Decentralized Finance

Practical Smart Contract Security

Instructors: **samczsun**, Dan Boneh, Arthur Gervais, Andrew Miller, Christine Parlour, Dawn Song

Paradigm

Stanford University

Imperial College London

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Berkeley UNIVERSITY OF CALIFORNIA

Introduction

- “How do I learn about smart contract security?”
  - Get hacked
  - Read blog posts
  - Try security challenges
  - Talk to an expert
- We’ll be focusing on the latter

Scheduled Programming

- When Safe Code Isn’t
- Uncovering a Four-Year Old Bug
- The 20 Million Dollar CTF
- How To Optimize Responsibly
- Cross-Chain Complications
- Escaping the Dark Forest

Next Up

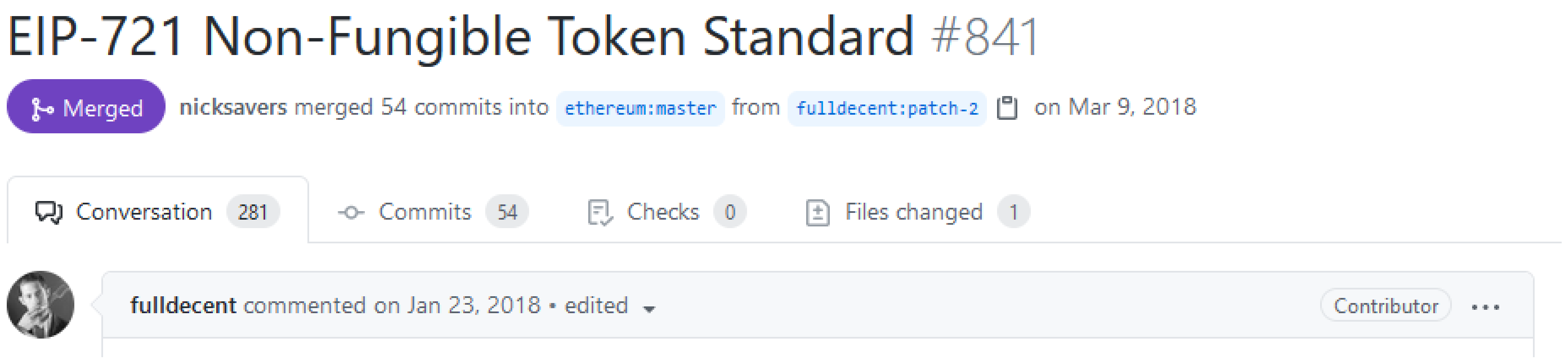
When Safe Code Isn’t

When Safe Code Isn’t

Safety is in the Eye of the Beholder

- What makes something safe?
- Who decides something is safe?
  - For a user: safety means protection against user error
  - For a programmer: safety means protection against malicious input
- What happens when two people don’t agree on the definition?

Safety is in the Eye of the Beholder



EIP-721 Non-Fungible Token Standard #841

Merged nicksavers merged 54 commits into `ethereum:master` from `fulldecent:patch-2` on Mar 9, 2018

Conversation 281 · Commits 54 · Checks 0 · Files changed 1

fulldecent commented on Jan 23, 2018 • edited

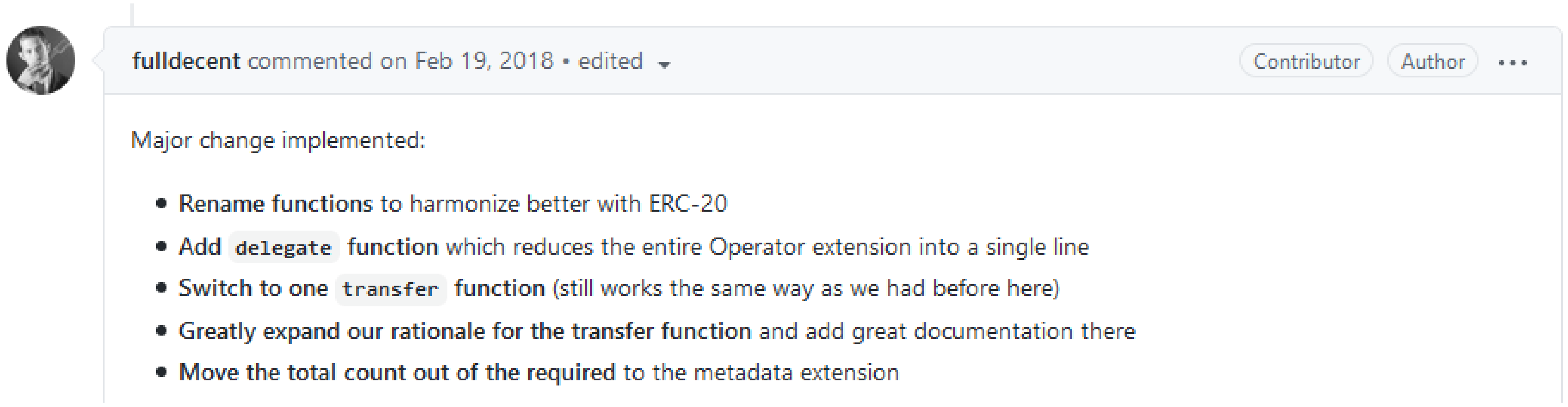
Contributor

Safety is in the Eye of the Beholder

```
/// @notice Approve a new owner to take your deed, or revoke approval by
///  setting the zero address. You may `approve` any number of times while
///  the deed is assigned to you, only the most recent approval matters.
/// @dev Throws if `msg.sender` does not own deed `_deedId` or if `_to` ==
///  `msg.sender`.
/// @param _deedId The deed you are granting ownership of
function approve(address _to, uint256 _deedId) external payable;

/// @notice Become owner of a deed for which you are currently approved
/// @dev Throws if `msg.sender` is not approved to become the owner of
///  `deedId` or if `msg.sender` currently owns `_deedId`.
/// @param _deedId The deed that is being transferred
function takeOwnership(uint256 _deedId) external payable;
```

Safety is in the Eye of the Beholder



fulldecent commented on Feb 19, 2018 • edited

Major change implemented:

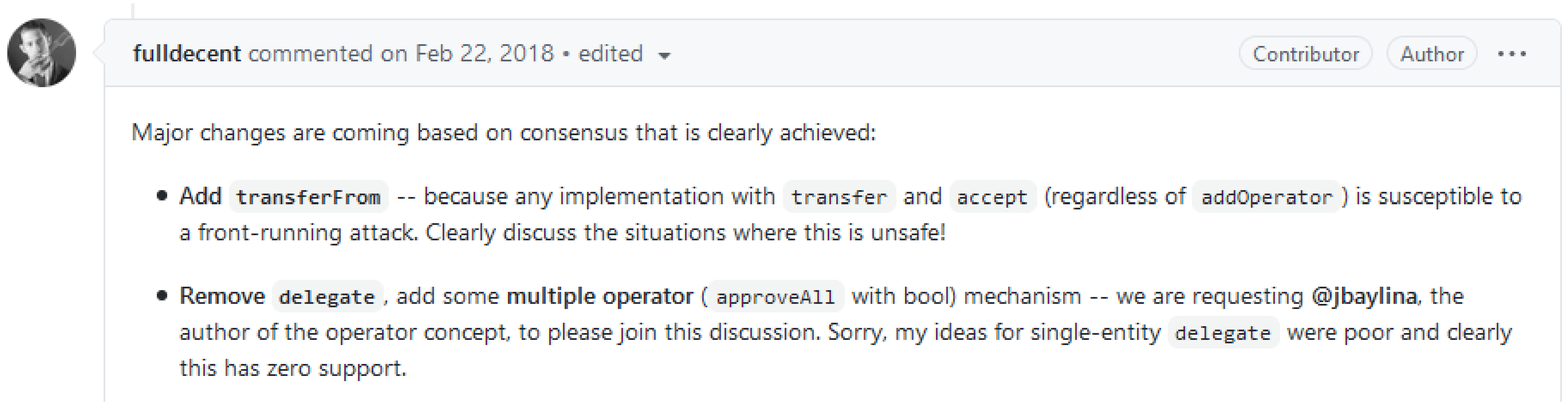
- **Rename functions** to harmonize better with ERC-20
- **Add** `delegate` **function** which reduces the entire Operator extension into a single line
- **Switch to one** `transfer` **function** (still works the same way as we had before here)
- **Greatly expand our rationale for the transfer function** and add great documentation there
- **Move the total count out of the required** to the metadata extension

Safety is in the Eye of the Beholder

```
/// @notice Set a new owner for a deed
/// @dev Throws unless `msg.sender` is the current deed owner, the "delegat
e
///  operator" of the current deed owner, or the "approved deed controller"
.
///  Throws if `_to` currently owns the deed. Throws if `_to` is the zero
///  address.
/// @param _to The new owner for the deed
/// @param _deedId The deed to transfer
function transfer(address _to, uint256 _deedId) external payable;

/// @notice Set or reaffirm the "approved deed controller" for a deed
/// @dev The zero address indicates there is no approved deed controller.
/// @dev Throws unless `msg.sender` is the current deed owner, or the
///  "delegate operator" of the current deed owner.
/// @param _approved The new approved deed controller
/// @param _deedId The deed to approve
function approve(address _approved, uint256 _deedId) external payable;
```

Safety is in the Eye of the Beholder



fulldecent commented on Feb 22, 2018 • edited

Major changes are coming based on consensus that is clearly achieved:

- Add `transferFrom` -- because any implementation with `transfer` and `accept` (regardless of `addOperator`) is susceptible to a front-running attack. Clearly discuss the situations where this is unsafe!
- Remove `delegate`, add some **multiple operator** (`approveAll` with bool) mechanism -- we are requesting **@jbaylina**, the author of the operator concept, to please join this discussion. Sorry, my ideas for single-entity `delegate` were poor and clearly this has zero support.

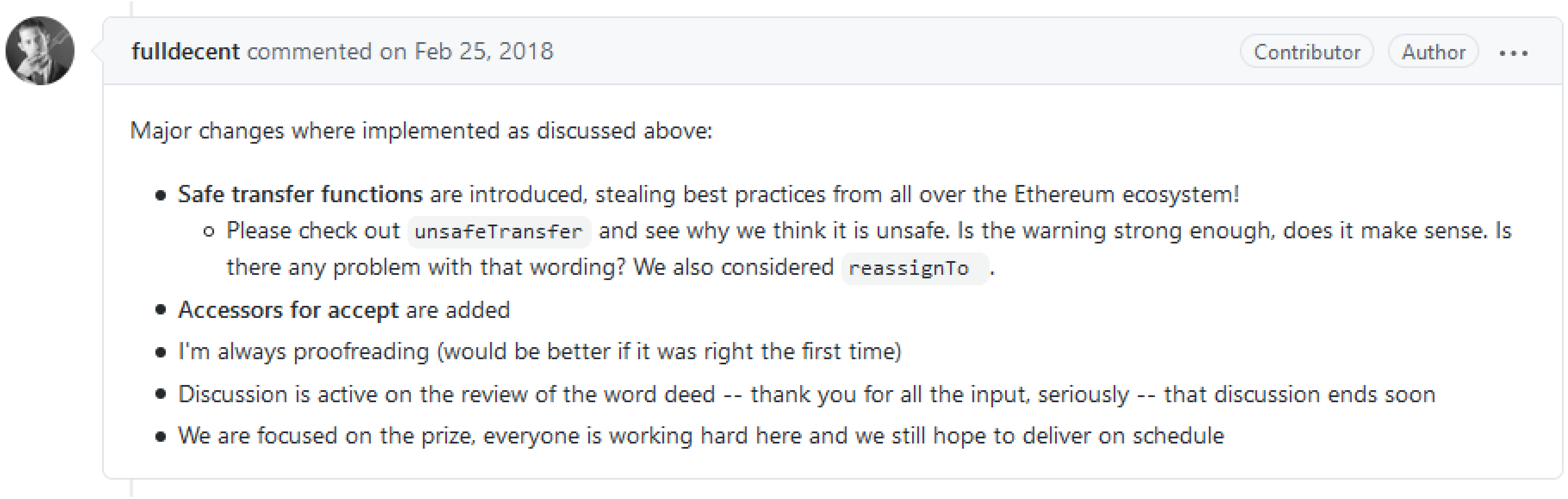
Safety is in the Eye of the Beholder

```
/// @notice Transfers the ownership of a deed -- warning the caller is
///  responsible to confirm that the sender is capable of receiving deeds
///  otherwise the deed may become inaccessible!
/// @dev Throws unless `msg.sender` is the current deed owner, the "delegate
///  operator" of the current deed owner, or the "approved deed controller".
///  Throws if `_to` currently owns the deed. Throws if `_to` is the zero
///  address.
/// @param _to The new owner for the deed
/// @param _deedId The deed to transfer
function transfer(address _to, uint256 _deedId) external payable;

/// @notice Transfers the ownership of a given deed from one address to
///  another address
/// @dev Throws unless `msg.sender` is the current deed owner, the "delegate
///  operator" of the current deed owner, or the "approved deed controller".
///  Throws if `_to` currently owns the deed. Throws if `_to` is the zero
///  address. Throws if the deed is not currently owned by _from.
/// @param _from The current owner for the deed
/// @param _to The new owner for the deed
/// @param _deedId The deed to transfer
function transferFrom(address _from, address _to, uint256 _deedId) external payable;

/// @notice Set or reaffirm the "approved deed controller" for a deed
/// @dev The zero address indicates there is no approved deed controller.
/// @dev Throws unless `msg.sender` is the current deed owner, or the
///  "delegate operator" of the current deed owner.
/// @param _approved The new approved deed controller
/// @param _deedId The deed to approve
function approve(address _approved, uint256 _deedId) external payable;
```

Safety is in the Eye of the Beholder



fulldecent commented on Feb 25, 2018

Contributor Author ...

