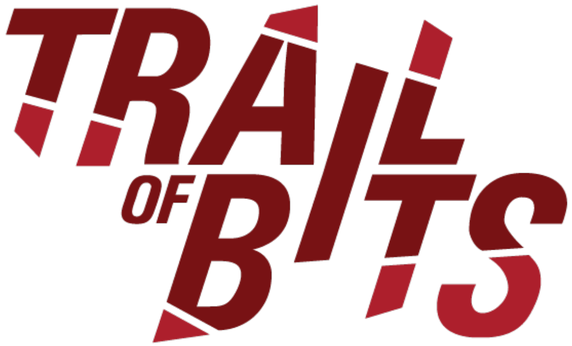
Fantastic Bugs and How to Squash Them;

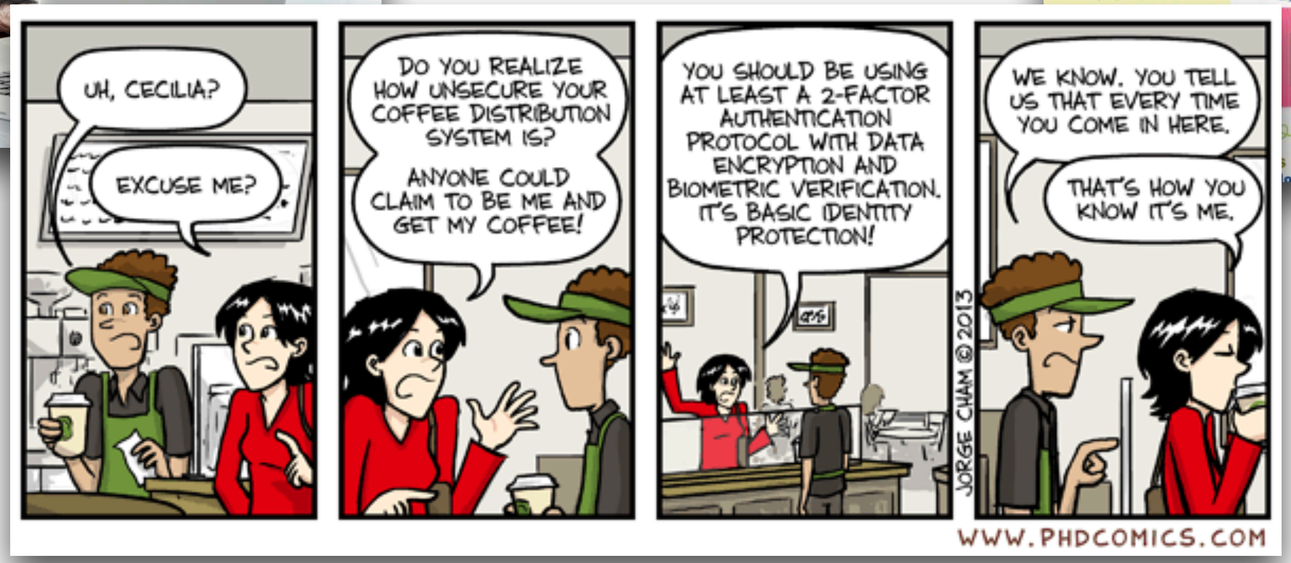
or, the Crimes of Solidity



Evan Sultanik

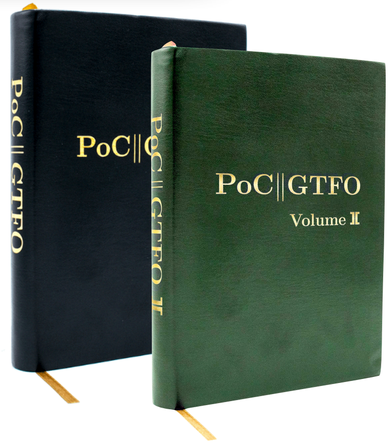
@ESultanik

`whoami`



TRAIL OF BITS

Drexel UNIVERSITY



What should you take away from this talk?

What should you take away from this talk?

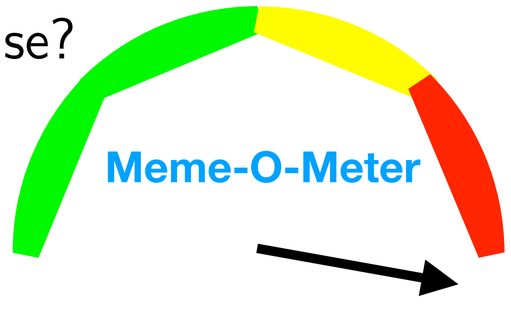
- Experienced Ethereum developers
  - ✓ Learn from the most common mistakes of your peers
  - ✓ Learn new tooling for improving your SDLC

What should you take away from this talk?

- Experienced Ethereum developers
  - ✓ Learn from the most common mistakes of your peers
  - ✓ Learn new tooling for improving your SDLC
- Programmers who are new to smart contracts
  - ✓ Learn what *not* to do
  - ✓ Learn what *to do*

What should you take away from this talk?

- Experienced Ethereum developers
  - ✓ Learn from the most common mistakes of your peers
  - ✓ Learn new tooling for improving your SDLC
- Programmers who are new to smart contracts
  - ✓ Learn what *not* to do
  - ✓ Learn what *to do*
- People interested in the technology
  - ✓ Learn about the state of the ecosystem
- Everyone else?



Outline

- Solidity the Language
- Solidity Implementation and Tooling
- On the Horizon
- Bugs!
- What You Can Do About It

Outline

- Solidity the Language
- Solidity Implementation and Tooling
- On the Horizon
- Bugs!
- What You Can Do About It

Solidity, the Language

Programming Language Checklist
by Colin McMillen, Jason Reed, and Elly Jones.

You appear to be advocating a new:
[ ] functional [ ] imperative [X] object-oriented [X] procedural [X] stack-based
[ ] "multi-paradigm" [ ] lazy [ ] eager [X] statically-typed [ ] dynamically-typed
[ ] pure [ ] impure [ ] non-hygienic [ ] visual [ ] beginner-friendly
[X] non-programmer-friendly [ ] completely incomprehensible
programming language. Your language will not work. Here is why it will not work.

Your language will not work.

You appear to believe that:
[ ] Syntax is what makes programming difficult
[ ] Garbage collection is free [ ] Computers have infinite memory
[X] Nobody really needs:
    [ ] concurrency [ ] a REPL [X] debugger support [ ] IDE support [ ] I/O
    [ ] to interact with code not written in your language
[ ] The entire world speaks 7-bit ASCII
[ ] Scaling up to large software projects will be easy
[ ] Convincing programmers to adopt a new language will be easy
[ ] Convincing programmers to adopt a language-specific IDE will be easy
[ ] Programmers love writing lots of boilerplate
[ ] Specifying behaviors as "undefined" means that programmers won't rely on them
[X] "Spooky action at a distance" makes programming more fun

You appear to believe that:
[ ] Syntax is what makes programming difficult
[ ] Garbage collection is free [ ] Computers have infinite memory
[X] Nobody really needs:
[ ] concurrency [ ] a REPL [X] debugger support [ ] IDE support [ ] I/O
[ ] to interact with code not written in your language
[ ] The entire world speaks 7-bit ASCII
[ ] Scaling up to large software projects will be easy
[ ] Convincing programmers to adopt a new language will be easy
[ ] Convincing programmers to adopt a language-specific IDE will be easy
[ ] Programmers love writing lots of boilerplate
[ ] Specifying behaviors as "undefined" means that programmers won't rely on them
[X] "Spooky action at a distance" makes programming more fun

Unfortunately, your language (has/lacks):
[ ] comprehensible syntax [ ] semicolons [ ] significant whitespace [ ] macros
[ ] implicit type conversion [ ] explicit casting [X] type inference
[ ] goto [ ] exceptions [X] closures [ ] tail recursion [ ] coroutines
[ ] reflection [X] subtyping [ ] multiple inheritance [X] operator overloading
[ ] algebraic datatypes [X] recursive types [ ] polymorphic types
[ ] covariant array typing [X] monads [ ] dependent types
[ ] infix operators [ ] nested comments [ ] multi-line strings [X] regexes
[ ] call-by-value [ ] call-by-name [ ] call-by-reference [ ] call-cc

The following philosophical objections apply:

The following philosophical objections apply:
[ ] Programmers should not need to understand category theory to write "Hello, World!"
[ ] Programmers should not develop RSI from writing "Hello, World!"
[ ] The most significant program written in your language is its own compiler
[ ] The most significant program written in your language isn't even its own compiler
[X] No language spec
[X] "The implementation is the spec"
  [ ] The implementation is closed-source [ ] covered by patents [ ] not owned by you
[X] Your type system is unsound [X] Your language cannot be unambiguously parsed
  [X] a proof of same is attached
  [ ] invoking this proof crashes the compiler
[ ] The name of your language makes it impossible to find on Google
[ ] Interpreted languages will never be as fast as C
[ ] Compiled languages will never be "extensible"
[ ] Writing a compiler that understands English is AI-complete
[ ] Your language relies on an optimization which has never been shown possible
[ ] There are less than 100 programmers on Earth smart enough to use your language
[ ] _____ takes exponential time
[ ] _____ is known to be undecidable

Your implementation has the following flaws:
[ ] CPUs do not work that way

Your implementation has the following flaws:

- [ ] CPUs do not work that way
- [ ] RAM does not work that way
- [ ] VMs do not work that way
- [X] Compilers do not work that way
- [ ] Compilers cannot work that way
- [ ] Shift-reduce conflicts in parsing seem to be resolved using rand()
- [ ] You require the compiler to be present at runtime
- [ ] You require the language runtime to be present at compile-time
- [X] Your compiler errors are completely inscrutable
- [X] Dangerous behavior is only a warning
- [ ] The compiler crashes if you look at it funny
- [ ] The VM crashes if you look at it funny
- [X] You don't seem to understand basic optimization techniques
- [X] You don't seem to understand basic systems programming
- [ ] You don't seem to understand pointers
- [ ] You don't seem to understand functions

Additionally, your marketing has the following problems:

- [ ] Unsupported claims of increased productivity
- [ ] Unsupported claims of greater "ease of use"
- [ ] Obviously rigged benchmarks
    - [ ] Graphics, simulation, or crypto benchmarks where your code just calls handwritten assembly through your FFI

Additionally, your marketing has the following problems:

[ ] Unsupported claims of increased productivity
[ ] Unsupported claims of greater "ease of use"
[ ] Obviously rigged benchmarks
    [ ] Graphics, simulation, or crypto benchmarks where your code just calls handwritten assembly through your FFI
    [ ] String-processing benchmarks where you just call PCRE
    [ ] Matrix-math benchmarks where you just call BLAS
[ ] Noone really believes that your language is faster than:
    [ ] assembly [ ] C [ ] FORTRAN [ ] Java [ ] Ruby [ ] Prolog
[ ] Rejection of orthodox programming-language theory without justification
[ ] Rejection of orthodox systems programming without justification
[ ] Rejection of orthodox algorithmic theory without justification
[ ] Rejection of basic computer science without justification

Taking the wider ecosystem into account, I would like to note that:

[ ] Your complex sample code would be one line in: _____
[ ] We already have an unsafe imperative language
[ ] We already have a safe imperative OO language
[ ] We already have a safe statically-typed eager functional language
[ ] You have reinvented Lisp but worse
[X] You have reinvented Javascript but worse
[ ] You have reinvented Java but worse
[ ] You have reinvented C++ but worse
[ ] You have reinvented PHP but worse

[ ] assembly [ ] C [ ] FORTRAN [ ] Java [ ] Ruby [ ] Prolog
[ ] Rejection of orthodox programming-language theory without justification
[ ] Rejection of orthodox systems programming without justification
[ ] Rejection of orthodox algorithmic theory without justification
[ ] Rejection of basic computer science without justification

Taking the wider ecosystem into account, I would like to note that:
[ ] Your complex sample code would be one line in: _____
[ ] We already have an unsafe imperative language
[ ] We already have a safe imperative OO language
[ ] We already have a safe statically-typed eager functional language
[ ] You have reinvented Lisp but worse
[X] You have reinvented Javascript but worse
[ ] You have reinvented Java but worse
[ ] You have reinvented C++ but worse
[ ] You have reinvented PHP but worse
[ ] You have reinvented PHP better, but that's still no justification
[ ] You have reinvented Brainfuck but non-ironically

In conclusion, this is what I think of you:
[X] You have some interesting ideas, but this won't fly.
[X] This is a bad language, and you should feel bad for inventing it.
[X] Programming in this language is an adequate punishment for inventing it.

In conclusion, this is what I think of you:

[X] You have some interesting ideas, but this won't fly.

[X] This is a bad language, and you should feel bad for inventing it.

[X] Programming in this language is an adequate punishment for inventing it.

```
if(1 | 0 < 1) {
    /* case 1 */
} else {
    /* case 2 */
}
```

```
if(1 | 0 < 1) {
```

C, C++, Javascript, Java, ...

```
} else {
    /* case 2 */
}
```

```
if(1 | 0 < 1) {
```

C, C++, Javascript, Java, ...

```
} else {
```

Such Language!

Solidity

Much Bugs!

WOW!

```
}
```

LEEEROYYY JENKINS!

```
if(1 | 0 < 1) {
```

C, C++, Javascript, Java, ...

```
} else {
```

Such Language!

Solidity

Much Bugs!

```
}
```

WOW!

LEEEROY

Lesson: Don't assume Solidity behaves like most other languages!

```
for (var i = 0; i < foo.length; ++i) {
    foo[i] = i;
}
```

What does `foo[1337]` look like after this?

```
for (var i = 0; i < foo.length; ++i) {
    foo[i] = i;
}
```

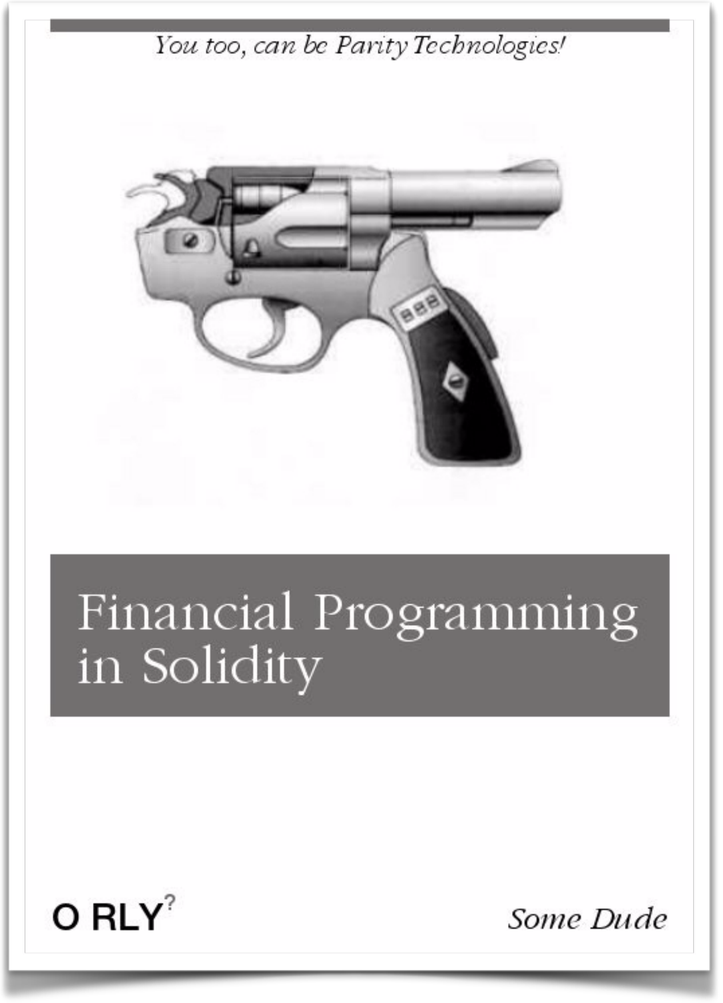
What does `foo[1337]` look like after this?

