Department of Information Engineering
The Chinese University of Hong Kong

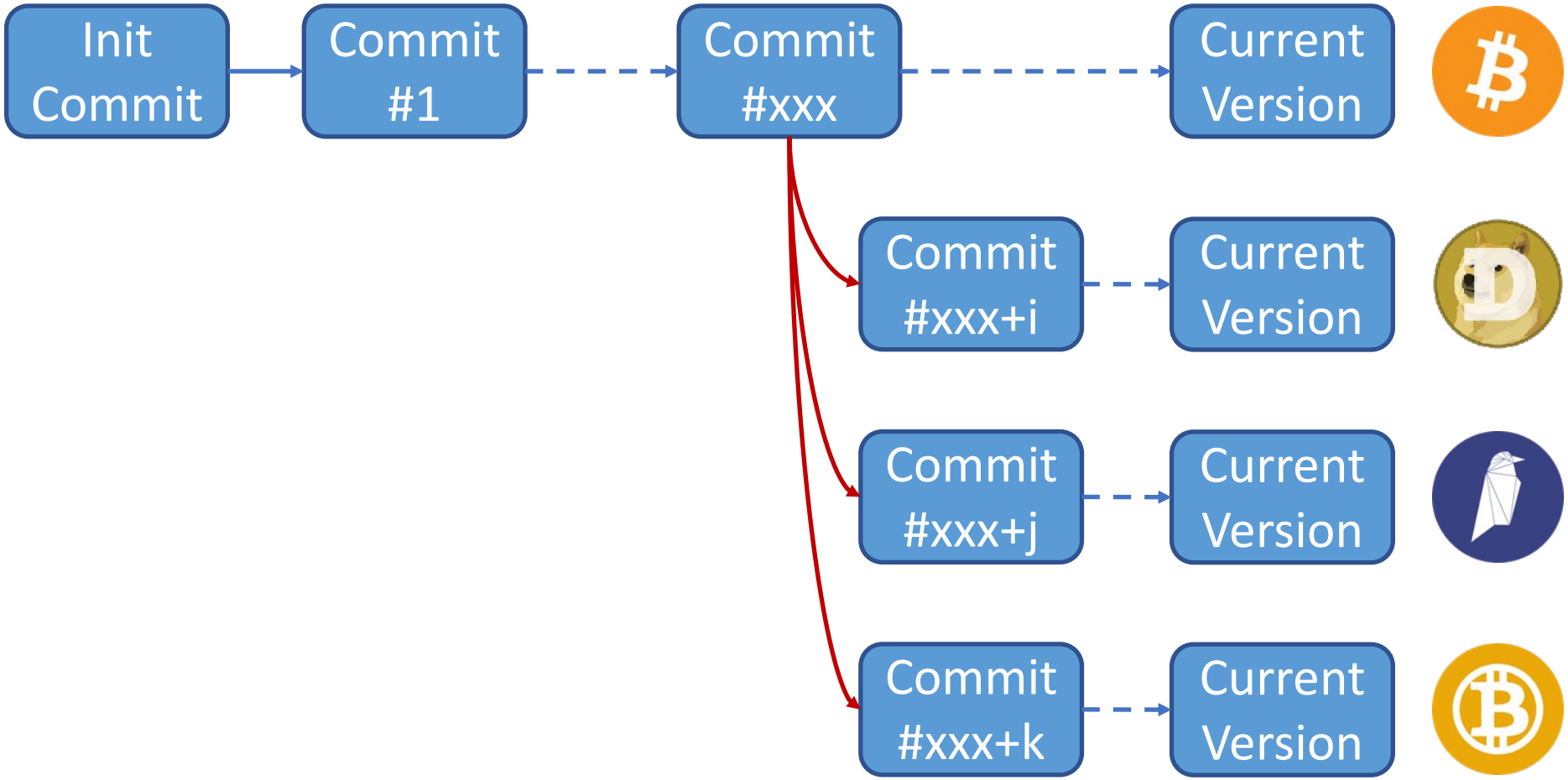
SMU
SINGAPORE MANAGEMENT UNIVERSITY

BlockScope: Detecting and Investigating Propagated Vulnerabilities in Forked Blockchain Projects

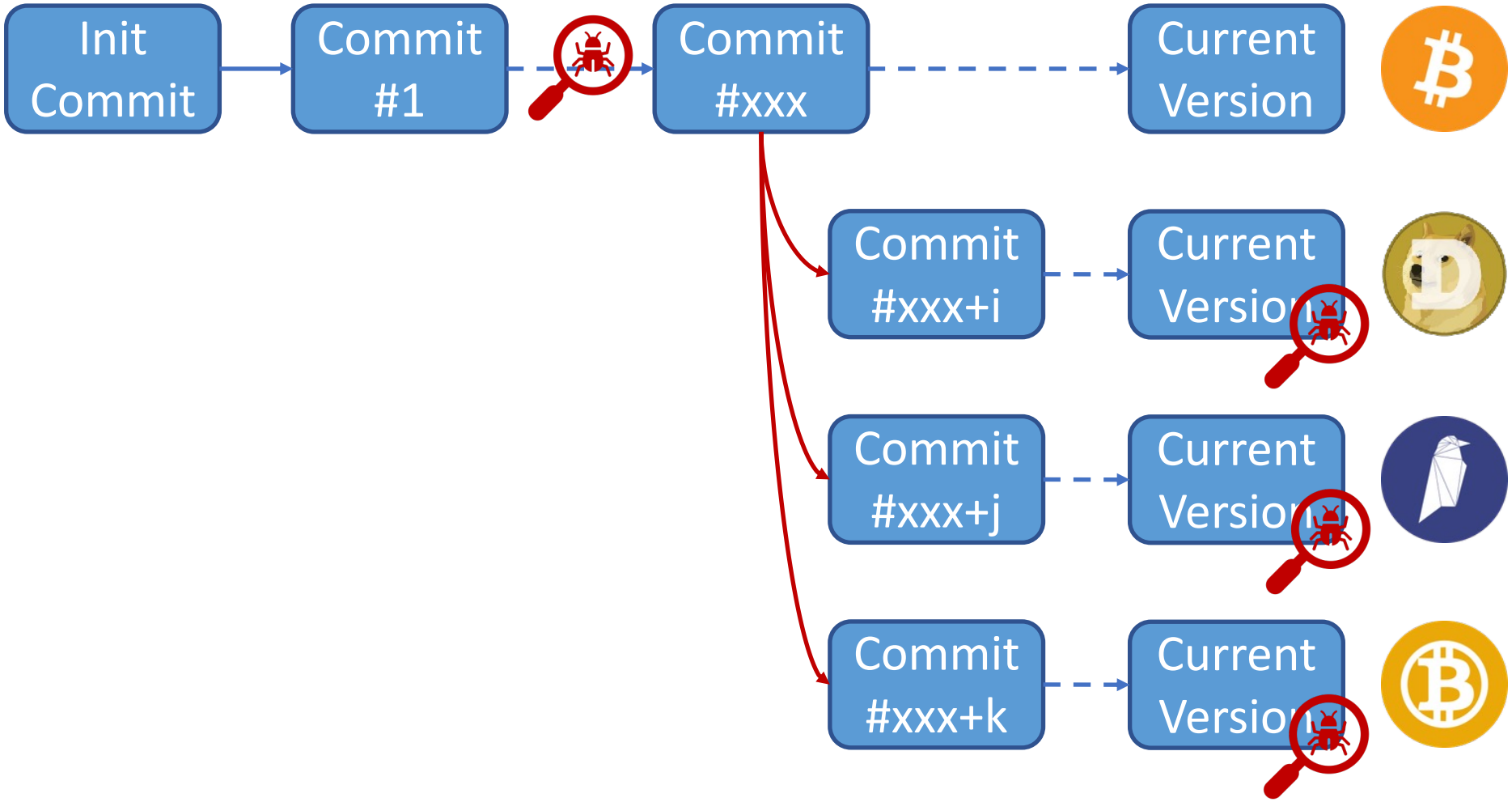
Xiao Yi[1], Yuzhou Fang[1], Daoyuan Wu[1*], Lingxiao Jiang[2]

https://github.com/VPRLab/BlkVulnReport

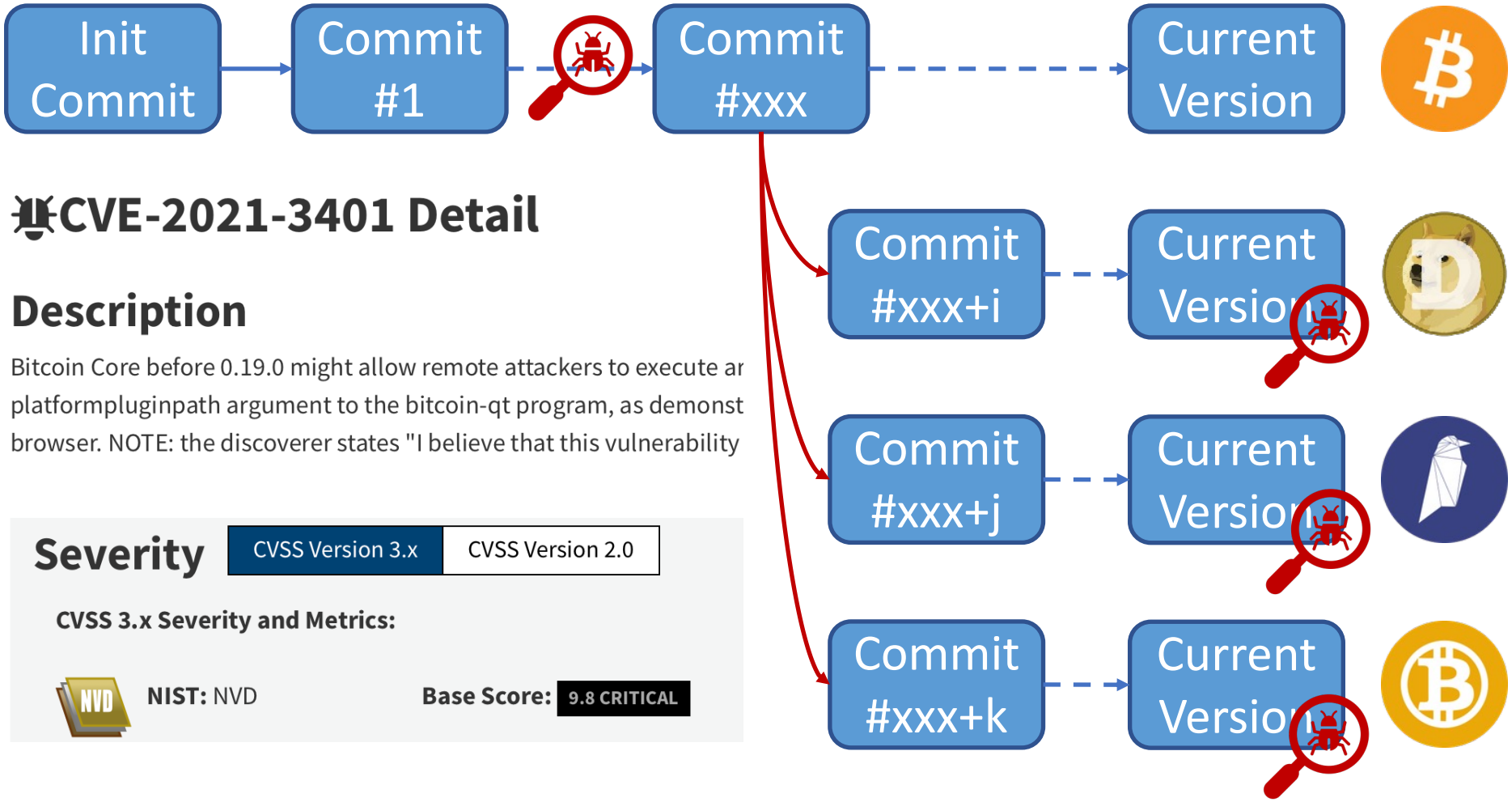
Motivation: Whether Bitcoin/Ethereum vulnerabilities propagated to their forked projects?



