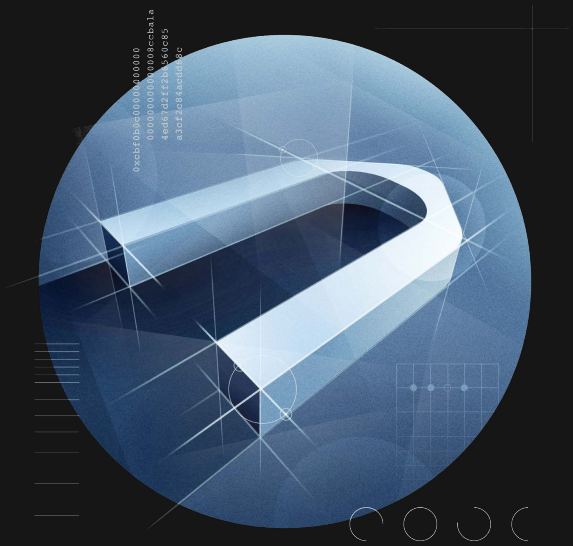


Program Analysis for High-Value Smart Contract Vulnerabilities (or *how to tame state explosion in smart contracts*)

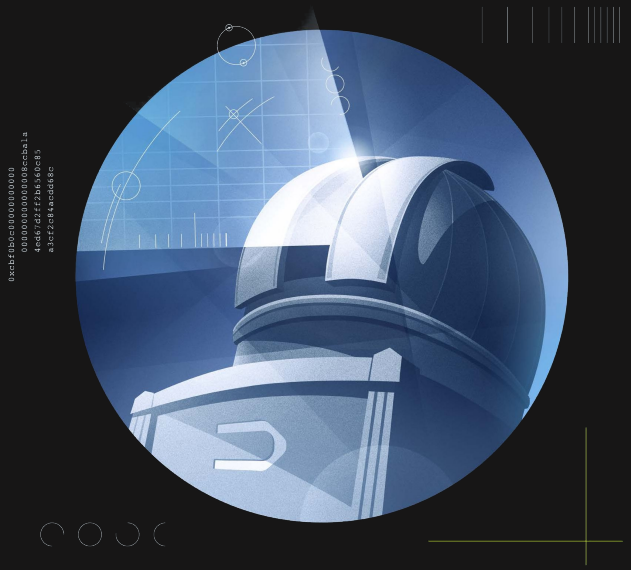
Yannis Smaragdakis

DEDAUB

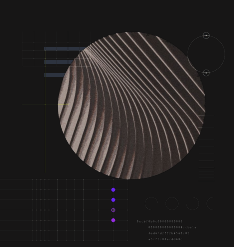
+ U.Athens

jointly with

Neville Grech, Sifis Lagouvardos, Konstantinos Triantafyllou, Ilias Tsatiris, Yannis Bollanos, Tony Rocco Valentine



Research positions available (ERC)



Smart Contracts?

- Perfect domain for program analysis/verification!
  - correctne$$ crucial

Move is a programming language designed for creating secure and verifiable smart contracts and other applications, especially on blockchains. It's known for its strong type system, resource-oriented design, and ability to formalize properties through verification, ensuring the reliability of the code.

Key features and characteristics of Move:

Safety and Security:

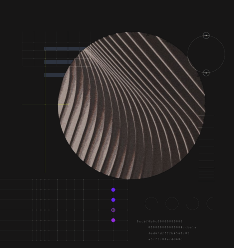
Move prioritizes safety and security, particularly when dealing with digital assets and transactions. It aims to prevent bugs and vulnerabilities that could compromise the integrity of on-chain systems.

Resource-Oriented Design:

Move utilizes resource types with "move" semantics, which directly represent digital assets like currency. This design ensures scarcity and prevents accidental duplication or loss of resources.

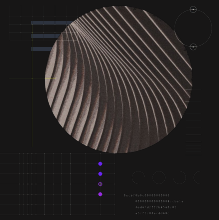
Formal Verification:

Move's type system and design allow for formal verification, meaning that it's possible to mathematically prove that the code behaves as intended. This adds an extra layer of security and reliability.



Smart Contracts?

- Perfect domain for program analysis/verification!
  - correctne$$ crucial
  - code public
  - executions public
  - manageable size / essential complexity



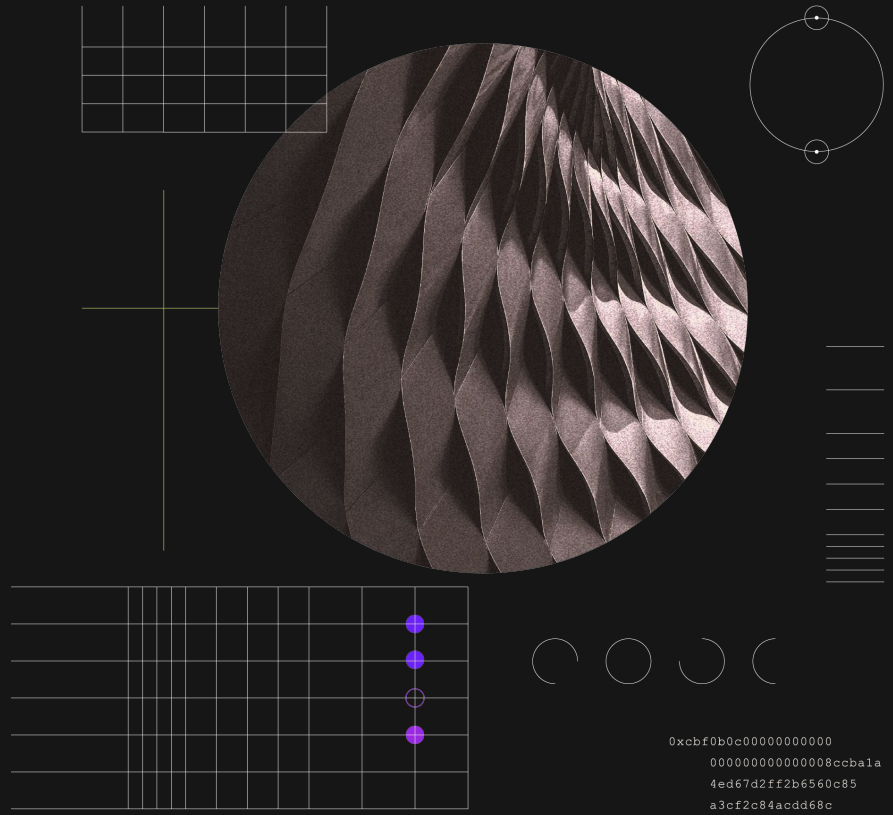
My Research/Dedaub Technology: Creating Programs that Understand Programs