Major changes where implemented as discussed above:

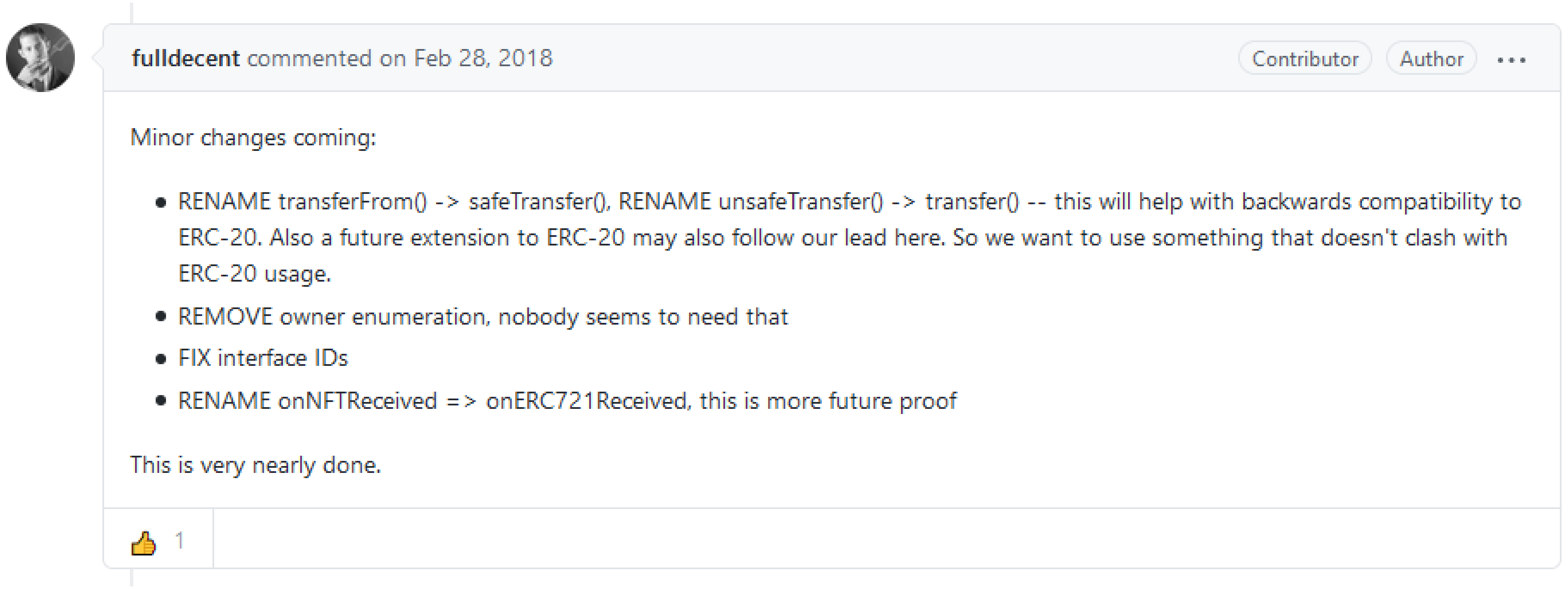
- **Safe transfer functions** are introduced, stealing best practices from all over the Ethereum ecosystem!
  - Please check out `unsafeTransfer` and see why we think it is unsafe. Is the warning strong enough, does it make sense. Is there any problem with that wording? We also considered `reassignTo` .
- **Accessors for accept** are added
- I'm always proofreading (would be better if it was right the first time)
- Discussion is active on the review of the word deed -- thank you for all the input, seriously -- that discussion ends soon
- We are focused on the prize, everyone is working hard here and we still hope to deliver on schedule

Safety is in the Eye of the Beholder

```
/// @notice Transfer ownership of a deed -- THE CALLER IS RESPONSIBLE
///  TO CONFIRM THAT `_to` IS CAPABLE OF RECEIVING DEEDS OR ELSE
///  THEY MAY BE PERMANENTLY LOST
/// @dev Throws unless `msg.sender` is the current deed owner, an authorized
///  operator, or the approved address for this deed. Throws if `_from` is
///  not the current owner of the deed. Throws if `_to` is the zero address.
///  Throws if `_deedId` is not a valid deed.
/// @param _from The new owner for the deed
/// @param _to The new owner for the deed
/// @param _deedId The deed to transfer
function unsafeTransfer(address _from, address _to, uint256 _deedId) external payable;

/// @notice Transfers the ownership of a given deed from one address to
///  another address
/// @dev Throws unless `msg.sender` is the current deed owner, an authorized
///  operator, or the approved address for this deed. Throws if `_from` is
///  not the current owner of the deed. Throws if `_to` is the zero address.
///  Throws if `_deedId` is not a valid deed. When transfer is complete,
///  this function also calls `onNFTReceived` on `_to` and throws if the return
///  value is not `keccak256("ERC721_ONNFTRECEIVED")`.
/// @param _from The current owner for the deed
/// @param _to The new owner for the deed
/// @param _deedId The deed to transfer
/// @param data Additional data with no specified format, sent in call to `_to`
function transferFrom(address _from, address _to, uint256 _deedId, bytes[] data) external payable;
```

Safety is in the Eye of the Beholder



fulldecent commented on Feb 28, 2018

Minor changes coming:

- RENAME transferFrom() -> safeTransfer(), RENAME unsafeTransfer() -> transfer() -- this will help with backwards compatibility to ERC-20. Also a future extension to ERC-20 may also follow our lead here. So we want to use something that doesn't clash with ERC-20 usage.
- REMOVE owner enumeration, nobody seems to need that
- FIX interface IDs
- RENAME onNFTReceived => onERC721Received, this is more future proof

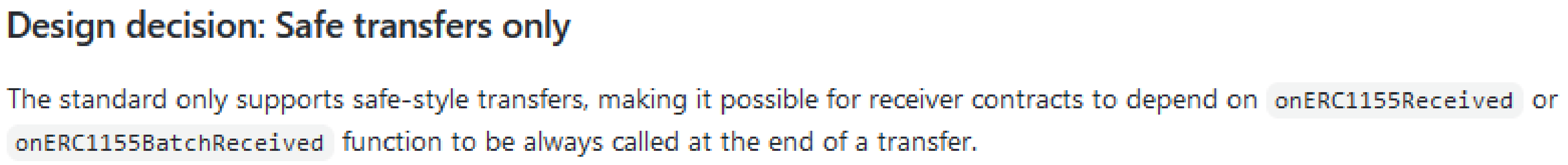
This is very nearly done.