Motivation: Whether Bitcoin/Ethereum vulnerabilities propagated to their forked projects?



Motivation: Whether Bitcoin/Ethereum vulnerabilities propagated to their forked projects?



CVE-2021-3401 Detail

Description

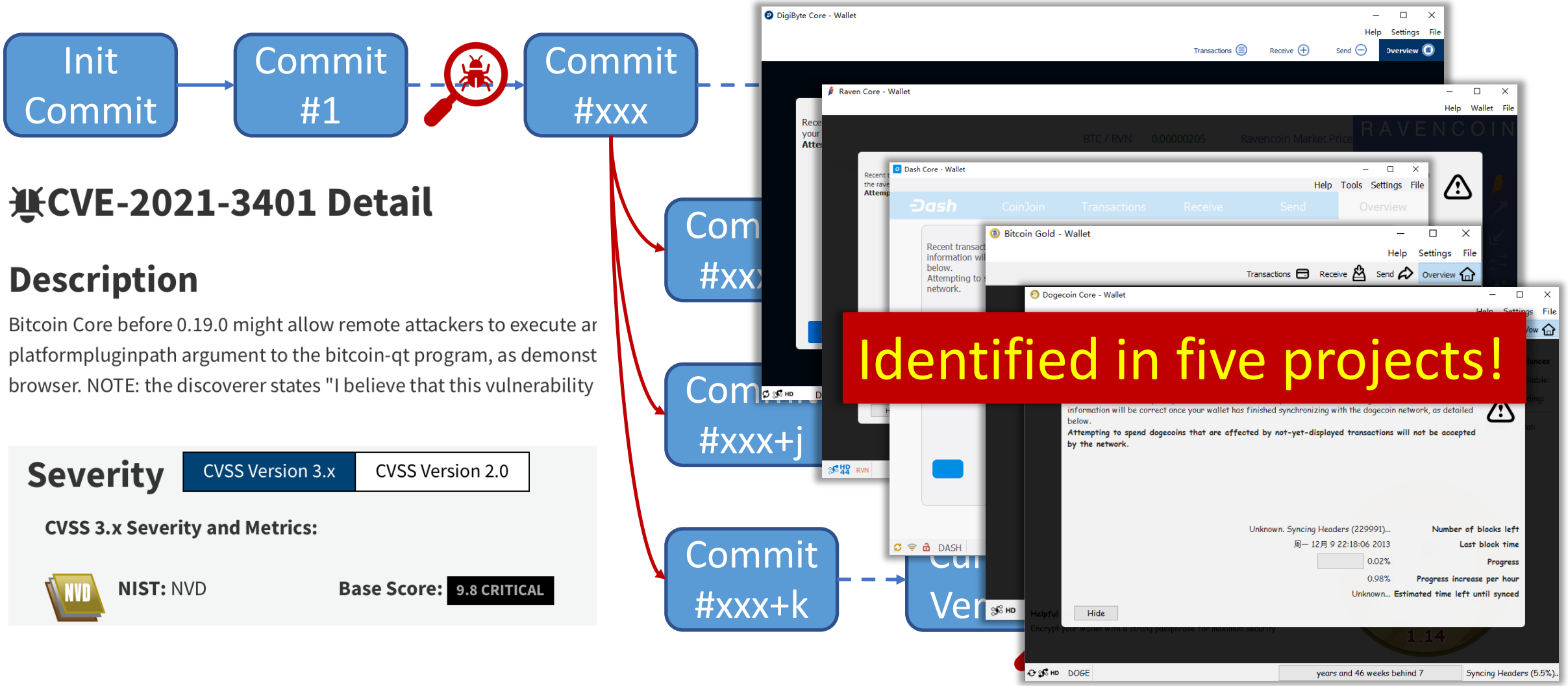
Bitcoin Core before 0.19.0 might allow remote attackers to execute ar platformpluginpath argument to the bitcoin-qt program, as demonst browser. NOTE: the discoverer states "I believe that this vulnerability

Severity CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

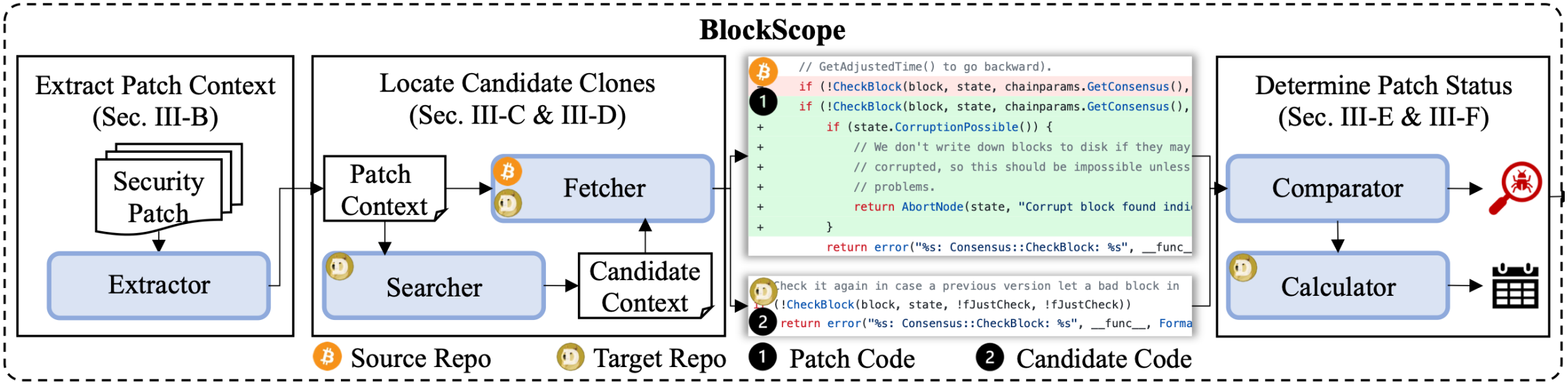
NVD **NIST:** NVD **Base Score:** 9.8 CRITICAL

Motivation: Whether Bitcoin/Ethereum vulnerabilities propagated to their forked projects?



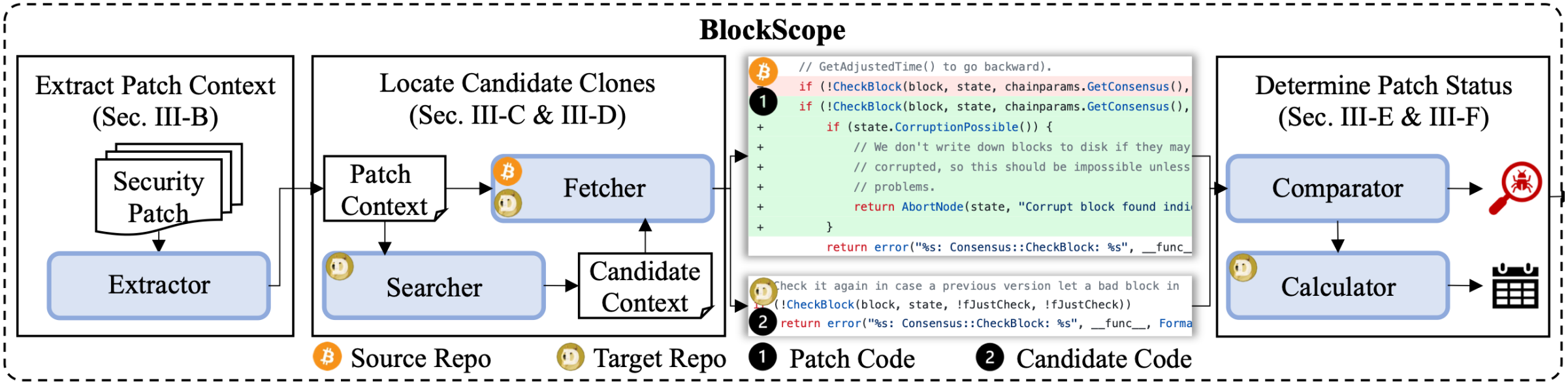
Our Tool: BlockScope

- A novel patch-based clone detection tool for propagated vulnerabilities in forked blockchain projects.



Our Tool: BlockScope

- A novel patch-based clone detection tool for propagated vulnerabilities in forked blockchain projects.



1. Leverage patch code contexts to locate only potentially relevant code

2. Adopt similarity-based code match for being immune to clone variants

Context-based Candidate Clone Search

UP context

Source patch code hunk from **Bitcoin**

```
1   AssertLockHeld(cs_main);                                          start statement (ss)
2   assert(pindex);
3   assert((pindex->phashBlock == nullptr) ||
4       (*pindex->phashBlock == block.GetHash()));
5   int64_t nTimeStart = GetTimeMicros();                  end statement (es) & key statement (ks)

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

10      return error("%s: Consensus::CheckBlock: %s", __func__, ...);      start statement (ss)
11  uint256 hashPrevBlock = pindex->pprev == nullptr ? uint256() : ...;
12  assert(hashPrevBlock == view.GetBestBlock());                          key statement (ks)
13  if (block.GetHash() == chainparams.GetConsensus().hashGenesisBlock) {
14      if (!fJustCheck)                                                   end statement (es)
```

DOWN context

Target candidate code hunk from **Dogecoin**

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

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

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

Context-based Candidate Clone Search

UP context — Source patch code hunk from **Bitcoin**

```
1   AssertLockHeld(cs_main);                                          start statement (ss)
2   assert(pindex);
3   assert((pindex->phashBlock == nullptr) ||
4       (*pindex->phashBlock == block.GetHash()));
5   int64_t nTimeStart = GetTimeMicros();                  end statement (es) & key statement (ks)

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

10      return error("%s: Consensus::CheckBlock: %s", __func__, ...);      start statement (ss)
11  uint256 hashPrevBlock = pindex->pprev == nullptr ? uint256() : ...;
12  assert(hashPrevBlock == view.GetBestBlock());                          key statement (ks)
13  if (block.GetHash() == chainparams.GetConsensus().hashGenesisBlock) {
14      if (!fJustCheck)                                                   end statement (es)
```

