Contracts for Contracts

Consolidating Smart Contracts with Behavioral Contracts

Guannan Wei, Danning Xie, Wuqi Zhang, Yongwei Yuan, Zhuo Zhang

PLDI 2024 @ Copenhagen, Denmark

PURDUE UNIVERSITY

PURPL

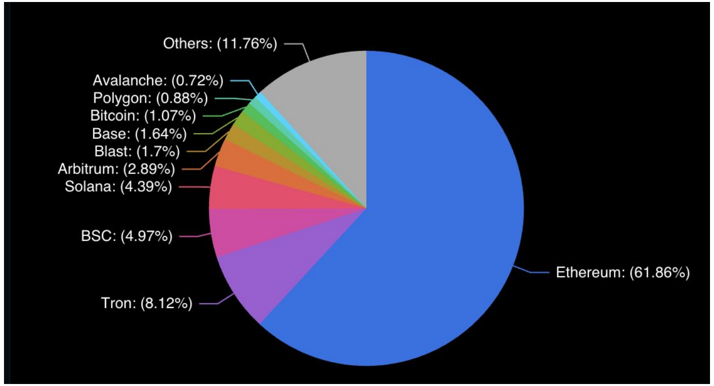
香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Smart Contracts

- **Smart contracts**
  - specifying assumptions/guarantees of transactions between business parties
  - autonomous programs signed and stored on a blockchain, enforcing the agreement between parties

Smart Contracts

- **Smart contracts**
  - specifying assumptions/guarantees of transactions between business parties
  - autonomous programs signed and stored on a blockchain, enforcing the agreement between parties
- **Solidity** - the most popular smart contract languages to build decentralized financial applications on the Ethereum blockchain



$59B total value locked on Ethereum

Smart Contracts - Solidity

- **Solidity** - the most popular smart contract languages to build decentralized financial applications on the Ethereum blockchain
- Bugs/attacks in Solidity programs can lead to real money loss!
  - Lack of clearly specified safety conditions.

Categories	Attacks		Bug Bounties	
	# Bugs	Fund loss	# Bugs	Bounties
Lending	1	$ 5,000K	2	$ 1,630K
Dexes	7	$ 13,950K	3	$ 65K
Yield	6	$ 20,300K	1	$ 10K
Services	3	$ 5,600K	2	$ 610K
Derivatives	-	-	2	$ 200K
Yield Aggregator	1	$ 2,100K	2	$ 300K
Real World Assets	2	$ 1,127K	1	$ 50K
Stablecoins	5	$211,360K	-	-
Indexes	-	-	1	$ 90K
NFT Marketplace	1	$ 20K	-	-
NFT Lending	2	$ 5,800K	-	-
Cross Chain	-	-	1	$10,000K
Others	-	-	1	$ 1,050K
Total	28	$265,257K	16	$14,005K

Zhang et al. ICSE 2023

Smart Contracts - Solidity

- **Solidity** - the most popular smart contract languages to build decentralized financial applications on the Ethereum blockchain
- Bugs/attacks in Solidity programs can lead to real money loss!
  - Lack of clearly specified safety conditions.

- ***Solidity has not provided enough mechanisms for programmers to specify & enforce safety conditions.***

Categories	Attacks		Bug Bounties	
	# Bugs	Fund loss	# Bugs	Bounties
Lending	1	$ 5,000K	2	$ 1,630K
Dexes	7	$ 13,950K	3	$ 65K
Yield	6	$ 20,300K	1	$ 10K
Services	3	$ 5,600K	2	$ 610K
Derivatives	-	-	2	$ 200K
Yield Aggregator	1	$ 2,100K	2	$ 300K
Real World Assets	2	$ 1,127K	1	$ 50K
Stablecoins	5	$211,360K	-	-
Indexes	-	-	1	$ 90K
NFT Marketplace	1	$ 20K	-	-
NFT Lending	2	$ 5,800K	-	-
Cross Chain	-	-	1	$10,000K
Others	-	-	1	$ 1,050K
Total	28	$265,257K	16	$14,005K

Zhang et al. ICSE 2023

Behavioral Contracts

- **Smart contracts**
  - specifying assumptions/guarantees of transactions between business parties
  - autonomous programs signed and stored on a blockchain, enforcing the agreement between parties
- **Behavioral contracts**
  - specifying assumptions/guarantees between software components

Behavioral Contracts

- **Smart contracts**
  - specifying assumptions/guarantees of transactions between business parties
  - autonomous programs signed and stored on a blockchain, enforcing the agreement between parties
- **Behavioral contracts**
  - specifying assumptions/guarantees between software components
  - allowing programmers to write executable specifications in the same host language
  - monitoring violations at runtime, further help with maintenance, debugging etc.

