev == nullptr ? uint256() : ...;
  assert(hashPrevBlock == view.GetBestBlock());
  if (block.GetHash() == chainparams.GetConsensus().hashGenesisBlock) {
      if (!fJustCheck)
```

```
bool ConnectBlock(const CBlock& block, CValidationState& state, ...,
    CCoinsViewCache& view, const CChainParams& chainparams, bool fJustCheck)
AssertLockHeld(cs_main);
const Consensus::Params& consensus = Params().GetConsensus(pindex->nHeight);
int64_t nTimeStart = GetTimeMicros();

if (!CheckBlock(block, state, !fJustCheck, !fJustCheck))

    return error("%s: Consensus::CheckBlock: %s", __func__, ...);
uint256 hashPrevBlock = pindex->pprev == NULL ? uint256() : ...;
assert(hashPrevBlock == view.GetBestBlock());
if (block.GetHash() == Params().GetConsensus(0).hashGenesisBlock) {
    if (!fJustCheck)
```

Fig. 3: Illustrating BlockScope’s context-based search process for finding candidate contexts in a target repository.