DOWN context

Target candidate code hunk from **Dogecoin**

Context-based Candidate Clone Search

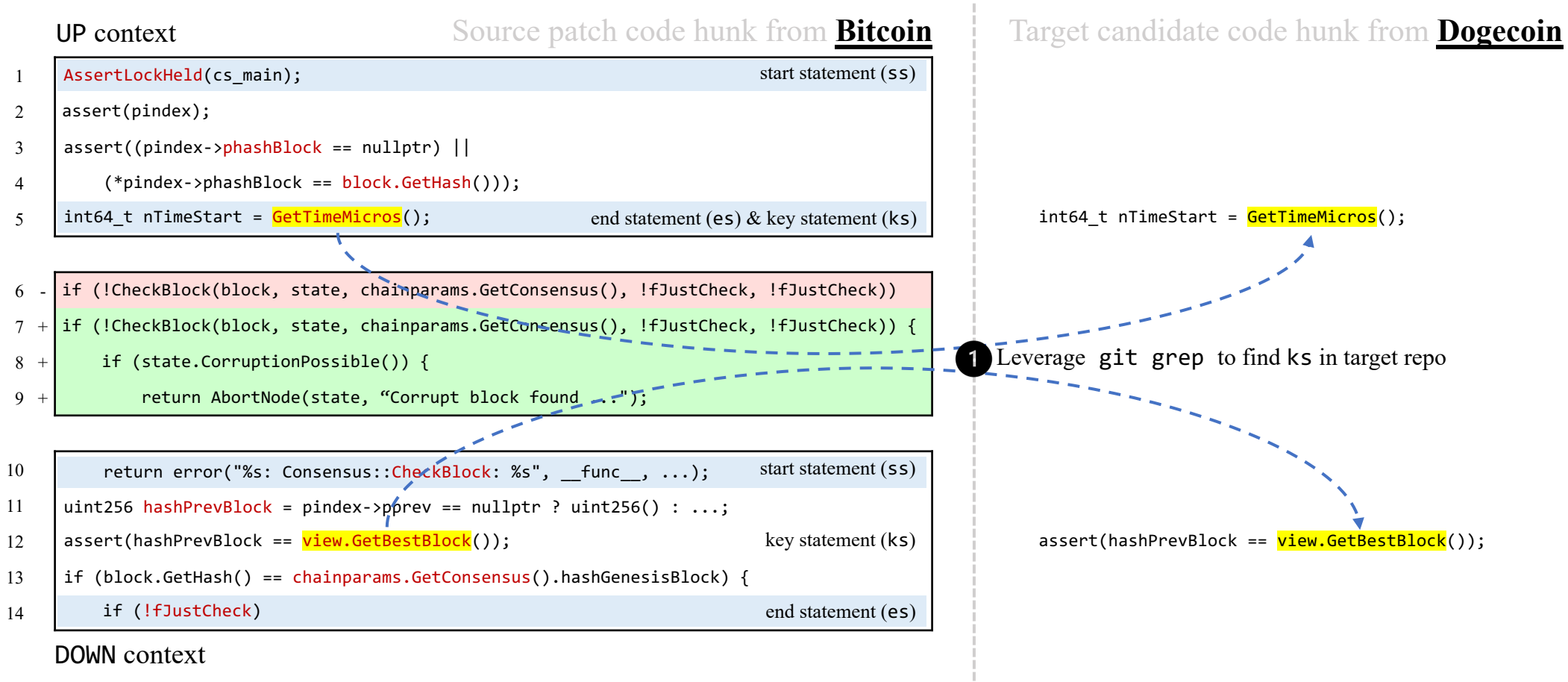
UP context — Source patch code hunk from **Bitcoin**

```
1   AssertLockHeld(cs_main);                                          start statement (ss)
2   assert(pindex);
3   assert((pindex->phashBlock == nullptr) ||
4       (*pindex->phashBlock == block.GetHash()));
5   int64_t nTimeStart = GetTimeMicros();                             end statement (es) & key statement (ks)

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

10      return error("%s: Consensus::CheckBlock: %s", __func__, ...);     start statement (ss)
11  uint256 hashPrevBlock = pindex->pprev == nullptr ? uint256() : ...;
12  assert(hashPrevBlock == view.GetBestBlock());                         key statement (ks)
13  if (block.GetHash() == chainparams.GetConsensus().hashGenesisBlock) {
14      if (!fJustCheck)                                                  end statement (es)
```