Behavioral Contracts

- **Smart contracts**
  - specifying assumptions/guarantees of transactions between business parties
  - autonomous programs signed and stored on a blockchain, enforcing the agreement between parties
- **Behavioral contracts**
  - specifying assumptions/guarantees between software components
  - allowing programmers to write executable specifications in the same host language
  - monitoring violations at runtime, further help with maintenance, debugging etc.

Applying "Design by Contract"

Bertrand Meyer

Interactive Software Engineering

As object-oriented techniques steadily gain ground in the world of software development, users and prospective users of these techniques are clamoring more and more loudly for a "methodology" of object-oriented software construction — or at least for some methodological guidelines. This article presents such guidelines, whose main goal is to help improve the reliability of software systems. *Reliability* is here defined as the combination of correctness and robustness or, more prosaically, as the absence of bugs.

Everyone developing software systems, or just using them, knows how pressing this question of reliability is in the current state of software engineering. Yet the rapidly growing literature on object-oriented analysis, design, and programming includes remarkably few contributions on how to make object-oriented software more reliable. This is surprising and regrettable, since at least three reasons justify devoting particular attention to reliability in the context of object-oriented development:

Smart Contracts + Behavioral Contracts

This Work - ConSol:
A linguistic extension to Solidity and a transpiler that supports expressing and monitoring rich pre/post-conditions as specifications.

- runtime validation
- improve readability & maintainability
- can be used to guide fuzzing, static verification, etc.

Smart Contracts + Behavioral Contracts

This Work - ConSol:

A linguistic extension to Solidity and a transpiler that supports expressing and monitoring rich pre/post-conditions as specifications.

```
function getPrice(address chainlink) returns (uint256) {
    (_, uint256 ethPrice, _, _, _) = chainlink.latestRoundData();
    return ORACLE.getRate() * ethPrice;
}
```

```
getPrice(a_chainlink_address)
```

Excerpted from the Sturdy contract

Smart Contracts + Behavioral Contracts

This Work - ConSol:
A linguistic extension to Solidity and a transpiler that supports expressing and monitoring rich pre/post-conditions as specifications.

```
getPrice(chainlink) returns (price)
ensures price * 0.95 < ORACLE.getLatestPrice() && price * 1.05 > ORACLE.getLatestPrice()
```

```
function getPrice(address chainlink) returns (uint256) {
    (_, uint256 ethPrice, _, _, _) = chainlink.latestRoundData();
    return ORACLE.getRate() * ethPrice;
}


getPrice(a_chainlink_address)
```

ensure the post-condition

Excerpted from the Sturdy contract

Smart Contracts + Behavioral Contracts

This Work - ConSol:

A linguistic extension to Solidity and a transpiler that supports expressing and monitoring rich pre/post-conditions as specifications.

```
getPrice(chainlink) returns (price)
ensures price * 0.95 < ORACLE.getLatestPrice() && price * 1.05 > ORACLE.getLatestPrice()
```

```
function getPrice(address chainlink) returns (uint256) {
    (_, uint256 ethPrice, _, _, _) = chainlink.latestRoundData();
    return ORACLE.getRate() * ethPrice;
}
```

ensure the post-condition

```
getPrice(a_chainlink_address)
```

check at call-sites during runtime

Excerpted from the Sturdy contract

Smart Contracts + Behavioral Contracts

This Work - ConSol:

A linguistic extension to Solidity and a transpiler that supports expressing and monitoring rich pre/post-conditions as specifications.

what if we want to say something about the argument `chainlink`?

getPrice(chainlink) returns (price)
*ensures price * 0.95 < ORACLE.getLatestPrice() && price * 1.05 > ORACLE.getLatestPrice()*

```
function getPrice(address chainlink) returns (uint256) {
    (_, uint256 ethPrice, _, _, _) = chainlink.latestRoundData();
    return ORACLE.getRate() * ethPrice;
}
```

