Enter the Hydra: Toward Principled Bug Bounties and Exploit-Resistant Smart Contracts

EPFL SuRI
18 June 2018
What’s a Smart Contract?
Smart contracts

- Small programs that run on blockchains
- Given trust in underlying blockchain, smart contracts are
  - Transparent
  - Irreversible
  - Tamper-resistant
- ...plus they can act upon crypto tokens = $money
Lots of recent interest in ETH...
Why? Suppose Alice and Bob want to trade..

Problem of Fair Exchange!
Trusted third-party (with public state)
Smart contract $\approx$ Trusted third-party (with public state)
No, not Floyd Mayweather...
Floyd 'Crypto' Mayweather promotes an ICO, again

You can call me Floyd Crypto Mayweather from now on...Hubii.Network

#ICO starts tomorrow! Smart contracts for sports?! #HubiiNetwork
#CryptoMediaGroup 😊

AUGUST 23
Crypto Tokens

• Application-specific cryptocurrency
• Mainly ERC20 tokens
  • Managed in Ethereum smart contracts
• $38+ billion token market cap
Crypto Tokens

- Sold in Initial Coin Offerings (ICOs)
  - a.k.a. Token Launch, Token Generation Events (TGEs), etc.
  - Like unregulated VC
  - Token like a share (kind of…)
- Since mid-2017, ICO funding outstripping early-stage Internet VC (!)
Crypto Tokens: ERC721

• “Non-fungible tokens”: Represent unique objects
SMART CONTRACT CHALLENGES

1. **Correctness**: Contracts often have fatal bugs!
2. **Confidentiality**: No private data.
3. **Authenticated data**: No good, trustworthy access to real-world data!
Side effects of the token mania

- Token smart contracts are compact
- Lots of money per contract
- Astonishing value per line of code
- Which makes for juicy targets...

<table>
<thead>
<tr>
<th>Token</th>
<th>Lines of Code</th>
<th>Value per line</th>
</tr>
</thead>
<tbody>
<tr>
<td>OmiseGo (OMG)</td>
<td>396</td>
<td>~$2.4M</td>
</tr>
<tr>
<td>Tether (USDT)</td>
<td>423</td>
<td>~$5.9M</td>
</tr>
<tr>
<td>EOS (EOS)</td>
<td>584</td>
<td>~$15.8M</td>
</tr>
</tbody>
</table>

Sources: coinmarketcap.com, 14 June 2018, and published contract source code
Some (in)famous smart contracts

• The DAO (June 2016)
  • Reentrancy bug ⇒ $50+ million stolen

• Parity multisig hack (July 2017)
  • Parity 1.5 client’s multisig wallet contract
  • Problem with library contract use ⇒ $30 million stolen
    …from 3 ICO wallets (Edgeless Casino, Swarm City, and æternity)

• Parity multisig hack—Redux! (Nov. 2017)
  • Problem with library contract ⇒ >$150 million frozen
    • …much from ICO wallets (Polkadot, $98 million)
Why not try to address correctness with…

• Formal verification
  • Absolutely!
  • But limited scaling
  • What if there’s a bug in the formal spec? (Turtles!)

• Static and dynamic verification
  • Absolutely!
  • But limited scope
N-Version programming
(Chen & Avizienis ’78, Knight-Leveson ‘86)
N-Version programming
(Chen & Avizienis ’78, Knight-Leveson ‘86)

Input $X$

Version 1

Version 2

Version 3

Majority Vote

Agreed output

$N$ software versions / heads
If something goes wrong...

Input X

Version 1

Version 2

Version 3

Majority Vote

Agreed output

N software versions / heads
What is N-version programming doing?