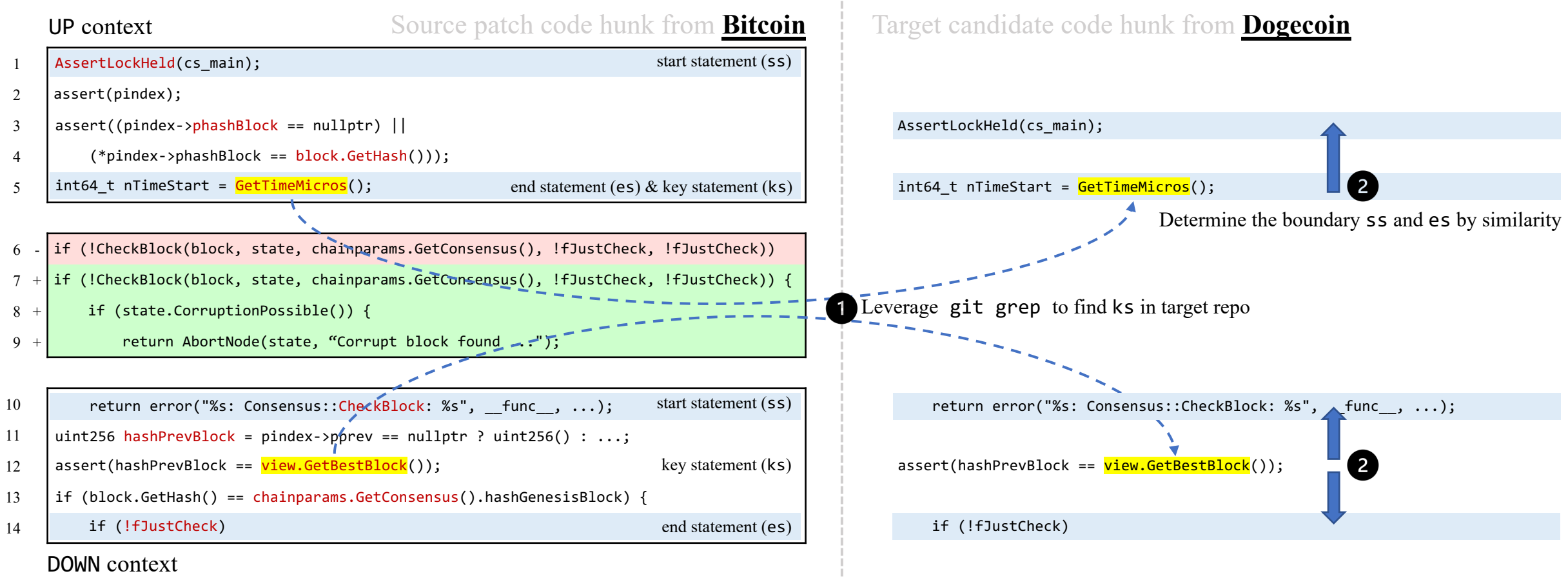
DOWN context

Target candidate code hunk from **Dogecoin**

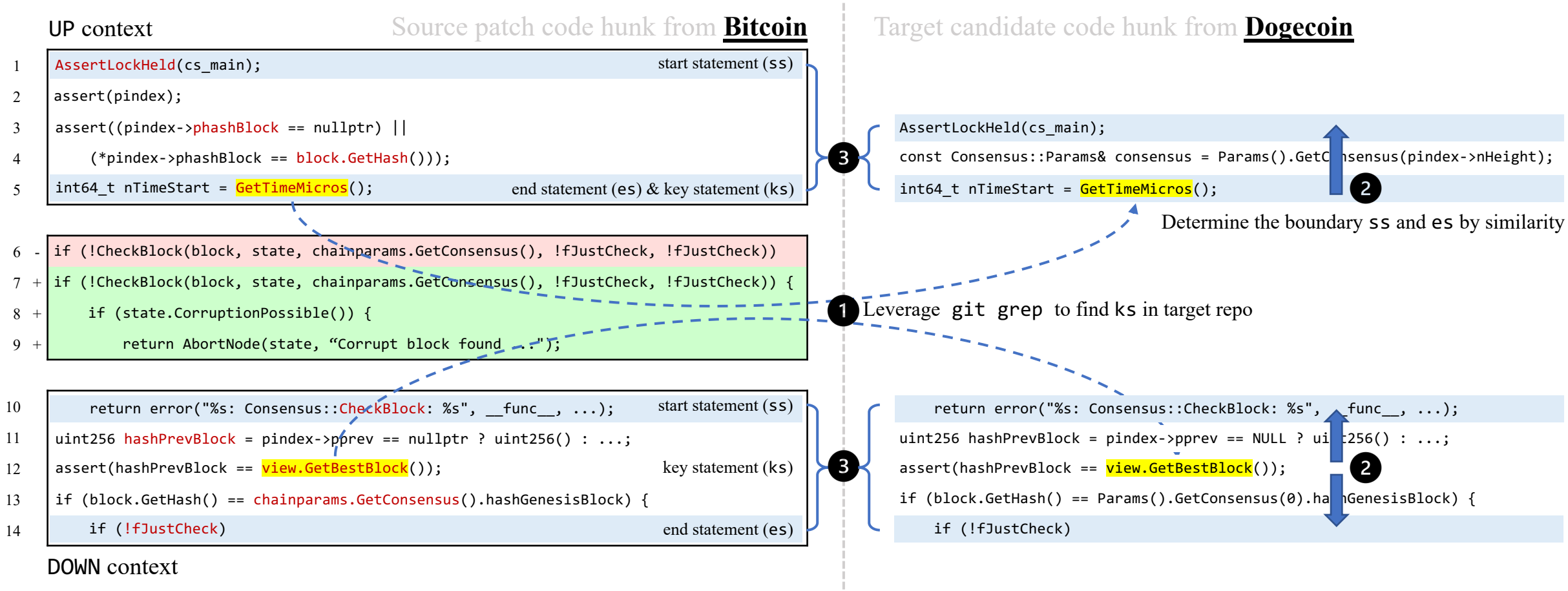
Context-based Candidate Clone Search



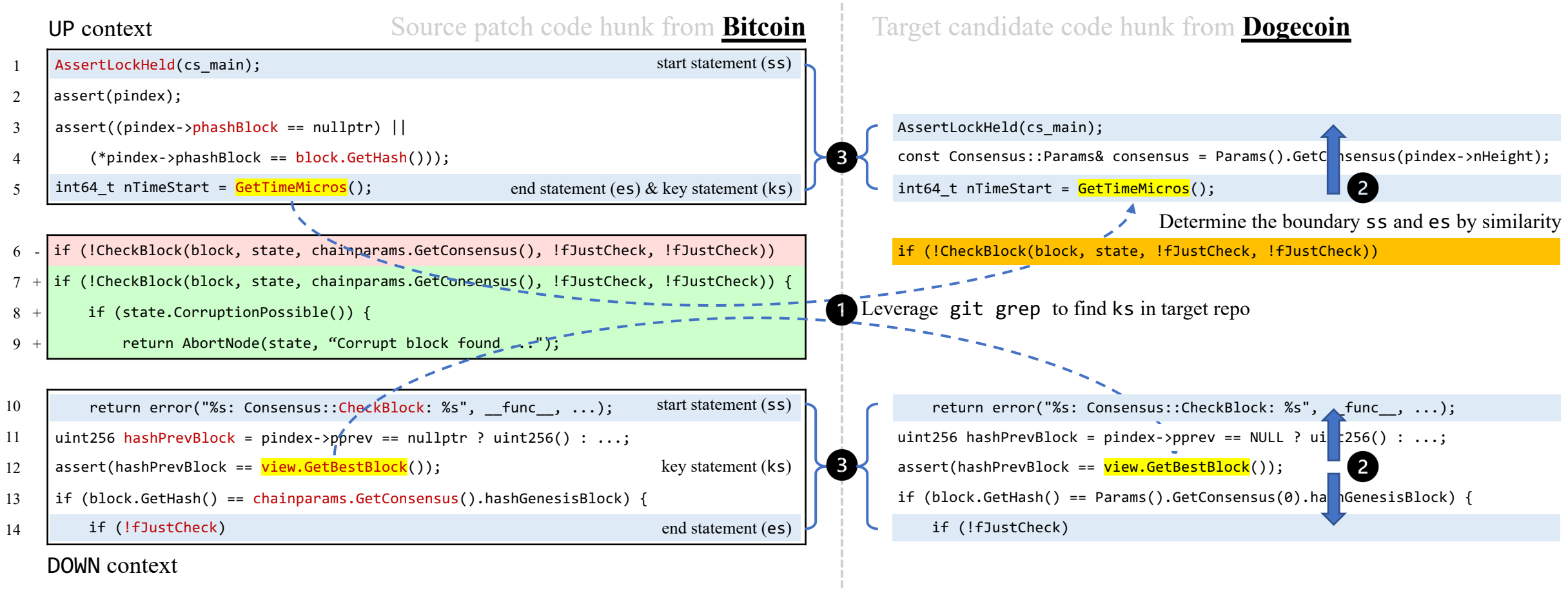
Context-based Candidate Clone Search



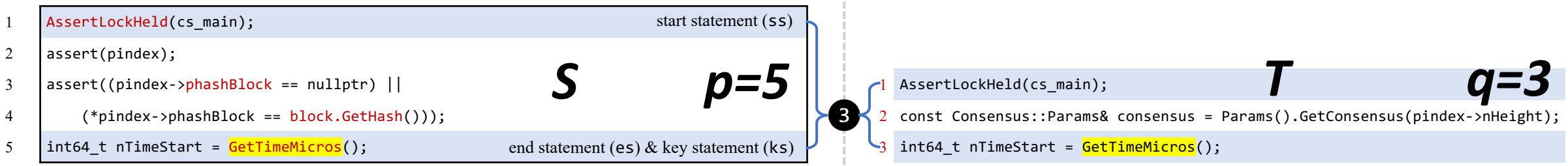
Context-based Candidate Clone Search



Context-based Candidate Clone Search

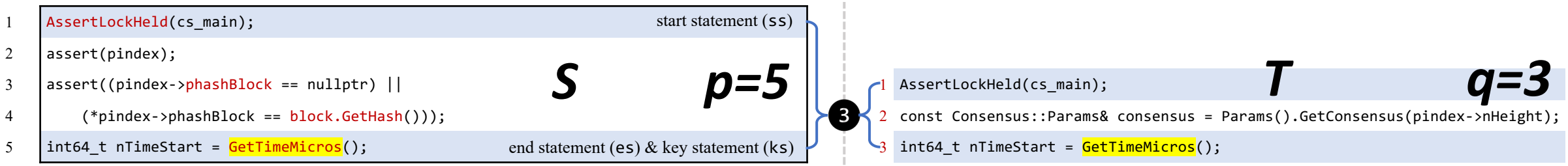


A New Way of Calculating Code Similarity



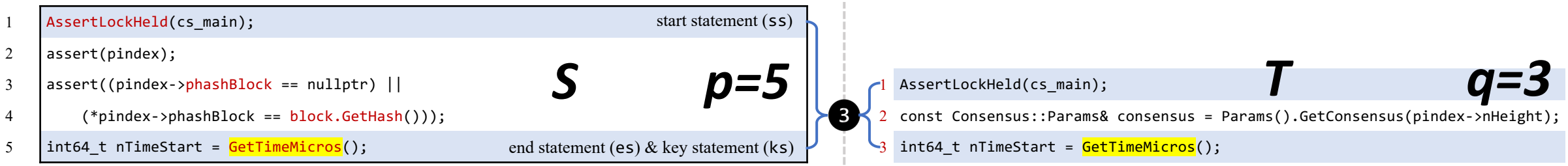
- Source code ***S*** with ***p*** statements and target code ***T*** with ***q*** statements.

A New Way of Calculating Code Similarity



- Source code ***S*** with ***p*** statements and target code ***T*** with ***q*** statements.
- 1. Pair-up each statement in *S* with the most similar statement in *T*, i.e.,
  - $\forall i \in [1, p]$, find j, s.t., $j = \underset{1 \le k \le q}{\text{argmax}} \text{strsim}(S_i, T_k)$.

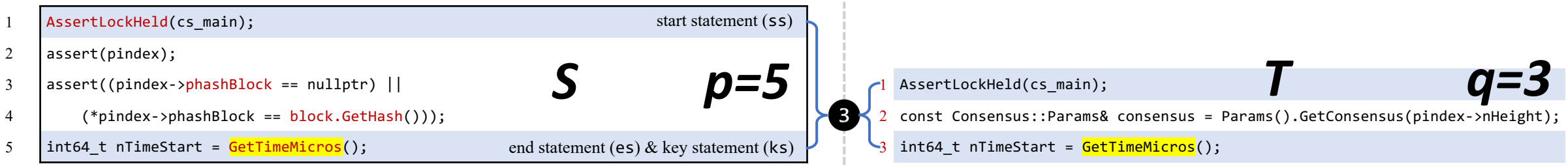
A New Way of Calculating Code Similarity



- Source code ***S*** with ***p*** statements and target code ***T*** with ***q*** statements.
- 1. Pair-up each statement in *S* with the most similar statement in *T*, i.e.,
  - $\forall i \in [1, p]$, find j, s.t., $j = \underset{1 \le k \le q}{\text{argmax}} \text{strsim}(S_i, T_k)$.

$p \neq q$ issue

A New Way of Calculating Code Similarity

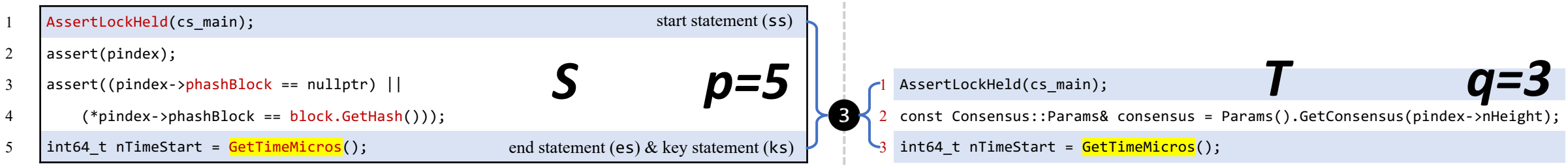


- Source code ***S*** with ***p*** statements and target code ***T*** with ***q*** statements.
- 1. Pair-up each statement in *S* with the most similar statement in *T*, i.e.,
  - $\forall i \in [1, p]$, find j, s.t., $j = \underset{1 \le k \le q}{\text{argmax}} \text{strsim}(S_i, T_k)$.

$p \neq q$ issue

- 2. Multiply $\text{strsim}(S_i, T_j)$ by a **reward factor** $r \in [0,1]$, i.e., $\text{strsim}(S_i, T_j) r^{|i-j|}$:
  - $r^{|i-j|}$ indicates: the greater $|i-j|$ the smaller the similarity between S_i and T_j.

A New Way of Calculating Code Similarity



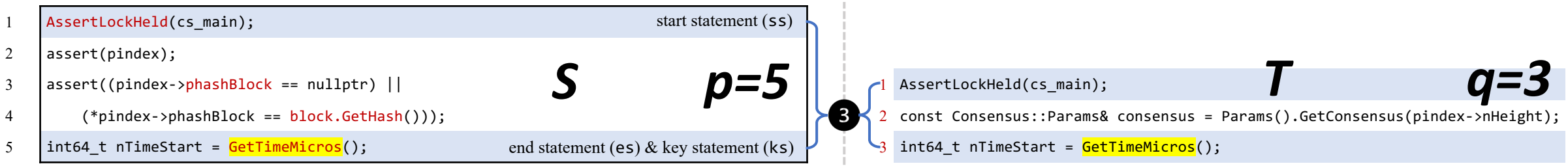
- Source code ***S*** with ***p*** statements and target code ***T*** with ***q*** statements.
- 1. Pair-up each statement in *S* with the most similar statement in *T*, i.e.,
  - $\forall i \in [1, p]$, find j, s.t., $j = \underset{1 \le k \le q}{\operatorname{argmax}} \operatorname{strsim}(S_i, T_k)$.

$p \neq q$ issue

- 2. Multiply $\operatorname{strsim}(S_i, T_j)$ by a **reward factor** $r \in [0,1]$, i.e., $\operatorname{strsim}(S_i, T_j) r^{|i-j|}$:
  - $r^{|i-j|}$ indicates: the greater $|i-j|$ the smaller the similarity between S_i and T_j.

code ordering issue

A New Way of Calculating Code Similarity



- Source code ***S*** with ***p*** statements and target code ***T*** with ***q*** statements.
- 1. Pair-up each statement in *S* with the most similar statement in *T*, i.e.,
  - $\forall i \in [1,p]$, find j, s.t., $j = \underset{1\le k\le q}{\operatorname{argmax}} \operatorname{strsim}(S_i, T_k)$.

$p \neq q$ issue

- 2. Multiply $\operatorname{strsim}(S_i, T_j)$ by a **reward factor** $r \in [0,1]$, i.e., $\operatorname{strsim}(S_i, T_j) r^{|i-j|}$:
  - $r^{|i-j|}$ indicates: the greater $|i-j|$ the smaller the similarity between S_i and T_j.

code ordering issue

- 3. Add up all the weighted similarities and normalize into [0,1], i.e.,
  - $\text{SIMILARITY}(S,T) = \frac{1}{p}\sum_{i=1}^{p} \operatorname{strsim}(S_i, T_j) r^{|i-j|}$.