- 160-bits integers

- first-class values

```
getPrice(a_chainlink_address)
```

Excerpted from the Sturdy contract

Smart Contracts + Behavioral Contracts

This Work - ConSol:

A linguistic extension to Solidity and a transpiler that supports expressing and monitoring rich pre/post-conditions as specifications.

what if we want to say something about the argument `chainlink`*?*

getPrice(chainlink) returns (price)
*ensures price * 0.95 < ORACLE.getLatestPrice() && price * 1.05 > ORACLE.getLatestPrice()*

```
function getPrice(address chainlink) returns (uint256) {
    (_, uint256 ethPrice, _, _, _) = chainlink.latestRoundData();
    return ORACLE.getRate() * ethPrice;
}
```

- *160-bits integers*
- *first-class values*
- *contains callable functions*

```
getPrice(a_chainlink_address)
```

Excerpted from the Sturdy contract

Smart Contracts + Behavioral Contracts

This Work - ConSol:
A linguistic extension to Solidity and a transpiler that supports expressing and monitoring rich pre/post-conditions as specifications.

what if we want to say something about the argument `chainlink`*?*

getPrice(chainlink) returns (price)
*ensures price * 0.95 < ORACLE.getLatestPrice() && price * 1.05 > ORACLE.getLatestPrice()*

```
function getPrice(address chainlink) returns (uint256) {
    (_, uint256 ethPrice, _, _, _) = chainlink.latestRoundData();
    return ORACLE.getRate() * ethPrice;
}
```

- *160-bits integers*
- *first-class values*
- *contains callable functions*

```
getPrice(a_chainlink_address)
```

cannot be checked when calling `getPrice`

Excerpted from the Sturdy contract

Smart Contracts + Behavioral Contracts

This Work - ConSol:
A linguistic extension to Solidity and a transpiler that supports expressing and monitoring rich pre/post-conditions as specifications.

★ Expressing and monitoring conditions for addresses and their latent calls.

Smart Contracts + Behavioral Contracts

This Work - ConSol:
A linguistic extension to Solidity and a transpiler that supports expressing and monitoring rich pre/post-conditions as specifications.

★ Expressing and monitoring conditions for addresses and their latent calls.

Contracts for Higher-Order Functions

Robert Bruce Findler[1] Matthias Felleisen
Northeastern University
College of Computer Science
Boston, Massachusetts 02115, USA

Abstract

Assertions play an important role in the construction of robust software. Their use in programming languages dates back to the 1970s. Eiffel, an object-oriented programming language, wholeheartedly adopted assertions and developed the "Design by Contract" philosophy. Indeed, the entire object-oriented community recognizes the value of assertion-based contracts on methods.

In contrast, languages with higher-order functions do not support assertion-based contracts. Because predicates on functions are,

1 Introduction

Dynamically enforced pre- and post-condition contracts have been widely used in procedural and object-oriented languages [11, 14, 17, 20, 21, 22, 25, 31]. As Rosenblum [27] has shown, for example, these contracts have great practical value in improving the robustness of systems in procedural languages. Eiffel [22] even developed an entire philosophy of system design based on contracts ("Design by Contract"). Although Java [12] does not support contracts, it is one of the most requested extensions.[1]

ICFP 2002

borrowing ideas from "Contracts for Higher-Order Functions", delaying the check of behaviors of functions as values

Specifying Conditions for Solidity Addresses

★ Expressing and monitoring conditions for addresses and their latent calls.

post-conditions of calling

chainlink

```
function getPrice(address chainlink) returns (uint256) {
    (_, uint256 ethPrice, _, uint256 updatedAt, _) = chainlink.latestRoundData();
    require(updatedAt > block.timestamp - 1 days);
    require(ethPrice > 0);
    return ORACLE.getRate() * ethPrice;
}
```