A program transformation $T$ takes $N \geq 1$ programs and creates new program $f^* := T(f_1, f_2, \ldots, f_N)$.
Some more definitions

• Let $\mathcal{I}$ be an ideal program specification
  • Conceptual! Doesn’t actually exist…

• Let $f$ be an implemented program

• An exploit is an input $X$ such that $\mathcal{I}(X) \neq f(X)$

• Intuition: Any deviation from intended behavior is a potentially serious bug

• Exploit set $E(f, \mathcal{I})$: set of exploits $X$ for $f$ and $\mathcal{I}$
Mind the gap

- Let $\mathcal{D}$ be a distribution over inputs $X$
- Definition of exploit gap:

\[
gap := \frac{\Pr_{X \in \mathcal{D}} \left[ X \in \bigcup_{i=1}^{N} E(f_i, \mathcal{I}) \right]}{\Pr_{X \in \mathcal{D}} \left[ X \in E(f^*, \mathcal{I}) \right]}
\]

- Affirmative gap ($>1$) means $T$ reduces exploits
- Bigger gap $\Rightarrow$ fewer relative bugs in $f^*$
- $\gap$ captures dependencies among heads

Exploits against $f_1, f_2, f_3 \ldots$
Exploits against $f^*$
Houston... we have a gap

\[
gap := \frac{\Pr_{X \in \mathcal{D}} \left[ X \in \bigcup_{i=1}^{N} E(f_i, I) \right]}{\Pr_{X \in \mathcal{D}} \left[ X \in E(f^*, I) \right]}
\]

Input \( X \)

\( f^* \)

\( f_1 \)

\( f_2 \)

\( f_3 \)

Version 1

Version 2

Version 3

\( N \) software versions / heads

Majority Vote

Agreed output
N-version-programming criticism

• Strong gap requires independence among heads
  • Correlations hurt!

• Knight-Leveson (1986):
  • “We reject the null hypothesis of full independence at a p-level of 5%”

• Eckhardt et al. (1991):
  • “We tried it at NASA and it wasn’t cost effective”
  • Worst case: 3 versions ⇒ 4x fewer errors
But not everything is a space shuttle…

- Not all software needs to be available at all times!
  - E.g., Smart contracts: How bad if it’s down for a while?
- In fact, often better no answer than the wrong one
  - Bugs are often harmful
- **N-of-N-Version Programming (NNVP)**
NNVP a.k.a. **Hydra Framework**

Idea: Strengthen majority vote of N-Version Programming
NNVP a.k.a. Hydra Framework

Unless all versions agree, abort!
NNVP a.k.a. **Hydra**

- **Aborting in NNVP:**
  Correctness ← Availability
- **NASA numbers much better for NNVP**
  - Some availability loss, but…
  - gap = 4,409 for $N = 3$ heads
  - gap = 34,546 for $N = 4$ heads
  - Probably even better!
Hydra creates a (strong) gap...

Serious bug in one head now rarely fatal...
Smart contracts are Hydra-friendly!

<table>
<thead>
<tr>
<th>Contract name</th>
<th>Exploit value (USD)</th>
<th>Root cause</th>
<th>Independence source</th>
<th>Exploit gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>The DAO* [19]</td>
<td>150M</td>
<td>Re-entrancy</td>
<td>language</td>
<td>✓</td>
</tr>
<tr>
<td>SmartBillions [20]</td>
<td>500K</td>
<td>Bug in caching mechanism</td>
<td>programmer</td>
<td>✓</td>
</tr>
<tr>
<td>HackerGold (HKG)* [21]</td>
<td>400K</td>
<td>Typo in code</td>
<td>programmer+language</td>
<td>✓</td>
</tr>
<tr>
<td>MakerDAO* [22]</td>
<td>85K</td>
<td>Re-entrancy</td>
<td>language</td>
<td>✓</td>
</tr>
<tr>
<td>Rubixi [23]</td>
<td>&lt;20K</td>
<td>Wrong constructor name</td>
<td>programmer+language</td>
<td>✓</td>
</tr>
<tr>
<td>Governmental [23]</td>
<td>10K</td>
<td>Exceeds gas limit</td>
<td>None?</td>
<td>✗</td>
</tr>
</tbody>
</table>

Hydra could probably have addressed cases in green and yellow vulnerabilities
Application: Bug Bounties
Bug bounties

• Reward for responsible disclosure of software vulnerabilities
• Key element of nearly all security assurance programs
  • E.g., Apple (up to $200k)
Some problems with bug bounties:

1. Bounties often fail to incentivize disclosure
   - Apple: ≤ $200k bounty
   - Zerodium: $1.5 million for certain iPhone jailbreaks

2. Time lag between reporting and action
   - Weaponization can happen after disclosure

3. Bounty administrator doesn’t always pay!
3. Bounty administrator doesn’t always pay!
The perfect bug bounty

1. **High leverage:** Small bounty incentivizes disclosure for valuable program

2. **Automatic payout:** Bounty hunter need not trust bounty administrator to pay
   - Censorship-resistant, verifiable

3. **Automatic remediation:** Immediate intervention in affected software
Bug bounties: The Rational Attacker’s Game

Program

Value: $A
Bug bounties: The Rational Attacker’s Game

Find
Exploit

Attack  Disclose

$A  $0

No bounty
Bug bounties: The Rational Attacker’s Game

Find

Exploit

Always attack!