BlockScope vs. State-of-the-art Tools

Patch-based code clone detection

ReDeBug [SP'12]

Hash tokenized contexts

Cannot detect Type-2

```
1  void foo() {
2      int x = intput();
3      if (x > MIN) {
4          int y = x * 10;
5          output(y);
6      }
7  }
```

Original Vuln Code

```
1  void foo() {
2    int x = intput();
3    if (x > minimum) {
4      int y = x * 10;
5      output(y);
6    }
7  }
```

Type-2 Clone

BlockScope vs. State-of-the-art Tools

Patch-based code clone detection

ReDeBug [SP'12] → VUDDY [SP'17]

Hash tokenized contexts

Cannot detect Type-2

Add variable abstraction

Cannot detect Type-3

```
1  void foo() {
2      int x = intput();
3      if (x > MIN) {
4          int y = x * 10;
5          output(y);
6      }
7  }
```

Original Vuln Code

```
1  void foo() {
2    int x = intput();
3    if (x > minimum) {
4      int y = x * 10;
5      output(y);
6    }
7  }
```

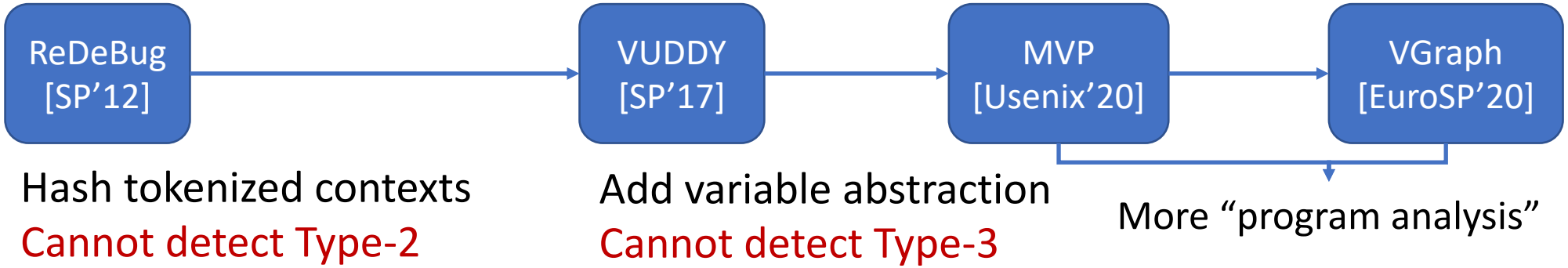
Type-2 Clone

```
1  void foo() {
2    int x = intput();
3    if (x > MIN) {
4      int z = x;
5      int y = x * 10;
6      output(y);
7    }
8  }
```

Type-3 Clone

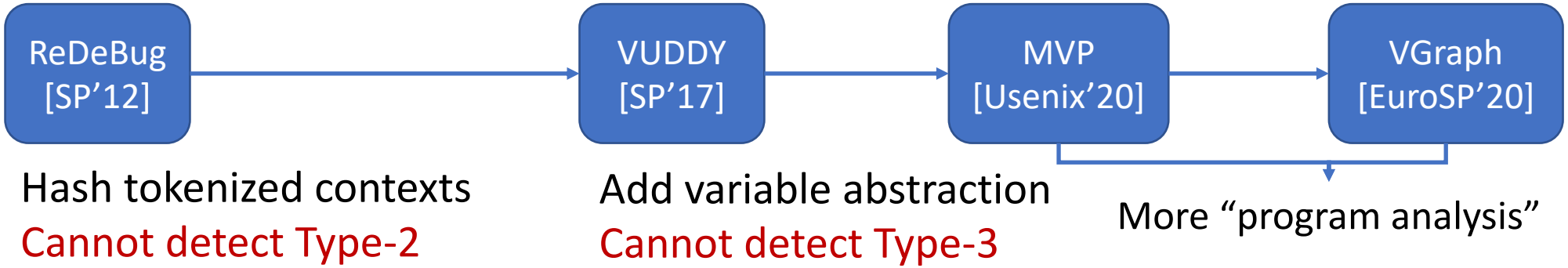
BlockScope vs. State-of-the-art Tools

Patch-based code clone detection



BlockScope vs. State-of-the-art Tools

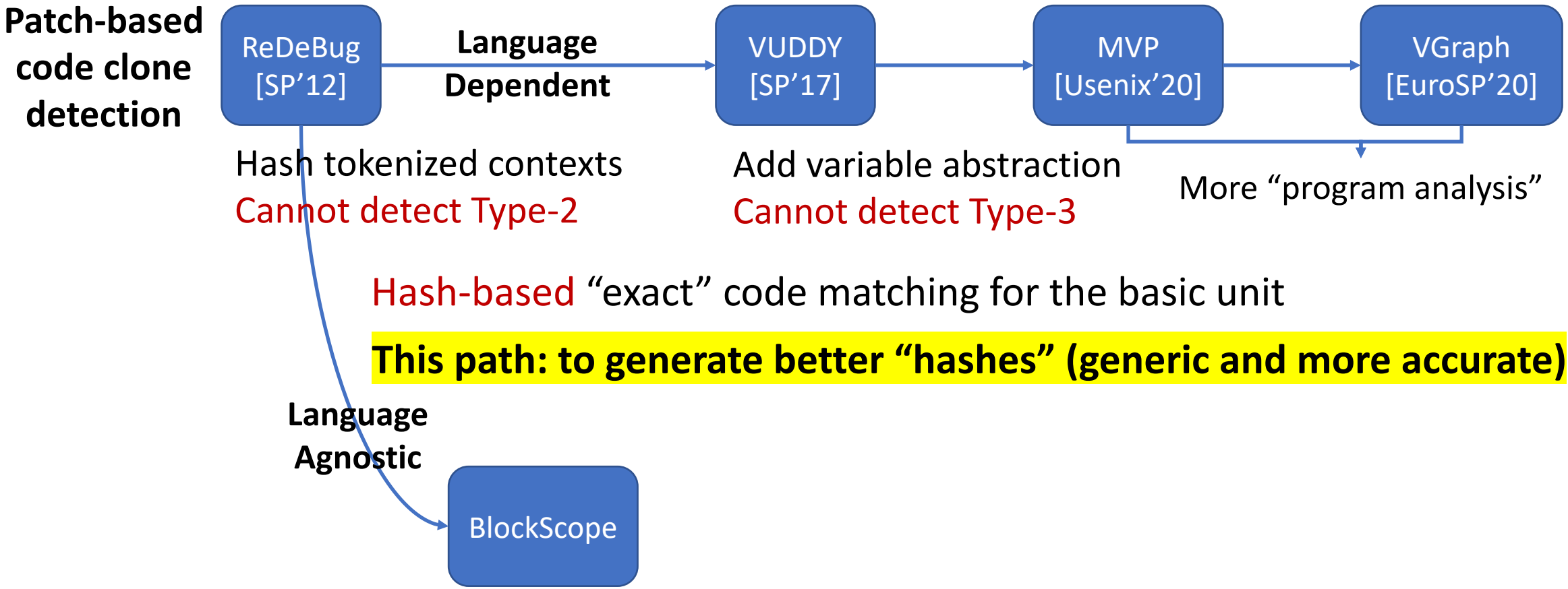
Patch-based code clone detection



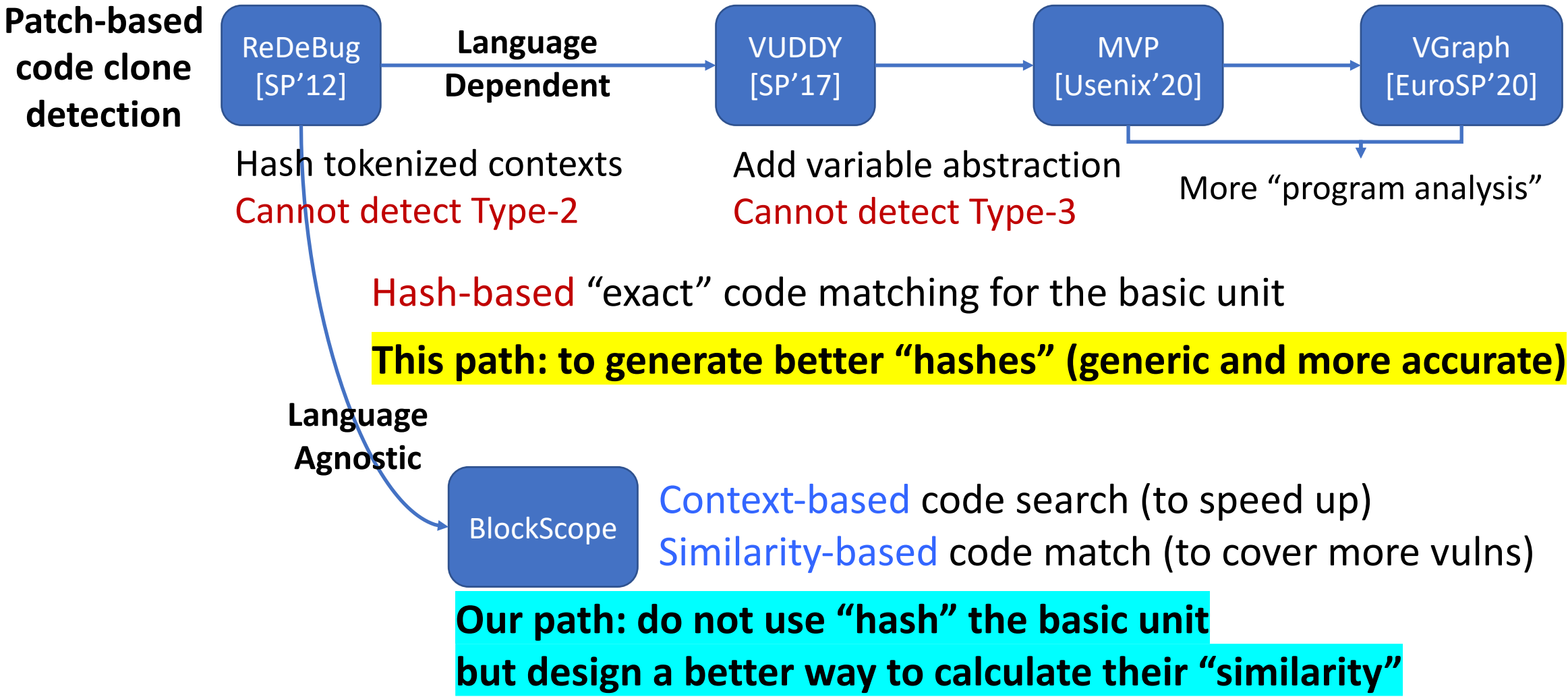
Hash-based "exact" code matching for the basic unit

This path: to generate better "hashes" (generic and more accurate)

BlockScope vs. State-of-the-art Tools



BlockScope vs. State-of-the-art Tools



Dataset

- Source/Target Code Repositories:

(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

- Security patches:
  - Within 5 years; cover different vulnerability types; applicable to most forked projects;
  - 32 from Bitcoin (including 4 CVEs);
  - 6 CVEs from Ethereum;

The Overall Accuracy and Performance

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)◇**	**53**	**51**	**245**	**3**	**189.6s (5.9s)◇**
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)◇**	**4**	**2**	**24**	**-**	**204.5s (34.1s)◇**

*: the numbers in (.) of these cells represent the average LOC per *project*.
◇: the numbers in (.) of these cells represent the average processing time per *patch*.