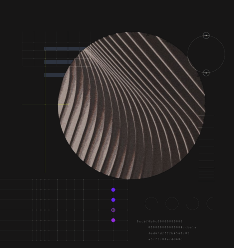
- Research in **Static Analysis**
  - create a model of all possible program behaviors
- Since 2018: applying to smart contracts

  [OOPSLA'18, ICSE'19, OOPSLA'20, PLDI'20, OOPSLA'21, CACM, OOPSLA'22, SBC'23, ISSTA'25]
- All analyses specified declaratively
  - logical rules (thousands of them)

```
LoopBoundBy(loop, var) :-
  InductionVar(i, loop),
  !InductionVar(var, loop),
  Flows(var, condVar), Flows(i, condVar),
  LoopExitCond(condVar, loop).
```

A Gadget


0xcbf0b0c00000000000
000000000000008ccba1a
4ed67d2ff2b6560c85
a3cf2c84acdd68c



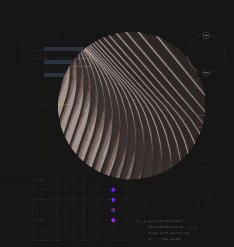
A Technical Topic: Transitively Closed Relations

- We all know transitively closed relations

  $r(x,y) \land r(y,z) \Rightarrow r(x,z)$

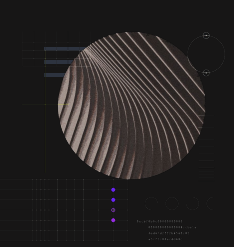
- In Datalog:

```
R(x,z) :- R(x,y), R(y,z).
```



Transitively Closed Relation

- Say we have Edge, want to compute its transitive closure, Path
- Base Case:
  ```
  Path(x,y) :- Edge(x,y).
  ```
- Then, straightforward:
  ```
  Path(x,z) :- Path(x,y), Path(y,z).
  ```



Transitively Closed Relation

- Say we have Edge, want to compute its transitive closure, Path
- Base Case:

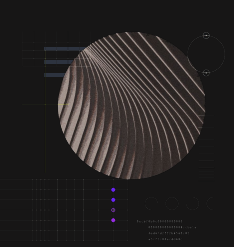
```
Path(x,y) :- Edge(x,y).
```

- Then, straightforward:

```
Path(x,z) :- Path(x,y), Path(y,z).
```

- Much better:

```
Path(x,z) :- Path(x,y), Edge(y,z).
```



Transitively Closed Relation

- Say we have Edge, want to compute its transitive closure, Path
- Base Case:

```
Path(x,y) :- Edge(x,y).
```

- Then, straightforward:

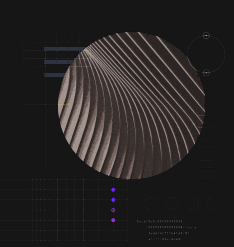
```
Path(x,z) :- Path(x,y), Path(y,z).
```

- Much better:

```
Path(x,z) :- Path(x,y), Edge(y,z).
```

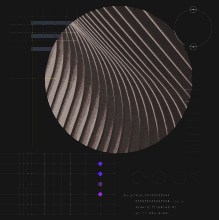
  - Evaluated as:

```
Path(x,z) :- ΔPath(x,y), Edge(y,z).
```



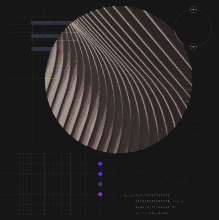
A Very Efficient Algorithm!

- ```
  Path(x,y) :- Edge(x,y).
  Path(x,z) :- Path(x,y), Edge(y,z).
  ```
- This is a pretty good algorithm for TC
  - likely optimal under some conditions (e.g., sparseness, trees)
  - not just in Datalog
  - but Datalog takes care of many efficiency concerns
  ```



A New Problem: Transitive Re-Closure (Incremental Closure)

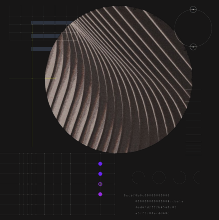
- We have `Path`, we are given a `Delta` with extra paths, compute `ExtPath`
- Can we avoid recomputing TC from scratch?
  - may even be impossible, e.g., no access to `Edge`, only `Path`



A New Problem: Transitive Re-Closure (Incremental Closure)

- We have `Path`, we are given a `Delta` with extra paths, compute `ExtPath`
- Can we avoid recomputing TC from scratch?
  - may even be impossible, e.g., no access to `Edge`, only `Path`
- Straightforward:

```
Path(x,y) :- Delta(x,y).
ExtPath(x,y) :- Path(x,y).
ExtPath(x,z) :- ExtPath(x,y), Path(y,z).
```

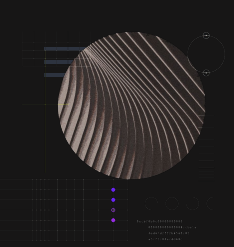


A New Problem: Transitive Re-Closure (Incremental Closure)

- We have `Path`, we are given a `Delta` with extra paths, compute `ExtPath`
- Can we avoid recomputing TC from scratch?
  - may even be impossible, e.g., no access to `Edge`, only `Path`
- Straightforward:

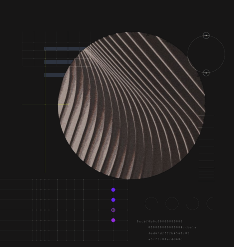
```
Path(x,y) :- Delta(x,y).
ExtPath(x,y) :- Path(x,y).
ExtPath(x,z) :- ExtPath(x,y), Path(y,z).
```

- Simply awful in performance!
  - (e.g., Dataset A: 28s + 215s, Dataset B: 8m + 419m)



Can We Emulate the Insight of Efficient TC?

- We need two new concepts:
  - `DeltaLeft(x,y)`: new path that starts with a delta edge on the left
  - `DeltaOneLeft(x,y)`: new path that starts with a delta edge on the left and contains no other delta edges
- Crucial: use negation for performance!

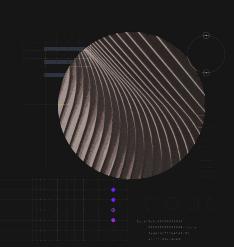


Transitive Re-Closure (Incremental Closure)

```
DeltaOneLeft(x,y) :- Delta(x,y), !Path(x,y).
DeltaOneLeft(x,z) :- Delta(x,y), Path(y,z), !Path(x,z).

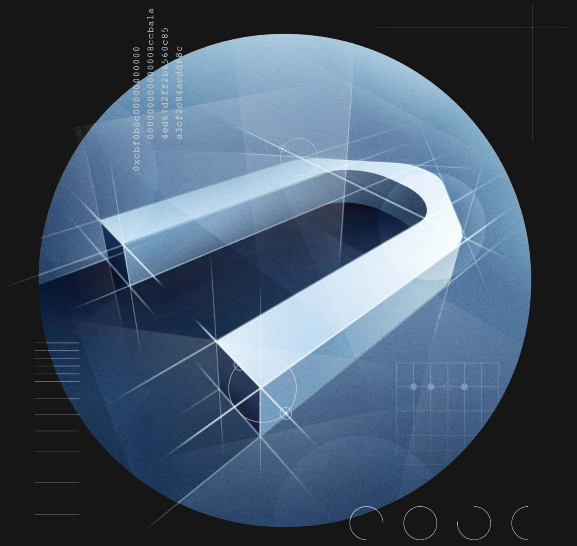
DeltaLeft(x,y) :- DeltaOneLeft(x,y).
DeltaLeft(x,z) :-
  DeltaOneLeft(x,y), DeltaLeft(y,z), !Path(x,z).