Lesson: Always use explicit types!

How to Write a Solidity Parser

(1) ☕

(2) Look up the official grammar



How to Write a Solidity Parser

(1) ☕

(2) Look up the official grammar

(3) 🍺

(4) Struggle to get a parser generator to accept it

How to Write a Solidity Parser

(1) ☕

(2) Look up the official grammar

(3) 🍺

(4) Struggle to get a parser generator to accept it

(5) 🍸

(6) Discover that the grammar is not correct

How to Write a Solidity Parser

(1) ☕

(2) Look up the official grammar

(3) 🍺

(4) Struggle to get a parser generator to accept it

(5) 🍸

(6) Discover that the grammar is not correct

(7) 💊

(8) Discover that existing parsers were #YOLO'd by hand

How to Write a Solidity Parser

(1) ☕

(2) Look up the official grammar

(3) 🍺

(4) Struggle to get a parser generator to accept it

(5) 🍸

(6) Discover that the grammar is not correct

(7) 💊

(8) Discover that existing parsers were #YOLO'd by hand

(9) 💉⚰️

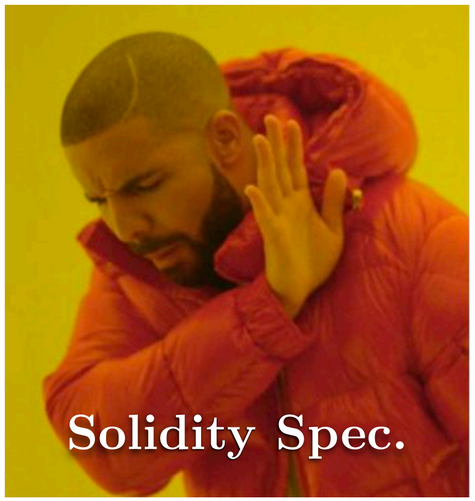
One Does Not Simply Implement the Shunting Yard Algorithm

One Does Not Simply

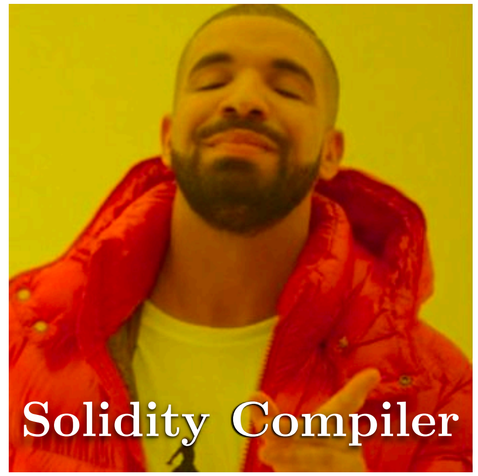
Implement the Shunting Yard Algorithm

```
contract C{
    struct myStruct{
        function(uint) my_func;
    }

    function test(){
        myStruct m;

        m.my_func = call_log;
        m.my_func(0);

        m.my_func = call_log2;
        m.my_func(0);
    }

    function call_log(uint a){
        Log(a);
    }

    function call_log2(uint a){
        Log2(a);
    }

    event Log(uint);
    event Log2(uint);
}
```

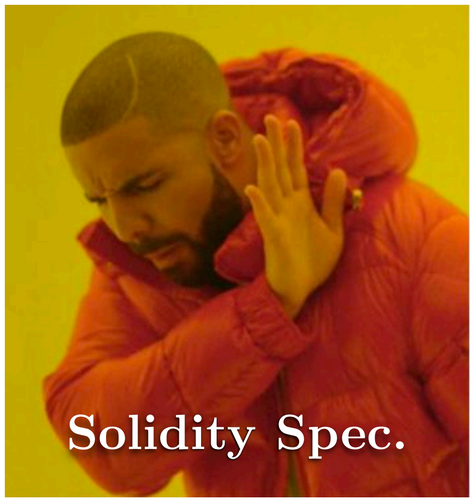
a struct that contains a pointer to a function



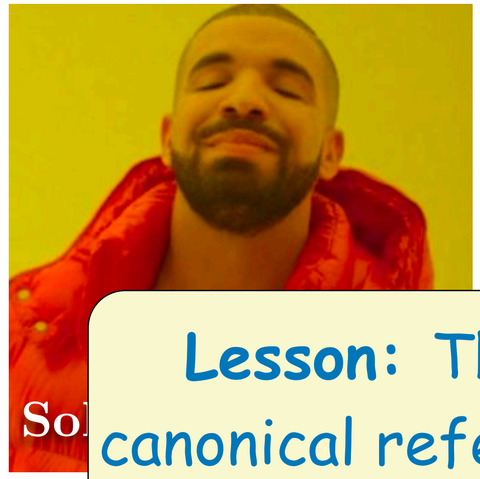

```
contract C{
    struct myStruct{
        function(uint) my_func;
    }

    function test(){
        myStruct m;

        m.my_func = call_log;
        m.my_func(0);

        m.my_func = call_log2;
        m.my_func(0);
    }

    function call_log(uint a){
        Log(a);
    }

    function call_log2(uint a){
        Log2(a);
    }

    event Log(uint);
    event Log2(uint);
}
```



a struct that contains a pointer to a function




```
contract C{
    struct myStruct{
        function(uint) my_func;
    }

    function test(){
        myStruct m;

        m.my_func = call_log;
        m.my_func(0);

        m.my_func = call_log2;
        m.my_func(0);
    }

    function call_log(uint a){
        Log(a);
    }

    function call_log2(uint a){
        Log2(a);
    }

    event Log(uint);
    event Log2(uint);
}
```



Lesson: The sole, canonical reference for Solidity is the source code of its sole compiler.

Solidity Parsing Using SmaCC: Challenges and Irregularities

Henrique Rocha Stephane Ducasse
Marcus Denker
INRIA Lille - Nord Europe
{henrique.rocha, stephane.ducasse,
marcus.denker}@inria.fr

Jason Lecerf
CEA-List
jason.clement.lecerf@gmail.com

Abstract

Solidity is a language used to implement smart contracts on a blockchain platform. Since its initial conception in 2014, Solidity has evolved into one of the major languages for the Ethereum platform as well as other blockchain technologies. Due to its popularity, there are many tools specifically designed to handle smart contracts written in Solidity. However, there is a lack of tools for Pharo to handle Solidity contracts. Therefore, we implemented a parser using SmaCC to serve as a base for further developing Solidity support in Pharo. In this paper we describe the parser creation, the irregularities we found in the Solidity grammar specification, and common practices on how to adapt the grammar to an LR type parser. Our experiences with parsing the Solidity language using SmaCC may help other developers trying to convert similar grammars.

Keywords Solidity, Parser, SmaCC, Blockchain, Ethereum.

1. Introduction

The Blockchain technology attracted a lot attention recently [LCO+16]. Blockchain is a distributed database, managed by a peer-to-peer network that stores a list of blocks or records. Ethereum [Eth14], and BitCoin [Nak09] are examples of blockchain technologies. Blockchains can be used for many applications such as cryptocurrency, digital wallets, adhoc networks, and remote transactions [LCO+16, HL16, LMH16, Dzi15, Eth14, Nak09]. One notable application of blockchain is the execution of smart contracts [LCO+16].

Smart contracts are what embedded procedures are for databases: programs executed in the blockchain to manage and transfer digital assets. When used in platforms like Ethereum, the contract language is Turing-complete [BDLF+16]. Therefore, smart contracts can be used in many different scenarios [LCO+16]. For example, there are smart contracts employed to simple storage [Eth17], and outsourced computation [LTKS15].

Solidity [Eth17] is a programming language loosely based on JavaScript, and it is used to specify smart contracts on blockchain platforms. Solidity was originally designed to be the primary smart contract language for the Ethereum platform. Even though other contract languages have been created for Ethereum [DAKM15], Solidity is still one of the major ones. Moreover, Solidity can also be used in other blockchain platforms such as Monax[1] and Hyperledger[2].

Probably because of its popularity, there are many tools to help integrate smart contracts written in Solidity with other languages and technologies [Eth17]. For example, we have Solidity compilers coded in C/C++ and NodeJs, third-party parsers and grammar specifications (JavaScript and ANTLR), plugins for IDEs and editors (IntelliJ, Visual Studio, Vim, Atom, and etc.). Such tool integration support developers of smart contracts. However, as far as we know, there is a lack of tools for Pharo Smalltalk to handle Solidity smart contracts. Moreover, most academic work towards smart contracts focuses on security [LCO+16, BDLF+16, DAK+15] and not in tool support.

In this paper, we plan to partially tackle this lack of tools problem by building a Solidity parser that runs on Pharo Smalltalk. We claim that with a parser and its generated AST (Abstract Syntax Tree), we will be able to develop strong tool support for Solidity contracts. For instance, it would be much easier to create code inspection tools on top of a functional parser than to rely on the purely textual content of the contract. To accomplish these goals, we used SmaCC

[1] https://monax.io/, verified 2017-06-19.
[2] https://www.hyperledger.org/, verified 2017-06-19.

Solidity Parsing Using SmaCC:
Challenges and Irregularities

Henrique Rocha Stephane Ducasse
Marcus Denker
INRIA Lille - Nord Europe
{henrique.rocha, stephane.ducasse,

Jason Lecerf
CEA-List
jason.clement.lecerf@gmail.com

Another interesting challenge we found to parse Solidity was that the language uses the same symbol (comma) as a separator for expression lists but also as an operator for the expression itself. ... This causes a serious problem because when the parser finds a comma in the input it does not know if it is an operator for the current expression (matching the Expression rule) or a separator to the current expression and the beginning of a new one (matching ExpressionList). This is a potential problem for any parser due to the ambiguity of matching either rule when encountering a comma.

plication of blockchain is the execution of smart contracts [LCO+16].

problem by building a Solidity parser that runs on Pharo Smalltalk. We claim that with a parser and its generated AST (Abstract Syntax Tree), we will be able to develop strong tool support for Solidity contracts. For instance, it would be much easier to create code inspection tools on top of a functional parser than to rely on the purely textual content of the contract. To accomplish these goals, we used SmaCC

[1] https://monax.io/, verified 2017-06-19.
[2] https://www.hyperledger.org/, verified 2017-06-19.

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

How Solid is Solidity?

An In-dept Study of Solidity's Type Safety

Master thesis

Supervisor
Prof. Silvia Crafa

Author
Matteo Di Pirro

SEPTEMBER 2018

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

...we found out that Solidity's type system is far from being safe with respect to any type of error:

in many occasions, contract interfaces are not consulted at compile-time, and this makes the execution raise an exception and the user waste money.

SEPTEMBER 2018

Solidity

Implementation and Tooling

The difference between an amateur and a professional is: you write your own compiler.

Ryan Stortz
@withzombies

There are contracts on the blockchain that calculate 1 with exponentiation. This actually costs people money...

```
    JUMPI(#0x200, %13),
  ]>,
  <SSA:BasicBlock ofs:0x24c insns:[
    %14 = SLOAD(#0x3),
    %15 = EXP(#0x100, #0x0),
    %16 = DIV(%14, %15),
    %17 = EXP(#0x2, #0xA0),
    %18 = SUB(%17, #0x1),
```

10:39 PM - 6 Mar 2018

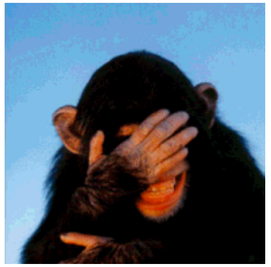
16 Block Trace

by Martin Holst Swende

18,538 invocations of `EXP`

Well over half were calculating 160 raised to the power of 1

Martin's GitHub profile pic:



16 Block Trace

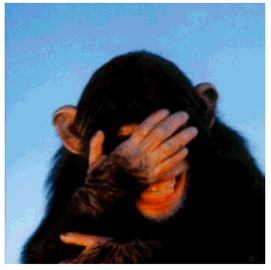
by Martin Holst Swende

18,538 invocations of `EXP`

Lesson: Solidity is bad at optimization, but getting better, kinda (more on this later)

Well over half were calculating 160 raised t

Martin's GitHub profile pic:



Exponentiation: How does it work?

```
// We need cleanup for EXP because 0**0 == 1, but 0**0x100 == 0
```

4 libsolidity/codegen/ExpressionCompiler.cpp — View

```
@@ -2069,7 +2069,9 @@ bool ExpressionCompiler::cleanupNeededForOp(Type::Category _type, Token::Value _
 {                                                    {
        if (Token::isCompareOp(_op) || Token::isShiftOp(_op))        if (Token::isCompareOp(_op) || Token::isShiftOp(_op))
                return true;                                 return true;
-       else if (_type == Type::Category::Integer && (_op == Token::Div || _op == Token::Mod))
+       else if (_type == Type::Category::Integer && (_op == Token::Div || _op == Token::Mod || _op == Token::Exp))
+               // We need cleanup for EXP because 0**0 == 1, but 0**0x100 == 0
+               // It would suffice to clean the exponent, though.
                return true;                                 return true;
        else                                         else
                return false;                                return false;
```

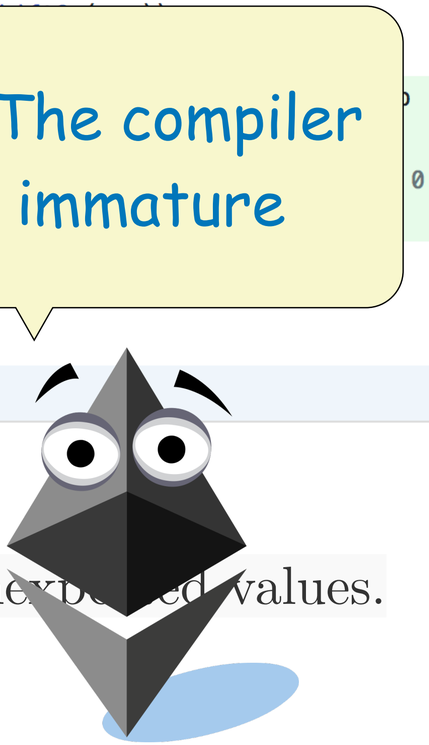
Using the ** operator with an exponent of type shorter than 256 bits can result in unexpected values.