Safety is in the Eye of the Beholder

```
/// @notice Transfers the ownership of an NFT from one address to another address
/// @dev Throws unless `msg.sender` is the current owner, an authorized
///  operator, or the approved address for this NFT. Throws if `_from` is
///  not the current owner. Throws if `_to` is the zero address. Throws if
///  `_tokenId` is not a valid NFT. When transfer is complete, this function
///  checks if `_to` is a smart contract (code size > 0). If so, it calls
///  `onERC721Received` on `_to` and throws if the return value is not
///  `bytes4(keccak256("onERC721Received(address,uint256,bytes)"))`.
/// @param _from The current owner of the NFT
/// @param _to The new owner
/// @param _tokenId The NFT to transfer
/// @param data Additional data with no specified format, sent in call to `_to`
function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external payable;

/// @notice Transfer ownership of an NFT -- THE CALLER IS RESPONSIBLE
///  TO CONFIRM THAT `_to` IS CAPABLE OF RECEIVING NFTS OR ELSE
///  THEY MAY BE PERMANENTLY LOST
/// @dev Throws unless `msg.sender` is the current owner, an authorized
///  operator, or the approved address for this NFT. Throws if `_from` is
///  not the current owner. Throws if `_to` is the zero address. Throws if
///  `_tokenId` is not a valid NFT.
/// @param _from The current owner of the NFT
/// @param _to The new owner
/// @param _tokenId The NFT to transfer
function transferFrom(address _from, address _to, uint256 _tokenId) external payable;
```

Safety is in the Eye of the Beholder



Design decision: Safe transfers only

The standard only supports safe-style transfers, making it possible for receiver contracts to depend on `onERC1155Received` or `onERC1155BatchReceived` function to be always called at the end of a transfer.

Safe Transfers

- User safety
  - Protects users from typos
  - Protects users from sending to the wrong address
- Programmer safety
  - Introduces new security risks!

Unsafe External Calls

- Threat model on the blockchain is different
  - Traditional programming: function calls are safe because it’s your own code or a library you trust
  - Smart contracts: function calls are unsafe because you might call an attacker who wants to steal your money

Unsafe External Calls

- During an external call, an attacker has full control
  - Interact with your contract again (reentrancy)
  - Interact with other contracts
- All external calls to non-trusted contracts may be unsafe!

Unsafe External Calls

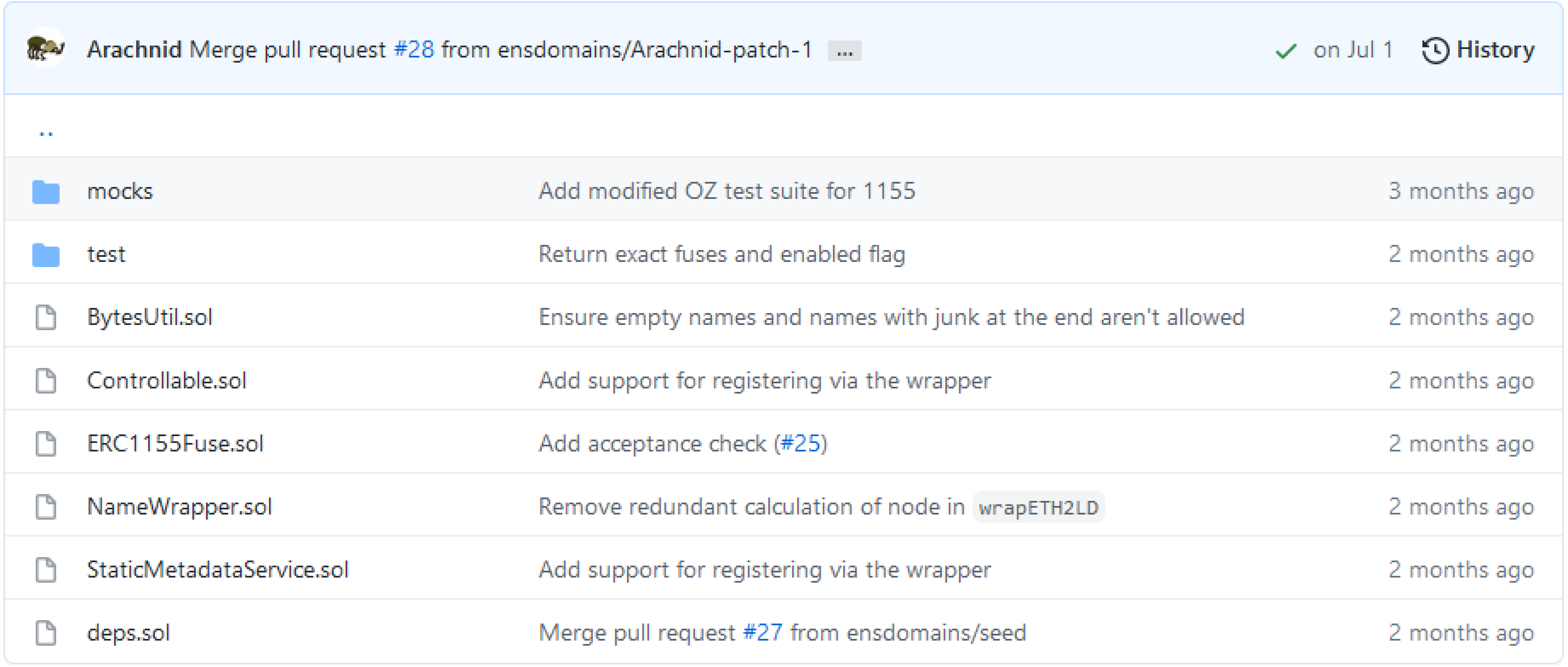
- How to determine if an external call is unsafe?
- Consider a hypothetical vulnerability
  - If it can be exploited without needing the external call, then the external call is redundant
  - Therefore, the external call must contribute something
- External call occurs during execution of function
  - What has the function already checked/changed?
  - What will the function check/change?

ENS Name Wrapper

- ERC-1155 token to wrap an ENS domain
- Allows ENS developers to expand abilities of domain owner
- Follow along
  - https://github.com/ensdomains/name-wrapper/tree/4726375

ENS Name Wrapper

- Identify the business logic



Arachnid Merge pull request #28 from ensdomains/Arachnid-patch-1 ... ✓ on Jul 1 History

..

mocks	Add modified OZ test suite for 1155	3 months ago
test	Return exact fuses and enabled flag	2 months ago
BytesUtil.sol	Ensure empty names and names with junk at the end aren't allowed	2 months ago
Controllable.sol	Add support for registering via the wrapper	2 months ago
ERC1155Fuse.sol	Add acceptance check (#25)	2 months ago
NameWrapper.sol	Remove redundant calculation of node in `wrapETH2LD`	2 months ago
StaticMetadataService.sol	Add support for registering via the wrapper	2 months ago
deps.sol	Merge pull request #27 from ensdomains/seed	2 months ago

ENS Name Wrapper

- Figure out the high-level user story
  - User wraps a domain, gets ERC1155 token
  - User unwraps a token, gets ENS domain back
  - User owns a token, can modify domain

ENS Name Wrapper

- Check for unsafe external calls

```
function wrap(
    bytes calldata name,
    address wrappedOwner,
    uint96 _fuses,
    address resolver
) public override {
    bytes32 node = _wrap(name, wrappedOwner, _fuses);
    address owner = ens.owner(node);

    require(
        owner == msg.sender ||
            isApprovedForAll(owner, msg.sender) ||
            ens.isApprovedForAll(owner, msg.sender),
        "NameWrapper: Domain is not owned by the sender"
    );
    ens.setOwner(node, address(this));
    if (resolver != address(0)) {
        ens.setResolver(node, resolver);
    }
}
```