Excerpted from the Sturdy contract

Specifying Conditions for Solidity Addresses

★ Expressing and monitoring conditions for addresses and their latent calls.

```
getPrice(chainlink) returns (price)
ensures price * 0.95 < ORACLE.getLatestPrice() && price * 1.05 > ORACLE.getLatestPrice()
where {
+     chainlink.latestRoundData() returns (_, answer, _, updatedAt, _)
+     ensures updatedAt > block.timestamp − 1 days && answer > 0
}
```

post-conditions of calling `chainlink`

```
function getPrice(address chainlink) returns (uint256) {
    (_, uint256 ethPrice, _, uint256 updatedAt, _) = chainlink.latestRoundData();
-   require(updatedAt > block.timestamp - 1 days);
-   require(ethPrice > 0);
    return ORACLE.getRate() * ethPrice;
}
```

Excerpted from the Sturdy contract

Specifying Conditions for Solidity Addresses

★ Expressing and monitoring conditions for addresses and their latent calls.

```
getPrice(chainlink) returns (price)
ensures price * 0.95 < ORACLE.getLatestPrice() && price * 1.05 > ORACLE.getLatestPrice()
where {
+    chainlink.latestRoundData() returns (_, answer, _, updatedAt, _)
+    ensures updatedAt > block.timestamp − 1 days && answer > 0
}
```

post-conditions of calling

`chainlink`

```
function getPrice(address chainlink) returns (uint256) {
    (_, uint256 ethPrice, _, uint256 updatedAt, _) = chainlink.latestRoundData();
    return ORACLE.getRate() * ethPrice;
}
```

Excerpted from the Sturdy contract

Specifying Conditions for Solidity Addresses

★ Expressing and monitoring conditions for addresses and their latent calls.

★ **Good for** decoupling repeated, low-level checks from business logic.

Specifying Conditions for Solidity Addresses

★ Expressing and monitoring conditions for addresses and their latent calls.

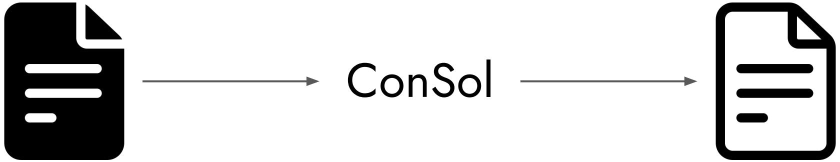
★ **Good for** decoupling repeated, low-level checks from business logic.

? **Need to** know when to perform the checks of address calls.

Challenge*: addresses are first-class values and can flow around!*

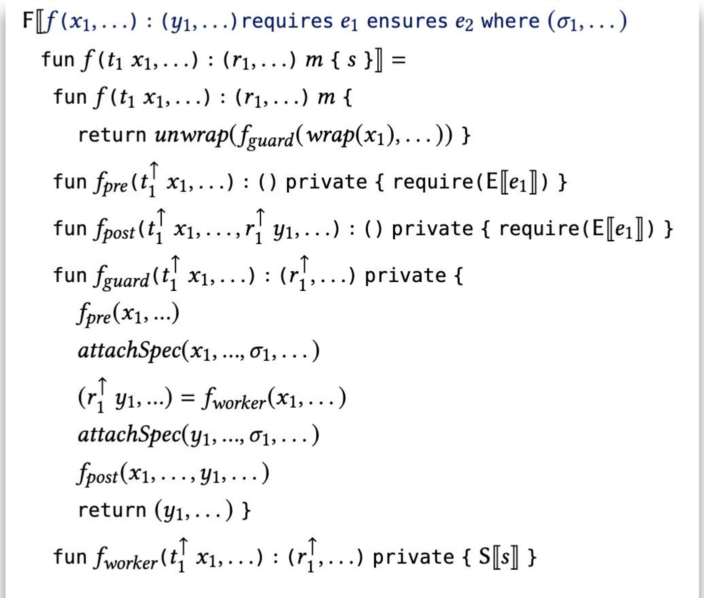
Implementation

- ConSol implements a whole-program transformation.



Solidity programs
with spec annotations.

Ordinary Solidity
programs.



