Writing Smart Contracts
10 Coding

Peter H. Gruber

Supported by the Algorand Foundation

Why good style?

Avoid errors

- Save time (searching for errors)
- Save embarrassment (if you don't find them)

Remember

- We usually overestimate how much we remember
- We usually underestimate the number of future changes

Collaborate

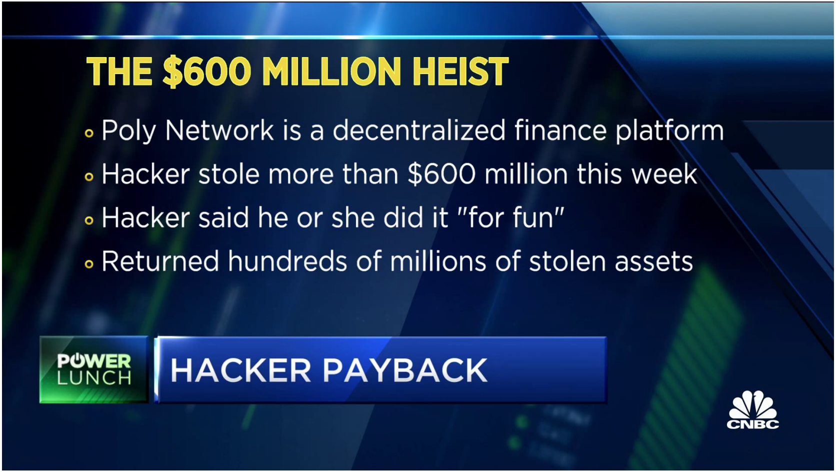
- Common standard with co-autohors, co-workers
- What happens if you change jobs?

Reproduce

- Document how every step from the data to your results
- Publish code to allow for reproducibility

Remember: If "code is law", then code quality is the **top priority**

Why good style?



THE $600 MILLION HEIST

- Poly Network is a decentralized finance platform
- Hacker stole more than $600 million this week
- Hacker said he or she did it "for fun"
- Returned hundreds of millions of stolen assets

HACKER PAYBACK

Polygon heist on Aug 10, 2021

Writing

Phase 1: Planning

- Start with an empty sheet of paper.
- Write down the contact logic.
- *Make a flowchart.
- Solve a sufficiently general problem.

Phase 2: Coding

Have a reader on your mind: your supervisor, a colleague, grandma. Would they understand what you are writing?

- Write clearly – don't be too clever.
- Use a standard code layout
  - Contract condition
  - Fee condition, security condition, . . .
- Readability: Empty lines + spaces
- Write and test a big contract in small pieces.
- Use versioning.
- Adopt a naming convention.

Blockchain arithmetic

Floating point is not unique

```
3.3 + 8.8 == 12.1
```

```
False
```

Integer arithmetic

- Integer amounts in “small” units (depends on ASA definition)
- Rounding down, e.g. `Int(3/4)`
- Start with large numbers, divide at the end

```
3/4 * Txn.amount()        # will always be zero
```

```
Txn.amount() * 3/4        # will work
```

- Make sure parts add up to one

```
Txn.amount() * 2/3                 # possible rounding error
Txn.amount() * 1/3                 # These 2 lines may not add up
```

```
Txn.amount() * 2/3                 # Offset possible rounding error
Txn.amount() - Txn.amount() * 2/3  # These 2 lines will always add up
```

Phase 3: Documenting

Code comments

- For programmers
  - Make your comments count: Do not translate code to English.
  - Explain *your* thinking and background.
- Document changes in a changelog.
- Code and comments should agree.

User Documentation

- For non-programmers
  - Address and bytecode (consider QR codes)
  - How to propose/calculate a valid transaction

Project Documentation

- Goal/economics/description of the project
- User guide
- Sample transactions and interpretation
- Data format (if needed), references
- Management of credentials

Phase 4: Testing

- Test in an organized, modular way
  - Make a test plan.
  - Verify code coverage ("did you check all cases?")
  - Devise a test suite (=test transactions)
- Test for different input and deliberately false input
- Test at boundary values (e.g. zero)

Test for attacks

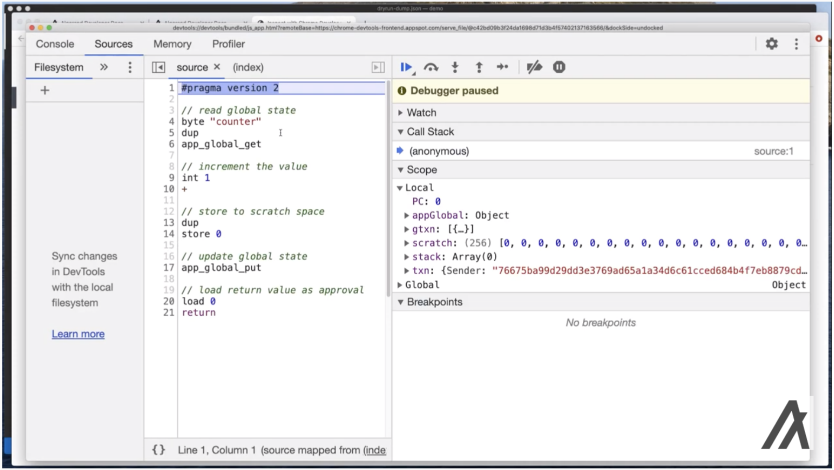
- Transaction fee attack
- Rekey attack
- Closeout attack
- Transaction group attack
- Cross-asset attack

Debugging

Debugging

- Main principle: prevention is better than repair
- Document the entire process
- Read the error message carefully
  If you don't understand it, google it
- Rerun notebook from the top
- Simplify program by commenting out code (#)
- Read your code carefully (print/read on tablet)
  Compare code to notes (step 1)
- Run the code in your head: is it doing what you think it should?
- Simplify/check expressions
  Nested brackets: run from innermost to outermost

Debugging



dryrun-dump.json — demo
devtools://devtools/bundled/js_app.html?remoteBase=https://chrome-devtools-frontend.appspot.com/serve_file/@c42bd09b3f24da1698d71d3b4f57402137163566/&dockSide=undocked
Console
Sources
Memory
Profiler
Filesystem
+
Sync changes in DevTools with the local filesystem
Learn more
source ×
(index)
Debugger paused
Watch
Call Stack
(anonymous)
source:1
Scope
Breakpoints
No breakpoints
{}
Line 1, Column 1 (source mapped from (inde

```
#pragma version 2

// read global state
byte "counter"
dup
app_global_get

// increment the value
int 1
+

// store to scratch space
dup
store 0

// update global state
app_global_put

// load return value as approval
load 0
return
```

```
▼ Local
    PC: 0
  ▶ appGlobal: Object
  ▶ gtxn: [{…}]
  ▶ scratch: (256) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
  ▶ stack: Array(0)
  ▶ txn: {Sender: "76675ba99d29dd3e3769ad65a1a34d6c61cced684b4f7eb8879cd…
▶ Global                                                        Object
```

- Install local node with use debugger
- `https://developer.algorand.org/articles/introducing-algorands-smart-contract-debugger/`

Audits

Audits

Auditing firms

- https://runtimeverification.com
- https://halborn.com
- https://www.coinfabrik.com
- https://www.trailofbits.com
- https://dapp.org.uk

Examples for Audits

https://github.com/runtimeverification/publications

Reading list for attacks

- The biggest smart contract hacks in history
  https://medium.com/solidified/d5a72961d15d
- Smart Contract Audits (good read)
  https://www.ulam.io/blog/smart-contract-audit/