ExtPath(x,y) :- Path(x,y).
ExtPath(x,y) :- DeltaLeft(x,y).
ExtPath(x,z) :- Path(x,y), DeltaLeft(y,z), !Path(x,z).
```



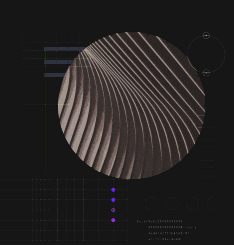
Input?

- (Dataset A: 28s + 11s, Dataset B: 8m + 4m)
- This should be a pretty good general transitive re-closure algorithm
  - (without taking advantage of special structure, e.g., SCCs, which can be added orthogonally)



Back to ...

Program Analysis for High-Value Smart Contract Vulnerabilities (or *how to tame state explosion in smart contracts*)



A Paradox

So much $ value, so much research, so little impact!

- Perez and Livshits [2021]: research tools produce lots of "true" warnings, only 0.27% of funds exploited
- Security experts consider automated tools to be near-worthless
  - @samczsun: "*tooling can't find the bugs that matter so at best we're just making sure people don't accidentally use blockhash on the current block or something*"
  - @gakonst: "*for an experienced contract author, it's never the automated tooling that finds the bugs that kill them*"

And Yet...

Dedaub
Vulnerability
Disclosures

Vulnerability Disclosures (since 2021)

Yield Skimming: Forcing Bad Swaps on Yield Farming

Vulnerability Disclosures (since 2021)

Yield Skimming: Forcing
Swaps on Yield Farming

Inside the War Room That Saved Primitive Finance

Jonah Michaels Follow

Mar 4 · 12 min read

Vulnerability Disclosures (since 2021)

Yield Skimming: Forcing Swaps on Yield Farming

Inside the War Primitive Fina

Jonah Michaels Follow

Mar 4 · 12 min read

"Look ma', no source!" Hacking a DeFi Service with No Source Code Available

Vulnerability Disclosures (since 2021)

Ethereum Pawn Stars: “$5.7M in hard assets? Best I can do is $2.3M”

e the War
itive Fina
ichaels Follow
2 min read

“Look ma’, no source!” Hacking a DeFi Service with No Source Code Available

Vulnerability Disclosures (since 2021)

Ethereum Pawn Stars: “$5
hard assets? Best I can do i
$2.3M”

Killing a Bad (Arbitrage) Bot

... to save its owners

urce!” Hacking a
n No Source

Vulnerability Disclosures (since 2021)

Ethereum Pawn Stars: “$5
hard assets? Best I can do i
$2.3M”

Vulnerability Disclosures (since 2021)

Phantom Functions and the Billion-Dollar No-op

By the Dedaub team

Harvest Finance

Uninitialized Proxies

Bug Fix Postmortem

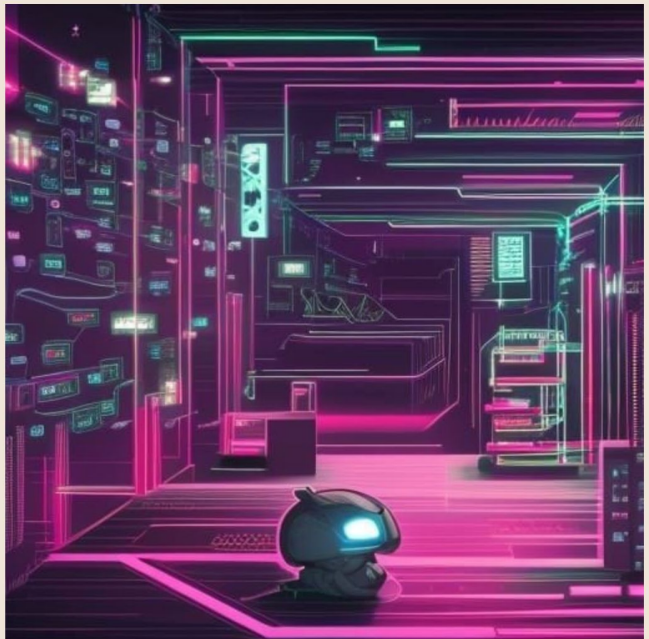
Vulnerability Disclosures (since 2021)

Phantom Functions and the Billion-Dollar No-op

By the Dedaub team

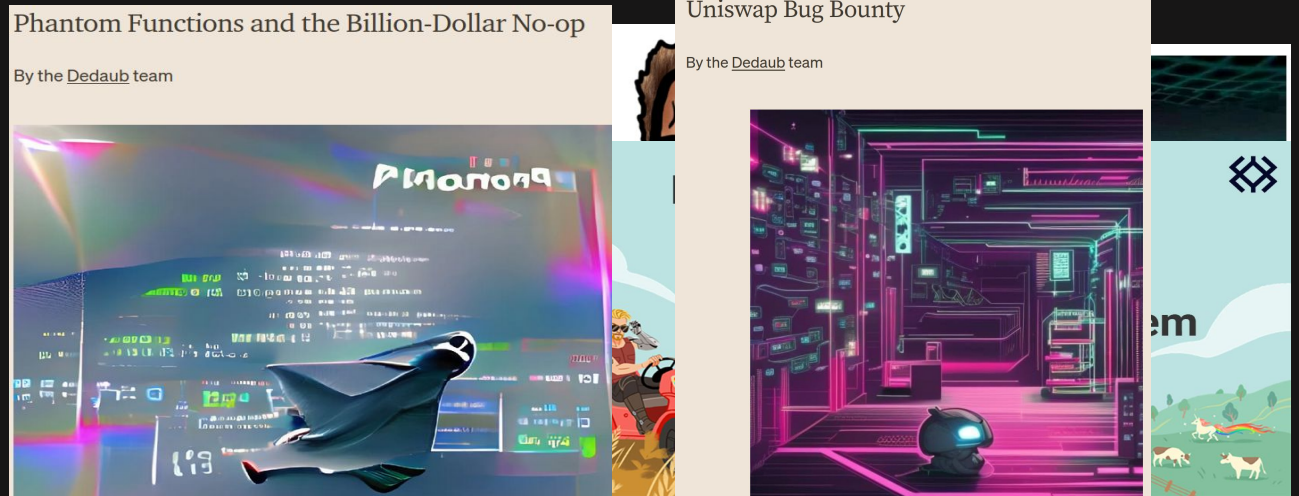
Uniswap Bug Bounty

By the Dedaub team

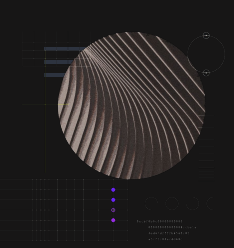


Vulnerability Disclosures (since 2021)

- Many major security vulnerabilities, 11 bug bounties of over $3M total
  - by DeFi Saver, Dinngo/Furucombo, Primitive, Armor, Vesper, BT Finance, Harvest, Multichain/Anyswap, Rari/Tribe DAO, Uniswap

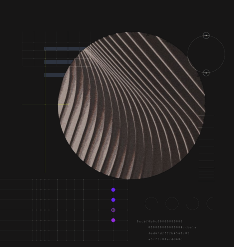


How???



Background I – Analysis Questions

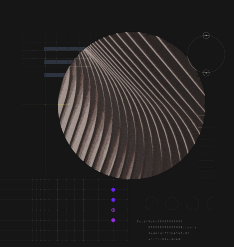
- **Taint** analysis is excellent example
  - *tainted* value: *controllable by an untrusted caller*
- Dominant in practice: most analysis questions hinge on tainting



Taint + Sensitive Operations

```
function withdraw(uint amount) {
  …
  token.transferFrom(owner, spender, amount);
  …
}
```

- Where does `owner` come from?
- Is the code even reachable for an untrusted caller?
- What about `spender`?

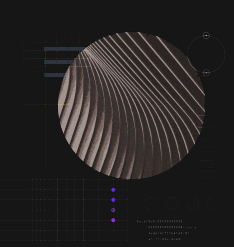


Taint + Reentrancy

```
function withdraw(uint amount) {

  if (credit[investor] >= amount) {
    investor.call.value(amount)();
    credit[investor] -= amount;
  }
}
```

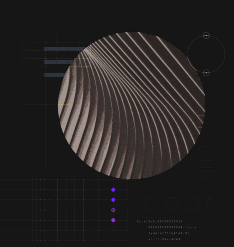
- Where does `investor` come from? Can it be contract?
- Is the code even reachable for an untrusted caller?



Taint + Reentrancy