$\mathsf{F}[\![f\,(x_1,\ldots) : (y_1,\ldots)$ `requires` e_1 `ensures` e_2 `where` $(\sigma_1,\ldots)$

`fun` $f\,(t_1\; x_1,\ldots) : (r_1,\ldots)\; m\; \{\; s\; \}]\!] =$

`fun` $f\,(t_1\; x_1,\ldots) : (r_1,\ldots)\; m\; \{$

`return` $unwrap(f_{guard}(wrap(x_1),\ldots))\; \}$

`fun` $f_{pre}(t_1^{\uparrow}\; x_1,\ldots) : ()$ `private` $\{$ `require`$(\mathsf{E}[\![e_1]\!])\; \}$

`fun` $f_{post}(t_1^{\uparrow}\; x_1,\ldots, r_1^{\uparrow}\; y_1,\ldots) : ()$ `private` $\{$ `require`$(\mathsf{E}[\![e_1]\!])\; \}$

`fun` $f_{guard}(t_1^{\uparrow}\; x_1,\ldots) : (r_1^{\uparrow},\ldots)$ `private` $\{$

$f_{pre}(x_1, ...)$

$attachSpec(x_1, ..., \sigma_1, \ldots)$

$(r_1^{\uparrow}\; y_1, ...) = f_{worker}(x_1,\ldots)$

$attachSpec(y_1, ..., \sigma_1, \ldots)$

$f_{post}(x_1,\ldots,y_1,\ldots)$

`return` $(y_1,\ldots)\; \}$

`fun` $f_{worker}(t_1^{\uparrow}\; x_1,\ldots) : (r_1^{\uparrow},\ldots)$ `private` $\{\; \mathsf{S}[\![s]\!]\; \}$

Fig. 4. The translation semantics of λ_{ConSol} (functions).

more details in the paper

Implementation - Addresses

- Change the value representation of addresses so that we can encode additional information about conditions.
- When calling the addresses, we decode and decide what condition of the address call should be checked.

```
F⟦f (x₁, ...) : (y₁, ...) requires e₁ ensures e₂ where (σ₁, ...)
  fun f (t₁ x₁, ...) : (r₁, ...) m { s }⟧ =
   fun f (t₁ x₁, ...) : (r₁, ...) m {
     return unwrap(f_guard(wrap(x₁), ...)) }
   fun f_pre(t₁↑ x₁, ...) : () private { require(E⟦e₁⟧) }
   fun f_post(t₁↑ x₁, ..., r₁↑ y₁, ...) : () private { require(E⟦e₁⟧) }
   fun f_guard(t₁↑ x₁, ...) : (r₁↑, ...) private {
     f_pre(x₁, ...)
     attachSpec(x₁, ..., σ₁, ...)
     (r₁↑ y₁, ...) = f_worker(x₁, ...)
     attachSpec(y₁, ..., σ₁, ...)
     f_post(x₁, ..., y₁, ...)
     return (y₁, ...) }
   fun f_worker(t₁↑ x₁, ...) : (r₁↑, ...) private { S⟦s⟧ }
```

Fig. 4. The translation semantics of λ_{ConSol} (functions).

more details in the paper

Implementation - Addresses

- Change the value representation of addresses so that we can encode additional information about conditions.

- When calling the addresses, we decode and decide what condition of the address call should be checked.

```
// spec for addr omitted
function f(address addr) {
  ...
  addr.g(x)
  ...
}
```

translates to

```
function f(guarded_address addr) {
  addr = attach_spec(addr, [encoded_spec])
  ...
  dispatch_g(addr, x)
  ...
}
```

Implementation - Addresses

- Change the value representation of addresses so that we can encode additional information about conditions.

- When calling the addresses, we decode and decide what condition of the address call should be checked.

```
// spec for addr omitted
function f(address addr) {
  ...
  addr.g(x)
  ...
}
```

translates to

```
function f(guarded_address addr) {
  addr = attach_spec(addr, [encoded_spec])
  ...
  dispatch_g(addr, x)
  ...
}
```

rewrite call-site, decide what needs to be checked

Implementation - Addresses

- Change the value representation of addresses so that we can encode additional information about conditions.

- When calling the addresses, we decode and decide what condition of the address call should be checked.

★ **Must be** careful for efficiency due to EVM's cost model!
Storing additional information causes more transaction fees.