ENS Name Wrapper

- Check for unsafe external calls

```
function _doSafeTransferAcceptanceCheck(
    address operator,
    address from,
    address to,
    uint256 id,
    uint256 amount,
    bytes memory data
) private {
    if (to.isContract()) {
        try
            IERC1155Receiver(to).onERC1155Received(
                operator,
                from,
                id,
                amount,
                data
            )
        returns (bytes4 response) {
            if (
                response != IERC1155Receiver(to).onERC1155Received.selector
            ) {
                revert("ERC1155: ERC1155Receiver rejected tokens");
            }
        } catch Error(string memory reason) {
            revert(reason);
        } catch {
            revert("ERC1155: transfer to non ERC1155Receiver implementer");
        }
    }
}
```

ENS Name Wrapper

- Is it exploitable?

```
function wrap(
    bytes calldata name,
    address wrappedOwner,
    uint96 _fuses,
    address resolver
) public override {
    bytes32 node = _wrap(name, wrappedOwner, _fuses);
    address owner = ens.owner(node);

    require(
        owner == msg.sender ||
            isApprovedForAll(owner, msg.sender) ||
            ens.isApprovedForAll(owner, msg.sender),
        "NameWrapper: Domain is not owned by the sender"
    );
    ens.setOwner(node, address(this));
    if (resolver != address(0)) {
        ens.setResolver(node, resolver);
    }
}
```

```
function _wrap(
    bytes memory name,
    address wrappedOwner,
    uint96 _fuses
) private returns (bytes32 node) {
    (bytes32 labelhash, uint256 offset) = name.readLabel(0);
    bytes32 parentNode = name.namehash(offset);

    require(
        parentNode != ETH_NODE,
        "NameWrapper: .eth domains need to use wrapETH2LD()"
    );

    node = _makeNode(parentNode, labelhash);

    _mint(node, name, wrappedOwner, _fuses);
    emit NameWrapped(node, name, wrappedOwner, _fuses);
}
```

ENS Name Wrapper

- What can we do with token ownership?
  - Look for functions with onlyTokenOwner modifier
  - Unwrap, burn fuses, set subnodes, set resolver/ttl, etc
- Unwrapping sounds pretty cool
  - Transfers underlying ENS domain to the owner of the token
  - Now we can do whatever we want with the ENS domain
  - After we’re done, return from the callback
  - We own the ENS domain, so permission check succeeds

Hashmasks

- Limited supply NFT
- Anyone could buy them during the sale, limit 20 per transaction
- Maximum of 16,384 NFTs to be minted
- Follow along
  - https://etherscan.io/address/0xc2c747e0f7004f9e8817db2ca4997657a7746928
  - Or just search for "Hashmasks" token

Hashmasks

```
function mintNFT(uint256 numberOfNfts) public payable {
    require(totalSupply() < MAX_NFT_SUPPLY, "Sale has already ended");
    require(numberOfNfts > 0, "numberOfNfts cannot be 0");
    require(numberOfNfts <= 20, "You may not buy more than 20 NFTs at once");
    require(totalSupply().add(numberOfNfts) <= MAX_NFT_SUPPLY, "Exceeds MAX_NFT_SUPPLY");
    require(getNFTPrice().mul(numberOfNfts) == msg.value, "Ether value sent is not correct");

    for (uint i = 0; i < numberOfNfts; i++) {
        uint mintIndex = totalSupply();
        if (block.timestamp < REVEAL_TIMESTAMP) {
            _mintedBeforeReveal[mintIndex] = true;
        }
        _safeMint(msg.sender, mintIndex);
    }

    /**
    * Source of randomness. Theoretical miner withhold manipulation possible but should be sufficient in a pragmatic sense
    */
    if (startingIndexBlock == 0 && (totalSupply() == MAX_NFT_SUPPLY || block.timestamp >= REVEAL_TIMESTAMP)) {
        startingIndexBlock = block.number;
    }
}
```

Hashmasks

```
function _safeMint(address to, uint256 tokenId, bytes memory _data) internal virtual {
    _mint(to, tokenId);
    require(_checkOnERC721Received(address(0), to, tokenId, _data), "ERC721: transfer to non ERC721Receiver implementer");
}

function _checkOnERC721Received(address from, address to, uint256 tokenId, bytes memory _data)
    private returns (bool)
{
    if (!to.isContract()) {
        return true;
    }
    bytes memory returndata = to.functionCall(abi.encodeWithSelector(
        IERC721Receiver(to).onERC721Received.selector,
        _msgSender(),
        from,
        tokenId,
        _data
    ), "ERC721: transfer to non ERC721Receiver implementer");
    bytes4 retval = abi.decode(returndata, (bytes4));
    return (retval == _ERC721_RECEIVED);
}
```

Hashmasks

```
function mintNFT(uint256 numberOfNfts) public payable {
    require(totalSupply() < MAX_NFT_SUPPLY, "Sale has already ended");
    require(numberOfNfts > 0, "numberOfNfts cannot be 0");
    require(numberOfNfts <= 20, "You may not buy more than 20 NFTs at once");
    require(totalSupply().add(numberOfNfts) <= MAX_NFT_SUPPLY, "Exceeds MAX_NFT_SUPPLY");
    require(getNFTPrice().mul(numberOfNfts) == msg.value, "Ether value sent is not correct");

    for (uint i = 0; i < numberOfNfts; i++) {
        uint mintIndex = totalSupply();
        if (block.timestamp < REVEAL_TIMESTAMP) {
            _mintedBeforeReveal[mintIndex] = true;
        }
        _safeMint(msg.sender, mintIndex);
    }

    /**
    * Source of randomness. Theoretical miner withhold manipulation possible but should be sufficient in a pragmatic sense
    */
    if (startingIndexBlock == 0 && (totalSupply() == MAX_NFT_SUPPLY || block.timestamp >= REVEAL_TIMESTAMP)) {
        startingIndexBlock = block.number;
    }
}
```

Unsafe External Calls

- Just because a function is called safe doesn’t mean it’s safe
- Don’t assume what a function does
  - If you’re not sure, check!
- Any external call may be unsafe
  - Consider positioning of the call and what you can do with it

Next Up

Uncovering a Four Year Old Bug

Uncovering a Four Year Old Bug

Standing the Test of Time

- The longer a contract goes unhacked, the more secure it must be
  - No low hanging fruit
  - Finding a bug requires understanding the logic like the back of your hand
- How do you ensure you achieve maximum coverage when reviewing a battle-tested contract?

