VETSC: Towards Automated Safety Vetting of Smart Contracts in Decentralized Applications

Yue Duan, Xin Zhao, Yu Pan, Shucheng Li,
Minghao Li, Fengyuan Xu, Mu Zhang

ACM CCS 2022

Smart Contract

Physical contracts

Trusted 3rd party

Slow speed

High cost

Smart Contract

- Spontaneous
- Low cost
- High speed
- Trusted

Everything comes with a price!

Security and Safety Issues

- Syntactic-level security issues
  - e.g., reentrancy, mishandled exceptions
- vel safety issues
- lenging to detect
- ed s mantic informatio abo d variables

Owner can cancel at ANYTIME

terminate a bidding process

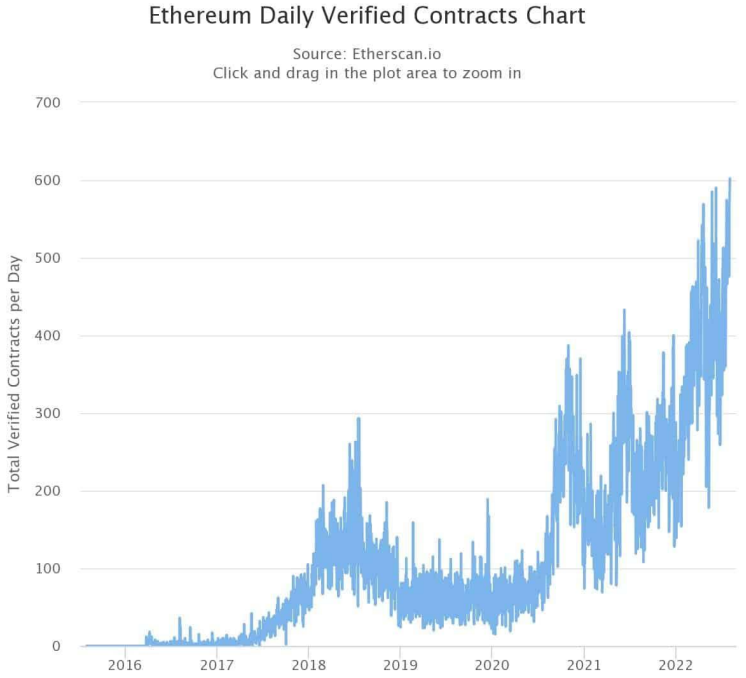
prior accepted bids

auction state

```
31 function cancelAuction(uint _i
32     Auction memory myAuction = auctions[_id];
33     uint bidsLen = accepted[_id]
34
35     // refund the last bid, if prior bids exis
36     if(bidsLen > 0) {
37         Bid memory lastBid = accepted[_id][bidsLen - 1];
38         if(!lastBid.from.send(lastBid.amount))
39     }
40     auctions[_id].active = false;
41     emit AuctionCanceled(msg.sender, _id);
42 }
```

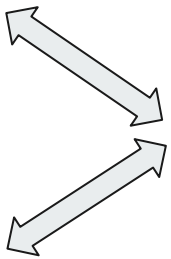
Security and Safety Issues

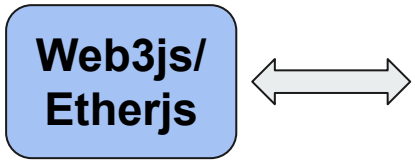
- Importance
  - growing fast
  - huge value locked in them

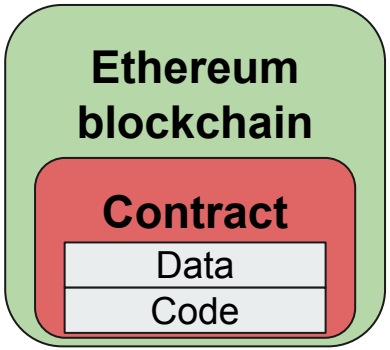


Decentralized Applications

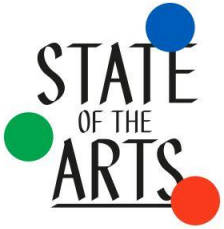
GUI Front-end **Middleware** **Backend**