```
function withdraw(uint amount) {
  require(msg.sender == DAO || msg.sender == owner);
  if (credit[investor] >= amount) {
    investor.call.value(amount)();
    credit[investor] -= amount;
  }
}
```

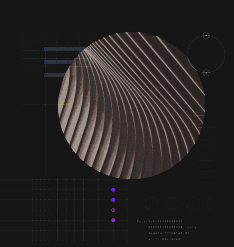
- Where does `investor` come from? Can it be contract?
- Is the code even reachable for an untrusted caller?



Background II – Analysis Answers?

- Principle:
  Static Analysis is a game of balancing 3 elements
  - **precision**
  - **completeness**
  - **performance**

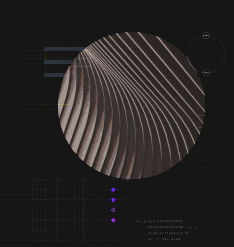
Static Analysis Is Poetry, Not Prose



Example

```
function whichPaths(uint x) public (returns uint y) {
  y = 3;
  if (x % 2 != 0) { y++; }
  if (x % 4 != 0) { y = y * y; }
}
```

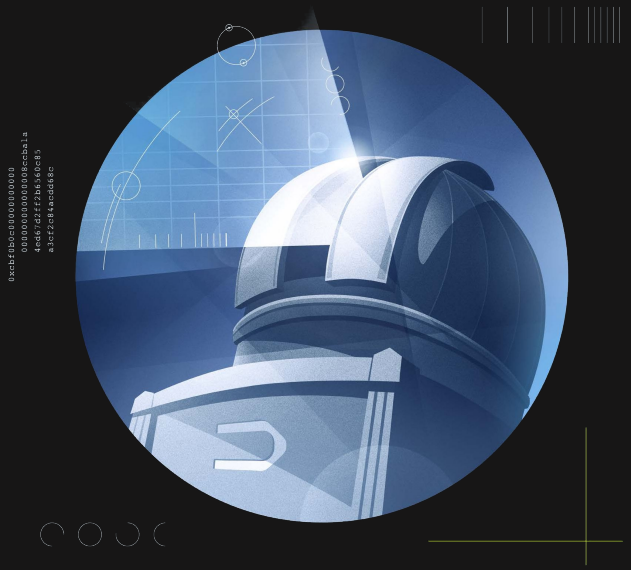
- Possible values: 3, 9, 16
  - cannot satisfy first path but not second: if not divisible by 2, certainly not divisible by 4
- If an analysis says: 3, 9, 4, 16, is it worse?



Static Analysis is Poetry, not Prose

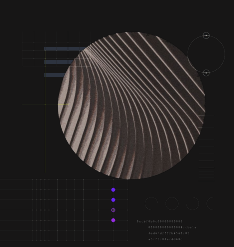
- Execution/model checking:
  The old man sat on the bench in the park. He watched as children played on the swings and slides. He smiled as he remembered his own childhood.

- Static analysis:
  Old man, children play, memories smile



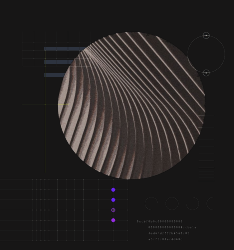
First Weapon: Symbolic Value-Flow (Symvalic) Analysis

[OOPSLA'21]



Symvalic (Symbolic + value-flow) Analysis

- What is it?
  - a precise, path-sensitive static analysis
  - that mixes values and symbolic expressions
    - Datalog fixpoint + symbolic reasoning
  - gets scalable precision through *dependencies*
    - a generalization of context sensitivity
  - main client: taint analysis



The Dirty Secret of Program Verification
(for security)

- ***Nothing works!***
- Execution-based approaches
  (symbolic/dynamic-symbolic execution, model checking)
  are precise but incomplete
  - state explosion problem
- Static analysis approaches are complete but imprecise

Execution-Based Approaches

(symbolic execution, model checking): *Horizontal*

```
address admin ;  // set up at construction, not in contract code

function withdrawToken (IERC20 token, uint256 amount, address sendTo) external {
  onlyAdmin();

  uint256 adjusted = amount * 103 / 100;

  if (amount >= 10000 && amount < 100000)
    token.transfer(sendTo,adjusted) ; …
}

function onlyAdmin() internal view {
  require (msg.sender == admin, "only admin");
}
```

adjusted	token	amount	sendTo	admin
1030	0x6f..	1000	0x3f6..	0x5a1..

Execution-Based Approaches

(symbolic execution, model checking): *Horizontal*

```
address admin ;  // set up at construction, not in contract code

function withdrawToken (IERC20 token, uint256 amount, address sendTo) external {
  onlyAdmin();

  uint256 adjusted = amount * 103 / 100;

  if (amount >= 10000 && amount < 100000)
    token.transfer(sendTo,adjusted) ; …
}

function onlyAdmin() internal view {
  require (msg.sender == admin, "only admin");
}
```

adjusted	token	amount	sendTo	admin
1030	0x6f..	1000	0x3f6..	0x5a1..

Execution-Based Approaches

(symbolic execution, model checking): *Horizontal*

```
address admin ;  // set up at construction, not in contract code

function withdrawToken (IERC20 token, uint256 amount, address sendTo) external {
  onlyAdmin();

  uint256 adjusted = amount * 103 / 100;

  if (amount >= 10000 && amount < 100000)
    token.transfer(sendTo,adjusted) ; …
}