- Accuracy:
  - Precision: 91.8% vs. 95%
  - Recall: 91.8% vs. 51.8%

The Overall Accuracy and Performance

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
Litecoin	423.3K	6	-	26	-	8.3s	5	1	26	-	16
Bitcoin SV	221.1K	11	1	18	2	10.6s	2	10	19	1	9.
Dash	380.3K	9	1	22	-	13.9s	7	3	21	1	17
Zcash	199.4K	9	2	19	2	8.4s	1	10	21	-	10
Bitcoin Gold	381.7K	10	1	21	-	8.8s	10	1	21	-	17
Horizen	178.9K	9	2	20	1	7.7s	1	10	21	-	12
Qtum	569.0K	-	-	31	1	12.0s	-	-	32	-	33
DigiByte	416.3K	10	1	21	-	10.7s	10	1	21	-	15
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)◇**	**53**	**51**	**245**	**3**	**189.6s (5.9s)◇**
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)◇**	**4**	**2**	**24**	**-**	**204.5s (34.1s)◇**

*: the numbers in (.) of these cells represent the average LOC per *project*.
◇: the numbers in (.) of these cells represent the average processing time per *patch*.

- Accuracy:

ReDeBug has less FPs, but too many FNs

The Overall Accuracy and Performance

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
Litecoin	423.3K	6	-	26	-	8.3s	5	1	26	-	16
Bitcoin SV	221.1K	11	1	18	2	10.6s	2	10	19	1	9.
Dash	380.3K	9	1	22	-	13.9s	7	3	21	1	17
Zcash	199.4K	9	2	19	2	8.4s	1	10	21	-	10
Bitcoin Gold	381.7K	10	1	21	-	8.8s	10	1	21	-	17
Horizen	178.9K	9	2	20	1	7.7s	1	10	21	-	12
Qtum	569.0K	-	-	31	1	12.0s	-	-	32	-	33
DigiByte	416.3K	10	1	21	-	10.7s	10	1	21	-	15
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)◇**	**53**	**51**	**245**	**3**	**189.6s (5.9s)◇**
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)◇**	**4**	**2**	**24**	**-**	**204.5s (34.1s)◇**

*: the numbers in (.) of these cells represent the average LOC per *project*.
◇: the numbers in (.) of these cells represent the average processing time per *patch*.

- Accuracy:

ReDeBug has less FPs, but too many FNs

- Performance:
  - Bitcoin: 109.9s vs. 189.6s
  - Ethereum: 13.3s vs. 204.5s

The Overall Accuracy and Performance

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
Litecoin	423.3K	6	-	26	-	8.3s	5	1	26	-	16
Bitcoin SV	221.1K	11	1	18	2	10.6s	2	10	19	1	9.
Dash	380.3K	9	1	22	-	13.9s	7	3	21	1	17
Zcash	199.4K	9	2	19	2	8.4s	1	10	21	-	10
Bitcoin Gold	381.7K	10	1	21	-	8.8s	10	1	21	-	17
Horizen	178.9K	9	2	20	1	7.7s	1	10	21	-	12
Qtum	569.0K	-	-	31	1	12.0s	-	-	32	-	33
DigiByte	416.3K	10	1	21	-	10.7s	10	1	21	-	15
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)◇**	**53**	**51**	**245**	**3**	**189.6s (5.9s)◇**
Binance	565.3K	1	-	5	-	2.2s	-	1	5	-	30
Avalanche	1070.1K	-	-	6	-	2.5s	-	-	6	-	55
Polygon	592.0K	-	-	6	-	2.3s	-	-	6	-	31
Celo	631.0K	1	-	5	-	2.7s	1	-	5	-	44
Optimism	630.6K	4	-	2	-	3.6s	3	1	2	-	43
Sum	**3.5M (697.8K)***	**6**	**-**	**24**	**-**	**13.3s (2.2s)◇**	**4**	**2**	**24**	**-**	**20 (34.**

*: the numbers in (.) of these cells represent the average LOC per *project*.
◇: the numbers in (.) of these cells represent the average processing time per *patch*.

- Accuracy:

ReDeBug has less FPs, but too many FNs

- Performance:

LOC significantly effects ReDeBug's performance

The Breakdown for Three Clone Types

- **Type-1&3** clones occupy **95.5%** of all the cases.

- BlockScope accuracy:
  - Type-1: 100%;
  - Type-2: 80%;
  - Type-3: 85.7%.
- ReDeBug accuracy:
  - Type-1: 85.7%;
  - Type-2: 0%;
  - Type-3: 26.8%.

Forked Project	Type-1		Type-2		Type-3		Sum	
	T	B;R	T	B;R	T	B;R	T	B;R
Dogecoin	6	(6;4)	-	-	10	(10;3)	16	(16;7)
Bitcoin Cash	1	(1;-)	-	-	-	-	1	(1;-)
Litecoin	5	(5;5)	-	-	1	(1;-)	6	(6;5)
Bitcoin SV	1	(1;-)	-	-	11	(10;2)	12	(11;2)
Dash	7	(7;7)	-	-	3	(2;-)	10	(9;7)
Zcash	1	(1;-)	2	(1;-)	8	(7;1)	11	(9;1)
Bitcoin Gold	9	(9;8)	-	-	2	(1;2)	11	(10;10)
Horizen	-	-	2	(2;-)	9	(7;1)	11	(9;1)
Qtum	-	-	-	-	-	-	-	-
DigiByte	7	(7;7)	1	(1;-)	3	(2;3)	11	(10;10)
Ravencoin	7	(7;7)	-	-	8	(7;3)	15	(14;10)
Sum	**44**	**(44;38)**	**5**	**(4;-)**	**55**	**(47;15)**	**104**	**(95;53)**
Binance	-	-	-	-	1	(1;-)	1	(1;-)
Avalanche	-	-	-	-	-	-	-	-
Polygon	-	-	-	-	-	-	-	-
Celo	1	(1;1)	-	-	-	-	1	(1;1)
Optimism	4	(4;3)	-	-	-	-	4	(4;3)
Sum	**5**	**(5;4)**	**-**	**-**	**1**	**(1;-)**	**6**	**(6;4)**

T, B, and R represent: the total number of vulnerabilities of each clone type, the number of vulnerabilities detected by BlockScope, and the number of vulnerabilities detected by ReDeBug, respectively.

Vulnerability Report Response

- Reported 110 vulnerabilities (101 TP + 9 FN);
  - 74 positive response;
  - CVE-2021-37491 of Dogecoin & CVE-2021-37492 of Ravencoin
  - 1 bug bounty from Binance;

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**

Vulnerability Report Response

- Reported 110 vulnerabilities (101 TP + 9 FN);
  - 74 positive response;
  - CVE-2021-37491 of Dogecoin & CVE-2021-37492 of Ravencoin
  - 1 bug bounty from Binance;
  - Dogecoin, Ravencoin, Dash, Bitcoin Gold, Litecoin, and Binance are the most active ones;

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**

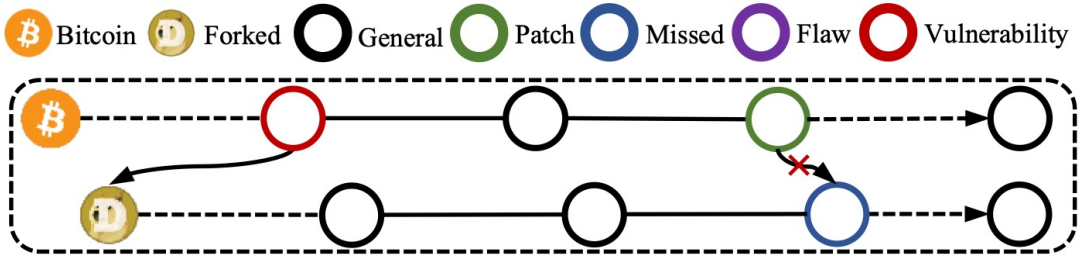
Vulnerability Report Response

- Reported 110 vulnerabilities (101 TP + 9 FN);
  - 74 positive response;
  - CVE-2021-37491 of Dogecoin & CVE-2021-37492 of Ravencoin
  - 1 bug bounty from Binance;
  - Dogecoin, Ravencoin, Dash, Bitcoin Gold, Litecoin, and Binance are the most active ones;
  - Bitcoin Cash, DigiByte, and Optimism did not respond to any of our 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**

How do vulnerabilities propagate to the forked projects?

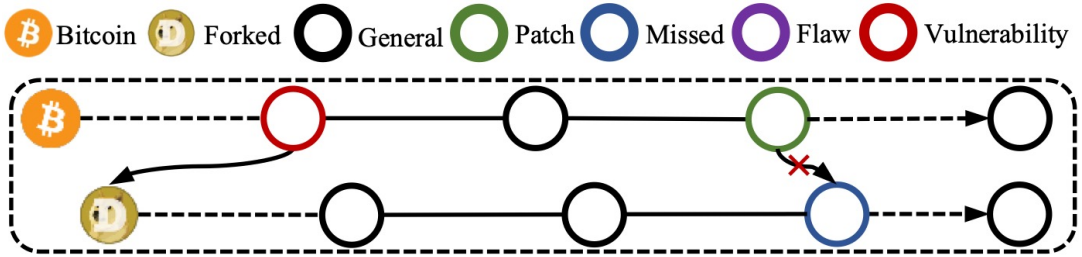
Investigation of Propagated Vulnerabilities



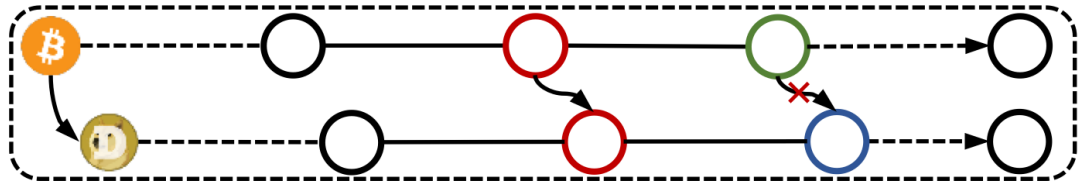
(a) The `fork` type: vulnerabilities directly forked in the beginning.

- 41 cases, e.g., CVE-2022-29177, CVE-2021-41173.

Investigation of Propagated Vulnerabilities



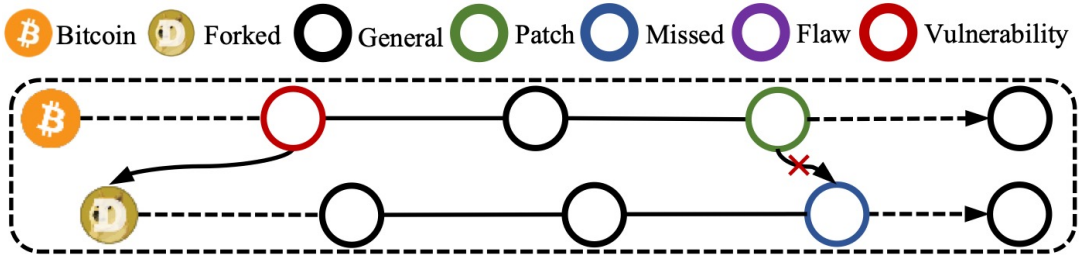
(a) The `fork` type: vulnerabilities directly forked in the beginning.



