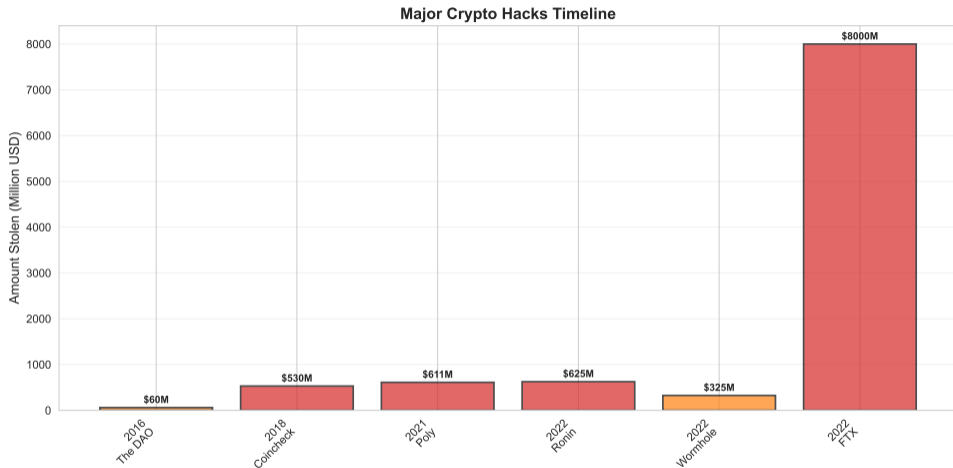


Lesson 23: Security and Hacks

Module 2: Blockchain Fundamentals

Digital Finance

The Stakes: Billions Lost to Hacks



Major Incidents:

- **2016:** The DAO (\$60M) – reentrancy
- **2021:** Poly Network (\$611M) – access control (returned)
- **2022:** Ronin Bridge (\$625M) – compromised keys
- **2022:** Wormhole Bridge (\$325M) – signature verification

`charts/lesson_23/threat_categories.pdf`

Reentrancy: The DAO Hack (2016)

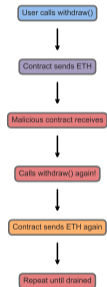
Vulnerability:

- Contract sends ETH before updating balance
- Recipient contract calls back
- Recursive withdrawals drain funds

Impact:

- 3.6M ETH stolen (\$60M)
- Led to Ethereum hard fork
- ETH/ETC split

Reentrancy Attack Flow



Prevention: Checks-Effects-Interactions pattern, reentrancy guards (OpenZeppelin)

Reentrancy Code Example

`charts/lesson_23/reentrancy_code.pdf`

Integer Overflow/Underflow

Problem (Pre-Solidity 0.8):

- uint256 max: $2^{256} - 1$
- Overflow: $\text{max} + 1$ wraps to 0
- Underflow: $0 - 1$ wraps to max
- Silent failures (no revert)

Example Attack:

- Balance: 100 tokens
- Transfer 200 (should fail)
- `balance - 200` underflows to huge number
- Exploit unlimited tokens

`charts/lesson_23/overflow_visualization.pdf`

Common Mistakes:

- Missing `onlyOwner` modifiers on critical functions
- Default function visibility (public vs internal)
- Incorrect permission checks

Example: Parity Wallet Hack (2017)

- Multi-sig library had unprotected `initWallet()` function
- Anyone could call and become owner
- Attacker initialized, then called `kill()` (selfdestruct)
- \$300M frozen permanently

Best Practice:

- Explicitly declare visibility (public, external, internal, private)
- Use `OpenZeppelin Ownable` and `AccessControl`
- Audit all privileged functions

`charts/lesson_23/flash_loan_attack.pdf`

Oracle Security: Defense Mechanisms

Vulnerable:

- Single DEX as price source
- Spot price (current block)
- Low liquidity pools

Secure:

- Time-Weighted Average Price (TWAP)
- Multiple oracle sources (Chainlink)
- High liquidity requirements
- Price deviation limits

charts/lesson_23/oracle_comparison.pdf

Bridge Hacks: Cross-Chain Vulnerabilities

Why Bridges are Risky:

- Centralized honeypots (large TVL)
- Complex multi-sig schemes
- Cross-chain validation challenges
- Centralized validators

Attack Types:

- Compromised validator keys
- Signature verification bugs
- Replay attacks

charts/lesson_23/bridge_architecture.pdf

Context:

- Ronin: Ethereum sidechain for Axie Infinity game
- Bridge: Transfer assets between Ethereum and Ronin
- Requires 5-of-9 validator signatures for withdrawals

Attack:

- ① Attacker compromised 4 Sky Mavis (developer) validator keys (spear phishing)
- ② Exploited backdoor in Axie DAO validator (5th signature)
- ③ Forged withdrawal transaction: 173,600 ETH + 25.5M USDC
- ④ Took 6 days to detect (low monitoring)

Root Cause: Centralization (5 of 9 validators controlled by single entity), poor key management

Wormhole Bridge Hack (2022): \$325M

Mechanism:

- Wormhole: Bridge between Ethereum and Solana
- Guardians validate cross-chain messages