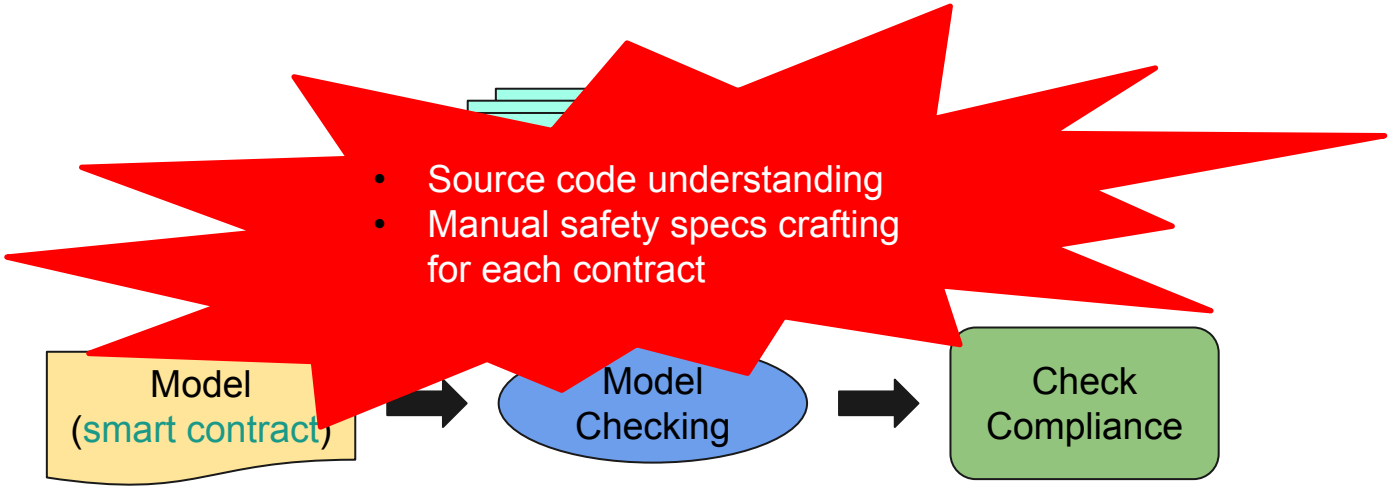




Semantic Level Issues



- Zeus [NDSS'18]
- VerX [Oakland'20]
- SmartPulse [Oakland'21]



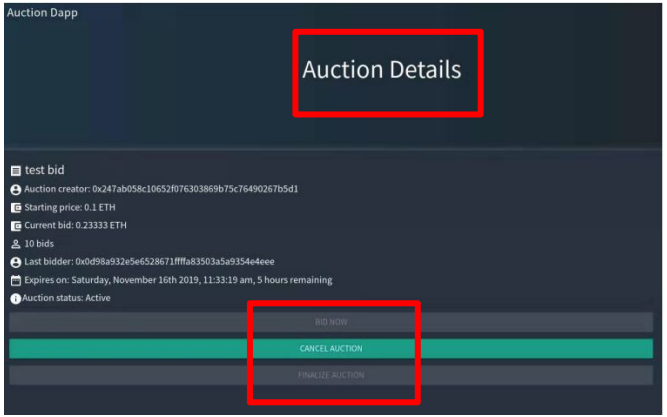
Semantic Level Issues

- **unreliable and maybe unavailable**
- **tedious manual work and error-prone**
- **do not consider DApp scenario**

Insights

- Front-end UI contains semantic information

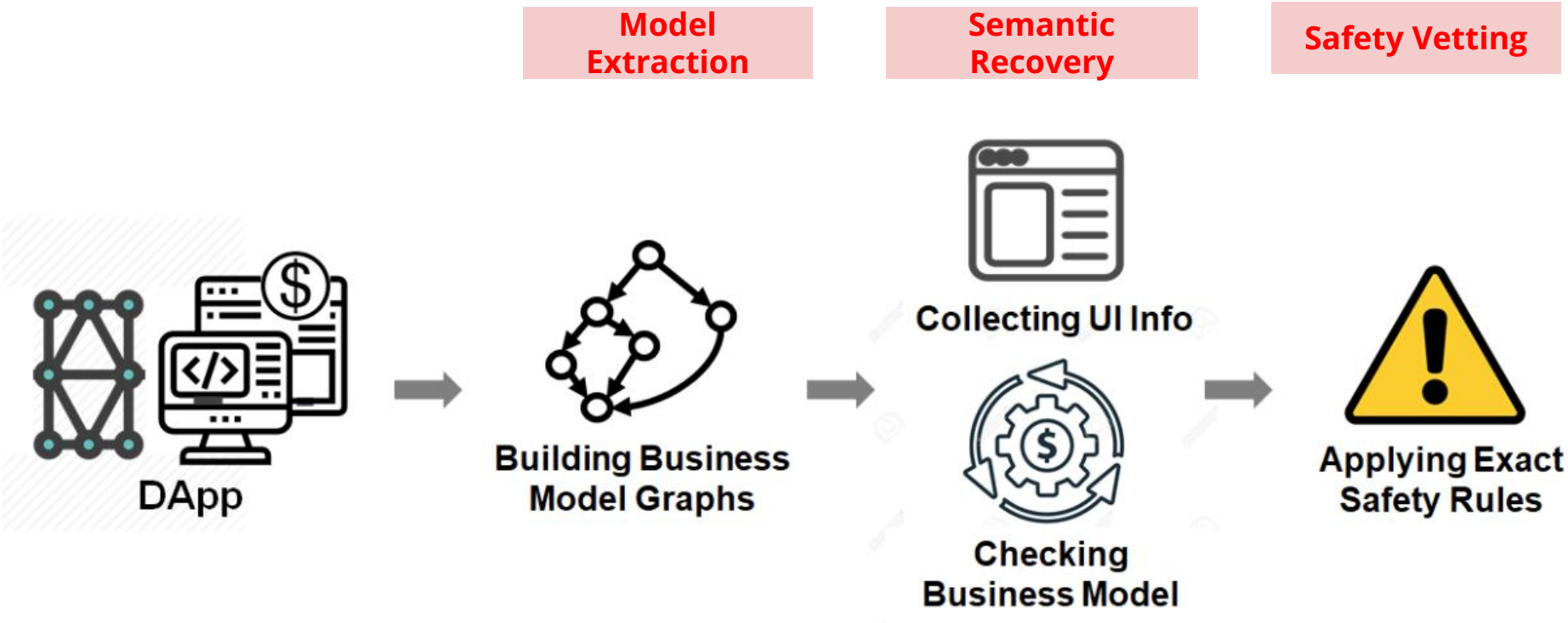
  => can help with code understanding



- Limited categories of business models

  => automate safety spec crafting

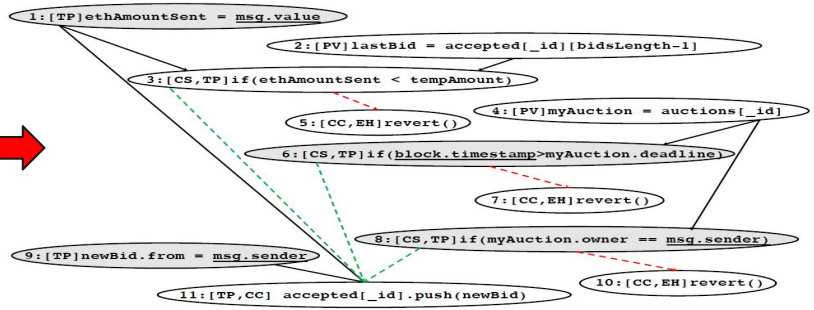
Our Solution: VetSC [CCS'22]