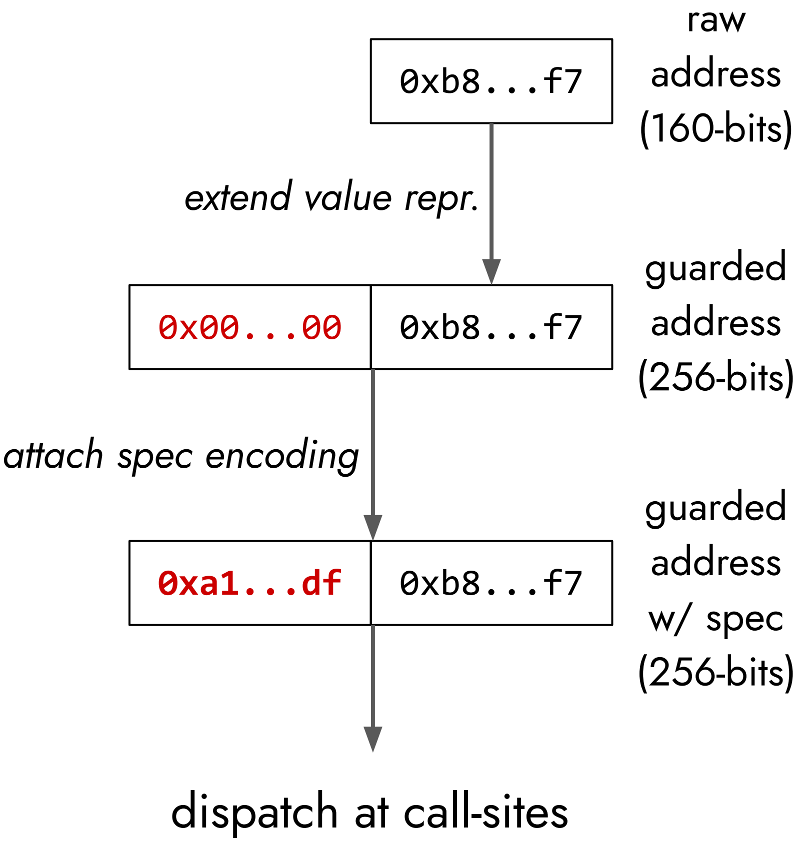
```
// spec for addr omitted
function f(address addr) {
  ...
  addr.g(x)
  ...
}
```

translates to

```
function f(guarded_address addr) {
  addr = attach_spec(addr, [encoded_spec])
  ...
  dispatch_g(addr, x)
  ...
}
```

Implementation - Addresses

- **Must be** careful for efficiency due to EVM's cost model!
  Storing additional information causes more transaction fees, *but the smallest unit of storage is 256 bits!*