Exponentiation: How does it work?

```
// We need cleanup for EXP because 0**0 == 1, but 0**0x100 == 0
```

4 libsolidity/codegen/ExpressionCompiler.cpp — View

```
@@ -2069,7 +2069,9 @@ bool ExpressionCompiler::cleanupNeededForOp(Type::Category _type, Token::Value _
 {                                                          {
       if (Token::isCompareOp(_op) || Token::isShiftOp(_op))      if (Token::isCompareOp(_op) || Token::isShiftOp(_op))
             return true;                                               return true;
-      else if (_type == Type::Category::Integer && (_op == Token::Div || _op == Token::Mod))
+      else if (_type == Type::Category::Integer && (_op == Token::Div || _op == Token::Mod || _op == Token::Exp))
+            // We need cleanup for EXP because 0**0 == 1, but 0**0x100 == 0
+            // It would
             return true;                                               return true;
       else                                                       else
             return false;                                              return false;
```



Using the ** operator with an exponent of type shorter than 256 bits can result in unexpected values.

Things Are Improving

Nick Oprisan @NicuOprisan · Nov 30

#Solidity needs to be more stable if we want to grow the dev community. As a dev I feel I can't keep-up with the releases. Add this to the growing rate of changes in the #JS community and you've got yourself a confused dev.

> **Chris** @ethchris
> Have you updated to #Solidity 0.5.0 already? We are about to release 0.5.1 in the next days! Yes, I know, it's crazy, it has not even been three weeks!
> Show this thread

4 4 10

Chris
@ethchris

Replying to @NicuOprisan

The breaking release before 0.5.0 was over two years ago. I think Solidity needs to get more flexible. We are planning breaking releases roughly every 6 months now. And I think it's fine, one reason being that you cannot change deployed code anyway.

4:12 AM - 30 Nov 2018

Things Are ~~Improving~~ Changing

Nick Oprisan @NicuOprisan · Nov 30

#Solidity needs to be more stable if we want to grow the dev community. As a dev I feel I can't keep-up with the releases. Add this to the growing rate of changes in the #JS community and you've got yourself a confused dev.

> **Chris** @ethchris
> Have you updated to #Solidity 0.5.0 already? We are about to release 0.5.1 in the next days! Yes, I know, it's crazy, it has not even been three weeks!
> Show this thread

4 4 10

Chris
@ethchris

Replying to @NicuOprisan

The breaking release before 0.5.0 was over two years ago. I think Solidity needs to get more flexible. We are planning breaking releases roughly every 6 months now. And I think it's fine, one reason being that you cannot change deployed code anyway.

4:12 AM - 30 Nov 2018

Things Are ~~Improving~~ Changing

Nick Oprisan @NicuOprisan · Nov 30

#Solidity needs to be more stable if we want to grow the dev community. As a dev I feel I can't keep-up with the releases. Add this to the growing rate of changes in the #JS community and you've got yourself a confused dev.

> **Chris** @ethchris
> Have you updated to #Solidity 0.5.0 already? We are about to release 0.5.1 in the next days! Yes, I know, it's crazy, it has not even been three weeks!
> Show this thread

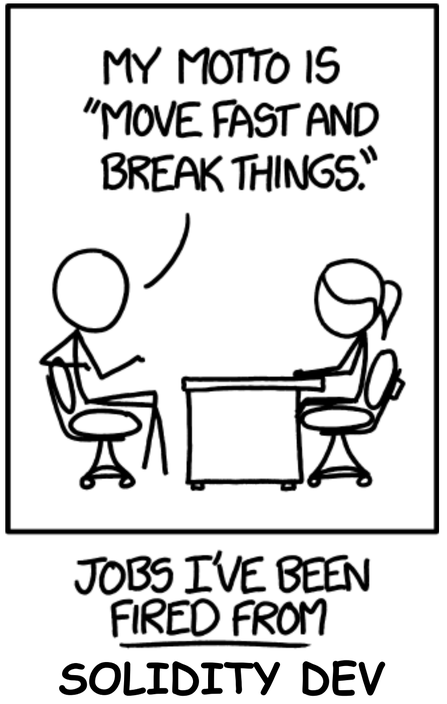
4 4 10

Chris
@ethchris

Replying to @NicuOprisan

The breaking release before 0.5.0 was over two years ago. I think Solidity needs to get more flexible. We are planning breaking releases roughly every 6 months now. And I think it's fine, one reason being that you cannot change deployed code anyway.

4:12 AM - 30 Nov 2018



Adapted from https://xkcd.com/1428/

Things Are ~~Improving~~ Changing

Nick Oprisan @NicuOprisan · Nov 30

#Solidity needs to be more stable if we want to grow the dev community. As a dev I feel I can't keep-up with the releases. Add this to the growing rate of changes in the #JS community and you've got yourself a confused dev.

> **Chris** @ethchris
> Have you updated to #Solidity 0.5.0 already? We are about to release 0.5.1 in the next days! Yes, I know, it's crazy, it has not even been three weeks!
> Show this thread

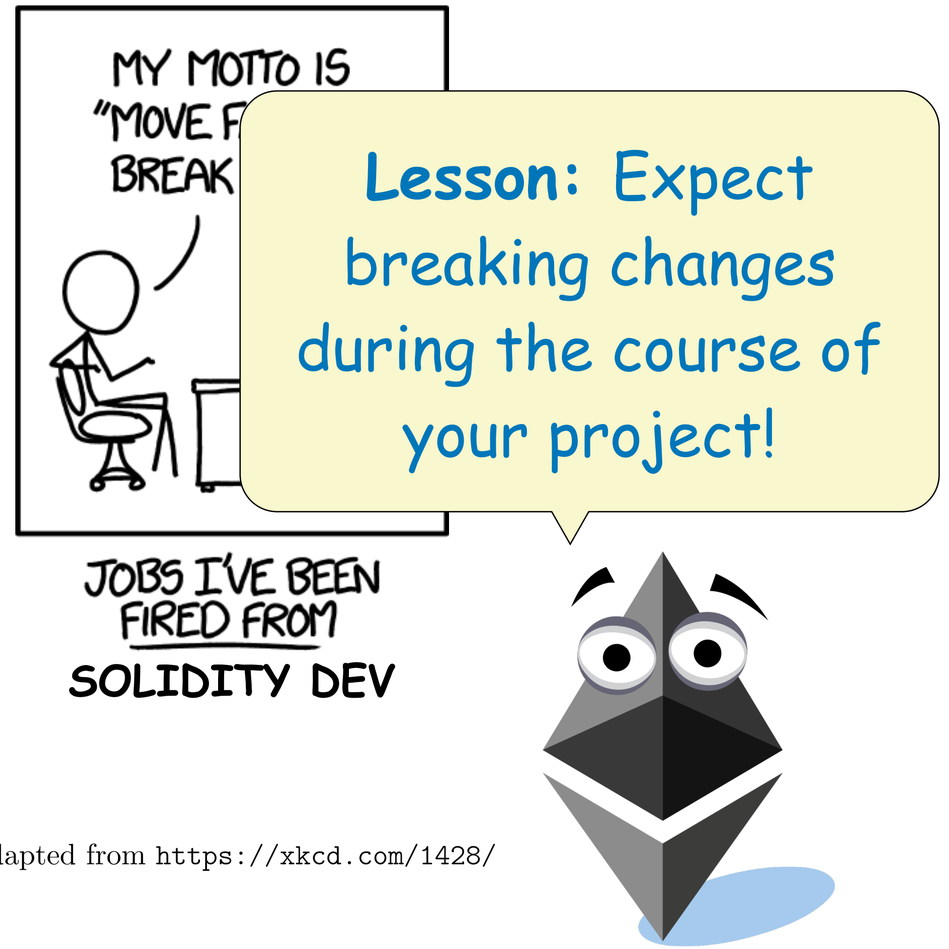
4 4 10

Chris
@ethchris

Replying to @NicuOprisan

The breaking release before 0.5.0 was over two years ago. I think Solidity needs to get more flexible. We are planning breaking releases roughly every 6 months now. And I think it's fine, one reason being that you cannot change deployed code anyway.

4:12 AM - 30 Nov 2018



Adapted from https://xkcd.com/1428/

Upgradable Contracts

Upgradable Contracts

Upgradable Contracts

Upgradable Contracts

😭

Lesson: If you absolutely have to use the DELEGATECALL proxy upgrade pattern, then you must always make sure the storage layout of your new contract matches the old one!

Backward Compatibility?

Backward Compatibility?

Backward Compatibility?

mo-seph commented 27 days ago • edited

Compliation fails for `solidity` using recommended install method. I'm on macos 10.13.6, and I've just installed brew to compile solc.

I've run

```
brew update
brew upgrade
brew tap ethereum/ethereum
```

I've tried installing the latest version, also tried 0.4.24. Here's a log with the latest version

mo-seph commented 27 days ago • edited

Compliation fails for `solidity` using recommended install method. I'm on macos 10.13.6, and I've just installed brew to compile solc.

I've run

```
brew update
brew upgrade
brew tap ethereum/ethereum
```

I've tried installing the latest version, also tried 0.4.24. Here's a log with the latest version

axic commented 26 days ago — Member

Since it works with 0.5.0, which has been released now, closing this issue.

👍 1 👎 1

axic closed this 26 days ago

mo-seph commented 27 days ago • edited

Compliation fails for `solidity` using recommended install method. I'm on macos 10.13.6, and I've just installed brew to compile solc.

I've run

```
brew update
brew upgrade
brew tap ethereum/ethereum
```

I've tried installing the latest version, also tried 0.4.24. Here's a log with the lates

⋮

axic commented 26 days ago — Member

Since it works with 0.5.0, which has been released now, closing this issue.

👍 1 👎 1

axic closed this 26 days ago

Lesson: Use `solc-select`!

Optimizations are Dangerous

- Compiler optimization still in active development
- Independent compiler audit in November of 2018 concluded optimizations are dangerous
- Numerous high severity bugs related to the optimizer, many excluded from the changlog
- There are likely latent bugs

Optimizations are Dangerous

- Compiler optimization still in active development
- Independent compiler audit in November of 2018 co
  optimizations are dangerous
- Numerous high severity bugs related to the optimiz
  excluded from the changlog
- There are likely latent bugs

Lesson: Don't turn on solc optimizations unless you really, really know what you are doing

On the Horizon

They're Proposing a New Intermediate Representation, YUL

```
Block = '{' Statement* '}'
Statement =
    Block |
    FunctionDefinition |
    VariableDeclaration |
    Assignment |
    Expression |
    Switch |
    ForLoop |
    BreakContinue
FunctionDefinition =
    'function' Identifier '(' TypedIdentifierList? ')'
    ( '->' TypedIdentifierList )? Block
VariableDeclaration =
    'let' TypedIdentifierList ( ':=' Expression )?
Assignment =
    IdentifierList ':=' Expression
Expression =
    FunctionCall | Identifier | Literal
If =
    'if' Expression Block
Switch =
    'switch' Expression Case* ( 'default' Block )?
Case =
    'case' Literal Block
```

```
ForLoop =
    'for' Block Expression Block Block
BreakContinue =
    'break' | 'continue'
FunctionCall =
    Identifier '(' ( Expression ( ',' Expression )* )? ')'
Identifier = [a-zA-Z_$] [a-zA-Z_0-9]*
IdentifierList = Identifier ( ',' Identifier)*
TypeName = Identifier | BuiltinTypeName
BuiltinTypeName = 'bool' | [us] ( '8' | '32' | '64' | '128' | '256' )
TypedIdentifierList = Identifier ':' TypeName ( ',' Identifier ':' TypeName )*
Literal =
    (NumberLiteral | StringLiteral | HexLiteral | TrueLiteral | FalseLiteral) ':'
TypeName
NumberLiteral = HexNumber | DecimalNumber
HexLiteral = 'hex' ('"' ([0-9a-fA-F]{2})* '"' | '\'' ([0-9a-fA-F]{2})* '\'')
StringLiteral = '"' ([^"\r\n\\] | '\\' .)* '"'
TrueLiteral = 'true'
FalseLiteral = 'false'
HexNumber = '0x' [0-9a-fA-F]+
DecimalNumber = [0-9]+
```

They're Proposing a New Intermediate Representation, YUL

```
Block = '{' Statement* '}'
Statement =
    Block |
    FunctionDefinition |
    VariableDeclaration |
    Assignment |
    Expression |
    Switch |
    ForLoop |
    BreakContinue
FunctionDefinition =
    'function' Identifier '(' TypedIdentifierList? ')'
    ( '->' TypedIdentifierList )? Block
VariableDeclaration =
    'let' TypedIdentifierList ( ':=' Expression )?
Assignment =
    IdentifierList ':=' Expression
Expression =
    FunctionCall | Identifier | Literal
If =
    'if' Expression Block
Switch =
    'switch' Expression Case* ( 'default' Block )?
Case =
    'case' Literal Block
```

```
ForLoop =
    'for' Block Expression Block Block
BreakContinue =
    'break' | 'continue'
FunctionCall =
    Identifier '(' ( Expression ( ',' Expression )* )? ')'
Identifier = [a-zA-Z_$] [a-zA-Z_0-9]*
IdentifierList = Identifier ( ',' Identifier)*
TypeName = Identifier | BuiltinTypeName
BuiltinTypeName = 'bool' | [us] ( '8' | '32' | '64' | '128' | '256' )
TypedIdentifierList = Identifier ':' TypeName ( ',' Identifier ':' TypeName )*
Literal =
    (NumberLiteral | StringLiteral | HexLiteral | TrueLiteral | FalseLiteral) ':'
TypeName
NumberLiteral = HexNumber | DecimalNumber
HexLiteral = 'hex' ('"' ([0-9a-fA-F]{2})* '"' | '\'' ([0-9a-fA-F]{2})* '\'')
StringLiteral = '"' ([^"\r\n\\] | '\\' .)* '"'
TrueLiteral = 'true'
FalseLiteral = 'false'
HexNumber = '0x' [0-9a-fA-F]+
DecimalNumber = [0-9]+
```

All of this has happened before ... and will happen again.