Model Extraction

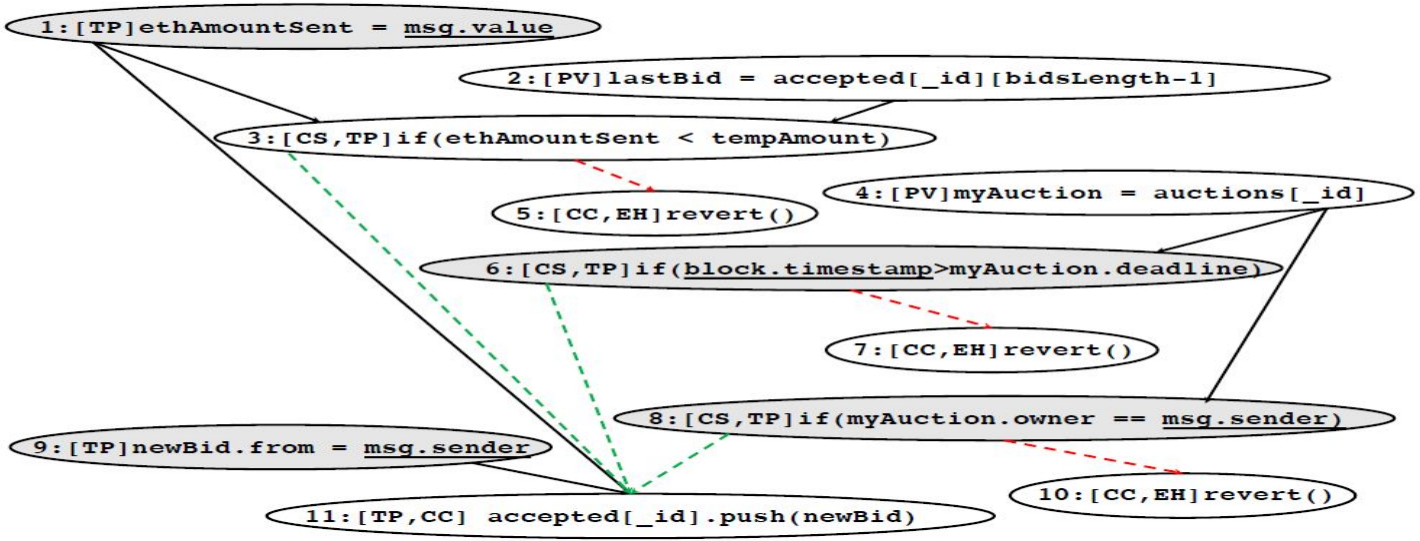
- Capture business logic from a given smart contract
- Solution
  - build **Business Model Graphs** (BMGs) to represent business model

```
1  function bidOnAuction(uint _id) public payable {
2      uint256 ethAmountSent = msg.value;
3
4      // owner cannot bid on his/her own merchandise
5      Auction memory myAuction = auctions[_id];
6      if(myAuction.owner == msg.sender) revert();
7
8      // check whether auction has expired
9      if(block.timestamp > myAuction.deadline) revert();
10
11     // check whether previous bids exist
12     uint bidsLength = accepted[_id].length;
13     uint256 tempAmount = myAuction.startPrice;
14     Bid memory lastBid;
15     if(bidsLength > 0) {
16         lastBid = accepted[_id][bidsLength-1];
17         tempAmount = lastBid.amount;
18     }
19
20     // check if bid price is greater than the current
        highest
21     if(ethAmountSent < tempAmount) revert();
22
23     // add the new bid to auction state
24     Bid memory newBid;
25     newBid.from = msg.sender;
26     newBid.amount = ethAmountSent;
27     accepted[_id].push(newBid);
28     emit BidSuccess(msg.sender, _id);
29 }
```



