Tutorial 3: The DAO Hack

Code vs Intent

ECOM215: Blockchain Economics and Digital Assets

Week 4 | Smart Contracts and Decentralised Applications

Semester B, 2025/2026

CASE BRIEF FOR STUDENTS

Please read before the tutorial. Estimated reading time: 15 minutes.

Background

In 2016, a project called "The DAO" (Decentralised Autonomous Organisation) became the largest crowdfunding campaign in history, raising over $150 million worth of Ether (ETH) in just 28 days. The DAO was designed to be a new kind of investment fund—one with no managers, no board of directors, and no physical location. Instead, all decisions would be made through smart contract code and token holder voting.

The DAO's premise was simple: investors would contribute ETH and receive DAO tokens in return. These tokens gave voting rights on which projects the fund should invest in. Profits from successful investments would be distributed to token holders automatically. No intermediaries, no fees, no human discretion—just code executing as written.

It was the embodiment of the blockchain ethos: "Code is law."

The Exploit

On 17 June 2016, an unknown attacker began draining funds from The DAO. By the time the attack was noticed, approximately 3.6 million ETH—worth around $60 million at the time—had been siphoned into a "child DAO" controlled by the attacker.

How did it happen?

The attacker exploited a vulnerability called a "reentrancy bug" in The DAO's smart contract code. Here's the simplified logic:

1. The DAO's "withdraw" function worked in two steps:
    - First, send the requested ETH to the user
    - Second, update the user's balance to reflect the withdrawal

2. The attacker created a malicious contract that, when receiving ETH, would immediately call the withdraw function again—before step 2 (updating the balance) could execute.
3. Because the balance hadn't been updated yet, The DAO thought the attacker still had funds to withdraw. The cycle repeated, draining ETH with each loop.

The bug was publicly known before the attack. Security researchers had warned The DAO's creators, but no fix had been implemented.

Crucially: The attacker did not "hack" The DAO in the traditional sense. They did not break encryption, guess passwords, or access systems illegally. They simply used the smart contract exactly as its code allowed. Every transaction was valid according to the rules written into the contract.

The Ethereum Community's Response

The attack created an existential crisis for Ethereum. The options were:

Option 1: Do nothing

- The code executed as written. The attacker followed the rules.
- Intervening would undermine the principle that "code is law."
- But: $60 million stolen, The DAO's investors wiped out, confidence in Ethereum damaged.

Option 2: Soft fork

- Freeze the attacker's funds by blacklisting their address.
- Less invasive than reversing transactions.
- But: Still sets precedent that the community can censor transactions.

Option 3: Hard fork

- Rewrite Ethereum's history to return the stolen funds to DAO investors.
- Investors made whole, attacker gets nothing.
- But: Destroys the principle of immutability. If history can be rewritten once, why not again?

After intense debate, a community vote, and significant controversy, the Ethereum Foundation implemented **Option 3**—a hard fork that returned the stolen ETH to investors.

The Aftermath

Ethereum Classic: Not everyone agreed with the hard fork. A minority of the community continued running the original, unforked Ethereum chain. This became "Ethereum Classic" (ETC)—a chain where the attacker kept the funds and "code is law" was upheld. ETC still exists today, though it's much smaller than Ethereum.

Legal questions: Was the attacker a thief or simply a clever user? They exploited a bug, but every transaction they made was technically valid. No charges were ever filed (the attacker's identity was never confirmed, though some later investigations claimed to identify them).

Industry impact: The DAO hack became a cautionary tale about smart contract security. It led to:

- Greater emphasis on code audits before deployment
- Development of formal verification methods
- The principle that smart contracts, while "immutable," exist within a social and economic context that cannot be fully automated

Key Facts Summary

Item	Detail
Date	June 2016
Amount raised	∼$150 million (ETH)
Amount stolen	∼$60 million (3.6 million ETH)
Vulnerability	Reentrancy bug
Resolution	Ethereum hard fork (funds returned)
Consequence	Ethereum/Ethereum Classic split

Questions to Consider

1. Was the attacker a "thief" or simply someone who used the code as written?
2. Was the hard fork the right decision? What precedent did it set?
3. If "code is law," what happens when the code has bugs?
4. Who should bear responsibility for smart contract vulnerabilities—developers, auditors, or users who invest without understanding the code?
5. Could something like The DAO hack happen today? What has changed?