They're Proposing a New Intermediate Representation, YUL

```
Block = '{' Statement* '}'
Statement =
    Block |
    FunctionDefinition |
    VariableDeclaration |
    Assignment |
    Expression |
    Switch |
    ForLoop |
    BreakContinue
VariableDeclaration =
    'let' TypedIdentifierList ( ':=' Expression )?
Assignment =
    IdentifierList ':=' Expression
Expression =
    FunctionCall | Identifier | Literal
If =
    'if' Expression Block
Switch =
    'switch' Expression Case* ( 'default' Block )?
Case =
    'case' Literal Block
```

The "If" production rule is never used!

```
ForLoop =
    'for' Block Expression Block Block
BreakContinue =
    'break' | 'continue'
FunctionCall =
    Identifier '(' ( Expression ( ',' Expression )* )? ')'
Identifier = [a-zA-Z_$] [a-zA-Z_0-9]*
IdentifierList = Identifier ( ',' Identifier)*
    ier | BuiltinTypeName
    'bool' | [us] ( '8' | '32' | '64' | '128' | '256' )
    t = Identifier ':' TypeName ( ',' Identifier ':' TypeName )*
Literal =
    (NumberLiteral | StringLiteral | HexLiteral | TrueLiteral | FalseLiteral) ':'
TypeName
NumberLiteral = HexNumber | DecimalNumber
HexLiteral = 'hex' ('"' ([0-9a-fA-F]{2})* '"' | '\'' ([0-9a-fA-F]{2})* '\'')
StringLiteral = '"' ([^"\r\n\\] | '\\' .)* '"'
TrueLiteral = 'true'
FalseLiteral = 'false'
HexNumber = '0x' [0-9a-fA-F]+
DecimalNumber = [0-9]+
```

All of this has happened before ... and will happen again.

They're Proposing a New Intermediate Representation, YUL

```
Block = '{' Statement* '}'
Statement =
    Block |
    FunctionDefinition |
    VariableDeclaration |
    Assignment |
    Expression |
    Switch |
    ForLoop |
    BreakContinue
VariableDeclaration =
    'let' TypedIdentifierList ( ':=' Expression )?
Assignment =
    IdentifierList ':=' Expression
Expression =
    FunctionCall | Identifier | Literal
If =
    'if' Expression Block
Switch =
    'switch' Expression Case* ( 'default' Block )?
Case =
    'case' Literal Block
```

The "If" production rule is never used!

The default switch case isn't followed by a ':'

```
ForLoop =
    'for' Block Expression Block Block
BreakContinue =
    'break' | 'continue'
FunctionCall =
    Identifier '(' ( Expression ( ',' Expression )* )? ')'
Identifier = [a-zA-Z_$] [a-zA-Z_0-9]*
IdentifierList = Identifier ( ',' Identifier)*
    ier | BuiltinTypeName
    'bool' | [us] ( '8' | '32' | '64' | '128' | '256' )
    t = Identifier ':' TypeName ( ',' Identifier ':' TypeName )*
Literal =
    (NumberLiteral | StringLiteral | HexLiteral | TrueLiteral | FalseLiteral) ':'
    hex' ( ([0-9a-fA-F]{2})*  |  \  ([0-9a-fA-F]{2})*  \ ')
StringLiteral = '"' ([^"\r\n\\] | '\\' .)* '"'
TrueLiteral = 'true'
FalseLiteral = 'false'
HexNumber = '0x' [0-9a-fA-F]+
DecimalNumber = [0-9]+
```

All of this has happened before ... and will happen again.

They're Proposing a New Intermediate Representation, YUL

"switch foo" is a legal production in this grammar

The "If" production rule is never used!

The default switch case isn't followed by a ':'

```
Block = '{' Statement* '}'
    FunctionDefinition |
    VariableDeclaration |
    Assignment |
    Expression |
    Switch |
    ForLoop |
    BreakContinue
VariableDeclaration =
    'let' TypedIdentifierList ( ':=' Expression )?
Assignment =
    IdentifierList ':=' Expression
Expression =
    FunctionCall | Identifier | Literal
If =
    'if' Expression Block
Switch =
    'switch' Expression Case* ( 'default' Block )?
Case =
    'case' Literal Block
```

```
    'for' Block Expression Block Block
BreakContinue =
    'break' | 'continue'
FunctionCall =
    Identifier '(' ( Expression ( ',' Expression )* )? ')'
Identifier = [a-zA-Z_$] [a-zA-Z_0-9]*
IdentifierList = Identifier ( ',' Identifier)*
    ier | BuiltinTypeName
    'bool' | [us] ( '8' | '32' | '64' | '128' | '256' )
    t = Identifier ':' TypeName ( ',' Identifier ':' TypeName )*
Literal =
    (NumberLiteral | StringLiteral | HexLiteral | TrueLiteral | FalseLiteral) ':'
    ([0-9a-fA-F]{2})*  |  \'  ([0-9a-fA-F]{2})*  \' )
StringLiteral = '"' ([^"\r\n\\] | '\\' .)* '"'
TrueLiteral = 'true'
FalseLiteral = 'false'
HexNumber = '0x' [0-9a-fA-F]+
DecimalNumber = [0-9]+
```

All of this has happened before ... and will happen again.

They're Proposing a New Intermediate Representation, YUL

"switch foo" is a legal production in this grammar

StringLiteral can't be represented without casting

The "If" production rule is never used!

The default switch case isn't followed by a ':'

```
Block = '{' Statement* '}'
    FunctionDefinition |
    VariableDeclaration |
    Assignment |
    Expression |
    Switch |
    ForLoop |
    BreakContinue
VariableDeclaration =
    'let' TypedIdentifierList ( ':=' Expression )?
Assignment =
    IdentifierList ':=' Expression
Expression =
    FunctionCall | Identifier | Literal
If =
    'if' Expression Block
Switch =
    'switch' Expression Case* ( 'default' Block )?
Case =
    'case' Literal Block
```

```
    'for' Block Expression Block Block
BreakContinue =
Identifier = [a-zA-Z_$] [a-zA-Z_0-9]*
IdentifierList = Identifier ( ',' Identifier)*
ier | BuiltinTypeName
'bool' | [us] ( '8' | '32' | '64' | '128' | '256' )
t = Identifier ':' TypeName ( ',' Identifier ':' TypeName )*
Literal =
    (NumberLiteral | StringLiteral | HexLiteral | TrueLiteral | FalseLiteral) ':'
StringLiteral = '"' ([^"\r\n\\] | '\\' .)* '"'
TrueLiteral = 'true'
FalseLiteral = 'false'
HexNumber = '0x' [0-9a-fA-F]+
DecimalNumber = [0-9]+
```

All of this has happened before ... and will happen again.

They're Proposing a New ermediate Representation, Y

EVM::SSA

SlithIR

"su is a legal production in this grammar

StringLiteral can't be represented without casting

The "If" production rule is never used!

The default switch case isn't followed by a ':'

```
Block = '
    ion |
    ration |
    |
    on |
    |
    Loop |
    BreakContinue
VariableDeclaration =
    'let' TypedIdentifierList ( ':=' Expression )?
Assignment =
    IdentifierList ':=' Expression
Expression =
    FunctionCall | Identifier | Literal
If =
    'if' Expression Block
Switch =
    'switch' Expression Case* ( 'default' Block )?
Case =
    'case' Literal Block
```

```
    'for' Block Expression Block Block
BreakContinue =
Identifier = [a-zA-Z_$] [a-zA-Z_0-9]*
IdentifierList = Identifier ( ',' Identifier)*
    ier | BuiltinTypeName
    'bool' | [us] ( '8' | '32' | '64' | '128' | '256' )
    t = Identifier ':' TypeName ( ',' Identifier ':' TypeName )*
Literal =
    (NumberLiteral | StringLiteral | HexLiteral | TrueLiteral | FalseLiteral) ':'
HexLiteral = 'hex' ('"' ([0-9a-fA-F]{2})* '"' | '\'' ([0-9a-fA-F]{2})* '\'')
StringLiteral = '"' ([^"\r\n\\] | '\\' .)* '"'
TrueLiteral = 'true'
FalseLiteral = 'false'
HexNumber = '0x' [0-9a-fA-F]+
DecimalNumber = [0-9]+
```

All of this has happened before ... and will happen again.

Solidity Alternatives

- Even more immature
- Lack of security tooling
- Different semantics!

Bugs!

Compiler Warnings

```
 1 pragma solidity ^0.4.9;
 2
 3 contract SafeMath {
 4     /**
 5       * @dev Adds two numbers, throws on overflow.
 6     */
 7     function add(uint256 _a, uint256 _b) internal pure returns (uint256) {
 8         uint256 c = _a + _b;
 9         if(c < _a) {
10             throw;
11         }
12
13         return c;
14     }
15 }
```

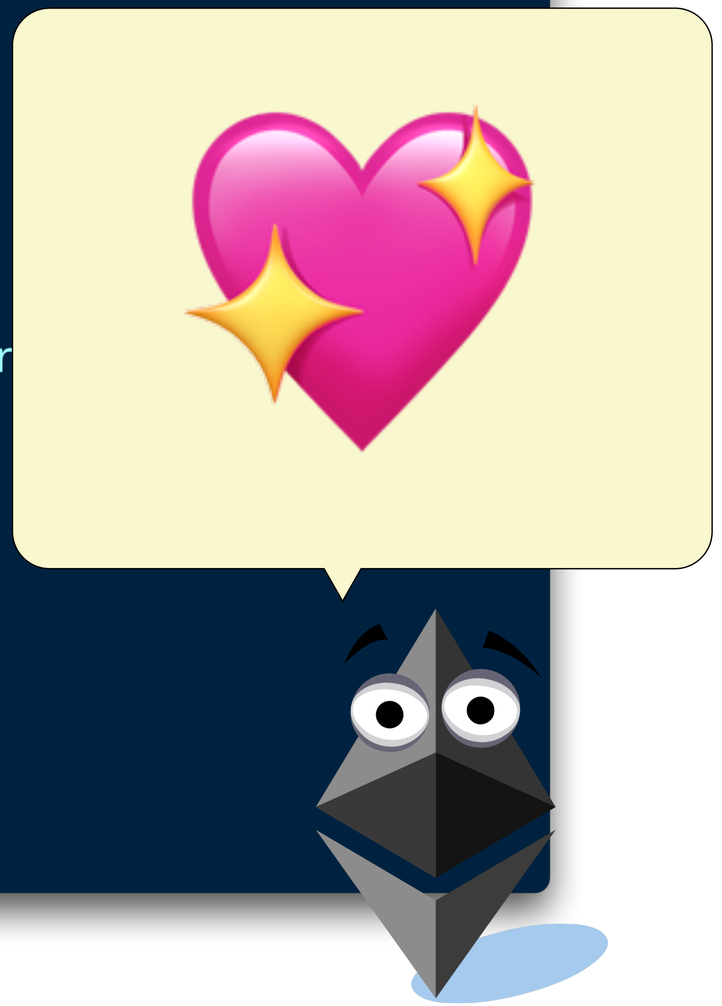
Compiler Warnings

```
 1 pragma solidity ^0.4.9;
 2
 3 contract SafeMath {
 4     /**
 5       * @dev Adds two numbers, throws on overflow.
 6     */
 7     function add(uint256 _a, uint256 _b) internal pure r
 8         uint256 c = _a + _b;
 9         if(c < _a) {
10             throw;
11         }
12
13         return c;
14     }
15 }
```

Warning: "throw" is deprecated in favour of "revert()", "require()" and "assert()"

Compiler Warnings

```
 1 pragma solidity ^0.4.9;
 2
 3 contract SafeMath {
 4     /**
 5       * @dev Adds two numbers, throws on overflow.
 6     */
 7     function add(uint256 _a, uint256 _b) internal pure r
 8         uint256 c = _a + _b;
 9         if(c < _a) {
10             throw; assert;
11         }
12
13         return c;
14     }
15 }
```



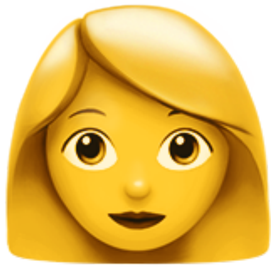
Compiler Warnings

Reentrancy

```
1 mapping (address => uint) private userBalances;
2
3 function withdrawBalance() public {
4     uint amountToWithdraw = userBalances[msg.sender];
5     require(msg.sender.call.value(amountToWithdraw)());
6     userBalances[msg.sender] = 0;
7 }
```

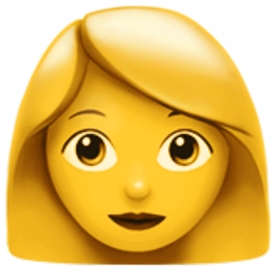
Reentrancy

```
1 mapping (address => uint) private userBalances;
2
3 function withdrawBalance() public {
4     uint amountToWithdraw = userBalances[msg.sender];
5     require(msg.sender.call.value(amountToWithdraw)());
6     userBalances[msg.sender] = 0;
7 }
```



Reentrancy

```
1 mapping (address => uint) private userBalances;
2
3 function withdrawBalance() public {
4     uint amountToWithdraw = userBalances[msg.sender];
5     require(msg.sender.call.value(amountToWithdraw)());
6     userBalances[msg.sender] = 0;
7 }
```

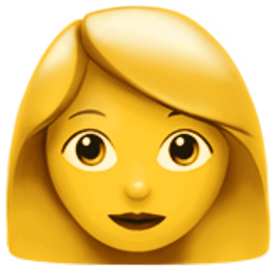


Deploy Attack Contract

```
 1 uint8 toWithdraw = 2;
 2 address target = ...;
 3
 4 function () public payable {
 5     if (--toWithdraw > 0) {
 6         pwn();
 7     }
 8 }
 9
10 function pwn() public {
11     target.withdrawBalance()
12 }
```

Reentrancy

```
1 mapping (address => uint) private userBalances;
2
3 function withdrawBalance() public {
4     uint amountToWithdraw = userBalances[msg.sender];
5     require(msg.sender.call.value(amountToWithdraw)());
6     userBalances[msg.sender] = 0;
7 }
```

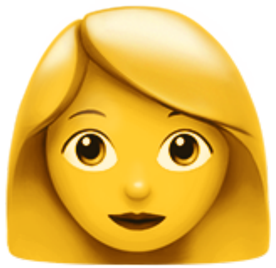


pwn()

```
 1 uint8 toWithdraw = 2;
 2 address target = ...;
 3
 4 function () public payable {
 5     if (--toWithdraw > 0) {
 6         pwn();
 7     }
 8 }
 9
10 function pwn() public {
11     target.withdrawBalance()
12 }
```

Reentrancy

```
1 mapping (address => uint) private userBalances;
2
3 function withdrawBalance() public {
4     uint amountToWithdraw = userBalances[msg.sender];
5     require(msg.sender.call.value(amountToWithdraw)());
6     userBalances[msg.sender] = 0;
7 }
```

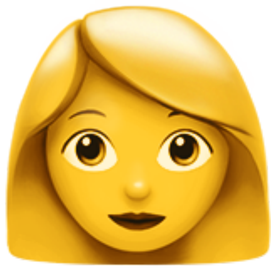


pwn()

```
 1 uint8 toWithdraw = 2;
 2 address target = ...;
 3
 4 function () public payable {
 5     if (--toWithdraw > 0) {
 6         pwn();
 7     }
 8 }
 9
10 function pwn() public {
11     target.withdrawBalance()
12 }
```