function onlyAdmin() internal view {
  require (msg.sender == admin, "only admin");
}
```

adjusted	token	amount	sendTo	admin
1030	0x6f..	1000	0x3f6..	0x5a1..

$\gamma = (\rho, \Sigma, H)$

Static Analysis Is Poetry, Not Prose

Value-Flow Static Analysis Approaches: *Vertical* (“sets of values”)

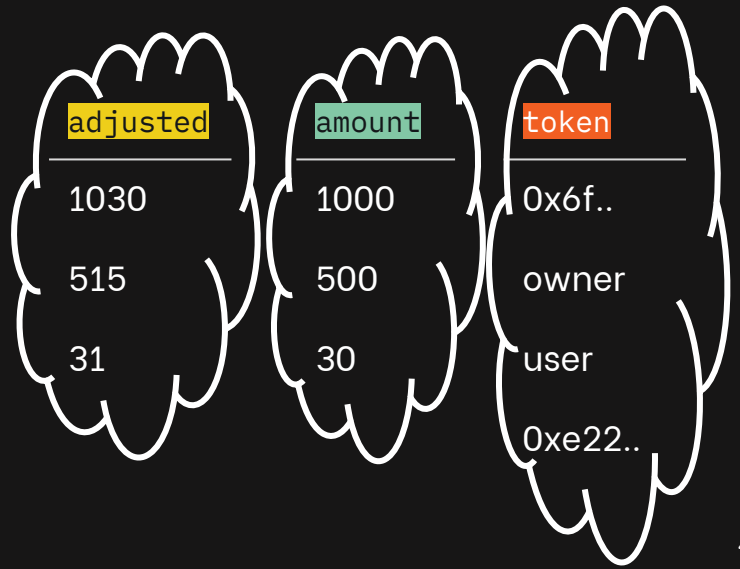
```
address admin ;  // set up at construction, not in contract code

function withdrawToken (IERC20 token, uint256 amount, address sendTo) external {
  onlyAdmin();

  uint256 adjusted = amount * 103 / 100;

  if (amount >= 10000 && amount < 100000)
    token.transfer(sendTo,adjusted) ; …
}

function onlyAdmin() internal view {
  require (msg.sender == admin, "only admin");
}
```



Solved state explosion …
destroyed precision

Symvalic Analysis Adds Dependencies

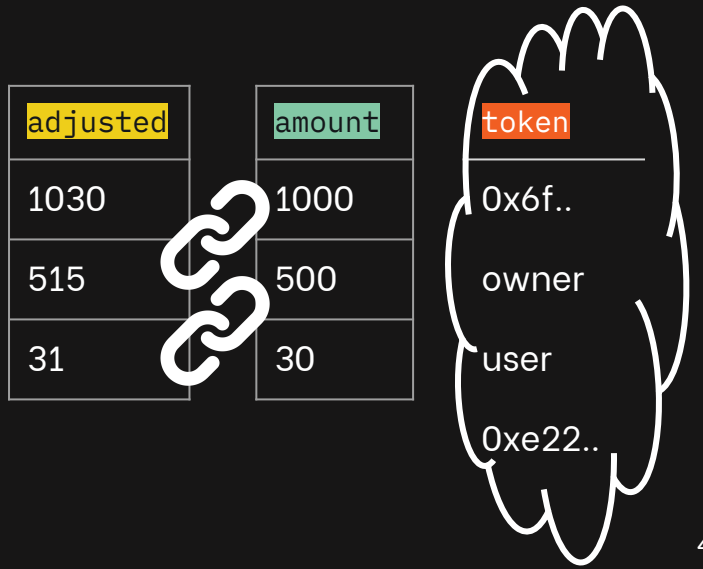
```
address admin ;  // set up at construction, not in contract code

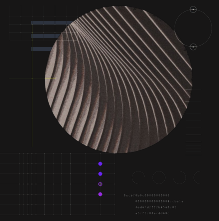
function withdrawToken (IERC20 token, uint256 amount, address sendTo) external {
  onlyAdmin();

  uint256 adjusted = amount * 103 / 100;

  if (amount >= 10000 && amount < 100000)
    token.transfer(sendTo,adjusted) ; …
}

function onlyAdmin() internal view {
  require (msg.sender == admin, "only admin");
}
```

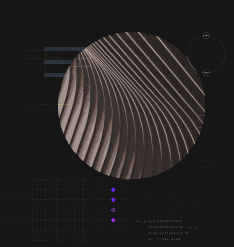




Symvalic Analysis Basics: Symbolic Expressions + Values

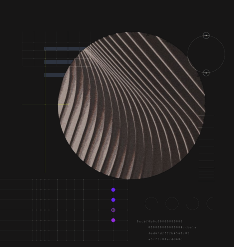
```
x    0x0
x    0x1
x    [LT, 0x0, <<owner-value>>]
x    [LT, <<owner-value>>, 0x0]
x    [LT, <<user1-value>>, 0x0]
y    0x0
y    0x1
y    0xc0
y    [AND, 0xff, [DIV, <<owner-value>>, 0x100]]
y    [ISZERO, [AND, 0xff, <<owner-value>>]]
```

Total: 52 479 rows



Symvalic Analysis with Dependencies

```
x    0x0                              [y -> 0x0, z -> 0x1]   [caller -> owner]
x    0x1                              [y -> 0x0, z -> 0x1]   [caller -> user]
x    [LT, 0x0, <<owner-value>>]       [y -> <<owner-value>>] [caller -> owner]
...
Total: 11 121 520 rows
```

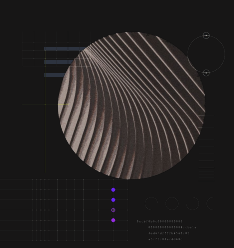


Symvalic Analysis with Dependencies