Model Extraction

Five
busir

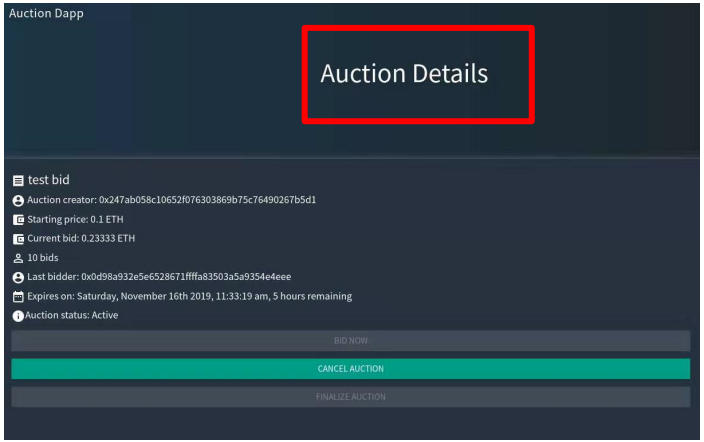


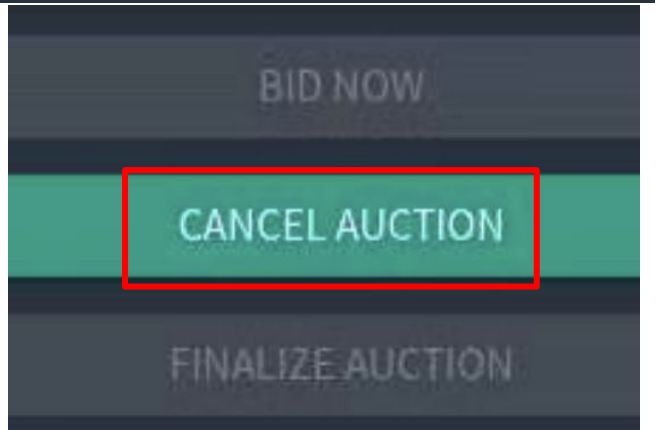

```
1  function bidOnAuction(uint _id) public payable {
```

```
24      Bid memory newBid;
25      newBid.from = msg.sender;
26      newBid.amount = ethAmountSent;
27      accepted[_id].push(newBid);
28      emit BidSuccess(msg.sender, _id);
29  }
```

Semantics Recovery

- Multiple levels of semantics
  - smart contract level
    - e.g., this is an Auction DApp
    - HTML parsing ⇒ NLP technique ⇒ semantics
  - function level
    - e.g., **cancelAuction**() is to cancel the auction
    - hook MetaMask & fuzz contract calling UI components
    - GUI text ⇒ NLP technique ⇒ semantics



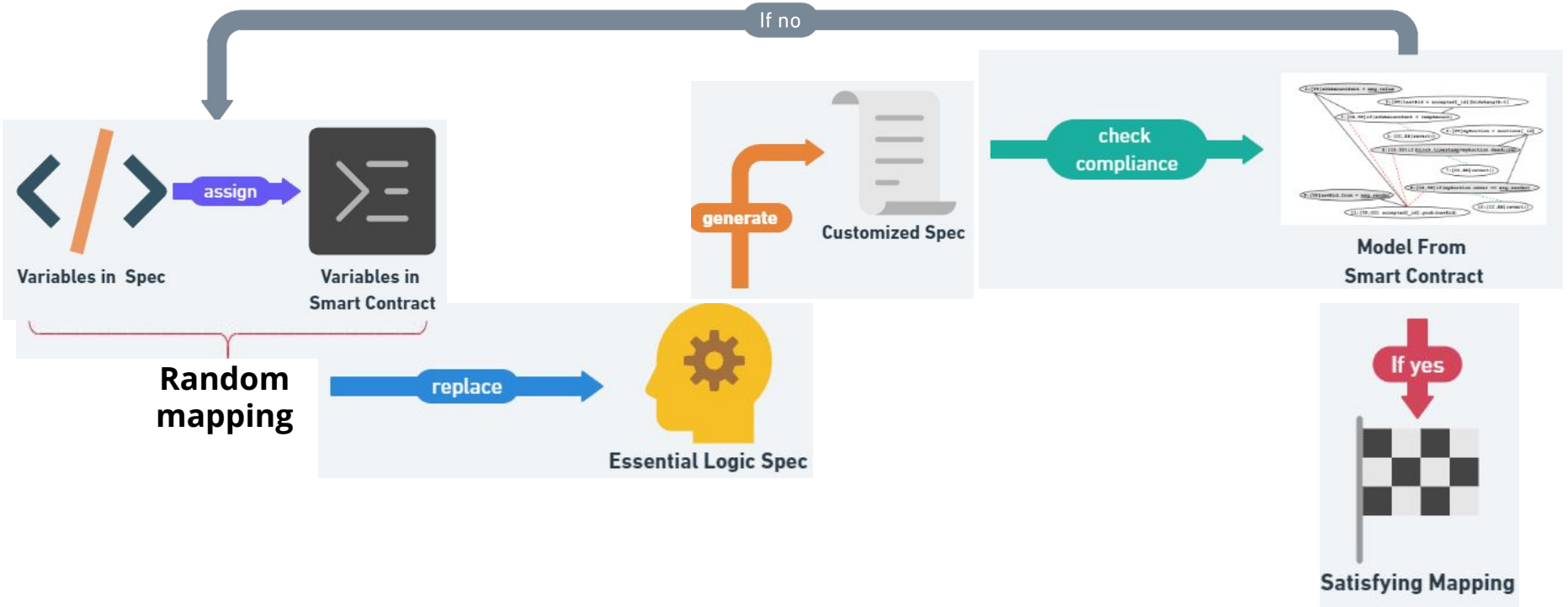


Semantics Recovery

- variable level
  - formalize it as a **model checking problem**
  - goal: find a mapping μ between
    - variables in smart contract code and variables in pre-defined essential specs
    - e.g., accepted[_id].active means auction state
    - such that the model complies with the customized specs

Function	Spec Type	Formal Spec
Bidding	Essential#1	$\square(current\_bid > highest\_bid \rightarrow \Diamond(highest\_bid := current\_bid \wedge highest\_bidder := current\_bidder))$
	Safety#1	$\square(current\_time > deadline \rightarrow \Diamond(execution\_state := revert))$
	Safety#2	$\square(auction.active == false \rightarrow \Diamond(execution\_state := revert))$
	Safety#3	$\square current\_bidder == auction.owner \rightarrow \Diamond(execution\_state := revert))$
Cancel	Essential#1	$auction.active := false$
	Safety#1	$\square(requester != auction.owner \rightarrow \Diamond(execution\_state := revert))$
	Safety#2	$\square(highest\_bidder != null \rightarrow \Diamond(execution\_state := revert))$