Reentrancy

```
1 mapping (address => uint) private userBalances;
2
3 function withdrawBalance() public {
4     uint amountToWithdraw = userBalances[msg.sender];
5     require(msg.sender.call.value(amountToWithdraw)());
6     userBalances[msg.sender] = 0;
7 }
```

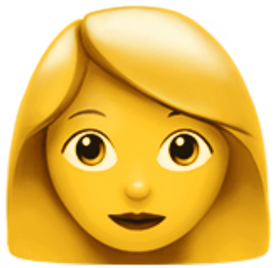


pwn()

```
1 uint8 toWithdraw = 2;
2 address target = ...;
3
4 function () public payable {
5     if (--toWithdraw > 0) {
6         pwn();
7     }
8 }
9
10 function pwn() public {
11     target.withdrawBalance()
12 }
```

Reentrancy

```
1 mapping (address => uint) private userBalances;
2
3 function withdrawBalance() public {
4     uint amountToWithdraw = userBalances[msg.sender];
5     require(msg.sender.call.value(amountToWithdraw)());
6     userBalances[msg.sender] = 0;
7 }
```

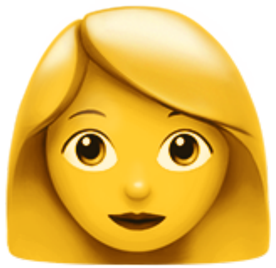


pwn()

```
 1 uint8 toWithdraw = 2;
 2 address target = ...;
 3
 4 function () public payable {
 5     if (--toWithdraw > 0) {
 6         pwn();
 7     }
 8 }
 9
10 function pwn() public {
11     target.withdrawBalance()
12 }
```

Reentrancy

```
1 mapping (address => uint) private userBalances;
2
3 function withdrawBalance() public {
4     uint amountToWithdraw = userBalances[msg.sender];
5     require(msg.sender.call.value(amountToWithdraw)());
6     userBalances[msg.sender] = 0;
7 }
```

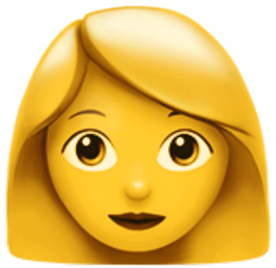


pwn()

```
 1 uint8 toWithdraw = 2;
 2 address target = ...;
 3
 4 function () public payable {
 5     if (--toWithdraw > 0) {
 6         pwn();
 7     }
 8 }
 9
10 function pwn() public {
11     target.withdrawBalance()
12 }
```

Reentrancy

```
1 mapping (address => uint) private userBalances;
2
3 function withdrawBalance() public {
4     uint amountToWithdraw = userBalances[msg.sender];
5     require(msg.sender.call.value(amountToWithdraw)());
6     userBalances[msg.sender] = 0;
7 }
```

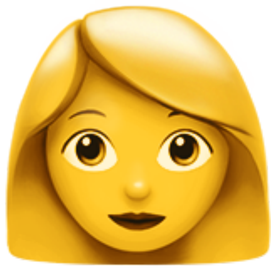


pwn()

```
 1 uint8 toWithdraw = 2;
 2 address target = ...;
 3
 4 function () public payable {
 5     if (--toWithdraw > 0) {
 6         pwn();
 7     }
 8 }
 9
10 function pwn() public {
11     target.withdrawBalance()
12 }
```

Reentrancy

```
1 mapping (address => uint) private userBalances;
2
3 function withdrawBalance() public {
4     uint amountToWithdraw = userBalances[msg.sender];
5     require(msg.sender.call.value(amountToWithdraw)());
6     userBalances[msg.sender] = 0;
7 }
```



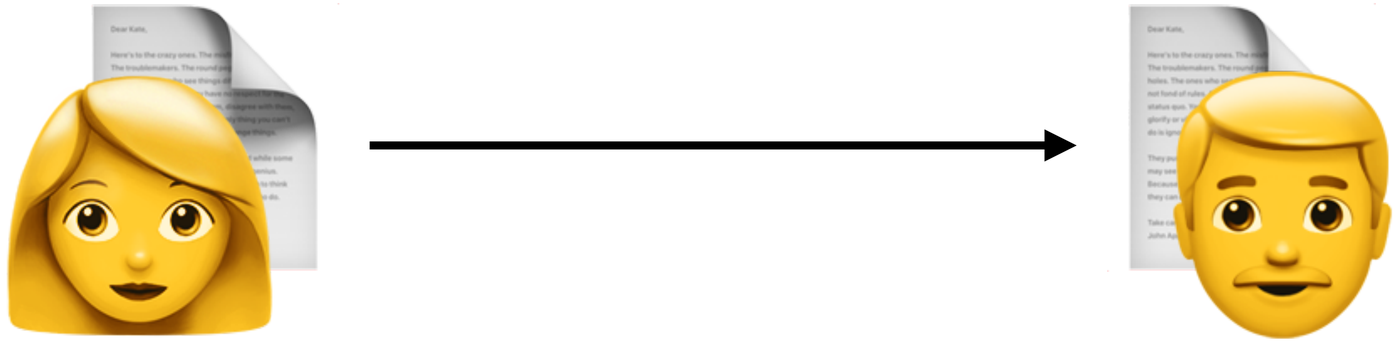
pwn()

```
1 uint8 toWithdraw = 2;
2 address target = ...;
3
4 function () public payable {
5     if (--toWithdraw > 0) {
6         pwn();
7     }
8 }
9
10 function pwn() public {
11     target.withdrawBalance()
12 }
```

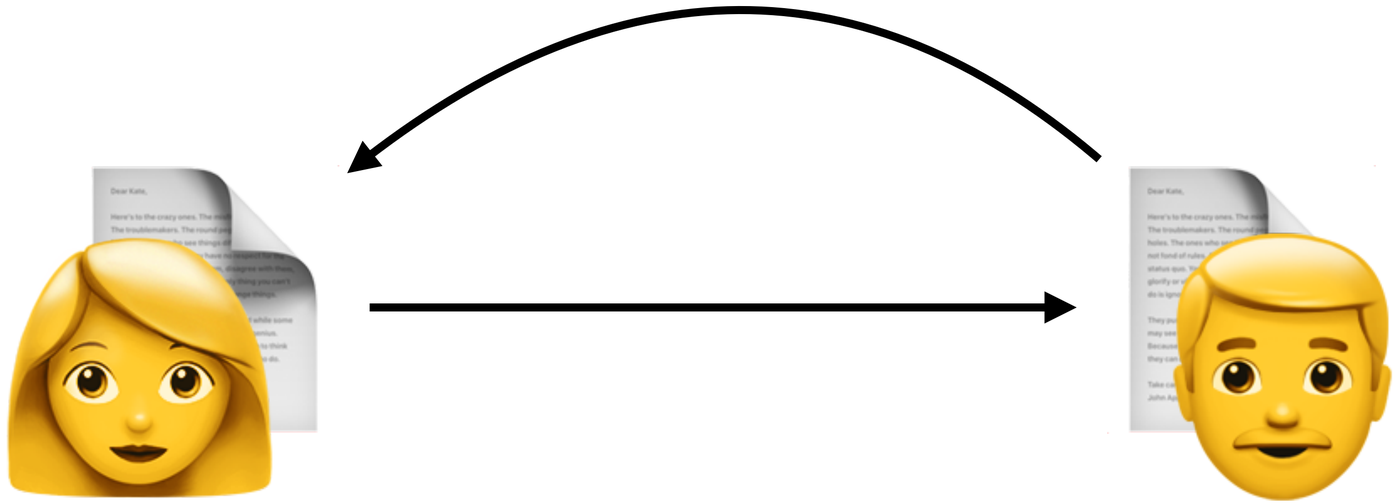
Lesson: Use the "checks, effects, interactions" pattern!

Malicious External Calls

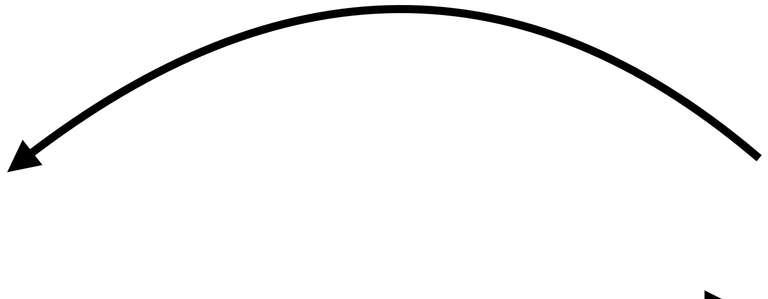
Malicious External Calls

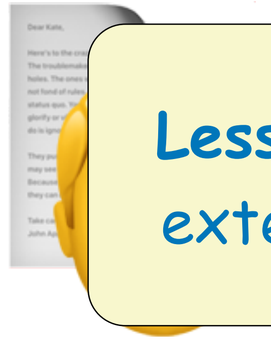


Malicious External Calls



Malicious External Calls





Lesson: Don't trust external contracts!

Zero Initialization

```
1 /**
2  * Checks if the address is an admin
3  */
4 modifier onlyAdmin() {
5     require(admins[msg.sender].revokedTimeStamp == 0,
6             'Admin was revoked.');
7     _;
8 }
```

Zero Initialization

```
/**
 * Checks if the address is an admin
 */
modifier onlyAdmin() {
    require(admins[msg.sender].revokedTimeStamp == 0,
            'Admin was revoked.');
    _;
}
```

```
enum UserStatus { Registered, Approved, Denied }

modifier onlyRegisteredUsers(address userAddress) {
    require(users[userAddress].status == UserStatus.Registered);
    _;
}
```

Zero Initialization

```
/**
 * Checks if the address is an admin
 */
modifier onlyAdmin() {
    require(admins[msg.sender].revokedTimeStamp == 0,
            'Admin was revoked.');
    _;
}
```

```
enum UserStatus { Registered, Approved, D

modifier onlyRegisteredUsers(address user
    require(users[userAddress].status == UserStatus.Regis ered);
    _;
}
```

Lesson: Unlike in most other languages, uninitialized keys will result in uninitialized memory, which is zeroed.

Array Length Manipulation

```
 1 contract Vulnerable {
 2     address public owner = msg.sender;
 3     uint256[] map;
 4
 5     function set(uint256 key, uint256 value) public payable {
 6         if (map.length <= key) {
 7             map.length = key + 1;
 8         }
 9
10         map[key] = value;
11     }
12 }
```

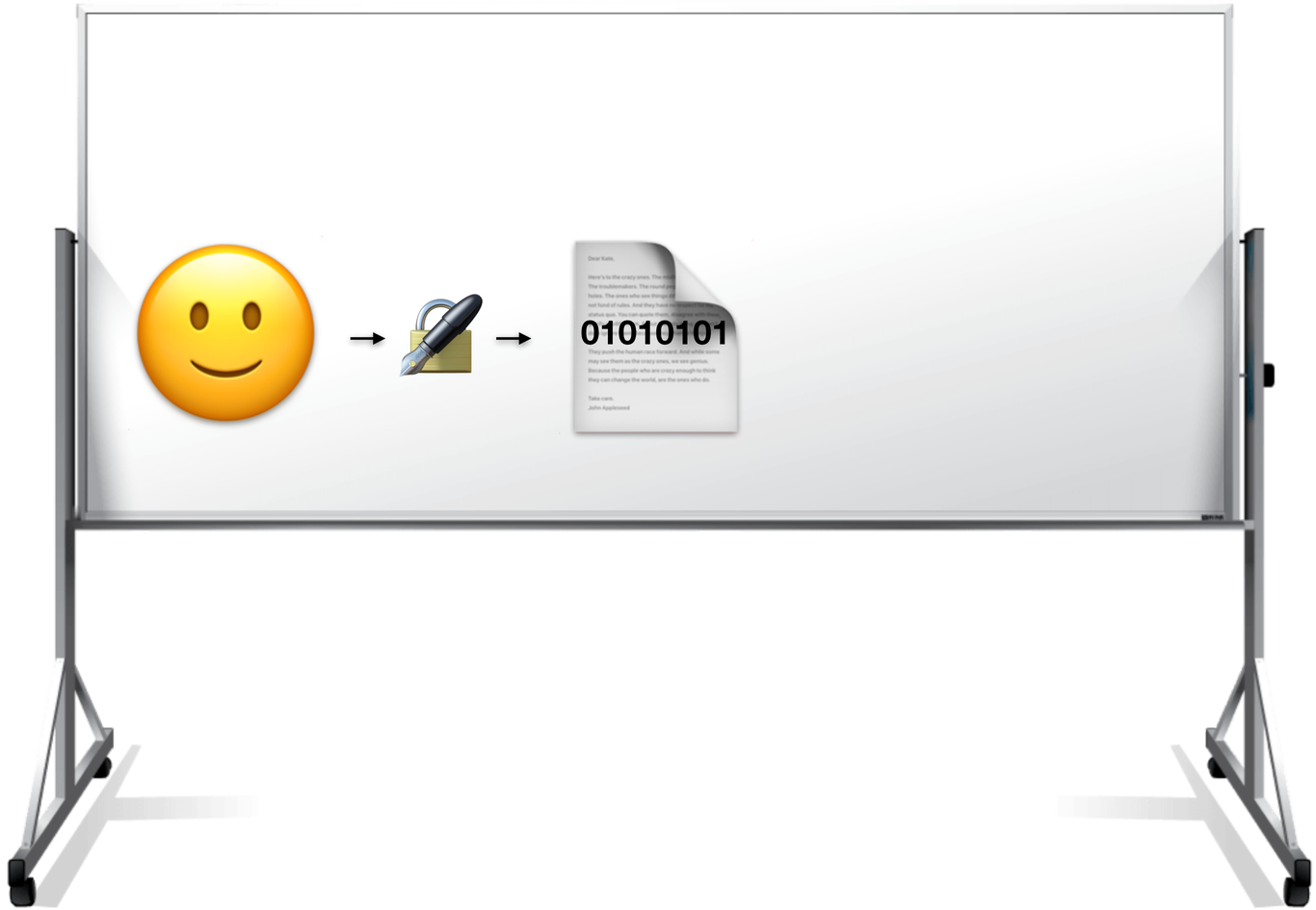
Array Length Manipulation

```
 1 contract Vulnerable {
 2     address public owner = msg.sender;
 3     uint256[] map;
 4
 5     function set(uint256 key, uint256 value)
 6         if (map.length <= key) {
 7             map.length = key + 1;
 8         }
 9
10         map[key] = value;
11     }
12 }
```

