Governance Attacks

Dan Boneh
Stanford University
Lots of governance experiments

Some of them not going so well:

- Attack on Beanstalk governance: $182M
- Attack on Yam Finance governance: thwarted
- Attack on Build Finance governance: $1.5M
- Attack on Mango governance: $115M
- Steem governance drama

What happened? What can we learn?
An Ethereum-based stablecoin **Bean**. Governance token **Stalk**.

**The problem:**
- an attacker can buy Stalk tokens,
- submit a proposal,
- vote on proposal, and
- have proposal execute all in a single transaction
Review: **flashloan**  (has many applications)
- a loan that is taken out and repaid back in a single transaction
- No risk to lender $\Rightarrow$ unbounded amount can be borrowed

**The attack:**
- Attacker took a huge flashloan from Aave, bought lots of Stalk,
- Passed a proposal to pay $80M to the attacker from the treasury,
- Sold the Stalk, and repaid the flashloan,
- Sent the proceeds to Tornado cash (donated $250K to Ukraine)
The lesson

The protocol relaunched four months later, and is still around.

**The lesson:** governance requires delays

- Require token holders to hold token for a long period of time
- Build a delay between proposal genesis, voting, and execution
A DeFi project running on Ethereum. Governed token YAM.

**What happened?** (within a single day)

- Attacker bought 224739 YAM with borrowed funds
- Attacker submitted a governance proposal granting it control of project reserves ($3M USD)
- Voted on proposal with borrowed 224739 YAM, hitting quorum
- Borrowed YAM exchanged for Eth and paid back
What happened next?

Yam Finance team: noticed the proposal shortly after it hit quorum

⇒ retained **cancel** power, and canceled the proposal.

**Lesson:**

- Illustrates the positive power of a veto
- The problem: power can never be taken away
  (unless voluntarily given away)
Build Finance DAO is a DAO providing services to other DAOs.

- No entity had the ability to cancel successful votes

The Build Finance DAO has been the target of a hostile governance takeover in which a malicious actor has put forward and succeeded with a proposal to take control of the Build token contract.

The attacker succeeded in the takeover by having a large enough vote in favour of the proposal and there were not enough countervotes to prevent the takeover from happening.

⇒ attacker drained entire treasure ($1.7M). Killed the project.
A margin trading protocol on Solana. Governance token **MNGO**.

**What happened?** (a price manipulation attack)

- Attacker leveraged the low liquidity of MNGO to spike its price from $0.03 to $0.91 (by borrowing $5M from FTX).

⇒ Attacker can profit by buying an early MNGO perpetual.

- The exploit left the protocol with $115M of bad debt.
- Attacker was left with 32M MNGO.
- Protocol was left with $70M USDC in its treasure ⇒ $45M shortfall.
What happened next?

Attacker submitted a governance proposal:

```
Repay bad debt

hi all, the mango treasury has about 70M USDC available to repay bad debt.
I propose the following. If this proposal passes, I will send the MSOL, SOL, and MNGO in this
account to an address announced by the mango team. The mango treasury will be used to cover
any remaining bad debt in the protocol, and all users without bad debt will be made whole. Any
bad debt will be viewed as a bug bounty / insurance, paid out of the mango insurance fund. By
voting for this proposal, mango token holders agree to pay this bounty and pay off the bad debt
with the treasury, and waive any potential claims against accounts with bad debt, and will not
pursue any criminal investigations or freezing of funds once the tokens are sent back as
described above.
```

Basically: attacker would send back $45M to cover the shortfall, and keep the remaining $70M
Attacker voted for their own proposal

... but they didn’t have enough MNGO

The lesson:
• a DAO should not vote on attack resolution strategy
  ... attacker can sway the vote
How did this end?

Avraham Eisenberg
@avi_eisen · Follow

Replying to @avi_eisen

I believe all of our actions were legal open market actions, using the protocol as designed, even if the development team did not fully anticipate all the consequences of setting parameters the way they are.

9:48 AM · Oct 15, 2022

Mango Labs Sues Trader to Recover Allegedly Stolen Tokens

- Eisenberg accused of raking in $114 million in 20 minutes
- Trader arrested in Puerto Rico on criminal charges over scheme

By Robert Burnson
January 25, 2023 at 6:21 PM PST
TORN governance token used to manage governance

Attacker’s proposal claims to use logic as an earlier legit proposal:

One innocuous-looking function added:
Once proposal passed, attacker caused the SelfDestruct

⇒ lets attacker replace proposal code with arbitrary code
  (a contract created with CREATE2 can be replaced after selfdestruct)

⇒ gave themselves 1.2M TORN (enough to pass any proposal)

Next: withdrew 380K TORN from gov. contract and sold for ETH
The lesson:

• can’t trust proposal description ...