Searching With A Fine Toothed Comb

- Reduce search space, but how?
- What exactly makes a vulnerability?
  - Code remains the same
  - User input changes
- Strategy #1: start looking where user input is processed

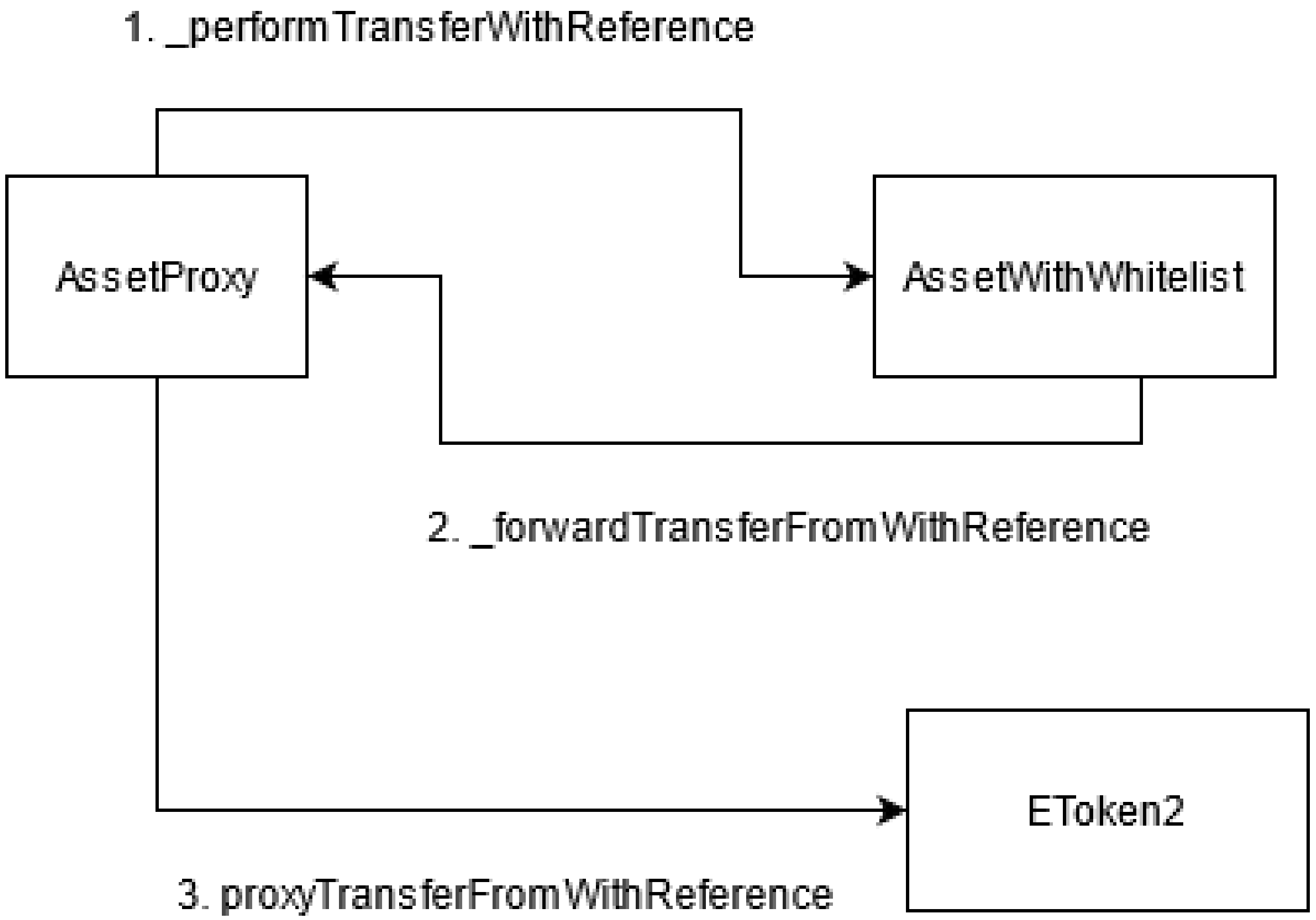
Searching With A Fine Toothed Comb

- What else makes a vulnerability?
  - Program does Bad Thing (send ether, selfdestruct, etc)
  - Bad Thing is triggered by special user input (otherwise it's not a vulnerability)
- Strategy #2: start looking where bad things can happen

Ambisafe

- ERC20 Platform-as-a-Service
- Every token proxies to the core contract
- A flaw in the core contracts would affect every token on the platform
- Want to try it yourself? Start here
  - https://etherscan.io/address/0x8400d94a5cb0fa0d041a3788e395285d61c9ee5e
  - UniBright token

Ambisafe


1. _performTransferWithReference
AssetProxy
AssetWithWhitelist
2. _forwardTransferFromWithReference
EToken2
3. proxyTransferFromWithReference

Potential Problem #1

```
/**
    * Transfers asset balance from the caller to specified receiver.
    *
    * @param _to holder address to give to.
    * @param _value amount to transfer.
    *
    * @return success.
    */
function transfer(address _to, uint _value) returns(bool) {
    return transferWithReference(_to, _value, '');
}

/**
    * Transfers asset balance from the caller to specified receiver adding specified comment.
    * Resolves asset implementation contract for the caller and forwards there arguments along with
    * the caller address.
    *
    * @param _to holder address to give to.
    * @param _value amount to transfer.
    * @param _reference transfer comment to be included in a EToken2's Transfer event.
    *
    * @return success.
    */
function transferWithReference(address _to, uint _value, string _reference) returns(bool) {
    return _getAsset()._performTransferWithReference(_to, _value, _reference, msg.sender);
}
```

Potential Problem #1

```
/**
 * Returns asset implementation contract for current caller.
 *
 * @return asset implementation contract.
 */
function _getAsset() internal returns(AssetInterface) {
    return AssetInterface(getVersionFor(msg.sender));
}
```

Potential Problem #1

```
/**
 * Returns asset implementation contract address assigned to sender.
 *
 * @param _sender sender address.
 *
 * @return asset implementation contract address.
 */
function getVersionFor(address _sender) constant returns(address) {
    return userOptOutVersion[_sender] == 0 ? latestVersion : userOptOutVersion[_sender];
}
```

Potential Problem #1

```
/**
 * Disagree with proposed upgrade, and stick with current asset implementation
 * until further explicit agreement to upgrade.
 *
 * @return success.
 */
function optOut() returns(bool) {
    if (userOptOutVersion[msg.sender] != 0x0) {
        return false;
    }
    userOptOutVersion[msg.sender] = latestVersion;
    return true;
}
```

Potential Problem #2

```
/**
 * Passes execution into virtual function.
 *
 * Can only be called by assigned asset proxy.
 *
 * @return success.
 * @dev function is final, and must not be overridden.
 */
function _performTransferWithReference(address _to, uint _value, string _reference, address _sender) onlyProxy() returns(bool) {
    if (isICAP(_to)) {
        return _transferToICAPWithReference(bytes32(_to) << 96, _value, _reference, _sender);
    }
    return _transferWithReference(_to, _value, _reference, _sender);
}

/**
 * Calls back without modifications.
 *
 * @return success.
 * @dev function is virtual, and meant to be overridden.
 */
function _transferWithReference(address _to, uint _value, string _reference, address _sender) internal returns(bool) {
    return proxy._forwardTransferFromWithReference(_sender, _to, _value, _reference, _sender);
}
```

Potential Problem #2

```
/**
 * Only assigned proxy is allowed to call.
 */
modifier onlyProxy() {
    if (proxy == msg.sender) {
        _;
    }
}
```