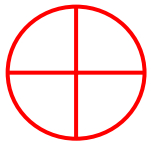
Semantics Recovery



Safety Vetting

variable mapping

Function/Domain	Smart Contract Variable	Spec Concept
bidOnAction	msg.value	*current_bid*
bidOnAction	msg.sender	*current_bidder*
bidOnAction	newBid.amount	*highest_bid*
bidOnAction	newBid.from	*highest_bidder*
cancelAction	msg.sender	*requester*
contract-wide	accepted[_id][bidsLength-1].amount	*highest_bid*
contract-wide	accepted[_id][bidsLength-1].from	*highest_bidder*
contract-wide	accepted[_id].active	*auction.active*



customized Specs

Safety#1	$\square(requester \mathrel{!}= auction.owner \rightarrow \Diamond(execution\_state := revert))$
Safety#2	$\square(highest\_bidder \mathrel{!}= null \rightarrow \Diamond(execution\_state := revert))$

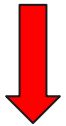
generic safety specs

Safety Vetting

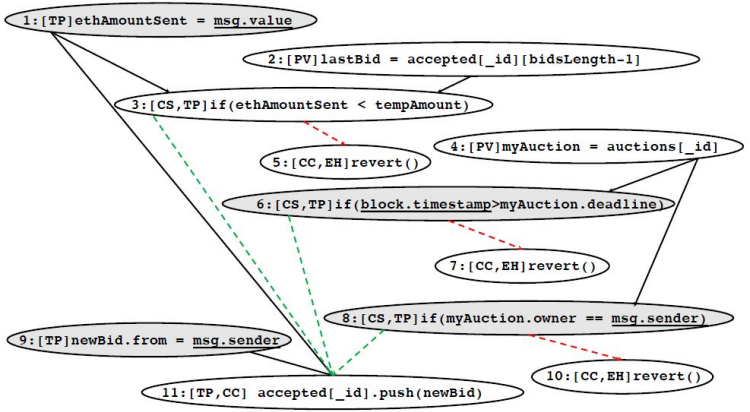
Customized Safety Specifications

□(`msg.sender` != auction.owner ⟶ ◇(execution_state = revert))

□(`accepted[_id][bidsLength-1].from` != null ⟶ ◇(execution_state = revert))



check against



Evaluation

#	Name	Unsafe Func Name	Code Logic	Major Widget Text/Context	UI == Logic?	Safety Issue in Smart Contracts	Violated Policy	Source Analyzability
1	cryptoatoms.org	-	-	-	Yes	-	-	Yes
2	proofoflove.digital	-	-	-	Yes	-	-	Yes
3	snailking	-	-	-	Yes	-	-	Yes
4	cryptominingwar	-	-	-	Yes	-	-	Yes
5	market.start.solar	-	-	-	Yes	-	-	No (Missing Source)
6	etheroll	-	-	-	Yes	-	-	No (Inlined Bytecode)
7	cryptokitties	`bid()`	Auction-Bid	*"buy"*	Ambiguity	N/A	N/A	Yes
8	hyperdragons	-	-	-				
19	note_dapp	-	-	-				
20	metacoin	-	-	-				
21	simple-vote	`vote()`	N/A	*"Start a vote"*	No Impl.	N/A	N/A	Yes
22	truffle-voting	`vote()`	Voting-Vote	*"Approve/Against/Abstain"*	Yes	Voting for an expired election	Voting-Vote-S1	Yes
23	Gnosis Safe	-	-	-	Yes	-	-	Yes
24	vote-dapp	-	-	-	Yes	-	-	Yes
25	EVotingDApp	-	-	-	Yes	-	-	Yes
26	Election	`vote()`	Voting-Vote	*"Vote"*	Yes	Voting for an expired election	Voting-Vote-S1	Yes
27	Election-DAPP	`vote()`	Voting-Vote	*"Approve/Against/Abstence"*	Yes	Voting for an expired election	Voting-Vote-S1	Yes
28	Vote	`vote()`	Voting-Vote	*"Submit"*	Yes	Voting for an expired election	Voting-Vote-S1	Yes
29	VotingDapp	`vote()`	Voting-Vote	*"Vote"*	Yes	Voting for an expired election	Voting-Vote-S1	Yes
30	VoteDapp	`vote()`	Voting-Vote	*"Vote"*	Yes	Voting for an expired election	Voting-Vote-S1	Yes
31	voting-DApp	`vote()`	Voting-Vote	*"Vote"*	Yes	Voting for an expired election	Voting-Vote-S1	Yes
32	VoteMe	-	-	-	Yes	-	-	Yes
33	Overview	`invest()`	CS-Invest	*"Buy tokens"*, *"Crowdsale"*	Yes	Invest an expired crowdsale	CS-Invest-S2	Yes
34	Crowdsale	-	-	-	Yes	-	-	Yes

Discovered 19 real-world issues

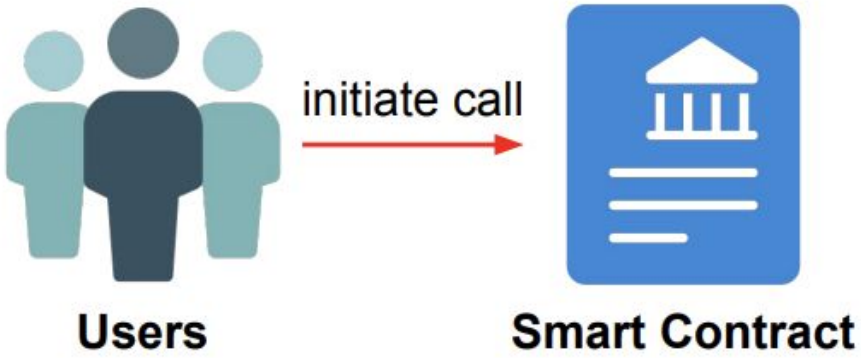
ACM CCS 2022 Best Paper Honorable Mention Award