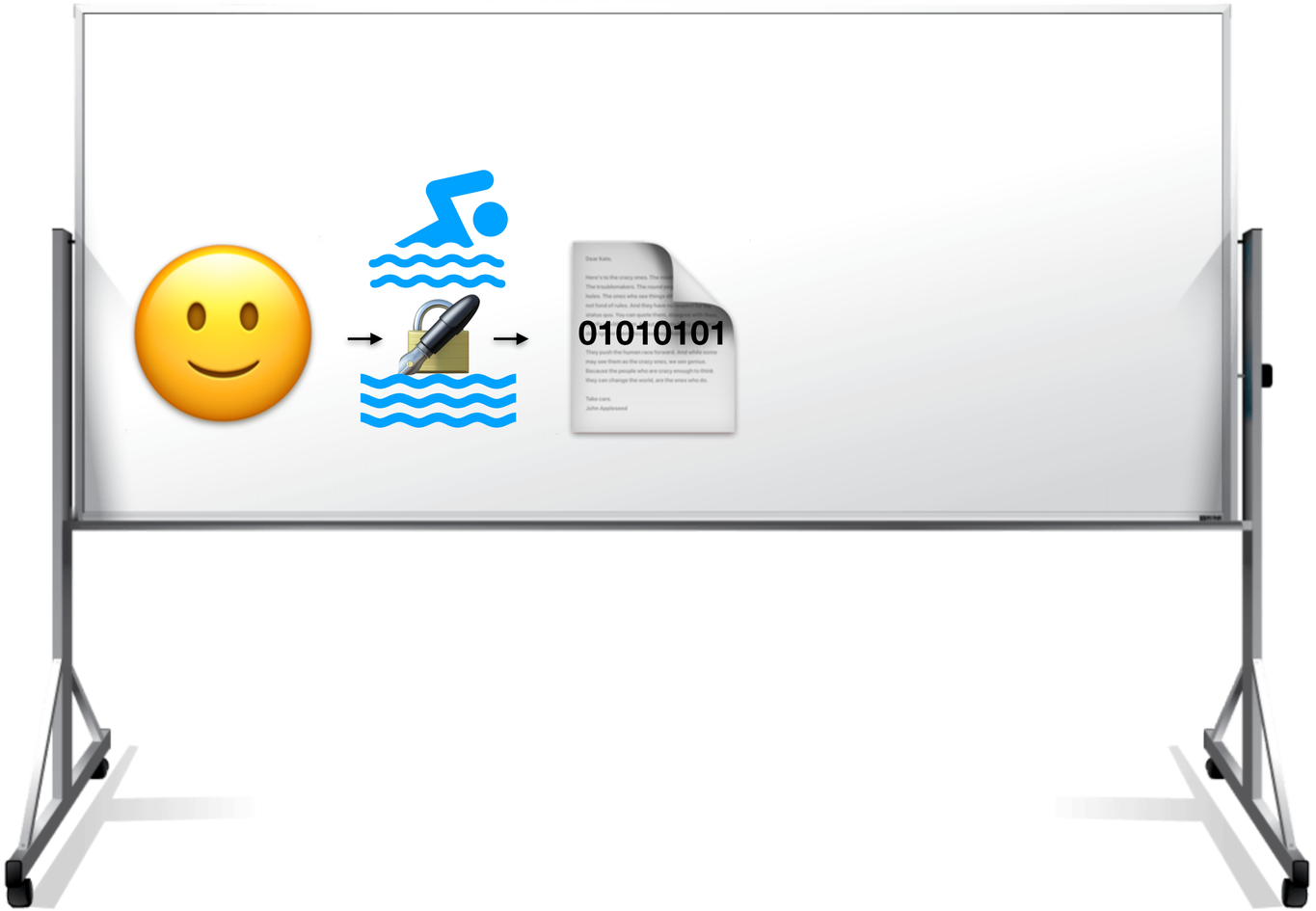
Lesson: Never manually manipulate the length of an array!

Transaction “Frontrunning”

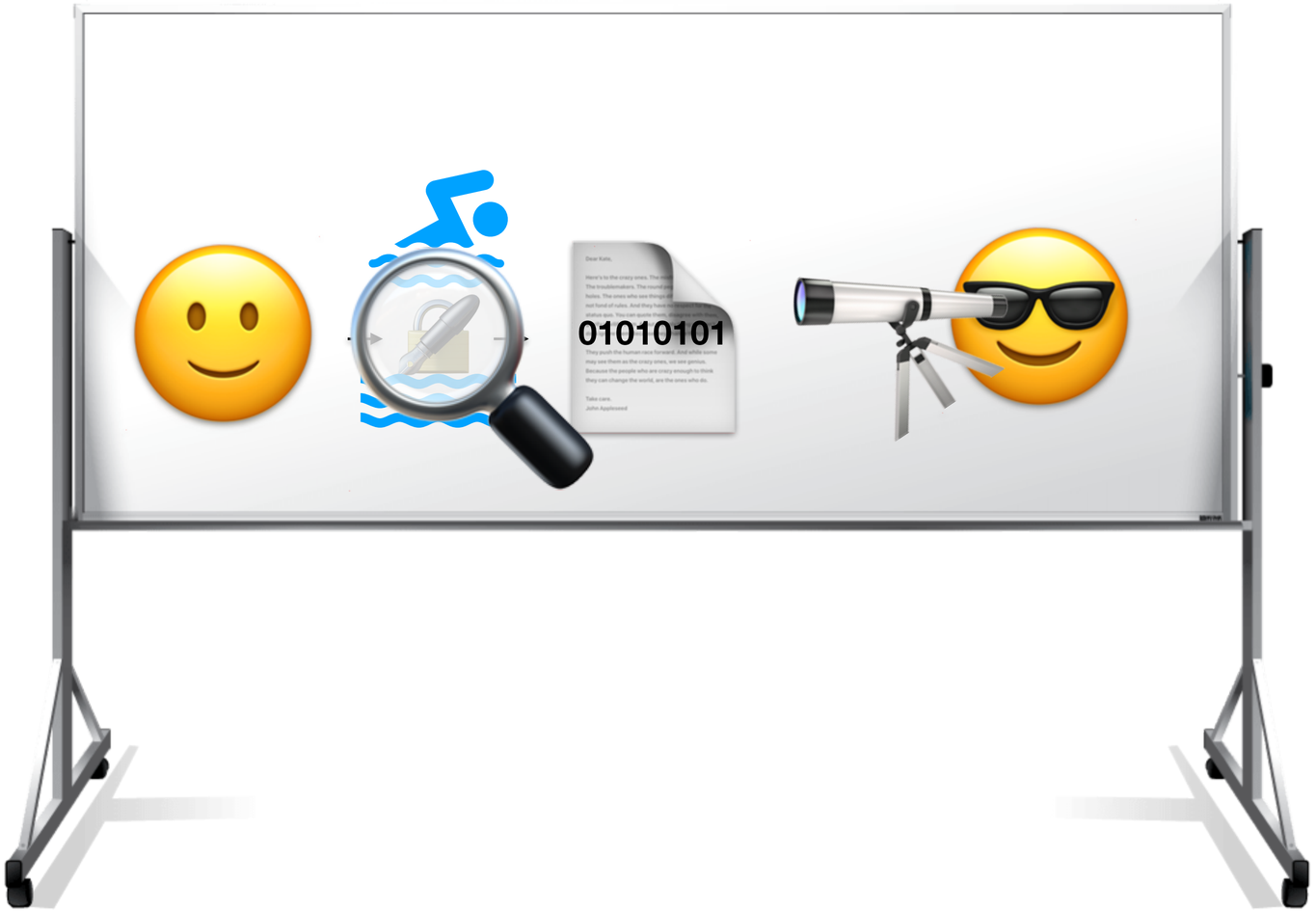
Transaction “Frontrunning”



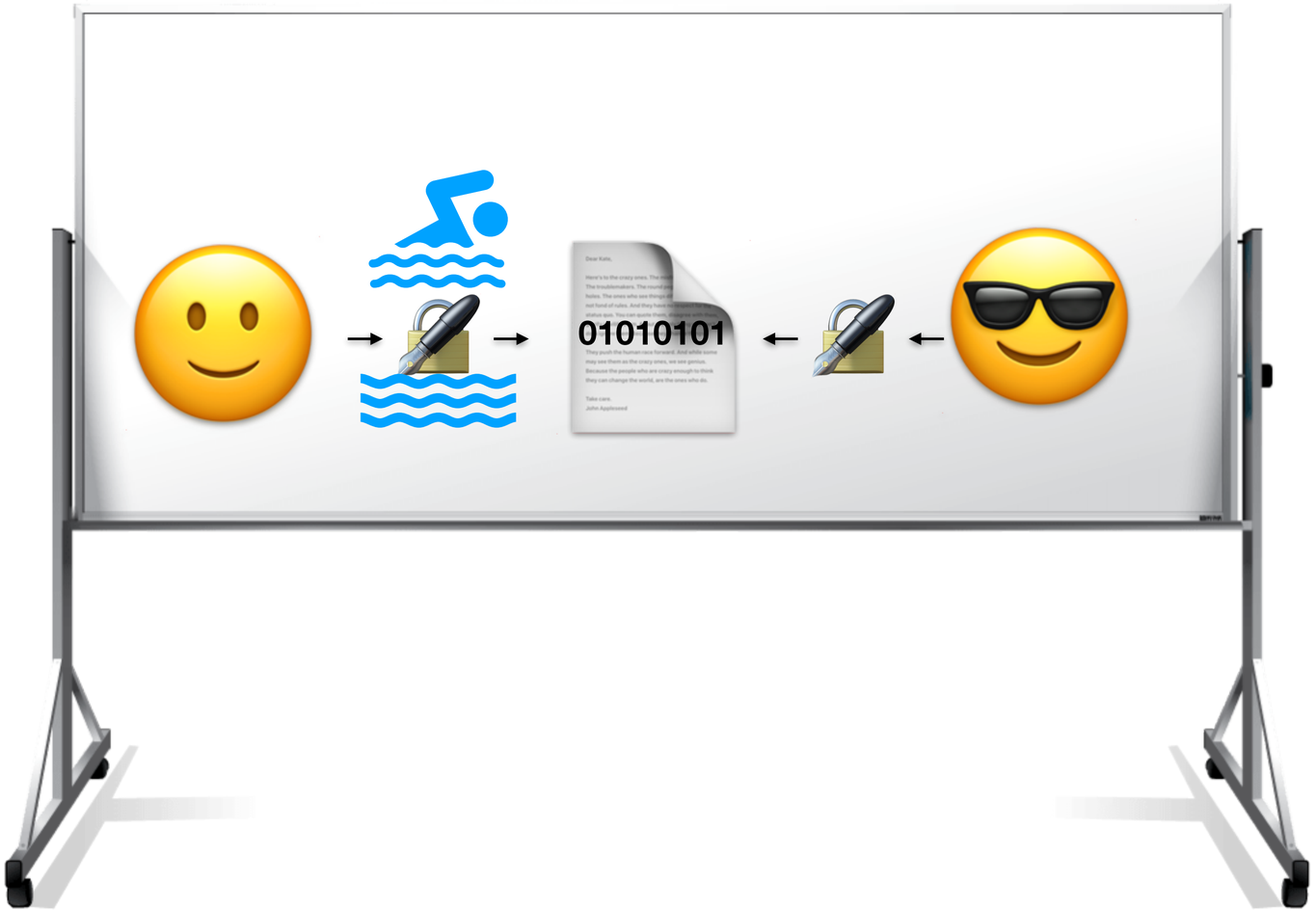
Transaction “Frontrunning”



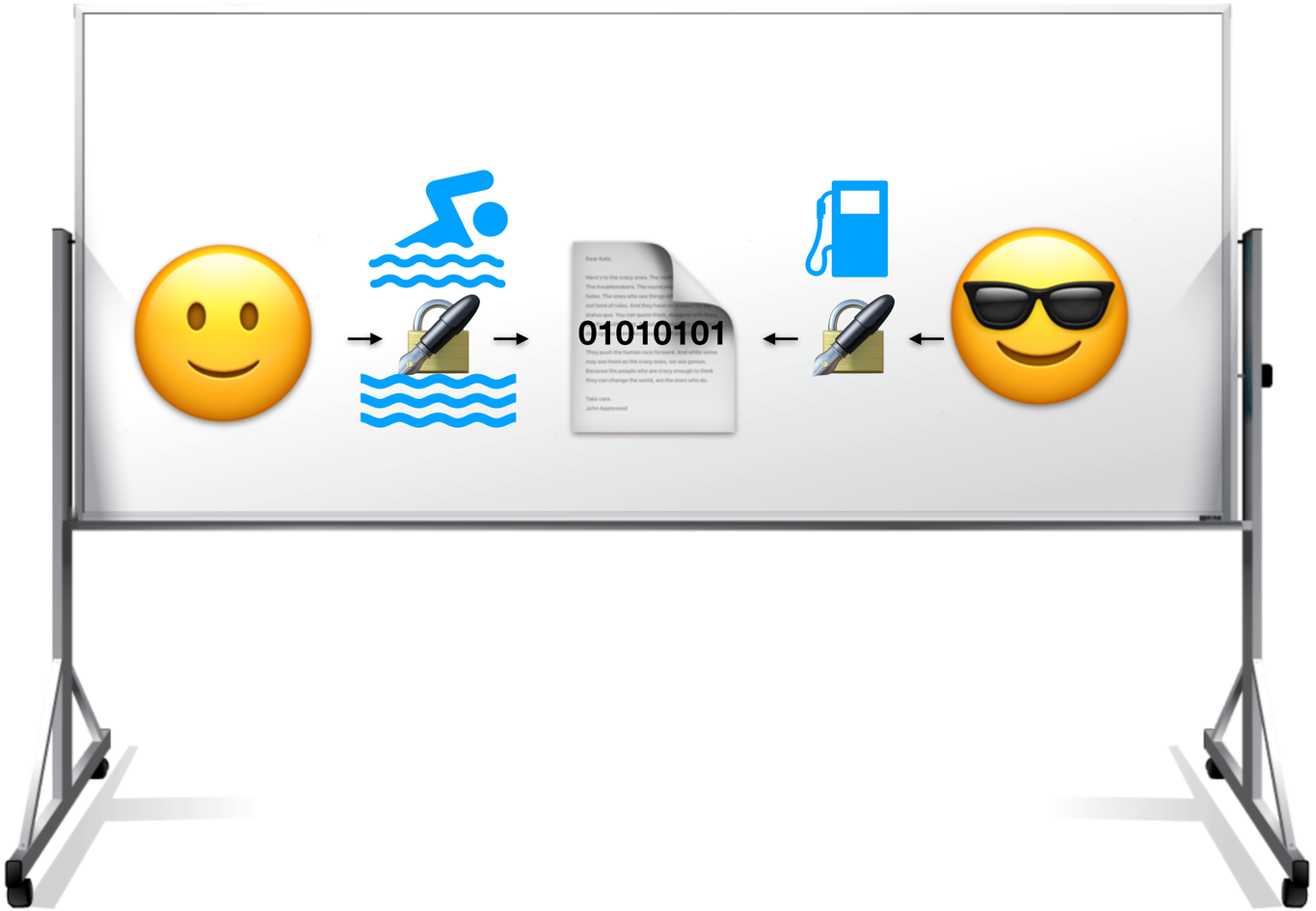
Transaction “Frontrunning”