```
x    0x0                          [y -> 0x0, z -> 0x1]   [caller -> owner]
x    0x1                          [y -> 0x0, z -> 0x1]   [caller -> user]
x    [LT, 0x0, <<owner-value>>]   [y -> <<owner-value>>] [caller -> owner]
...
Total: 11 121 520 rows
```

Local dependencies

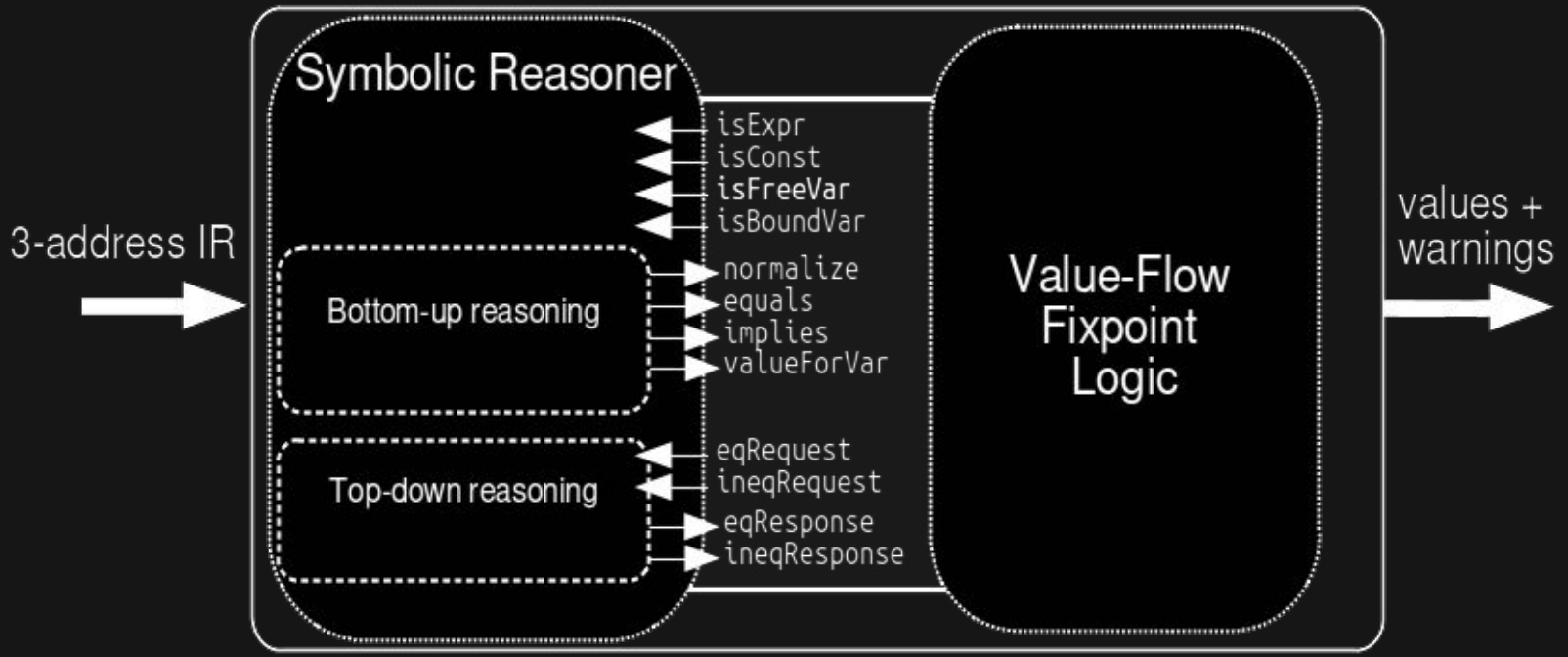
“Global” dependencies



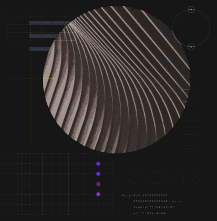
Approach

- Datalog-based analysis rules
- Appealing to symbolic solver/theorem prover also expressed as Datalog rules
- Limited top-down reasoning
  - “solve equations”
- Bottom up reasoning, up to bounded expression size

Architecture

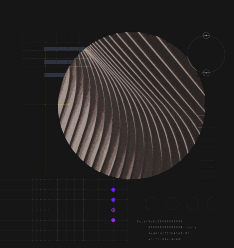


Second Weapon: Corpus Analysis



Static Analysis Gets Us Answers. What Is the Question?

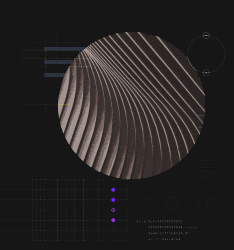
- Example: "*is the first argument of a* **`swap`** *call* ***tainted****?*"
- How do we know that `swap` is special?
  How do we know that the first argument has monetary significance?
- One answer: have humans specify
  - not optimal…



Corpus Analysis: Learn from What Deployed Contracts Do!

Which function signatures/arguments …

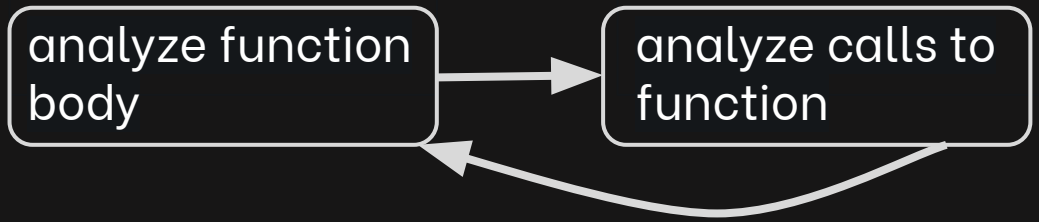
- typically have monetary significance
  - e.g., flow to `transfer`/`transferFrom`
- perform initialization
- do a `delegatecall`
- return values that can be manipulated by an untrusted caller (by changing contract state)
- allow reentrancy
- check permissions of their caller
- perform guarded/unguarded external calls of monetary significance
- …



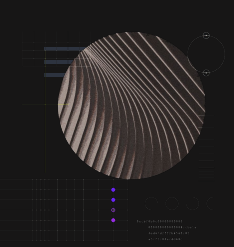
Corpus Analysis Summarizes Behavior in Two Ways

- *What does a contract do?*
  - so that its callers can be analyzed better
- *What is the usual behavior of a contract's/function's callers?*
  - so that callers can be analyzed for deviations

All Recursively, Non-Trivially



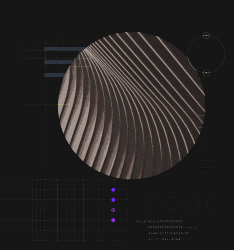
- E.g.,
  - **`bar`** is a function known to call functions on its second argument
  - (contract A) function **`foo`** makes external call to **`bar`** (contract X) with tainted second argument
  - (contract A) **`foo`** is reentrant



Overall

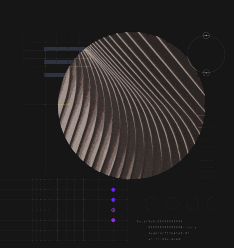
- Most corpus analysis insights are simple:
  - e.g., “*first argument of* **`swap`** *rarely tainted, but it is here*”
- We have the benefit of a service with all deployed Ethereum contracts
