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

• Reward given for responsible disclosure of vulnerabilities

• Linus’s Law
  • “Given enough eyeballs, all bugs are shallow.”
Bug bounties

- Reward given for responsible disclosure of vulnerabilities
- Linus’s Law
  - “Given enough eyeballs, all bugs are shallow.”
- Key element of nearly all security assurance programs
  - E.g., Apple (up to $200k)

<table>
<thead>
<tr>
<th>COMPANY</th>
<th>NEW</th>
<th>REWARD</th>
<th>SWAG</th>
<th>HALL OF FAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Password</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>123 Contact Form</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>99designs</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Abacus</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>ABN Amro</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Acorns LLC</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Acquia</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Active Campaign</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>
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!
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!
The perfect bug bounty

1. “Strong exploit gap”: Small bounty incentivizes disclosure for valuable program

2. Automatic remediation: Immediate intervention in affected software

3. Automatic payout: Bounty hunter need not trust bounty administrator to pay
   - Censorship-resistant, verifiable
Bug bounties: The Rational Attacker’s Game

Program

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

Find
Exploit

Attack
$A$

Disclose
$0$

No bounty
Bug bounties: The Rational Attacker’s Game

Find
Exploit
Always attack!

Attack
$A

Disclose
$0

No bounty
Bug bounties: The Rational Attacker’s Game

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

Find

Exploit

Disclose if $B > A$

Attack

$A$

$B$

Classic bounty: $B$
Our goal

Find
Exploit

Attack

$A/gap'$

Disclose

$B$

Hydra bounty $B$
Our goal

Find Exploit

Attack

$A/gap$

$B$

Disclose

Exploit gap

Hydra bounty $B$
Our goal

Find

Exploit

Disclose

Attack

Disclose if

$B > A / \text{gap}^*$

$A / \text{gap}^*$

$B$

Exploit

Hydra bounty $B$
How is this possible?

Program
Value: $A

Disclose, i.e., don’t attack, if $B > \frac{A}{\text{gap}^*}$!
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

$N$ software versions / heads

Majority Vote

Agreed output

✓
What is N-version programming doing?

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

![Diagram of N-version programming]

Input $X$  

Version 1  

Version 2  

Version 3  

Majority Vote  

output $Y$
Some more definitions

• Let $\mathcal{I}$ be an *ideal* program specification
• 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 gap:

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

• **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

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

\(f^*\)

Input \(X\)

\(f_1\)

Version 1

\(f_2\)

Version 2

\(f_3\)

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 $\Rightarrow$ 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
• \textit{N-of-N-Version Programming (NNVP)}
NNVP a.k.a. **Hydra**

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

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

- Aborting in NNVP:
  - Correctness $\gg$ Availability
- Eckhardt et al. numbers much better for NNVP
  - Some availability loss, but…
  - 30 – 5087 times fewer failures!
- Probably even better!
  - Eckhardt et al. didn’t identify independent bugs
Example: N-Version Programming

N software versions

Input

Version 1

Version 2

Version 3

Output selector

Agreed output

✗
Example: Hydra

```
input

Head 1
Head 2
Head 3

N software versions

Fault manager

Agreed output
```

Software versions = ?

Example: Hydra
Example: Hydra

input

Head 1

Head 2

Head 3

N software versions

= ?

Fault manager

Agreed output

= ?
Example: Hydra

input

Head 1

Head 2

Head 3

N software versions

Fault manager

Agreed output

$N$ software versions = ?
Hydra creates a (strong) exploit gap...

A serious bug / exploit is no longer fatal...
Our goal

Find Exploit

Attack

$A/gap^*$

Disclose

$B$

Exploit gap
The perfect bug bounty

1. "Strong exploit gap": Small bounty incentivizes disclosure for valuable program

2. Automatic remediation: Immediate intervention in affected software

3. Automatic payout: Bounty hunter need not trust bounty administrator to pay
   • Censorship-resistant, verifiable
Target Application: Smart Contracts
Smart contracts

- Small programs that run on blockchains
- Given trust in underlying blockchain, smart contracts are
  - Incorruptible
  - Transparent
  - Irreversible
- ...plus they can act upon
  
  **crypto tokens = $money**
Lots of recent interest in ETH...

$USD

< $7 billion + < $15 billion < > $29 billion
Suppose Alice and Bob want to trade

10 Bob’s Bubble Tokens (BBT) → 1 ETH

Problem of *Fair Exchange*!
Trusted third-party (with public state)
Virtual trusted third-party (with public state)
No, not Floyd Mayweather...
Floyd 'Crypto' Mayweather promotes an ICO, again

Mashable
AUG 24, 2017

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

• Mainly ERC20 tokens
  • Application-specific currency in Ethereum smart contracts

• Sold in Initial Coin Offerings (ICO)
  • a.k.a. Token Launch, Token Generation Events (TGEs), etc.
  • Like unregulated VC

• Since mid-2017, ICO funding outstripping early-stage Internet VC (!)
Side effects of the mania

- ERC20 smart contracts are compact
- Lots of money per contract
- Astonishing value per line of code
- Which can cause problems...