$A$

$0$

No bounty
Bug bounties: The Rational Attacker’s Game

Find
Exploit

Attack
$A$

Disclose
$B$

Classic bounty: $B$
Bug bounties: The Rational Attacker’s Game

Disclose if $B > A$!

Classic bounty: $B$
Our goal: High leverage

Find
Exploit

Attack
$A/gap

Disclose
$B
Our goal: High leverage

For $\text{gap} \gg 1$
Our goal: High leverage

Find
Exploit

Disclose
$B > \frac{A}{\text{gap}}$

Attack
$A/\text{gap}^*$

Exploit gap
Wait a minute…

Program
Value: $A$

Disclose, i.e.,
don’t attack
even though
$B < A$ ?!
The Hydra Framework for Bug Bounties

Input $X$ -> Head 1 -> Head 2 -> Head 3 -> Agreed output $Y$
The Hydra Framework for Bug Bounties

Input $X$

Head 1

Head 2

Head 3

Fault manager

$\neq$

Abort

Pay $bounty$

$bounty$
The Hydra Hacker’s Dilemma

Claim bounty ($B$) now?  

Try to break all heads ($A$)?
Recall:

\[
gap := \frac{\Pr_{X \in \mathcal{D}} \left[ X \in \bigcup_{i=1}^{N} \mathcal{E}(f_i, \mathcal{I}) \right]}{\Pr_{X \in \mathcal{D}} \left[ X \in \mathcal{E}(f^*, \mathcal{I}) \right]}
\]
Hydra Framework $\rightarrow$ High leverage

- Suppose strong rational adversary discovers bugs as fast as all honest bounty hunters.
- Suppose:
  - Contract worth $A$
  - Bounty $B$
- Then (we prove) adversary discloses if:
  \[ B > A / (\text{gap} + 1). \]
Example

• Recall: NASA experiments imply:
  • $\text{gap} = 4,409$ for $N = 3$ heads
  • $\text{gap} = 34,546$ for $N = 4$ heads

• So…
  • After $\text{Approx} $1 billion contract (e.g., OmiseGo)
  • $N = 4$
  • $\$30k$ bounty incentivizes adversary to disclose!
The perfect bug bounty

1. **High leverage:** Small bounty incentivizes disclosure for valuable program

2. **Automatic payout:** Bounty hunter need not trust bounty administrator to pay
   - Censorship-resistant, verifiable

3. **Automatic remediation:** Immediate intervention in affected software
It’s a smart contract! It’s automatically automatic!

Input $X$

Head 1

Head 2

Head 3

$\text{Pay } \$\text{bounty}$

$\text{Pay } \$\text{bounty}$

$\text{f*}$
The perfect bug bounty

1. **High leverage:** Small bounty incentivizes disclosure for valuable program

2. **Automatic payout:** Bounty hunter need not trust bounty administrator to pay
   • Censorship-resistant, verifiable

3. **Automatic remediation:** Immediate intervention in affected software
How to remediate if contract fails?

• The DAO ($50+ million stolen)
  • **Remedy:** Fork returned money (in ETH-land) to victims

• Parity multisig hack ($30 million stolen)
  • *(Partial)* **Remedy:** White hats “stole” $78 mil.; returned money to victims
    • (Two co-authors of Hydra paper among these hackers…)

• Parity multisig hack—Redux! ($150 million frozen)
  • *(Proposed) Remedy:* Unfreeze funds and return to victims
The Hydra Framework for Bug Bounties

```
Fault manager = ?
```

Abort + Return $$$
The perfect bug bounty

1. “Strong exploit gap”: Small bounty incentivizes disclosure for valuable program
2. **Automatic payout**: Bounty hunter need not trust bounty administrator to pay
   • Censorship-resistant, verifiable
3. **Automatic remediation**: Immediate intervention in affected software
Smart contracts: Perfect bug-bounty targets