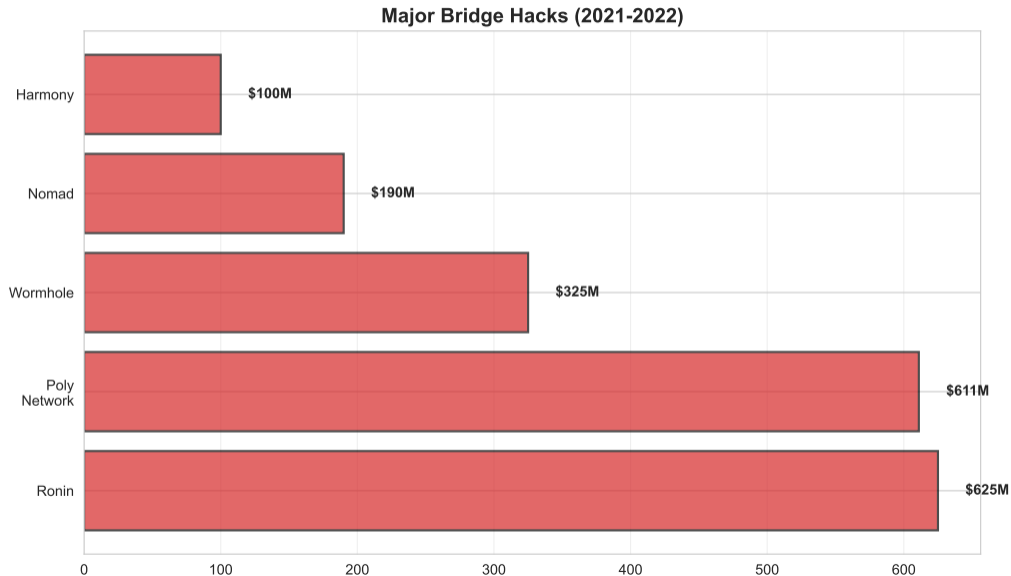
Vulnerability:

- Signature verification function had bug
- Attacker bypassed guardian signature check
- Minted 120,000 wrapped ETH on Solana without depositing on Ethereum

Outcome:

- Attacker bridged fake wETH to Ethereum, sold for real ETH
- Jump Crypto (parent company) refunded users (\$325M bailout)

Lesson: Even well-funded projects have critical bugs, audits not foolproof



Phishing and Social Engineering

Common Attacks:

- Fake wallet extensions (clipboard hijack)
- Malicious dApp frontends
- Discord/Telegram impersonation
- Fake support ("verify your wallet")
- Address poisoning

Example:

- User clicks malicious link
- Connects wallet to fake dApp
- Signs `setApprovalForAll` transaction
- Attacker drains NFTs/tokens

`charts/lesson_23/phishing_flow.pdf`

Address Poisoning Attack

`charts/lesson_23/address_poisoning.pdf`

Rug Pulls: Exit Scams

Types:

- **Liquidity Removal:** Dev drains LP, token worthless
- **Backdoor Functions:** Hidden mint/burn/pause
- **Honeypot:** Can buy but not sell

Example: Squid Game Token (2021)

- Market cap: \$3M in days
- Could buy, not sell (code restriction)
- Developers cashed out, disappeared

charts/lesson_23/rug_pull_pattern.pdf

Scenario:

- Protocol governed by token holders (1 token = 1 vote)
- Attacker acquires majority via flash loan or market buy
- Proposes malicious governance action (drain treasury)
- Votes pass, execute attack, repay flash loan

Example: Beanstalk (2022, \$182M)

- Attacker took flash loan of governance tokens
- Voted to approve malicious proposal
- Drained \$182M from protocol
- Repaid flash loan in same transaction

Defense: Time-locks on proposals, vote escrow (lock tokens for voting), quorum requirements

Audit Process:

- Security firm reviews smart contract code
- Identifies vulnerabilities, best practice violations
- Provides report with severity levels (critical, high, medium, low)
- Cost: \$50K–\$500K+ depending on complexity

Limitations:

- Audits are snapshots (code changes after audit)
- Cannot find all bugs (Wormhole was audited)
- Economic attacks (flash loans, governance) hard to model
- Audit shopping (only publish favorable audits)

Best Practice: Multiple audits (Trail of Bits, OpenZeppelin, Consensys Diligence), public bug bounties

Bug Bounties: Whitehat Incentives

Programs:

- Pay security researchers for finding bugs
- Tiered rewards (critical: **\$1M+**)
- Platforms: Immunefi, HackerOne

Examples:

- Aurora: **\$6M** bounty (critical)
- Wormhole: **\$10M** (post-hack)
- MakerDAO: **\$10M** max

charts/lesson_23/bug_bounty_tiers.pdf

For Developers:

- Use audited libraries (OpenZeppelin)
- Multiple audits before mainnet launch
- Time-locks on upgrades and governance
- Comprehensive testing (unit, integration, fuzzing)
- Bug bounty programs
- Incident response plan

For Users:

- Hardware wallets for large amounts
- Verify contract addresses (bookmarks, not Google)
- Revoke token approvals regularly (revoke.cash)
- Never share seed phrases
- Be skeptical of high yields (if too good to be true...)

- **Reentrancy:** External calls before state updates, use checks-effects-interactions
- **Overflow/Underflow:** Solidity 0.8+ auto-checks, or SafeMath
- **Oracle Manipulation:** Use TWAP, Chainlink, avoid single DEX price
- **Bridge Hacks:** Centralized validators, key compromise (\$2B+ stolen)
- **Phishing:** Social engineering, verify addresses, revoke approvals
- **Best Practices:** Audits, bug bounties, time-locks, user vigilance

Next Lesson: Regulation and future – MiCA, SEC, real-world assets, careers