<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>~$1.59M</td>
</tr>
<tr>
<td>Tether (USDT)</td>
<td>423</td>
<td>~$1.11M</td>
</tr>
<tr>
<td>EOS (EOS)</td>
<td>584</td>
<td>~$1.01M</td>
</tr>
</tbody>
</table>

Sources: coinmarketcap.com, 3 Nov., 8:20 a.m. and published contract source code
Some (in)famous smart contracts

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

• Parity multisig hack
  • Parity 1.5 client’s multisig wallet contract
  • Bad use of delegatecall $\Rightarrow$ $30$ million stolen
    …from 3 ICO wallets (Edgeless Casino, Swarm City, and æternity)
  • Remedy: White hats stole $78$ mil.; returned money to victims
    • (Two authors of paper among these hackers…)
Some (in)famous smart contracts

• Last week! Parity multisig hack—Redux!
  • Bad use of delegatecall $\Rightarrow$ >$150$ million frozen
    • …much from ICO wallets (Polkadot, $98$ million)
    • Accidentally triggered (???)
  • Easy fix (with hard fork): Unfreeze funds
Some (in)famous smart contracts

- Last week! Parity multisig hack—Redux!
  - Bad use of delegatecall ⇒ >$150 million frozen
  - Accidentally triggered
  - Easy fix (with hard fork): Unfreeze funds

Three lessons:

(1) Hard to write correct smart contracts!
(2) Often clear ways to remediate failures
### (3) 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
Smart contracts: perfect bug-bounty targets

• Vulnerable:
  • Bug-prone / hard to code correctly
  • Many $$$ per line of code
• But promising:
  • Hydra-friendly
  • Reasonable remediation possible
    • E.g., return the money
  • Automatic bounty payout possible
    • They’re smart contracts, after all!
• Bonus: Automatic value-at-risk assessment
  • First opportunity to reason about bounty amounts in principled way!
The perfect bug bounty

1. “Strong exploit gap”: Small bounty incentivizes disclosure for valuable program

2. **Automatic remediation**: Immediate intervention in affected software

3. **Automatic payout**: Bounty hunter need not trust bounty administrator to pay
   - Censorship-resistant, verifiable
Hydra Framework and Implementation
The Hydra Framework

Input $X$  

Head 1  

Head 2  

Head 3  

Agreed output $Y$  

$f^*$
The Hydra Framework

Input X

Head 1

Fault manager

 Abort

Head 2

Head 3

f*
The Hydra Framework

Input $X$

Head 1

Head 2

Head 3

Fault manager

Abort

+ Return

$$$

$ f^*$
What does framework buy us?

- Recall:

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

- Suppose we have an adversary and a collection of honest players finding / sampling bugs (from distribution \( \mathcal{D} \))
  - Adversary at rate \( \lambda_A \)
  - Honest players at rate \( \lambda_H \)
What does framework buy us?

• Suppose:
  • Contract is worth $balance$

• Then:
  • Example: $\lambda_{\mathcal{H}} = \lambda_{\mathcal{A}}$, i.e., $\mathcal{A}$ and honest players equally powerful…
  • $\Pr[\mathcal{A} \text{ compromises all heads}] = \frac{1}{\text{gap} + 1}$
  • And $\mathcal{A}$ otherwise incentivized to disclose bug if:

\[
$bounty > \frac{1}{\text{gap}+1} \cdot balance.$
\]
Example

• Results of Eckhardt et al. experiment imply:
  • gap = 4,409 for N = 3 heads
  • gap = 34,546 for N = 4 heads

• Thus, e.g.,
  • If contract worth $675 million (e.g., OmiseGo)
  • N = 4
  • $\text{bounty}$ of $20k incentivizes $A$ to disclose!

\[ \text{\textcolor{green}{bounty}} > \frac{1}{\text{gap}+1} \cdot \text{\textcolor{green}{balance}}. \]
Implemented Hydra bounty contracts

• ERC20
  • Standard token-management contract
  • $N = 3$
  • $\texttt{bounty} = 3\text{ETH} \approx $900.
  • **Deployed** @ 0xf4ee935a3879ff07362514da69c64df80fa28622

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

• Unfortunately, blockchains are messy…
• Suppose adversary $A$ discovers bug $X$
• $A$ should disclose fast to prevent honest user claiming $\$bounty$
Bug withholding

• But $A$ 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 $t$
  + Lots of cover traffic

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

Hydra Contract

$\text{C}(X') \text{ C}(X''')$

$\text{C}(X') \text{ C}(X'')$

$bounty$
Solution?