• Vulnerable:
  • Bug-prone / hard to code correctly
  • Many $$ per line of code

• But promising:
  • Hydra-friendly
  • Support (1) High leverage; (2) Automated payout; and (3) Reasonable remediation
  • **Bonus**: Automatic value-at-risk assessment
    • First opportunity to reason about bounty amounts in principled way!
Implementation

• ERC20
  • Standard token-management contract
  • $N = 3$
  • $bounty = 3\text{ETH} \approx 1500$
  • Deployed @0xf4ee935a3879ff07362514da69c64df80fa28622

• Generalized Monty-Hall game
  • Extension of Monty Hall game to $K$ out of $M$ doors
  • In progress
Submarine Commitments
Bug withholding

• Suppose adversary $A$ discovers bug $X$
• $A$ should disclose fast to prevent honest user claiming $bounty$
• Or should she?
Bug withholding

• Unfortunately, blockchains are messy…

• $A$ can front-run honest user!

• So $A$ can withhold $X$ and keep looking for full exploit of $f^*$

• Ruins our whole bounty analysis!
  • No immediate incentive to disclose compromise of individual heads!
Solution?

• Idea 1: Must commit in block $t-1$ to reveal claim in block $t$

• Problem: $A$ commits in every round and front-runs reveal!
Solution?

- Idea 2: Must commit $\text{deposit}$ in block $t-1$ to reveal claim in block $t$
Solution?

- Idea 2: Must commit $deposit$ in block $t-1$ to reveal claim in block $t$
- Problem: $deposit$ sent to Hydra Contract is publicly visible
  - So $A$ can front-run commit!

In general, if $A$ can observe honest users’ behavior, she can front-run them!
Solution: Submarine Commitment

- **Commit** sends $\text{deposit}$ to random address
- People send money to fresh addresses all the time!
- So **Commit** looks like ordinary traffic...
  - No visible association with Hydra Contract
Solution: *Submarine Commitment*

- \( R \) is specially constructed
- Only HydraContract can recover money from \( R \), with key \( \kappa \)
- **Reveal** sends key \( \kappa \)
- Key \( \kappa \) allows fund recovery by HydraContract
- Thus we can:
  - Commit \( \$\text{deposit} \) stealthily and
  - Prevent front-running!
Submarine Commitments

- Security analysis a bit involved:
  - New, strong adversarial model introduced for blockchains
Submarine Commitments

• We prove tight bounds on adversary’s front-running ability
• E.g., to protect $100,000 bounty with reasonable parameters in Ethereum, need $\text{deposit} = $278
• New, practical Ethereum implementation not in paper
  • We’re implementing it…
  • Good to protect against general front-running
The Hydra Project \textit{(alpha)}

Hydra is a cutting-edge Ethereum contract development framework for:

- decentralized security and bug bounties
- rigorous cryptoeconomic security guarantees
- mitigating programmer and compiler error

READ THE PAPER  TRY THE ALPHA  CHAT ON RIOT

www.thehydra.io

Paper to appear in USENIX Security 2018
Suppose that N players share cryptocurrency using an M-of-N multisig scheme. If N-M+1 players disappear, the remaining ones have a problem: They've permanently lost their funds. In this blog, we propose a solution to this critical problem using the power of the trusted hardware.

The Social Workings of a Contract
by Karen Levy on Wednesday January 17, 2018 at 01:00 PM

Guest blogger Prof. Karen Levy describes how contracts often include terms that are unenforceable, purposefully vague, or never meant to be enforced, how this helps set expectations, and what this means for smart contracts.

May 10-11, 2018
IC3 Spring Retreat in NYC
IC3 faculty, students and industry members gather twice per year to discuss the major technical challenges and innovative solutions to widespread blockchain adoption.

February 26, 2018 - March 2, 2018
Financial Cryptography and Data Security 2018 and the 5th Workshop on Bitcoin and Blockchain Research.
Prof. Sarah Meiklejohn is co-Program Chair for FC18 and Prof. Ittay Eyal is co-Program Chair for the 5th Workshop on Bitcoin and Blockchain Research.