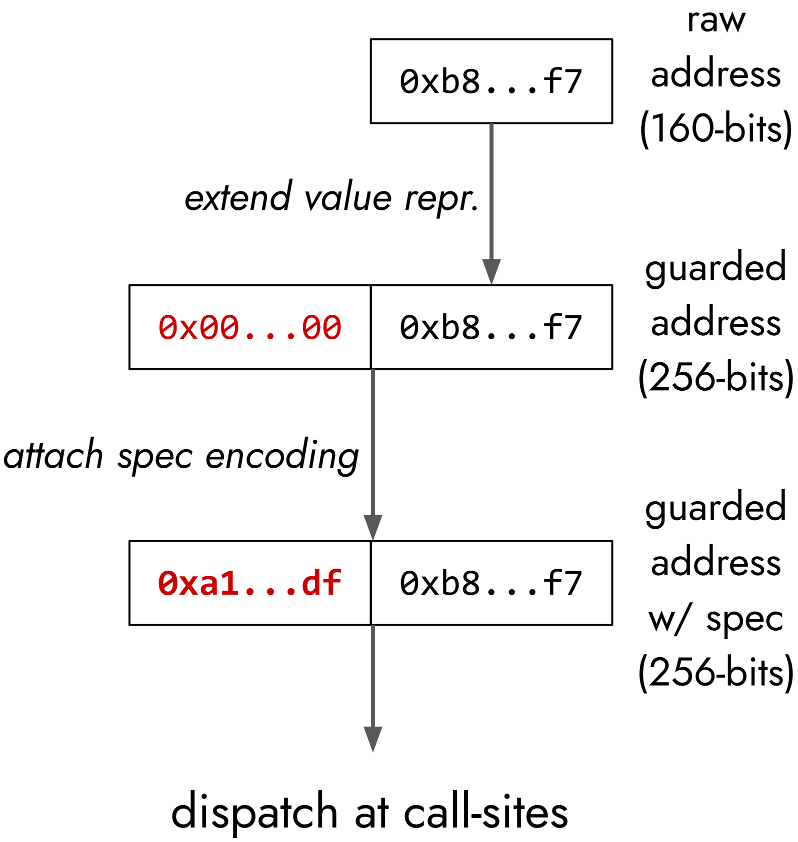


Implementation - Addresses

- **Must be** careful for efficiency due to EVM's cost model!
  Storing additional information causes more transaction fees, *but the smallest unit of storage is 256 bits!*

- **Solution:** bit-stealing
  extend 160-bits address values to 256-bits, use the 96-MSBs to encode spec provenance information.

  very little overhead!



Evaluation

- Can we use ConSol to express security defenses and decouple them from the main business logic?
- How much overhead is introduced in ConSol-translated programs?

Evaluation - Effectiveness

- Can we use ConSol to express security defenses and decouple them from the main business logic?

- Case studies:
  20 real-world attacks
  154.32M loss in total

Table 1. Summary of studied cases. *LR* denotes LoC Reduced.

Project	Date	Loss ($)	Root Cause of Vulnerability	LR (%)
Qubit [17]	01-28-22	80M	Zero Address Function Call	15.38
TecraSpace [80]	02-04-22	63K	Any Token is Destroyed	50.00
Umbrella [86]	03-20-22	700K	Integer Over/Underflow	33.33
XCarnival [90]	06-26-22	3.87M	Infinite Number of Loans	26.32
BadGuys [65]	09-02-22	NFT	Missing Airdrop Eligibility Check	94.12
EFLeverVault [50]	10-14-22	1M	Business Logic Flaw	25.00
N00d [8]	10-26-22	29K	Reentrancy	11.11
Dexible [61]	02-17-23	1.5M	Arbitrary External Call	11.76
SushiSwap [72]	04-09-23	3.3M	Unchecked User Input	54.55
SwaposV2 [18]	04-16-23	468K	Erroneous Accounting	25.00
Unknown [81]	05-31-23	111K	Missing Slippage Check	30.00
Sturdy [9]	06-12-23	800K	Readonly Reentrancy	57.14
LEVUSDC [28]	06-15-23	105K	Access Control	33.33
AzukiDAO [70]	07-03-23	69K	Invalid Signature Verification	48.15
Bao [6]	07-04-23	46K	Inflation Manipulate	83.33
Miner [54]	02-15-24	466K	Lack of Validation	83.33
YearnFinance [5]	04-13-23	11.6M	Misconfiguration	-
ZunamiProtocol [44]	08-14-23	2M	Price Manipulation	-
KyberSwap [10]	11-22-23	48M	Precision Loss	-
Time [67]	12-06-23	188.9K	Arbitrary Address Spoofing Attack	-

Evaluation - Effectiveness

- Can we use ConSol to express security defenses and decouple them from the main business logic?
- Case studies:
  20 real-world attacks
  154.32M loss in total
- Effective as low-level assertions, simplify the code with better readability

Table 1. Summary of studied cases. *LR* denotes LoC Reduced.

Project	Date	Loss ($)	Root Cause of Vulnerability	LR (%)
Qubit [17]	01-28-22	80M	Zero Address Function Call	15.38
TecraSpace [80]	02-04-22	63K	Any Token is Destroyed	50.00
Umbrella [86]	03-20-22	700K	Integer Over/Underflow	33.33
XCarnival [90]	06-26-22	3.87M	Infinite Number of Loans	26.32
BadGuys [65]	09-02-22	NFT	Missing Airdrop Eligibility Check	94.12
EFLeverVault [50]	10-14-22	1M	Business Logic Flaw	25.00
N00d [8]	10-26-22	29K	Reentrancy	11.11
Dexible [61]	02-17-23	1.5M	Arbitrary External Call	11.76
SushiSwap [72]	04-09-23	3.3M	Unchecked User Input	54.55
SwaposV2 [18]	04-16-23	468K	Erroneous Accounting	25.00
Unknown [81]	05-31-23	111K	Missing Slippage Check	30.00
Sturdy [9]	06-12-23	800K	Readonly Reentrancy	57.14
LEVUSDC [28]	06-15-23	105K	Access Control	33.33
AzukiDAO [70]	07-03-23	69K	Invalid Signature Verification	48.15
Bao [6]	07-04-23	46K	Inflation Manipulate	83.33
Miner [54]	02-15-24	466K	Lack of Validation	83.33
YearnFinance [5]	04-13-23	11.6M	Misconfiguration	-
ZunamiProtocol [44]	08-14-23	2M	Price Manipulation	-
KyberSwap [10]	11-22-23	48M	Precision Loss	-
Time [67]	12-06-23	188.9K	Arbitrary Address Spoofing Attack	-