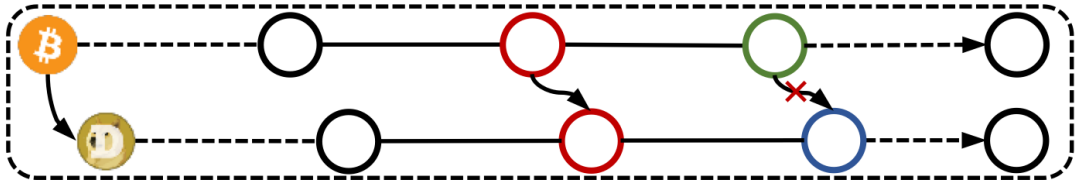
(b) The `fetch` type: vulnerabilities fetched from vulnerable commits.

- 41 cases, e.g., CVE-2022-29177, CVE-2021-41173.
- 25 cases, e.g., CVE-2021-3401, CVE-2020-26265, CVE-2020-26264, CVE-2020-26260.

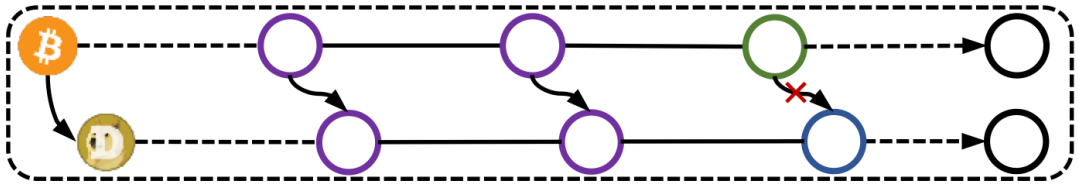
Investigation of Propagated Vulnerabilities



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



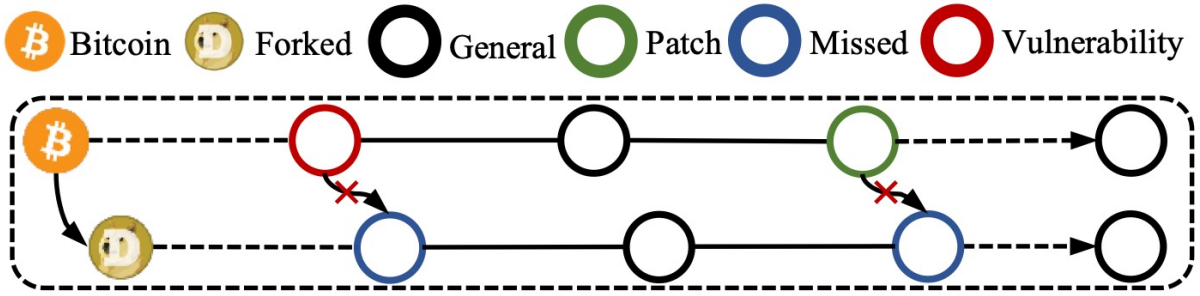
(b) The `fetch` type: vulnerabilities fetched from vulnerable commits.



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

- 41 cases, e.g., CVE-2022-29177, CVE-2021-41173.
- 25 cases, e.g., CVE-2021-3401, CVE-2020-26265, CVE-2020-26264, CVE-2020-26260.
- 44 cases, e.g., Bitcoin PR#16512.

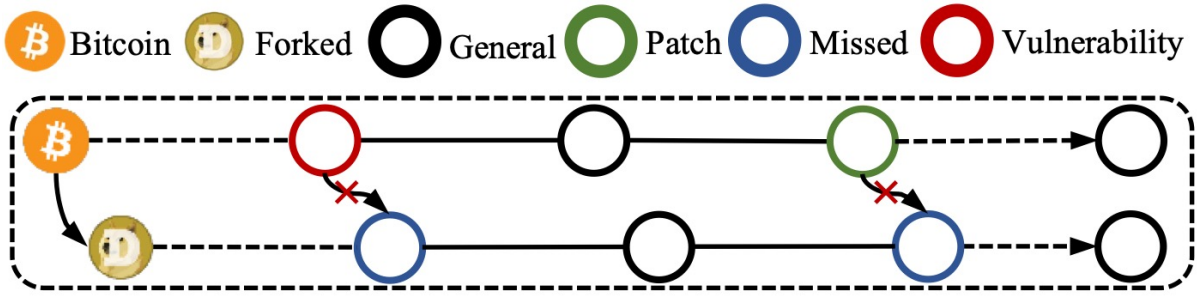
Our Limitation



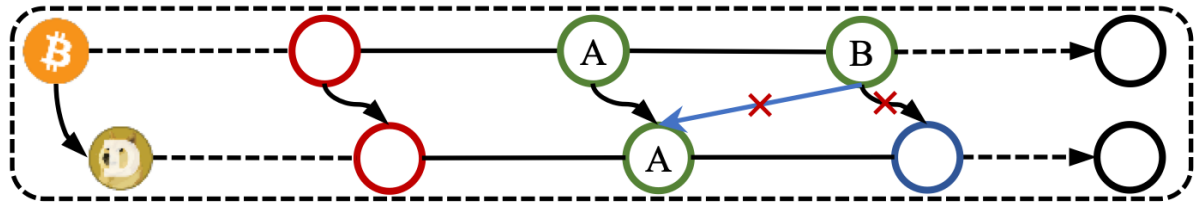
(a) FP-I: no clone, and thus no vulnerability.

- FP-I: 7 cases, e.g., CVE-2018-17145, CVE-2019-15947, Bitcoin PR#12561, Bitcoin PR#14249.

Our Limitation



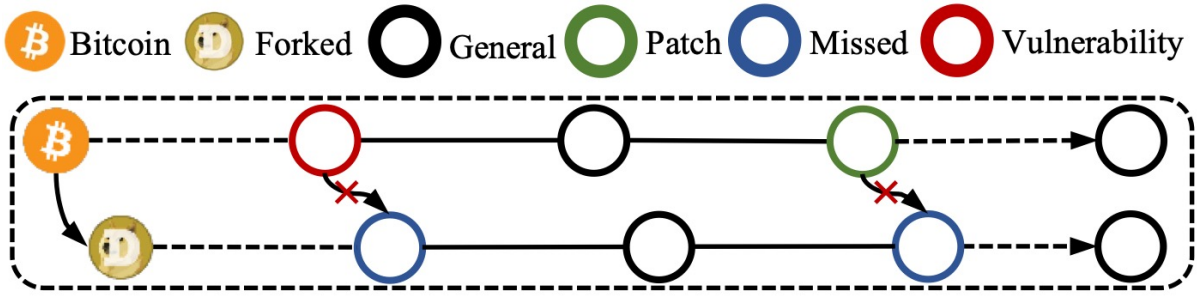
(a) FP-I: no clone, and thus no vulnerability.



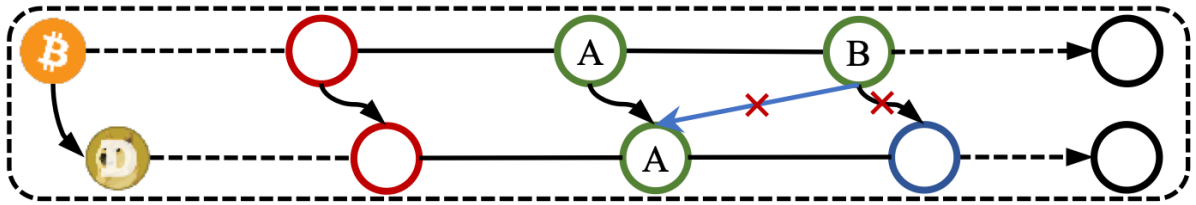
(b) FP-II: patch outdated.

- FP-I: 7 cases, e.g., CVE-2018-17145, CVE-2019-15947, Bitcoin PR#12561, Bitcoin PR#14249.
- FP-II: 2 cases, e.g., Bitcoin PR#12561, Bitcoin PR#13808.

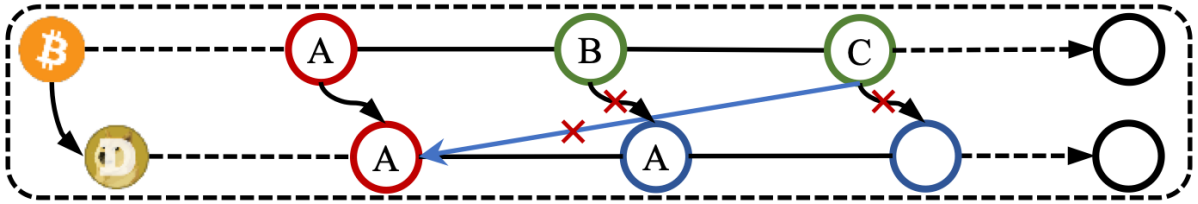
Our Limitation



(a) FP-I: no clone, and thus no vulnerability.



(b) FP-II: patch outdated.



(c) FN: target code outdated.

- FP-I: 7 cases, e.g., CVE-2018-17145, CVE-2019-15947, Bitcoin PR#12561, Bitcoin PR#14249.
- FP-II: 2 cases, e.g., Bitcoin PR#12561, Bitcoin PR#13808.
- FN: 9 cases, e.g., Bitcoin PR#10345, Bitcoin PR#11568, Bitcoin PR#13907.

Department of Information Engineering
The Chinese University of Hong Kong

How long does it take for the forked projects to fix the propagated vulnerabilities?

Determining Fixed Cases' Delay

- Interval between the **patch's commit date** in the source project and the **patch's release date** in the target project.

Forked Project	# Fixed Cases		
	Detected	Truth	Err*
Dogecoin	1	1	-
Bitcoin Cash	23	25	(2;-)
Litecoin	22	22	-
Bitcoin SV	1	1	-
Dash	11	10	(-;1)
Zcash	2	1	(-;1)
Bitcoin Gold	14	14	-
Horizen	1	-	(-;1)
Qtum	28	28	(1;1)
DigiByte	14	14	-
Ravencoin	3	3	-
Sum	**120**	**119**	**(3;4)**
Binance	5	5	-
Avalanche	3	3	-
Polygon	6	6	-
Celo	4	4	-
Optimism	1	1	-
Sum	**19**	**19**	**-**

* represents (the number of missed cases; the number of mistake cases).