Potential Problem #3

```
/**
 * Performs transfer call on the EToken2 by the name of specified sender.
 *
 * Can only be called by asset implementation contract assigned to sender.
 *
 * @param _from holder address to take from.
 * @param _to holder address to give to.
 * @param _value amount to transfer.
 * @param _reference transfer comment to be included in a EToken2's Transfer event.
 * @param _sender initial caller.
 *
 * @return success.
 */
function _forwardTransferFromWithReference(address _from, address _to, uint _value, string _reference, address _sender)
    onlyImplementationFor(_sender) returns(bool) {
    return etoken2.proxyTransferFromWithReference(_from, _to, _value, etoken2Symbol, _reference, _sender);
}
```

Potential Problem #3

```
/**
 * Only asset implementation contract assigned to sender is allowed to call.
 */
modifier onlyImplementationFor(address _sender) {
    if (getVersionFor(_sender) == msg.sender) {
        _;
    }
}
```

Potential Problem #4

```
/**
 * Prforms allowance transfer of asset balance between holders wallets.
 *
 * Can only be called by asset proxy.
 *
 * @param _from holder address to take from.
 * @param _to holder address to give to.
 * @param _value amount to transfer.
 * @param _symbol asset symbol.
 * @param _reference transfer comment to be included in a Transfer event.
 * @param _sender allowance transfer initiator address.
 *
 * @return success.
 */
function proxyTransferFromWithReference(address _from, address _to, uint _value, bytes32 _symbol, string _reference, address _sender)
    onlyProxy(_symbol) returns(bool) {
    return _transfer(getHolderId(_from), _createHolderId(_to), _value, _symbol, _reference, getHolderId(_sender));
}
```

Potential Problem #4

```
/**
 * Emits Error if called not by asset proxy.
 */
modifier onlyProxy(bytes32 _symbol) {
    if (_isProxy(_symbol)) {
        _;
    } else {
        _error('Only proxy: access denied');
    }
}

function _isProxy(bytes32 _symbol) constant internal returns(bool) {
    return proxies[_symbol] == msg.sender;
}
```

Digging Deeper

```
function _transfer(uint _fromId, uint _toId, uint _value, bytes32 _symbol, string _reference, uint _senderId) internal
    checkSigned(_senderId, 1) returns(bool) {
    // Should not allow to send to oneself.
    if (_fromId == _toId) {
        _error('Cannot send to oneself');
        return false;
    }
    // Should have positive value.
    if (_value == 0) {
        _error('Cannot send 0 value');
        return false;
    }
    // Should have enough balance.
    if (_balanceOf(_fromId, _symbol) < _value) {
        _error('Insufficient balance');
        return false;
    }
    // Should allow references.
    if (bytes(_reference).length > 0 && !isEnabled(sha3(_symbol, Features.TransferWithReference))) {
        _error('References feature is disabled');
        return false;
    }
    // [snip]
```

Digging Deeper

```
    // [snip]

    // Should have enough allowance.
    if (_fromId != _senderId && _allowance(_fromId, _senderId, _symbol) < _value) {
        _error('Not enough allowance');
        return false;
    }
    // Adjust allowance.
    if (_fromId != _senderId) {
        assets[_symbol].wallets[_fromId].allowance[_senderId] -= _value;
    }
    _transferDirect(_fromId, _toId, _value, _symbol);
    // Internal Out Of Gas/Throw: revert this transaction too;
    // Recursive Call: safe, all changes already made.
    eventsHistory.emitTransfer(_address(_fromId), _address(_toId), _symbol, _value, _reference);
    _proxyTransferEvent(_fromId, _toId, _value, _symbol);
    return true;
}
```

Digging Deeper

```
/**
 * Ask asset Proxy contract to emit ERC20 compliant Transfer event.
 *
 * @param _fromId holder id to take from.
 * @param _toId holder id to give to.
 * @param _value amount to transfer.
 * @param _symbol asset symbol.
 */
function _proxyTransferEvent(uint _fromId, uint _toId, uint _value, bytes32 _symbol) internal {
    if (proxies[_symbol] != 0x0) {
        // Internal Out Of Gas/Throw: revert this transaction too;
        // Recursive Call: safe, all changes already made.
        Proxy(proxies[_symbol]).emitTransfer(_address(_fromId), _address(_toId), _value);
    }
}
```

Digging Deeper

- If we can replace proxies[_symbol], we can emit a fake transfer event
- However, recall the onlyProxy modifier from earlier
  - We can’t just replace the proxy before the transfer
- We need an unsafe external call after proxyTransferFromWithReference is called

Digging Deeper

- No obvious unsafe external calls in _transfer
- However, there’s a checkSigned modifier

Digging Deeper

```
modifier checkSigned(uint _holderId, uint _required) {
    if (!isCosignerSet(_holderId) || _checkSigned(holders[_holderId].cosigner, _holderId, _required)) {
        _;
    } else {
        _error('Cosigner: access denied');
    }
}

function _checkSigned(Cosigner _cosigner, uint _holderId, uint _required) internal returns(bool) {
    return _cosigner.consumeOperation(sha3(msg.data, _holderId), _required);
}

function setCosignerAddress(Cosigner _cosigner) checkSigned(_createHolderId(msg.sender), 1) returns(bool) {
    if (!_checkSigned(_cosigner, getHolderId(msg.sender), 1)) {
        _error('Invalid cosigner');
        return false;
    }
    holders[_createHolderId(msg.sender)].cosigner = _cosigner;
    return true;
}
```

Digging Deeper

- There’s an unsafe external call to a user specified cosigner
- During this call, we can swap out the proxy to the victim token proxy
- Now we can emit arbitrary transfer events
- However, Ambisafe is a permissioned platform, so low severity exploit

Digging Deeper

- Time to look at other pieces of business logic
- Notice recovery logic

Digging Deeper

```
function grantAccess(address _from, address _to) returns(bool) {
    if (!isCosignerSet(getHolderId(_from))) {
        _error('Cosigner not set');
        return false;
    }
    return _grantAccess(getHolderId(_from), _to);
}

function _grantAccess(uint _fromId, address _to) internal checkSigned(_fromId, 2) returns(bool) {
    // Should recover to previously unused address.
    if (getHolderId(_to) != 0) {
        _error('Should recover to new address');
        return false;
    }
    // We take current holder address because it might not equal _from.
    // It is possible to recover from any old holder address, but event should have the current one.
    address from = holders[_fromId].addr;
    holders[_fromId].addr = _to;
    holderIndex[_to] = _fromId;
    // Internal Out Of Gas/Throw: revert this transaction too;
    // Recursive Call: safe, all changes already made.
    eventsHistory.emitRecovery(from, _to, msg.sender);
    return true;
}
```

Digging Deeper

- Seems useless, unless we can somehow know who’s about to receive tokens for the first time
- As it turns out, we can

Life of a Transaction

- Click send
- Wait a bit
- Mined

Life of a Transaction

- Click send
- Signed locally
- Sent to connected Ethereum node
- Propagated via P2P network (mempool)
- Selected by miner
- Mined
- (Optional) Reorged and reinserted into the mempool