Proxy Hunting: Understanding and Characterizing Proxy-based Upgradeable Smart Contracts in Blockchains

William E Bodell III, Sajad Meisami, Yue Duan

USENIX Security 2023

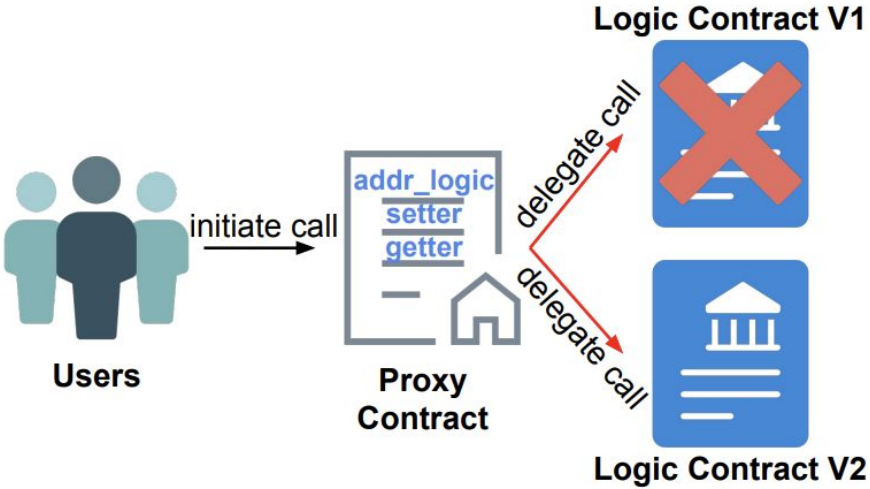
Smart Contract



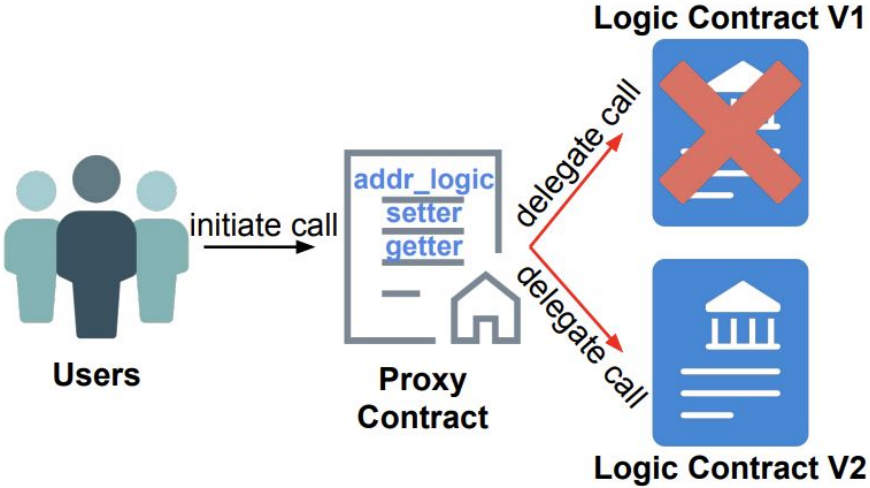
- Immutability serves as the foundation for security
- Smart contracts need regular updates
  - security reasons
  - non-security reasons

Upgradeable Smart Contract

- Proxy-based upgradeable smart contract
  - split one contract into at least two
    - proxy contract
    - logic contract
  - user interacts with the proxy
  - proxy contract calls to the actual logic

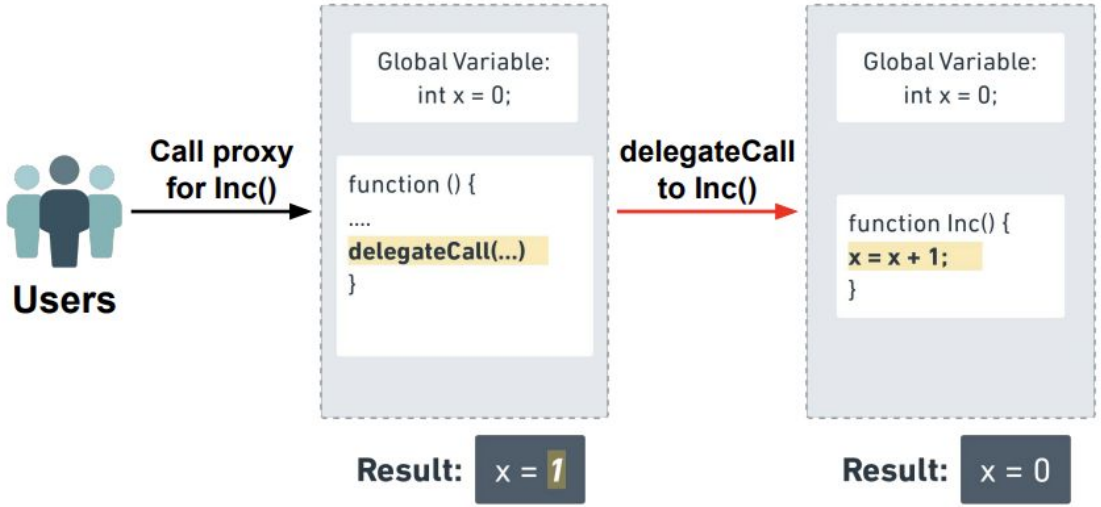


Upgradeable Smart Contract



- ***addr_logic***: a global variable that stores the address of the current logic contract
- ***setter***: a function that sets ***addr_logic***
- ***getter***: a function that retrieves ***addr_logic***