Further Reading (Optional)

- The original DAO white paper (search "DAO white paper 2016")

- Ethereum Foundation's blog post on the hard fork decision
- "The DAO Hack" chapter in *The Infinite Machine* by Camila Russo

Session Timeline

Time	Activity
0:00–0:08	Context setting: summarise case, check understanding
0:08–0:20	Discussion Question 1
0:20–0:32	Discussion Question 2
0:32–0:44	Discussion Question 3
0:44–0:52	Discussion Question 4
0:52–1:00	Synthesis and key takeaways

Discussion Questions with Guidance

Question 1: Was the attacker a thief?

"The attacker used the smart contract exactly as its code allowed. Every transaction was valid. Was this theft, or just clever use of a system?"

Let students debate both sides. Points that may emerge:

"Yes, it was theft":

- Intent matters. The attacker clearly intended to take funds that weren't rightfully theirs.
- Legal systems distinguish between what you *can* do and what you're *allowed* to do.
- Just because a door is unlocked doesn't mean you can enter and take things.
- The social contract among DAO investors was to fund projects, not to have funds drained.

"No, it wasn't theft":

- The attacker followed the contract's rules perfectly.
- Investors agreed to be bound by the code when they invested.
- In traditional finance, exploiting pricing errors or inefficiencies is often legal (arbitrage).
- If we call this theft, where do we draw the line? All MEV extraction? All DeFi exploits?

Key insight to draw out: This case exposes the tension between *code as written* and *code as intended*. Smart contracts can execute the former but cannot understand the latter.

Question 2: Was the hard fork the right decision?

"The Ethereum community voted to rewrite history and return the funds. Was this the correct choice? What are the implications?"

Arguments for the hard fork:

- Pragmatic: Saved $60M, protected investor confidence, kept Ethereum viable

- Ethereum was young; letting a catastrophic failure stand could have killed the project
- The "immutability" principle is valuable, but it's not absolute—human systems always have override mechanisms
- Democratic: The community voted and the majority supported the fork

Arguments against the hard fork:

- Sets dangerous precedent: If history can be rewritten once, why not again?
- Who decides which hacks deserve reversal? FTX? Mt. Gox? Where does it end?
- Undermines the core value proposition of blockchain (immutability, censorship resistance)
- Creates moral hazard: Investors may take more risks if they expect bailouts

Key insight to draw out: There is no "neutral" position. Doing nothing is also a choice with consequences. The fork revealed that blockchains are social systems, not just technical ones.

Question 3: Who bears responsibility for smart contract bugs?

"The vulnerability was publicly known before the attack. Security researchers had warned The DAO's developers. Who should bear responsibility—developers, auditors, investors, or the attacker?"

Developers:

- Wrote flawed code and didn't fix known vulnerabilities
- Rushed to launch without adequate testing
- But: Were volunteers/early-stage; industry standards didn't exist yet

Auditors (if any):

- Security reviews existed but missed the vulnerability (or warnings were ignored)
- But: Audits are not guarantees; they're point-in-time assessments

Investors:

- Invested $150M in experimental, unaudited code
- Most did not read or understand the smart contract
- But: Can we expect retail investors to audit code? Is that realistic?

The attacker:

- Ultimately, they chose to exploit rather than report
- But: If code is law, they did nothing "wrong" by the system's own rules

Key insight to draw out: Responsibility is distributed, but the current system puts most risk on end users who are least equipped to evaluate smart contract security. This has implications for regulation and investor protection.

Question 4: Could this happen today? What has changed?

"It's now 2026. Has the industry learned from The DAO? Could a similar attack happen again?"

What has improved:

- Audit industry matured (Trail of Bits, OpenZeppelin, Certora, etc.)
- Formal verification tools exist
- Bug bounty programs incentivise responsible disclosure
- More conservative coding patterns (checks-effects-interactions)
- Time-locks and governance delays common for large protocols

What hasn't changed:

- DeFi exploits still happen regularly (∼$3B+ stolen in 2022 alone)
- Speed-to-market pressure leads to shortcuts
- Composability creates unexpected interactions
- Many users still invest without understanding risks
- "Code is law" philosophy persists in some communities

Key insight to draw out: Technical improvements are real, but the fundamental tension between innovation speed and security remains. Flash loan attacks, oracle manipulation, and governance exploits are modern equivalents. The scale of losses has actually increased, not decreased.

Extension Question (if time permits)

"An attacker drains $100M from a DeFi protocol today. Should the protocol's governance execute an emergency intervention to reverse it? How is this different from what Ethereum did?"

This connects to modern DeFi governance debates. Most protocols now have multisigs or emergency mechanisms that *could* intervene—so the question is not "can they?" but "should they?" and "under what circumstances?"

End of Tutorial 3 Materials