• can’t trust proposal code ...

both need to be carefully vetted before every vote!
... and now for something completely different

More on ZK proofs
What is a zk-SNARK? (intuition)

SNARK: a succinct proof that a certain statement is true

Example statement: “I know an $m$ such that $\text{SHA256}(m) = 0$”

- SNARK: the proof is “short” and “fast” to verify
  
  [if $m$ is 1GB then the trivial proof (the message $m$) is neither]

- zk-SNARK: the proof “reveals nothing” about $m$ (privacy for $m$)
Commercial interest in SNARKs

Many more building applications that use SNARKs
Why so much commercial interest?

Babai-Fortnow-Levin-Szegedy 1991:

In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with unreliable software.

“Checking Computations in Polylogarithmic Time”
Babai-Fortnow-Levin-Szegedy 1991:

In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with unreliable software.

“Checking Computations in Polylogarithmic Time”
Outsourcing computation: (no need for zero knowledge)

L1 chain quickly verifies the work of an off-chain service

Examples:

- **Scalability**: proof-based Rollups (zkRollup)
  
  off-chain service processes a batch of Tx;
  
  L1 chain verifies a succinct proof that Tx were processed correctly

- **Bridging blockchains**: proof of consensus (zkBridge)
  
  enables transfer of assets from one chain to another
Some applications require zero knowledge (privacy):

- **Private Tx on a public blockchain:**
  - zk proof that a private Tx is valid (Tornado cash, Zcash, IronFish, Aleo)
  - Governance with privacy: voting, delegation, treasury

- **Compliance:**
  - Proof that a private Tx is compliant with banking laws (Espresso)
  - Proof that an exchange is solvent in zero-knowledge (Raposa)
Many non-blockchain applications

Blockchains drive the development of SNARKs

... but many non-blockchain applications
Why are all these applications possible now?

**The breakthrough:** new fast SNARK provers

- Proof generation time is linear (or quasilinear) in computation size
- Many beautiful ideas ... will cover in lectures

a large bibliography:  a16zcrypto.com/zero-knowledge-canon
What is a SNARK?
NARK: Non-interactive ARgument of Knowledge

Public program: \( C(x, w) \rightarrow \mathbb{F} \)

Preprocessing (setup): \( S(C) \rightarrow \) public parameters \((pp, vp)\)
A preprocessing NARK is a triple \((S, \ P, \ V)\):

- \(S(C) \rightarrow\) public parameters \((pp, vp)\) for prover and verifier
- \(P(pp, x, w) \rightarrow\) proof \(\pi\)
- \(V(vp, x, \pi) \rightarrow\) accept or reject
NARK: requirements (informal)

Prover $P(pp, x, w)$

Verifier $V(vp, x, \pi)$

proof $\pi$ → accept or reject

Complete: $\forall x, w: C(x, w) = 0 \Rightarrow \Pr[V(vp, x, P(pp, x, w)) = \text{accept}] = 1$

Adaptively knowledge sound: $V$ accepts $\Rightarrow$ $P$ “knows” $w$ s.t. $C(x, w) = 0$

(an extractor $E$ can extract a valid $w$ from $P$)

Optional: Zero knowledge: $(C, pp, vp, x, \pi)$ “reveal nothing new” about $w$
SNARK: a **Succinct ARgument of Knowledge**

A **strongly succinct preprocessing NARK** is a triple $(S, P, V)$:

- $S(C) \rightarrow$ public parameters $(pp, vp)$ for prover and verifier
- $P(pp, x, w) \rightarrow$ **short** proof $\pi$ ; \[ \text{len}(\pi) = O_{\lambda}(\log(|C|)) \]
- $V(vp, x, \pi)$ **fast to verify** ; \[ \text{time}(V) = O_{\lambda}(|x|, \, \log(|C|)) \]

$V$ has no time to read $C$ !!
SNARK: a **Succinct** ARgument of Knowledge

SNARK: a NARC (complete and knowledge sound) that is **succinct**

zk-SNARK: a SNARK that is also **zero knowledge**
The trivial SNARK is not a SNARK

(a) Prover sends $w$ to verifier,
(b) Verifier checks if $C(x, w) = 0$ and accepts if so.