Determining Fixed Cases' Delay

- Interval between the **patch's commit date** in the source project and the **patch's release date** in the target project.
- Find the commits that added the patch by **`git blame`**:

Forked Project	# Fixed Cases		
	Detected	Truth	Err*
Dogecoin	1	1	-
Bitcoin Cash	23	25	(2;-)
Litecoin	22	22	-
Bitcoin SV	1	1	-
Dash	11	10	(-;1)
Zcash	2	1	(-;1)
Bitcoin Gold	14	14	-
Horizen	1	-	(-;1)
Qtum	28	28	(1;1)
DigiByte	14	14	-
Ravencoin	3	3	-
Sum	**120**	**119**	**(3;4)**
Binance	5	5	-
Avalanche	3	3	-
Polygon	6	6	-
Celo	4	4	-
Optimism	1	1	-
Sum	**19**	**19**	**-**

* represents (the number of missed cases; the number of mistake cases).

src/qt/bitcoin.cpp Example of the output of **`git blame`**.

```
202d853b  201      }
202d853b  202  }
202d853b  203
a2714a5c  204  static int qt_argc = 1;
797fef7b  205  static const char* qt_argv = "qtum-qt";
a2714a5c  206
a2714a5c  207  BitcoinApplication::BitcoinApplication(...):
a2714a5c  208      QApplication(qt_argc, const_cast<char **>(...)),
9096276e  209      coreThread(nullptr),
71e0d908  210      m_node(node),
9096276e  211      optionsModel(nullptr),
```

Determining Fixed Cases' Delay

- Interval between the **patch's commit date** in the source project and the **patch's release date** in the target project.
- Find the commits that added the patch by **`git blame`**:
  - Added by two commits: a2714a5c & 797fef7b;
  - a2714a5c is earlier, thus determined as the "true" commit.

Forked Project	# Fixed Cases		
	Detected	Truth	Err*
Dogecoin	1	1	-
Bitcoin Cash	23	25	(2;-)
Litecoin	22	22	-
Bitcoin SV	1	1	-
Dash	11	10	(-;1)
Zcash	2	1	(-;1)
Bitcoin Gold	14	14	-
Horizen	1	-	(-;1)
Qtum	28	28	(1;1)
DigiByte	14	14	-
Ravencoin	3	3	-
Sum	**120**	**119**	**(3;4)**
Binance	5	5	-
Avalanche	3	3	-
Polygon	6	6	-
Celo	4	4	-
Optimism	1	1	-
Sum	**19**	**19**	**-**

* represents (the number of missed cases; the number of mistake cases).

src/qt/bitcoin.cpp Example of the output of **`git blame`**.

```
202d853b  201      }
202d853b  202  }
202d853b  203
a2714a5c  204  static int qt_argc = 1;
797fef7b  205  static const char* qt_argv = "qtum-qt";
a2714a5c  206
a2714a5c  207  BitcoinApplication::BitcoinApplication(...):
a2714a5c  208      QApplication(qt_argc, const_cast<char **>(...)),
9096276e  209      coreThread(nullptr),
71e0d908  210      m_node(node),
9096276e  211      optionsModel(nullptr),
```

Determining Fixed Cases' Delay

- Interval between the **patch's commit date** in the source project and the **patch's release date** in the target project.
- Find the commits that added the patch by **`git blame`**:
  - Added by two commits: a2714a5c & 797fef7b;
  - a2714a5c is earlier, thus determined as the "true" commit.
- Crawl the commit's GitHub page to find its release date.

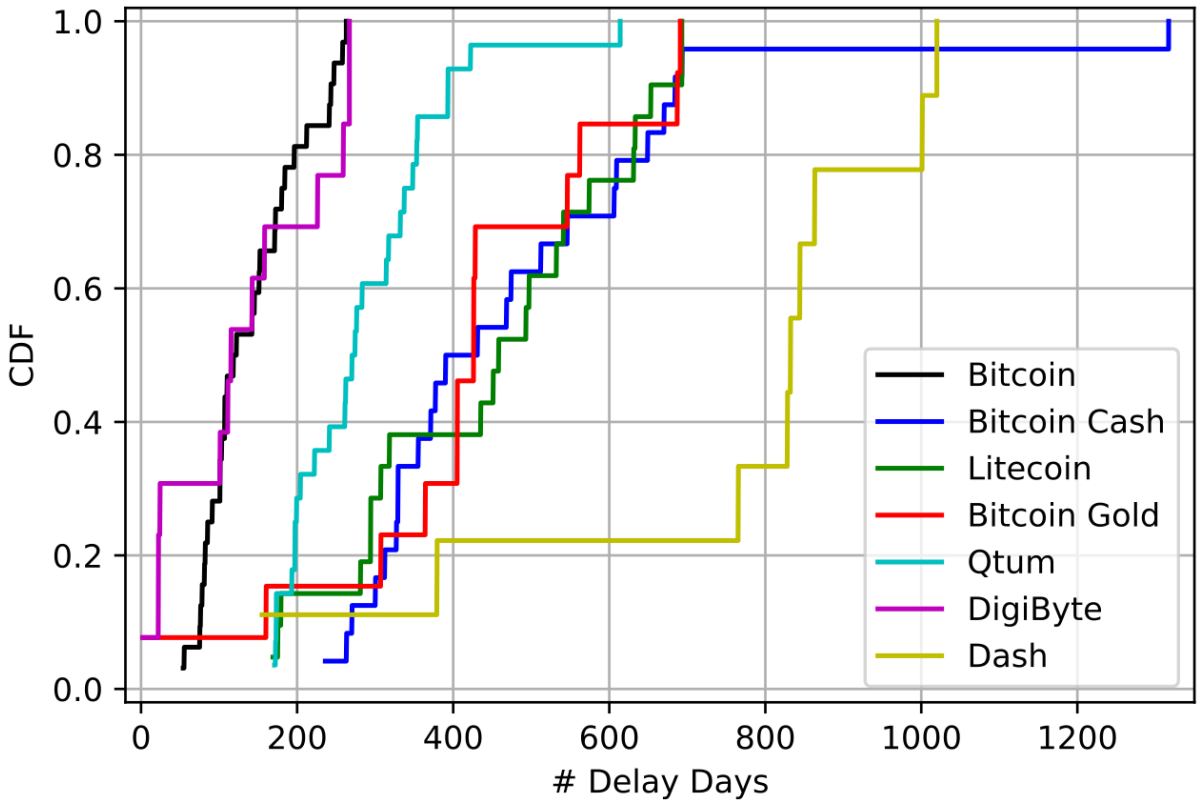
Forked Project	# Fixed Cases		
	Detected	Truth	Err*
Dogecoin	1	1	-
Bitcoin Cash	23	25	(2;-)
Litecoin	22	22	-
Bitcoin SV	1	1	-
Dash	11	10	(-;1)
Zcash	2	1	(-;1)
Bitcoin Gold	14	14	-
Horizen	1	-	(-;1)
Qtum	28	28	(1;1)
DigiByte	14	14	-
Ravencoin	3	3	-
Sum	**120**	**119**	**(3;4)**
Binance	5	5	-
Avalanche	3	3	-
Polygon	6	6	-
Celo	4	4	-
Optimism	1	1	-
Sum	**19**	**19**	**-**

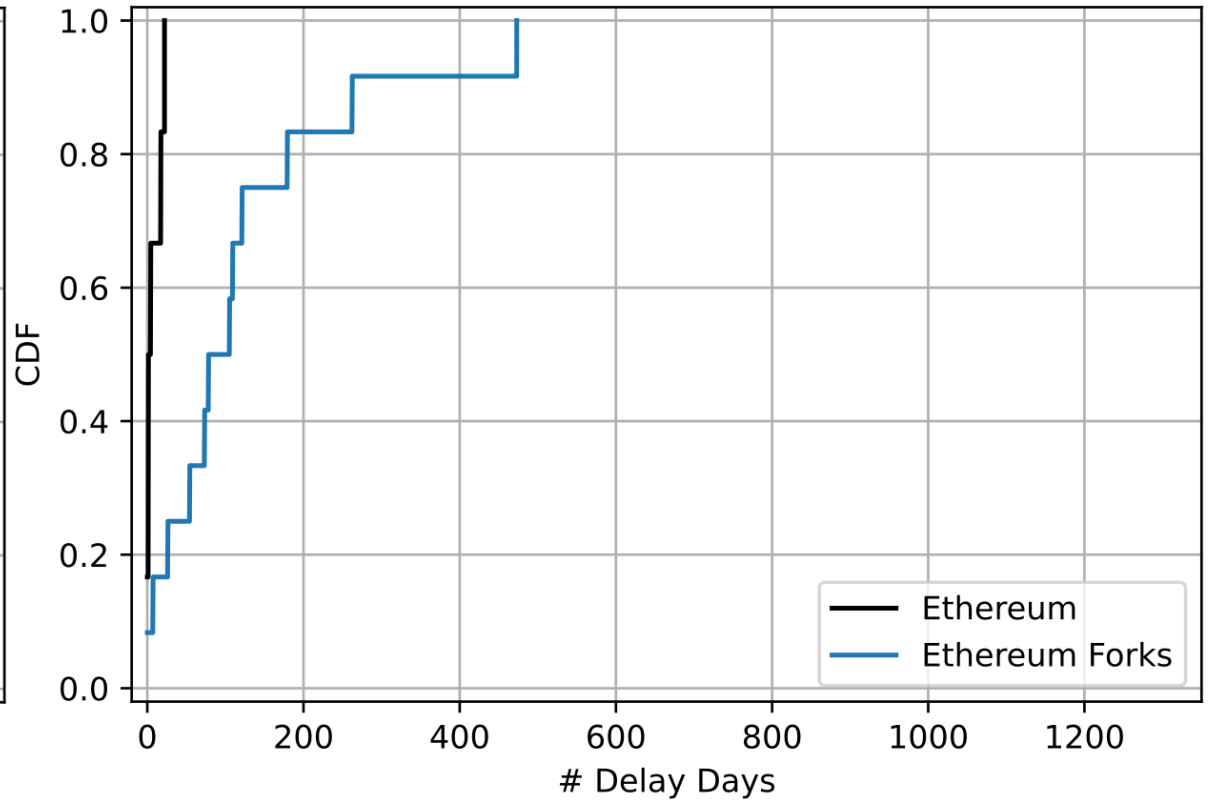
* represents (the number of missed cases; the number of mistake cases).

src/qt/bitcoin.cpp Example of the output of **`git blame`**.

```
202d853b  201      }
202d853b  202  }
202d853b  203
a2714a5c  204  static int qt_argc = 1;
797fef7b  205  static const char* qt_argv = "qtum-qt";
a2714a5c  206
a2714a5c  207  BitcoinApplication::BitcoinApplication(...):
a2714a5c  208      QApplication(qt_argc, const_cast<char **>(...)),
9096276e  209      coreThread(nullptr),
71e0d908  210      m_node(node),
9096276e  211      optionsModel(nullptr),
```

Patch Delay Analysis





- Only DigiByte can catch up with Bitcoin's schedule.
- Dash is particularly slow.
- Ethereum's forks generally perform better than Bitcoin's forks.

Thank You!

- **BlockScope**: For the effective and efficient detection of multiple types of cloned vulnerabilities.

- Detected **101 true vulnerabilities** in 16 Bitcoin and Ethereum forked projects; 2 new CVEs of Dogecoin and Ravencoin; a bug bounty from Binance.

- Conducted a **deep investigation** on vulnerability propagation and patching processes.

Questions?