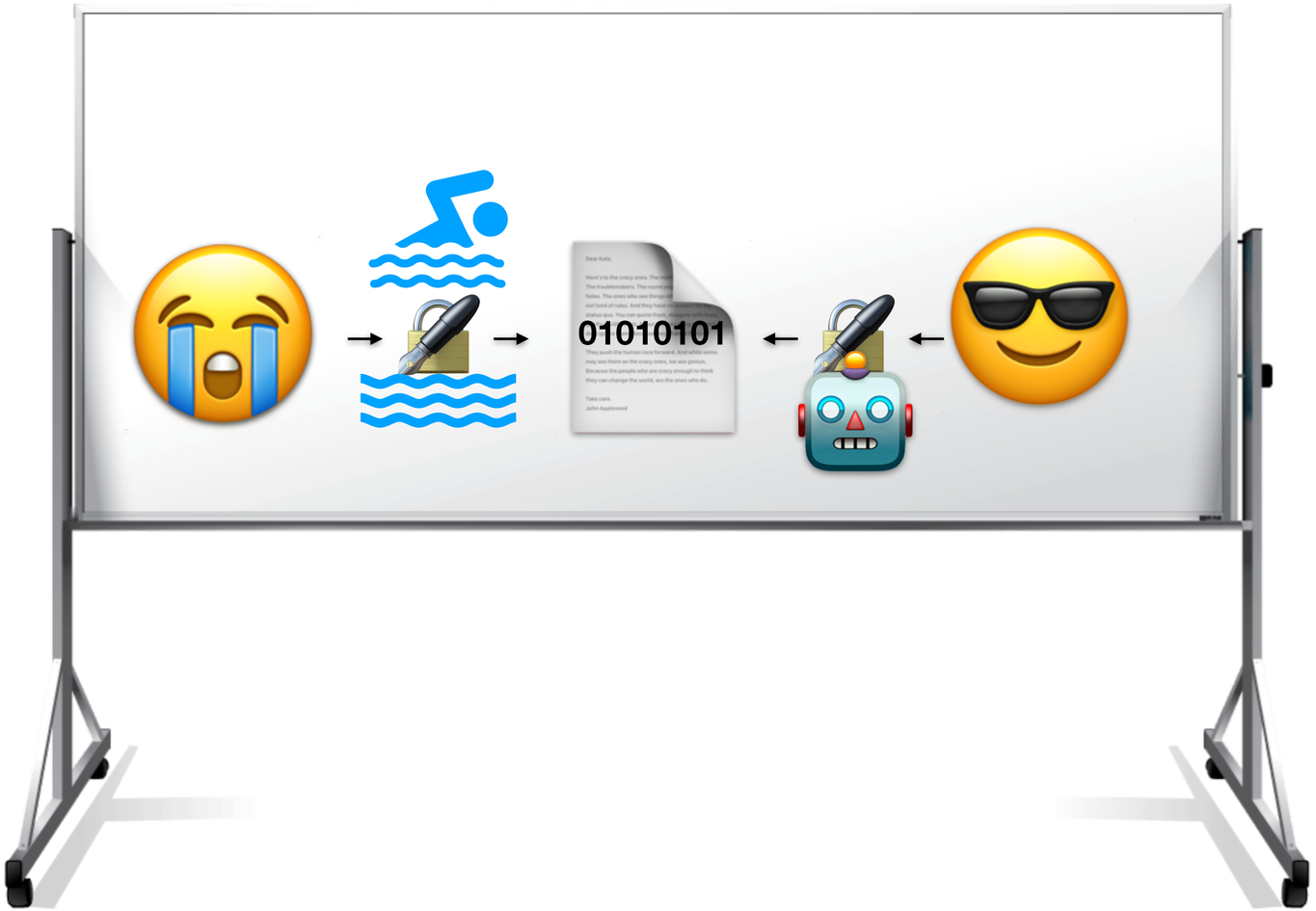
Transaction “Frontrunning”



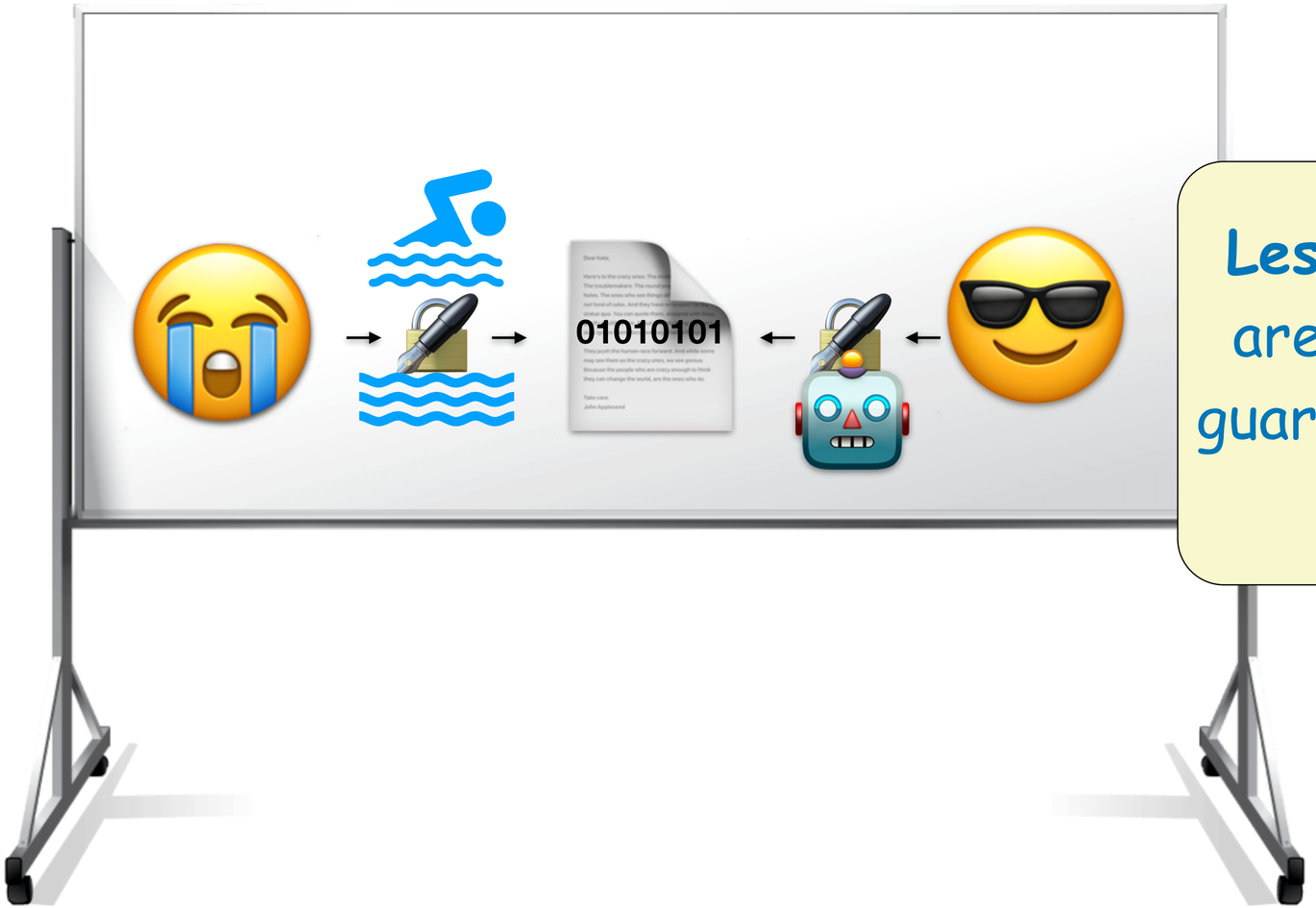
Transaction “Frontrunning”



Transaction “Frontrunning”



Transaction “Frontrunning”



Lesson: Transactions are public, and aren’t guaranteed to be mined in order

Randomness

- **The blockchain does not provide any cryptographically secure source of randomness**
  - Block hashes are random, but miners can manipulate them
  - Miners can also influence timestamps

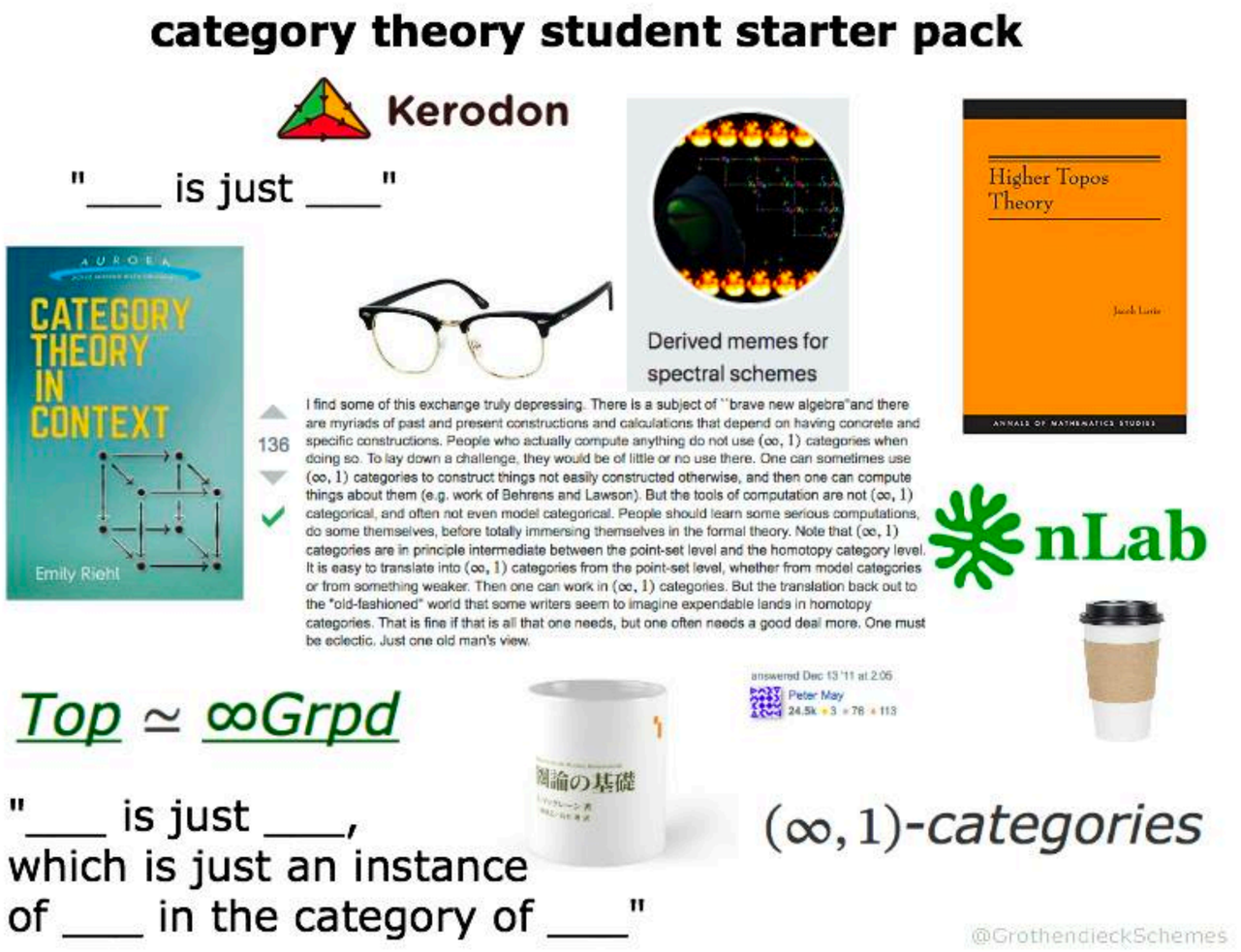
Randomness

- **The blockchain does not provide any cryptographically secure source of randomness**
  - Block hashes are random, but miners can manipulate them
  - Miners can also influence timestamps
- **Everything in a contract is publicly visible**
  - Random numbers can't be generated until after all lottery entries have been recorded

Randomness

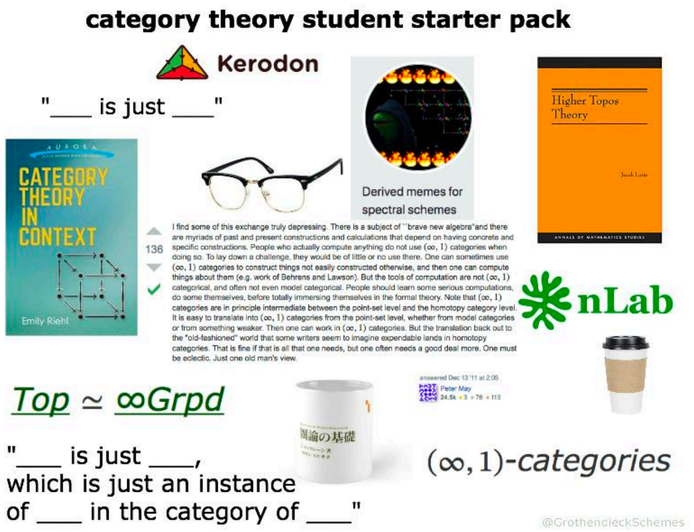
- **The blockchain does not provide any cryptographically secure source of randomness**
  - Block hashes are random, but miners can manipulate them
  - Miners can also influence timestamps
- **Everything in a contract is publicly visible**
  - Random numbers can't be generated until after all lottery entries have been recorded
- **Computers will always be faster than the blockchain**
  - Any number a contract could generate can be pre-calculated off-chain faster

Don’t try to be clever with number theory



Don’t try to be clever with number theory

```
winner = entries[blockHash % entries.length];
```



Everybody with me!

You can't do random on a blockchain

Everybody with me!

You can't do ran
on a blockchain

Lesson: If you really need randomness, use a trusted off-chain oracle.