**Problems with this:**

1. $w$ might be long: we want a “short” proof
2. computing $C(x, w)$ may be hard: we want a “fast” verifier
3. $w$ might be secret: prover might not want to reveal $w$ to verifier
How to define “knowledge soundness”? 
Definitions: knowledge soundness

**Goal:** if $V$ accepts then $P$ “knows” $w$ s.t. $C(x, w) = 0$

What does it mean to ”know” $w$ ??

**informal def:** $P$ knows $w$, if $w$ can be “extracted” from $P$
Zero knowledge

Where is Waldo?
Zero knowledge

A story about the lady sipping tea
(S, P, V) is honest-verifier zero knowledge if for every $x \in \mathbb{F}^n$
proof $\pi$ “reveals nothing” about $w$, other than its existence

What does it mean to “reveal nothing” ??

**Informal def:** $\pi$ “reveals nothing” about $w$ if the verifier can generate $\pi$ by itself $\implies$ it learned nothing new from $\pi$

(S, P, V) is zero knowledge if there is an efficient alg. $Sim$ s.t. $(pp, vp, \pi) \leftarrow Sim(C, x)$ “look like” the real $pp, vp$ and $\pi$.

Main point: $Sim(C, x)$ simulates $\pi$ without knowledge of $w$
Our old example

Prover wants to prove that it knows the solution to a Sudoku puzzle.

ZK: a simulator Sim can generate the proof transcript given only the statement.

Prover Peggy

Verifier Victor

can be made non-interactive

yes/no
Our old example

Prover wants to prove that it knows the solution to a Sudoku puzzle

ZK: a simulator Sim can generate the proof transcript given only the statement

Prover Peggy  →  proof \( \pi \)  →  Verifier Victor

can be made non-interactive
An example ZK protocol

Proof of shuffle revisited

public statement:

754  935  484  153

484  153  754  935

Goal:
prove that bottom row is a shuffle of the top row
The rules

Prover can do: \( \text{ReRand}(r) \)

Verifier: given \( 754 \), \( 754 \), and \( r \)

can confirm that right box is ReRand of left box.
The (interactive) protocol

**Statement:**
random shuffle of bottom row

**Witness:**
\( r_1, r_2, r_3, r_4 \)

**Bot \( \rightarrow \) Rand:**
\( r_1 \), \( r_2 \), \( r_3 \), \( r_4 \)

**Top \( \rightarrow \) Rand:**

**Random Bit \( b \) in \( \{0,1\} \):**

\( b = 0 \): \( (s_3, s_2, s_4, s_1) \)
\( b = 1 \): \( (r_3 s_4, r_4 s_1, r_1 s_3, r_2 s_2) \)

**Verify**
ReRand

**Yes/No**
Why is this sound?

Suppose bottom is not a shuffle of top

**Claim**: prover can answer correctly when $b = 0$ or when $b = 1$, but cannot answer both correctly !!

⇒ verifier catches prover with prob. $\frac{1}{2}$.

⇒ Repeat 128 times to gain overwhelming confidence
Why is this knowledge sound?

Let $P^*$ be a malicious prover

- Extractor: get $P^*$ to answer both $b = 0$ and $b = 1$

\[
\begin{align*}
\text{bot } &\rightarrow \text{ rand} & b = 0: (s_3, s_2, s_4, s_1) \\
\text{top } &\rightarrow \text{ rand} & b = 1: (r_3s_4, r_4s_1, r_1s_3, r_2s_2)
\end{align*}
\]

Now, dividing bottom by top extracts the witness $(r_1, r_2, r_3, r_4)$
Why is this ZK?

statement: 

random bit \( b \) in \( \{0,1\} \)

some \((t_1, t_2, t_3, t_4)\) compatible with top \(\rightarrow\) rand or bot \(\rightarrow\) rand
Why is this ZK?

Simulator: need to generate transcript

• step 1: choose some random \((t_1, t_2, t_3, t_4)\)
• step 2: choose a random bit \(b\) in \(\{0, 1\}\)
• step 3: compute a compatible rand:
  shuffle and ReRand top or bottom using \((t_1, t_2, t_3, t_4)\)

\[\Rightarrow\] output transcript
END OF LECTURE
The (interactive) protocol

**Prover**

witness: $r_1, r_2, r_3, r_4$

**Verifier**

$r_1$ | $r_2$ | $r_3$ | $r_4$
--- | --- | --- | ---
754 | 935 | 484 | 153

$s_1$ | $s_2$ | $s_3$ | $s_4$
--- | --- | --- | ---
935 | 153 | 484 | 754