Evaluation - Efficiency

- How much overhead is introduced in ConSol-translated programs?

- Benchmark programs:
  - 16 real-world attacks
  - 23 contracts from ERC20, ERC721, and ERC1202 (collected by Li et al, PLDI 20)
- Baseline: low-level assertion-patched contracts

Evaluation - Efficiency

- How much overhead is introduced in ConSol-translated programs?

- Results on 16 attacks:

  0.207% more gas consumption

Project	#Tx	by Assertions GFI ($)	by Assertions GIR (%)	by ConSol GFI ($)	by ConSol GIR (%)
Qubit [17]	0	-	-	-	-
TecraSpace [80]	4245	0.000	0.000	0.000	0.000
Umbrella [86]	58	0.001	0.111	0.001	0.015
XCarnival [90]	365	0.016	0.029	0.040	0.072
BadGuys [65]	950	0.003	0.096	0.005	0.166
EFLeverVault [50]	21	0.027	0.089	0.031	0.102
N00d [8]	111	0.009	0.547	0.009	0.571
Dexible [61]	54	0.126	0.230	0.178	0.324
SushiSwap [72]	202	0.007	0.099	0.007	0.106
SwaposV2 [18]	7	0.003	0.048	0.004	0.068
Unknown [81]	10	0.381	0.002	0.438	0.003
Sturdy [9]	23	0.940	1.126	0.941	1.128
LEVUSDC [28]	45	0.008	0.042	0.008	0.044
AzukiDAO [70]	2937	0.019	0.227	0.022	0.257
Bao [6]	15	0.001	0.005	0.002	0.018
Miner [54]	3922	0.002	0.007	0.011	0.030
Avg.	-	0.110	0.190	0.121	0.207

Evaluation - Efficiency

- How much overhead is introduced in ConSol-translated programs?

- Results on 16 attacks:

  0.207% more gas consumption

- Results on ERC20/721/1202: **0.290%** more gas consumption

Contract	BEC	USDT	ZRX	THETA	INB	HEDG	DAI	EKT	XIN	HOT	SWP	VOTE
Original	960,999	62,426	51,468	51,540	53,738	53,941	53,696	51,911	51,375	51,525	55,728	210,395
ConSol	965,478	62,910	51,468	51,777	53,986	54,110	53,865	52,307	51,609	51,773	55,886	210,543
GIR (%)	0.47	0.78	0.00	0.46	0.46	0.31	0.31	0.76	0.46	0.48	0.28	0.07
LR (%)	39.10	30.83	0.00	38.89	38.89	50.00	50.00	40.83	34.44	44.44	50.00	20.83

Contract	DOZ	MCHH	CC	CLV	LAND	CARDS	KB	TRINK	PACKS	BKC	EGG
Original	2,163,066	221,235	214,598	246,986	215,732	214,770	214,731	214,484	260,566	301,808	215,428
ConSol	2,166,247	221,526	215,082	247,471	216,201	215,123	215,079	214,965	260,920	302,199	215,791
GIR (%)	0.15	0.13	0.23	0.20	0.22	0.16	0.16	0.22	0.14	0.13	0.17
LR (%)	43.19	33.33	35.71	41.67	33.33	33.81	37.05	35.71	33.33	37.05	37.50

Summary

- **ConSol**: Behavioral Contracts for Smart Contracts
- **Expressive**: specifying and monitoring behaviors of latent address calls
- **Efficient**: marginal gas consumption on Ethereum

- Prototype implementation and evaluation data: https://github.com/Kraks/contract-for-contract

Thank you!