Pre-Signed Transfers

```
function signedTransfer(
    address to,
    uint256 tokens,
    address feeRecipient,
    uint256 fee,
    uint256 expiry,
    bytes32 nonce,
    uint8 v, bytes32 r, bytes32 s) external returns (bool success) {
    bytes32 releaseHash = keccak256(abi.encodePacked(
       "\x19\x01",
       DOMAIN_SEPARATOR,
       keccak256(abi.encode(SIGNEDTRANSFER_TYPEHASH, to, tokens, feeRecipient, fee, expiry, nonce))
    ));

    address from = ecrecover(releaseHash, v, r, s);

    approvals[from][msg.sender] = add(tokens, fee);

    transferFrom(from, to, tokens);
    transferFrom(from, feeRecipient, fee);

    return true;
}
```

Pre-Signed Transfers

```
 1 function signedTransfer(
 2     address to,
 3     uint256 tokens,
 4     address feeRecipient,
 5     uint256 fee,
 6     uint256 expiry,
 7     bytes32 nonce,
 8     uint8 v, bytes32 r, bytes32 s) external returns (bool success)
 9     bytes32 releaseHash = keccak256(abi.encodePacked(
10         "\x19\x01",
11         DOMAIN_SEPARATOR,
12         keccak256(abi.encode(SIGNEDTRANSFER_TYPEHASH, to, tokens, f
13     ));
14
15     address from = ecrecover(releaseHash, v, r, s);
16
17     approvals[from][msg.sender] = add(tokens, fee);
18
19     transferFrom(from, to, tokens);
20     transferFrom(from, feeRecipient, fee);
21
22     return true;
23 }
```

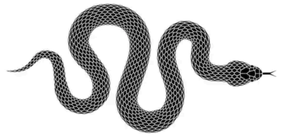
Lesson: Always check the return value of ecrecover! Better yet, avoid it!

What Can You Do About It?

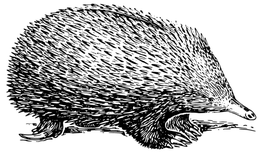
What can be done?

`https://github.com/trailofbits/...`

Manticore Symbolic Execution

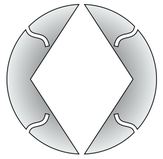


Slither Static Analysis



Echidna Property Based Fuzzer

Rattle EVM to SSA Lifter

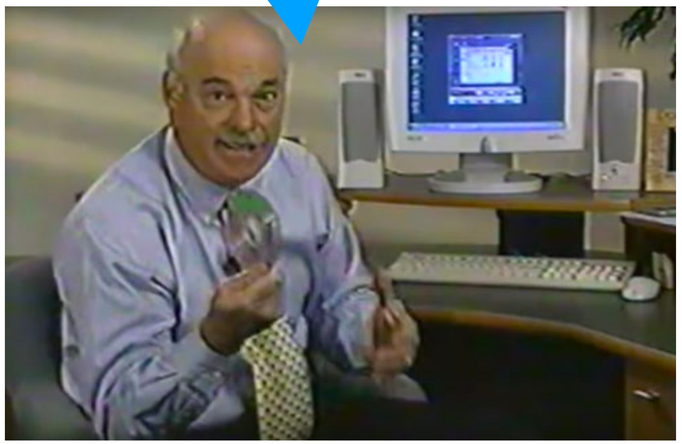


Etheno Test Framework Integration

Ethersplay Visual EVM Disassembler

pyevmasm Bytecode Analysis

Buy our free,
open source products.



evm-opcodes
VM Reference

blockchain-security-contacts
it's surprisingly hard to disclose bugs

not-so-smart-contracts
common vulnerability database

awesome-ethereum-security
security best practices

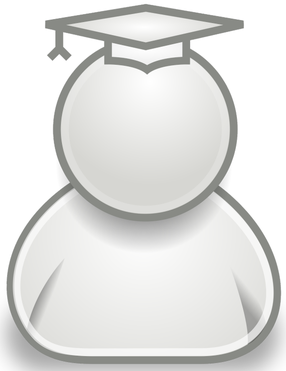
(Not So) Smart Contracts

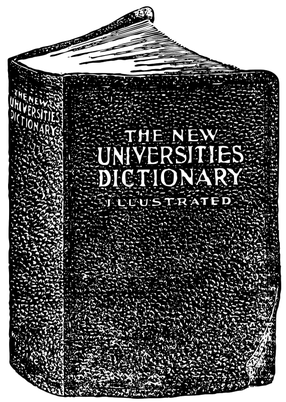
Educational Tool

Learn about EVM and Solidity Vulnerabilities

Working Examples of Contracts

Real Vulnerabilities Found in the Wild





Reference Material

Useful when Auditing Code

`https://github.com/crytic/not-so-smart-contracts`

Community Information

Awesome Ethereum Security

- **What?** Curated list of community-maintained and open-source references
- **Why?** Everything in one place: no more searching through stack overflow, github, and reddit
- **Features**
  - Resources for secure development, CTFs & wargames, and even specific podcast episodes
  - Identifies security tools for visualization, linting, bug finding, verification, and reversing
  - Pointers to related communities

Blockchain Security Contacts

- **What?** Comprehensive list of security contacts for blockchain applications
- **Why?** Projects worth $10MM+ should have a way to engage with security researchers
- **Features**
  - Vuln disclosure program best practices
  - Deployed addresses template for dapps
  - Existing contact info for over 100 projects (Blockchains, dapps, ERC20 and 721 tokens, Exchanges, Wallet software)

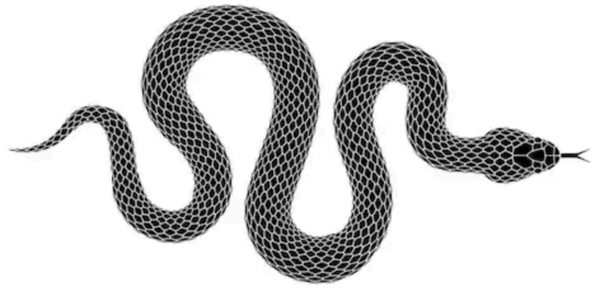
`https://github.com/crytic/awesome-ethereum-security` and `/blockchain-security-contacts`

Slither

Smart Contract Static Analysis

- Solidity and Vyper vulnerability detection
- Low false positives
- Easily integrates into CI pipeline
- Very fast (milliseconds)
- Supports advanced value- and taint-tracking
- Python-based detector API

- **Inputs:** Solidity code
- **Outputs:**
  - Detected errors (extensive list of vulnerability detectors included)
  - Warnings of poor coding practices
  - Inheritance graph and contract summary



Slither is open source!

`https://github.com/crytic/slither`

Slither Installation and Usage

```
$ pip3 install slither-analyzer
```

then

```
$ slither contract.sol
```

or

```
$ truffle compile; slither .
```

That's literally it!

Slither Installation and Usage

```
$ pip3 install slither-analyzer
```

then

```
$ slither contract.sol
```

or

```
$ truffle compile; slither .
```

That's literally it!

Lesson: Slither is super easy and quick! No excuse not to integrate it in your CI pipeline.

Problem: Test for New Bugs

```
contract Simple {
    function f(uint a){
        // .. lot of paths and conditions

        if (a == 65) {
            // leads to a bug here
        }
    }
}
```

Problem: Test for New Bugs

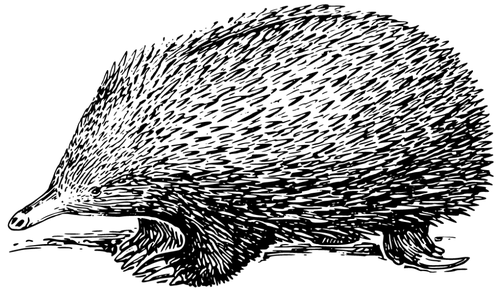
```
contract Simple {
    function f(uint a){
        // .. lot of paths and

        if (a == 65) {
            // leads to a bug h
        }
    }
}
```

Echidna

Smart Contract Property Tester

- Generates and execute many contract inputs
- Generate intelligent, grammar-based inputs
- Seamlessly integrate into developer workflows
- Run thousands of generated inputs per second
- Automatically generate minimal testcases
- Highly extensible via Haskell API

- **Inputs:** Solidity code and tests
- **Outputs:**
  - List of invariants Echidna was able to violate
  - Minimal call sequence to trigger discovered violations



Echidna is open source!

`https://github.com/crytic/echidna`

Echidna Example

```
➜ echidna git:(master) ✗ cat solidity/cli.sol
pragma solidity ^0.4.16;

contract Test {
  bool private flag0=true;
  bool private flag1=true;

  function set0(int val) returns (bool){
    if (val % 10 == 0) {flag0 = false;}
  }
  function set1(int val) returns (bool){
    if (val % 10 == 0 && flag0) {flag1 = false;}
  }
  function echidna_alwaystrue() returns (bool){
    return(true);
  }
  function echidna_sometimesfalse() returns (bool){
    return(flag0 || flag1);
  }
}
➜ echidna git:(master) ✗ ./echidna-test solidity/cli.sol
── solidity/cli.sol ──
  ✗ "echidna_sometimesfalse" failed after 36 tests and 681 shrinks.

      | Call sequence: set0(7946810797001355118938603703351564369838113269809310950469780);
      |                set1(80453298035196525130529691613626476953794039948107547184640199506677 60);

  ✓ "echidna_alwaystrue" passed 100 tests.
  ✗ 1 failed, 1 succeeded.
➜ echidna git:(master) ✗
```

Echidna Example

```
➜ echidna git:(master) ✗ cat solidity/cli.sol
pragma solidity ^0.4.16;

contract Test {
  bool private flag0=true;
  bool private flag1=true;

  function set0(int val) returns (bool){
    if (val % 10 == 0) {flag0 = false;}
  }
  function set1(int val) returns (bool){
    if (val % 10 == 0 && flag0) {flag1 = false;}
  }
  function echidna_alwaystrue() returns (bool){
    return(true);
  }
  function echidna_sometimesfalse() returns (bool){
    return(flag0 || flag1);
  }
}
➜ echidna git:(master) ✗ ./echidna-test solidity/cli.sol
── solidity/cli.sol ──
  ✗ "echidna_sometimesfalse" failed after 36 tests and 681 shrinks.

      | Call sequence: set0(7946810797001355118938603703351564369838113269809310950469780);
      |                set1(804532980351965251305296916136264769537940399481075471846401995066776O);

  ✓ "echidna_alwaystrue" passed 100 tests.
  ✗ 1 failed, 1 succeeded.
➜ echidna git:(master) ✗
```

Lesson: Echidna is not as fast as Slither, but it is still quick enough to be useful in your CI pipeline. Unlike Slither, it is capable of discovering wholly new classes of bugs.

Manticore

Smart Contract Verifier

- Uses symbolic execution of EVM
- Deeply explores possible contract states across multiple transactions and contracts
- Discover functions directly from bytecode
- Detect contract flaws like int overflows, uninitialized memory/storage usage, and more
- Verify customized program assertions
- Highly scriptable and extensible via Python API

- **Inputs:** Solidity code (optional) or raw EVM bytecode
- **Outputs:**
  - List of detected flaws
  - Verified properties
  - Execution traces of discovered paths

Manticore is open source!

`https://github.com/trailofbits/manticore`

Manticore Example

```
contract Simple {
    function f(uint a){
        // .. lot of paths and conditions

        if (a == 65) {
            revert();
        }
    }
}
```

Manticore Example

```
contract Simple {
    function f(uint a){
        // .. lot of paths and conditions

        if (a == 65) {
           revert();
        }
    }
}
```

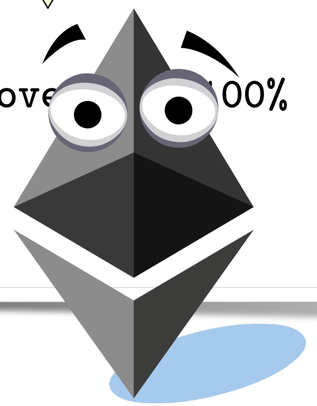
```
$ manticore simple.sol
2018-02-28 17:06:21,650: [25981] m.main:INFO: Beginning analysis
2018-02-28 17:06:21,803: [25981] m.ethereum:INFO: Starting symbolic transaction: 1
2018-02-28 17:06:22,098: [25981] m.ethereum:INFO: Generated testcase No. 0 - REVERT
2018-02-28 17:06:23,185: [25981] m.ethereum:INFO: Generated testcase No. 1 - REVERT
2018-02-28 17:06:24,206: [25981] m.ethereum:INFO: Finished symbolic transaction: 1 | Code Coverage: 100% |
Terminated States: 3 | Alive States: 1
2018-02-28 17:06:24,213: [32058] m.ethereum:INFO: Generated testcase No. 2 - STOP
2018-02-28 17:06:25,269: [25981] m.ethereum:INFO: Results in /examples/mcore_zua0Yl
```

Manticore Example

```
contract Simple {
    function f(uint a){
        // .. lot of paths and conditions

        if (a == 65) {
           revert();
        }
    }
}
```

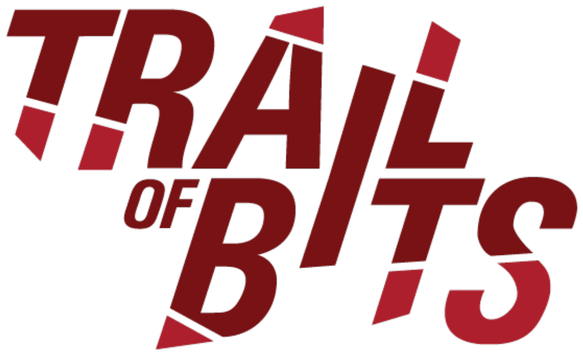
```
$ manticore simple.sol
2018-02-28 17:06:21,650: [25981] m.main:INFO: Beginning analysis
2018-02-28 17:06:21,803: [25981] m.ethereum:INFO: Starting symbolic ti
2018-02-28 17:06:22,098: [25981] m.ethereum:INFO: Generated testcase No. 0 - REVERT
2018-02-28 17:06:23,185: [25981] m.ethereum:INFO: Generated testcase No. 1 - REVERT
2018-02-28 17:06:24,206: [25981] m.ethereum:INFO: Finished symbolic transaction: 1 | Code Cove    00% |
Terminated States: 3 | Alive States: 1
2018-02-28 17:06:24,213: [32058] m.ethereum:INFO: Generated testcase No. 2 - STOP
2018-02-28 17:06:25,269: [25981] m.ethereum:INFO: Results in /examples/mcore_zua0Yl
```

Manticore can verify that your code satisfies its invariants, but it can take a long time to run!



Conclusions

- Solidity isn't a great language, but we're stuck with it (for now)
- Don't assume Solidity behaves like a "normal" language
- Don't trust the Solidity documentation; the sole compiler implementation is canon
- Don't enable Solidity compiler optimizations
- Avoid the "DELEGATECALL" upgrade pattern
- Don't trust calls to external contracts
- Remember that everything on the blockchain is public
- Don't assume transactions will be mined in order (or at all!)
- Read "(Not So) Smart Contracts"
- Add Slither and Echidna into your CI pipeline
- Use Manticore to verify the correctness of your contracts



Thanks!

@ESultanik

Thanks!

@ESultanik

TRAIL OF BITS

Acknowledgements

Ryan Stortz	`@withzombies`
Jay Little	`@computerality`
Josselin Feist	`@montyly`
Stefan Edwards	`@lojikil`
JP	`@japesinator`

Et pl. al.