  - works well with Dedaub’s public tooling

Insights



Thoughts on Static Analysis in Industry vs Academic Research

- Industry: great when stupid solutions work well
- Academia: catastrophic
  - anecdotes: initializers, ML for parameter settings

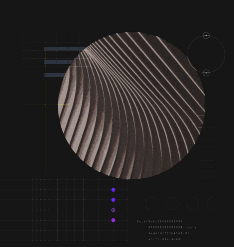


Consider Two Questions

For a ***good*** analysis:

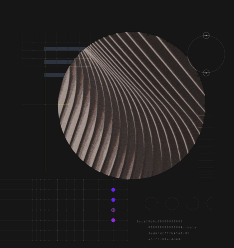
- Out of 100 contracts, how many would you expect to be flagged?

- Out of 100 flagged contracts, how many warnings do you expect to be valid?



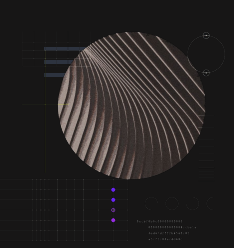
Thoughts on Static Analysis for Security

- Realistic warning rates?
  - AccessibleSelfDestruct: 5.11%
  - ArithmeticErrorHighConfidence: 0.43%
  - BadRandomness: 16.1%
  - BlockReachableByInconsistentAssertionPaths: 0.43%
  - CallToThis: 0.00%
  - FlashLoanCallbackUncheckedSender: 0.00%
  - NoChainidInECDSASignedData: 2.98%
  - ProxyForTransfer: 3.40%
  - ProxyForTransferFrom: 0.00%
  - ProxyForTransferFromLowConfidence: 1.70%
  - ProxyForTransferFromMediumConfidence: 0.43%
  - ReachableAssertionFailure: 23.83%
  - Reentrancy: 0.85%
  - SuspiciousFunctionCallScaling: 0.00%
  - TaintedDelegateCall: 0.85%
  - UniswapPriceManipulationPotentialHighConfidence: 0.00%
  - UniswapTaintedTokenHighConfidence: 0.00



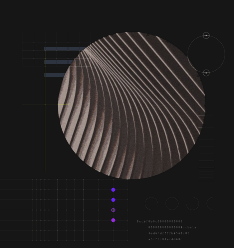
Thoughts on Static Analysis in Industry vs Academic Research

- We have the wrong metrics for anything that counts
- Warning rates at 0.5% seem useless
  - "*199 out of 200 contracts are already correct, why is it interesting to get that number to 200?*"
- But it's these warnings that find high-value vulnerabilities!



Thoughts on Static Analysis in Industry vs Academic Research

- We have the wrong metrics for anything that counts
- <60% precision in an analysis is hardly a publishable result!
- But even 5% precision is awesome for high-value vulnerabilities
  - $$$ 1-of-20 times!



Consider Two Questions

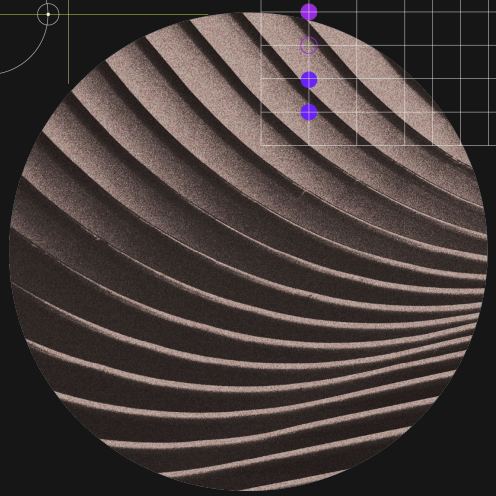
For a “good” analysis:

- Out of 100 contracts, how many would you expect to be flagged?

0.5 ?

- Out of 100 flagged contracts, how many warnings do you expect to be valid?

5 ?



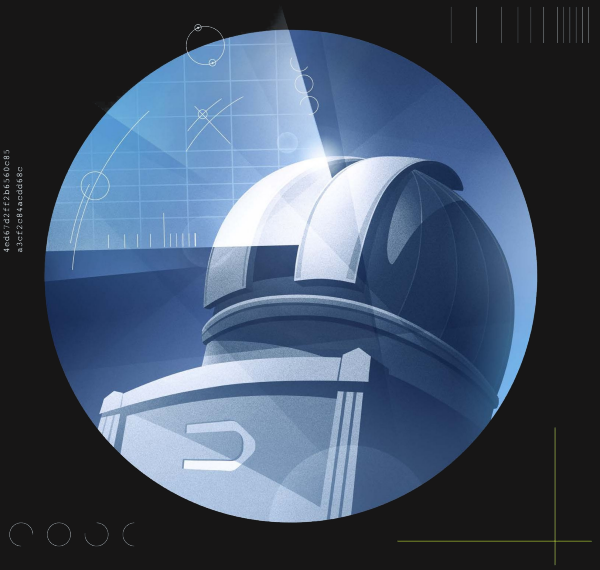
0xcbf0bc00000000000
0000000000000008ccba1a
4ed67d2f2b650c85
a3cf2c84acdd68c

That’s all, folks!

Yannis Smaragdakis

DEDAUB

library.dedaub.com



Research positions available (ERC)