Upgradeable Smart Contract

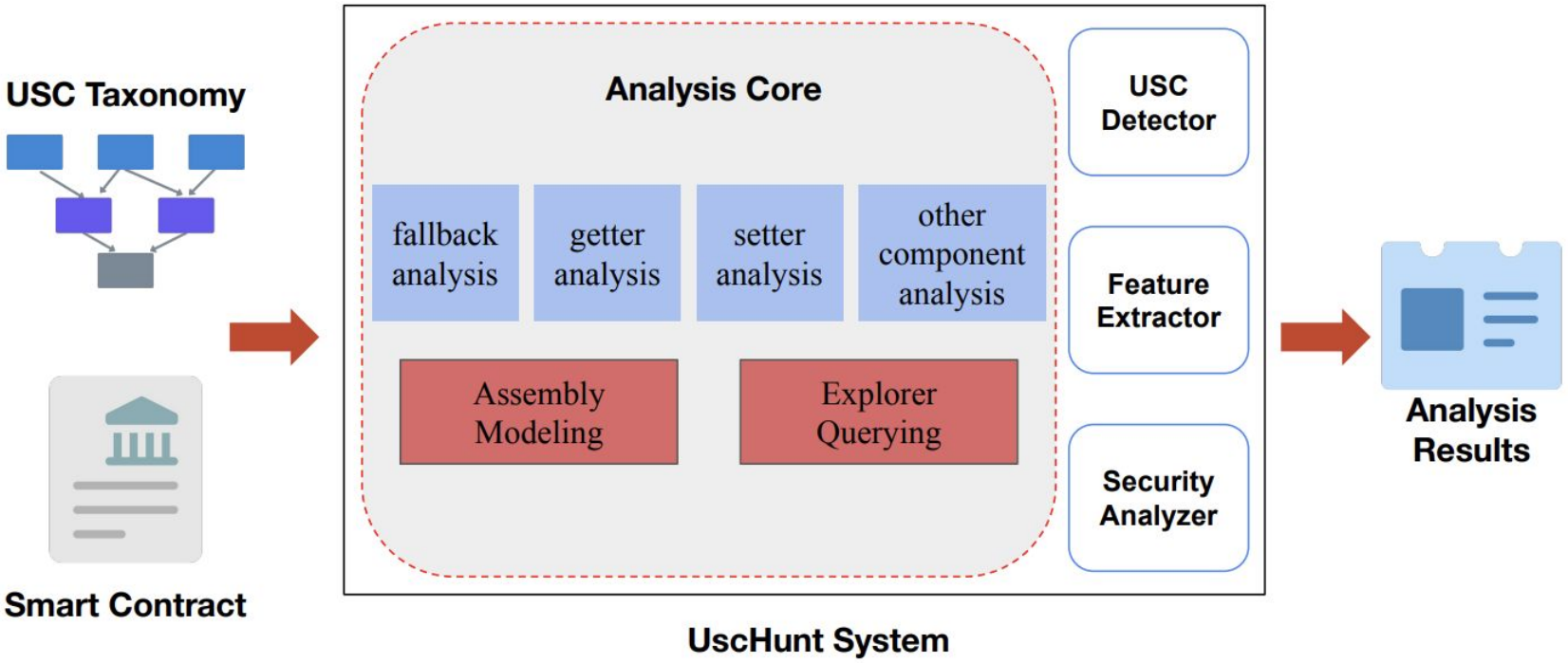


- ***deletegatecall***
  - a special instruction
  - call an external function but execute the callee function in the caller's context

Upgradeable Smart Contract Study

- Conduct a study on USCs with 800K+ smart contracts
  - **Importance**
    - How widely used are USCs in today's mainstream blockchains? How much USD worth of cryptocurrencies and tokens is currently held by USCs? How have these numbers changed over time?
  - **Unique Behaviors and Design Patterns**
    - How can we characterize the uniqueness of USCs in the real world? How are USCs implemented? What are the unique behaviors and design patterns?
  - **Security and Safety Risks**
    - What are the security and safety issues associated with USCs? What is the possible impact of each issue?

UscHunt Overview



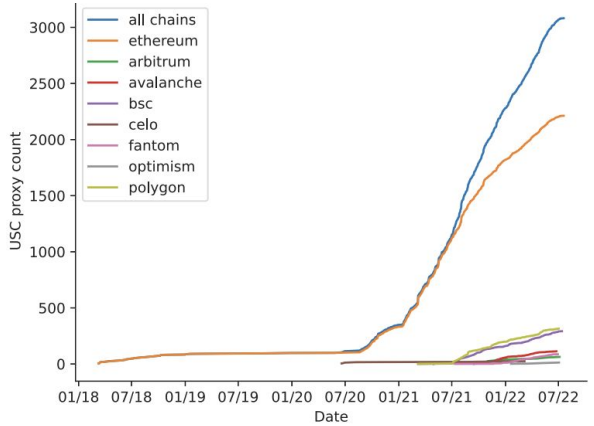
Findings - Importance

Table 1: Percentage of USCs on Each Blockchain

Chain	Total Count	USC Proxy Count	USC Proxy Percent
Ethereum	482,889	5,384	1.11%
Arbitrum	4,684	189	4.04%
Avalanche	29,759	282	0.95%
BSC	261,068	1,507	0.58%
Celo	917	56	6.11%
Fantom	16,893	218	1.29%
Optimism	960	60	6.25%
Polygon	64,487	1,119	1.74%
Total	861,657	8,815	1.02%