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

- Idea 2: Must commit \( \text{deposit} \) in block \( t-1 \) to reveal claim in block \( t \)
- Problem: \( \text{deposit} \) 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 $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*

- But actually, $R$ is specially constructed
- Only HydraContract can recover money from $R$, with key $κ$
- **Reveal** sends key $K$
- Key $κ$ 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

\[ \mathcal{F}_{\text{withhold}} \text{ with } P = \{P_0, P_1, \ldots, P_m\}, (\delta, \rho)\text{-adversary } A, \text{ blocksize } s, \text{ target height } n \]

Init: \( B \leftarrow \emptyset, B.\text{Height} \leftarrow 0, \text{MaxHeight} \leftarrow 0, \text{Mempool} \leftarrow \emptyset \)

On receive \( \text{("post", } r \text{)} \) from \( P_i \) \( \text{ // } P_i \text{ submits tx} \)
  assert \( \text{ValidTx}(r; B, \text{Mempool}) \)
  \( \text{tag}(r) \leftarrow (B.\text{Height}, P_i) \quad \text{ // Label tx with current chain height and sender} \)
  \( \text{Mempool} \leftarrow \text{Mempool} \cup r \)
  send \( \text{Mempool} \) to \( A \)

On receive \( \text{("add block", } B \text{)} \) from \( A \) \( \text{ // } A \text{ extends blockchain} \)
  if \( B.\text{Height} = n \) then
    output \( B \); halt \( \text{ // To complete chain, } A \text{ adds arbitrary } n + 1^{\text{th}} \text{ block} } \)
  assert \( (|B| = s) \land (B \subseteq \text{Mempool}) \)
  assert \( \exists r \in \text{Mempool} - B \ s.t. \ (\text{tag}(r) = (h, P_0)) \land (h \leq B.\text{Height} - \delta) \)
  \( \text{ // Ensure delay at most } \delta \text{ for } P_0 \text{’s transactions} \)
  \( B.\text{Height} \leftarrow B.\text{Height} + 1 \)
  \( B_{\text{maxHeight}} \leftarrow B \text{ // Add new block to chain} \)
  \( \text{Mempool} \leftarrow \text{Mempool} - B \text{ // Remove processed txs from Mempool} \)
  \( \text{MaxHeight} \leftarrow \max(B.\text{Height}, \text{MaxHeight}) \)
  send \( B \) to \( P_0 \)

On receive \( \text{("rewind", } r \text{)} \) from \( A \) \( \text{ // } A \text{ rewinds by } r \text{ blocks} \)
  assert \( \text{MaxHeight} - (B.\text{Height} - r) \leq \rho \)
  \( \text{ // Ensure that } A \text{ rewinds by no more than } \rho \)
  \( \text{Mempool} \leftarrow \text{Mempool} \cup \{B_i\} \in [B.\text{Height} - r + 1, \text{MaxHeight}] \)
  \( \text{ // Return rewound transactions to Mempool} \)
  \( B.\text{Height} \leftarrow B.\text{Height} - r \)

Figure 2: Ideal functionality \( \mathcal{F}_{\text{withhold}} \) for \((\delta, \rho)\)-adversary \( A \)
Submarine Commitments

• Security analysis a bit involved:
  • New, strong adversarial model introduced for blockchains
• Standard cryptographic modeling of adversaries... but with money

![Figure 4: Adversarial game $\text{Exp}_A^{\text{btyrace}}$](image)
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 $deposit = $278
• Caveat: Most practical Ethereum implementation requires EIP-86
  • On roadmap…
The Hydra Project [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

www.thehydra.io
Initiative for CryptoCurrencies and Contracts (IC3)

IC3: ADVANCING THE SCIENCE AND APPLICATIONS OF BLOCKCHAINS

Latest on Blog

To Sink Frontrunners, Send in the Submarines
by Lorenz Breidenbach, Phil Daian, Ari Juels, and Florian Tramèr on Monday August 28, 2017 at 05:01 AM

We discuss a novel scheme for preventing (miner) frontrunning in Ethereum.

Who Has Your Back in Crypto?
by Emin Gün Sirer on Saturday August 26, 2017 at 01:55 PM

Between miners, businesses and developers, people think that the developers have their best interests at heart. I discuss why this is a fallacy.

News & Events

October 31, 2017
Coin Desk: IC3 Blockchain Initiative Expands Research Team to Europe

Friday October 6, 2017
Blockchains Workshop
Cornell Tech is opening its new campus to the community in a series of public events. The Blockchains Workshop is co-organized by IC3. Registration is open to the public.