Table 2: Total value held in USCs

Chain	Native Value	Token Value	Total Value
Ethereum	$68.1M	$2.6B	$2.7B
Arbitrum	$16.1K	$2.5M	$2.5M
Avalanche	$6.9K	$4.7M	$4.7M
BSC	$177K	$12.5M	$12.7M
Celo	$322.6M	$106.2M	$428.8M
Fantom	$304.6K	$3M	$3.3M
Optimism	$0	$425.5K	$425.5K
Polygon	$43.3K	$1.4M	$1.4M



Findings - Patterns and Behaviors

	Pattern Name	Inherited Storage	Eternal Storage	Unstructured Storage			Mastercopy / Singleton	Transparent Proxy	Multiple Implementations		Registry Proxies	
	Sub-Pattern	-	-	Non-standard	EIP-1967: Standard Storage Slots	EIP-1822: UUPS	-	-	EIP-1538: VTable	EIP-2535: Diamond	Beacon	Registry
Syntactic Features	Target location	Inherited contract	-	Proxy contract	Proxy contract	Proxy and logic	Proxy and logic	Proxy contract	Proxy contract	Proxy contract	External contract	External contract
	Target type	-	-	bytes32	bytes32	bytes32	address	-	mapping (bytes4 => address)	mapping (bytes4 => Facet struct)	address	mapping (___ => address)
	Target scope	State variable	State variable	Constant	Constant	Literal in fallback	State variable	-	State variable	Structure variable	State variable	State variable
	Target inheritance	Inherited by proxy and logic	-	-	-	Logic must inherit Proxiable	-	-	-	-	-	-
	Setter location	-	-	Proxy contract	Proxy contract	Logic contract	Logic contract	Proxy contract	Proxy contract	Logic contract	External contract	External contract
	Getter location	-	-	Proxy contract	Proxy contract	Proxy contract (fallback)	Proxy contract (fallback)	Proxy contract	Proxy contract (fallback)	Proxy contract (fallback)	External contract	External contract
	Mappings of each type	-	Yes	-	-	-	-	-	-	-	-	-
Semantic Features	Constant storage offset	No	No	Yes, slot varies	Yes, hashed string is `eip1967.proxy.implementation`	Yes, hashed string is `PROXIABLE`	No	-	No	No, but struct may be stored in storage slot	No, but beacon address may be stored in storage slot	No, but registry address may be stored in storage slot
	Storage layout coupling	Yes	Yes	No	No	No	Yes	-	-	-	-	-
	Simultaneous upgrades	-	-	-	-	-	-	-	-	-	Yes	Yes
	Removable upgradeability	-	-	No	No	Upgrade to logic w/o setter	Upgrade to logic w/o setter	-	-	Remove Diamond CutFacet	Update to beacon w/o setter	Update to registry w/o setter
	Transparent admin check	-	-	-	-	-	-	Yes	-	-	-	-
	Scattered implementations	-	-	-	-	-	-	-	Yes	Yes	-	-

Findings - Security Issues

Table 7: Detected USC-related Security Issues

Chain	Implementation Issues				Policy Issues	
	Storage Layout Clashes (Between Proxy and Logic)	Storage Layout Clashes (Between Logic Versions)	Function Selector Collisions	Insufficient Compatibility Checks In Logic Setter	Vulnerabilities Cannot Be Patched Immediately	Upgradeability Can Be Removed Accidentally
Ethereum	36	3	0	1,017	24	150
Arbitrum	1	0	0	87	0	12
Avalanche	1	1	0	137	3	12
BSC	7	0	0	337	0	23
Celo	0	0	0	25	0	0
Fantom	3	0	0	166	0	2
Optimism	0	0	0	33	0	5
Polygon	8	0	0	441	3	9
Total	56	4	0	2,243	30	213

Findings - Security Issues

```
1  contract Proxy {
2    Logic public target;
3    function setTarget(Logic _target) {
4      target = _target;
5    }
6    function() external payable {
7      ...
8      delegatecall(gas,sload(target), ...)
9    }
10 }
11
12 contract Logic {
13   address public otherAddr;
14   function setOtherAddr(address _other) {
15     otherAddr = _other;
16   }
17 }
```

Listing 1: Synthetix Proxy and Logic Contracts

- The storage layout clash is in the first lines of each contract: the Proxy contract stores its addr_logic as a Logic contract type state variable in the first storage slot (Ln.2), while the logic contract declares another variable (otherAddr) in the same position (Ln.13).

Findings - Security Issues

(1) **Checks contract call result.** In the best case, the *setter* should attempt to call a function that is expected to be in the new logic contract, and verify that it returns the correct value, as EIP-1822 does:

```
require(bytes32(PROXIABLE_MEM_SLOT) ==
Proxiable(newAddress).proxiableUUID())
```

(2) **Checks address is a contract.** It may be sufficient to check that the new address is indeed a contract:

```
require(extcodesize(newLogicAddr) > 0)
```

(3) **Checks address is not zero.** A common yet insufficient check ensures the new address is not zero:

```
require(_implementation != address(0))
```

(4) **Checks new address is not the same as old.** The least sufficient check, no bearing on compatibility:

```
require(currentImpl != _newImpl)
```

Findings - Security Issues

```
1  contract EnclavesDEXProxy {
2    address public proposedImpl;
3    uint256 public proposedTimestamp;
4
5    function propose(address _proposed) {
6      ...
7      proposedImpl = _proposed;
8      proposedTimestamp = now + 2 weeks;
9    }
10   function upgrade() {
11     ...
12     require(proposedTimestamp < now);
13     impl = proposedImpl;
14   }
15 }
```

Listing 4: Example of a USC proxy with a time-delay.

Thanks!!