Digging Deeper

- Scan mempool for transactions which result in someone receiving tokens for the first time
- Frontrun by granting them access to our account ID
  - Effectively backdoors their address
- When enough addresses have been backdoored, steal tokens

Uncovering a Four Year Old Bug

- Sometimes, a bug will go unfound for years
- Need to understand the implications of every line of code
- Analyze the contract methodically for maximum coverage

Next Up

The 20 Million Dollar CTF

The 20 Million Dollar CTF

Real World Security

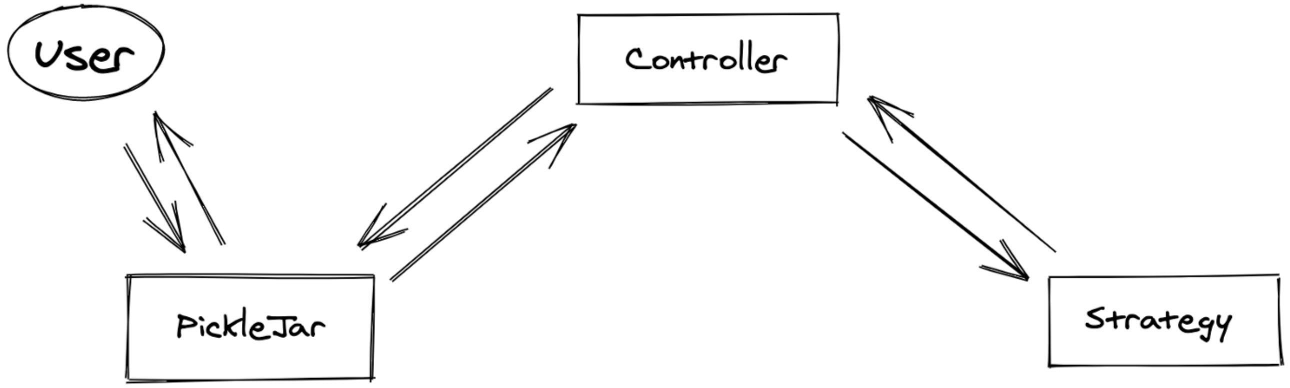
- Most bugs aren’t complicated
  - Missing only owner modifier
  - Private function is public
  - Integer overflow
- Sometimes, bugs can get very complicated
  - Known as exploit chaining
  - Extremely satisfying to pull off

Pickle Finance

- DeFi protocol for yield farming
- Users deposit stablecoins to Pickle Jars and get pTokens

Pickle Finance

- Strategists use deposited tokens to generate returns
- Returns are deposited into the Pickle Jar, increasing the value of pTokens
- Follow along
  - https://github.com/pickle-finance/protocol/tree/4d7ecfa



Strategies

- Controller sends tokens to strategy
- Strategy sends tokens to other protocols
- Other protocols sometimes airdrop tokens back
  - COMP, UNI, etc
- There needs to be a way to recover airdropped tokens
  - Solution: primary asset (named ‘want’) is locked, controller can retrieve all other tokens

Strategies

```
// Controller only function for creating additional rewards from dust
function withdraw(IERC20 _asset) external returns (uint256 balance) {
    require(msg.sender == controller, "!controller");
    require(want != address(_asset), "want");
    balance = _asset.balanceOf(address(this));
    _asset.safeTransfer(controller, balance);
}
```

Bug #1

- What happens if a protocol tokenizes deposits?
  - Compound: Deposit DAI, receive cDAI
  - Aave: Deposit DAI, receive aDAI
- Not protected from controller withdrawal!
- Severity: low, because we assume controller is not malicious

Controller

- Supports setting governance parameters
  - Change fees
  - Add new strategies
- Also allows users to swap between two Pickle Jars

Controller

```
// Function to swap between jars
function swapExactJarForJar(
    address _fromJar, // From which Jar
    address _toJar, // To which Jar
    uint256 _fromJarAmount, // How much jar tokens to swap
    uint256 _toJarMinAmount, // How much jar tokens you'd like at a minimum
    address payable[] calldata _targets,
    bytes[] calldata _data
) external returns (uint256) {
    require(_targets.length == _data.length, "!length");

    // Only return last response
    for (uint256 i = 0; i < _targets.length; i++) {
        require(_targets[i] != address(0), "!converter");
        require(approvedJarConverters[_targets[i]], "!converter");
    }
```

Bug #2

- If it walks like a duck and quacks like a duck, it might be an evil contract about to ruin your day
- Controller didn’t verify that jars were legitimate
- Malicious user could specify their own jar
- Severity: low, because the only thing they could do is force a strategy to deleverage

Proxy Logic

- Controller executed proxy logic using delegatecall

```
function _execute(address _target, bytes memory _data)
    internal
    returns (bytes memory response)
{
    require(_target != address(0), "!target");

    // call contract in current context
    assembly {
        let succeeded := delegatecall(
            sub(gas(), 5000),
            _target,
            add(_data, 0x20),
            mload(_data),
            0,
            0
        )

        // snip
    }
}
```

Proxy Logic

- One logic was written for Curve to allow for:
  - Burning LP tokens for underlying tokens
  - Minting LP tokens using underlying tokens
- Curve’s interface changed slightly, so the proxy had generic support

Proxy Logic

```
function add_liquidity(
    address curve,
    bytes4 curveFunctionSig,
    uint256 curvePoolSize,
    uint256 curveUnderlyingIndex,
    address underlying
) public {
    uint256 underlyingAmount = IERC20(underlying).balanceOf(address(this));

    // curveFunctionSig should be the abi.encodedFormat of
    // add_liquidity(uint256[N_COINS],uint256)
    // The reason why its here is because different curve pools
    // have a different function signature

    uint256[] memory liquidity = new uint256[](curvePoolSize);
    liquidity[curveUnderlyingIndex] = underlyingAmount;

    bytes memory callData = abi.encodePacked(
        curveFunctionSig,
        liquidity,
        uint256(0)
    );

    IERC20(underlying).safeApprove(curve, 0);
    IERC20(underlying).safeApprove(curve, underlyingAmount);
    (bool success, ) = curve.call(callData);
    require(success, "!success");
}
```

Bug #3

- Curve proxy logic allows:
  - Calling any arbitrary function (curveFunctionSig)
  - With one arbitrary parameter, because of the way arrays are ABI-encoded (underlyingAmount)
  - To an arbitrary contract (curve)
- However, underlyingAmount depends on the balance of underlying token
- Severity: medium, since you can only specify one parameter

Exploit Chaining

- How do we turn three low/med severity vulnerabilities to one critical severity vulnerability?
- Consider our toolbox:
  - Bug #1: Controller can withdraw cTokens from strategies
  - Bug #2: Controller doesn’t verify jars
  - Bug #3: Curve proxy logic can call any function on any contract with one attacker-controlled parameter

Exploit Chaining

- Step 1: Create contracts pretending to be Pickle Jars (bug #2)
- Step 2: Execute swapExactJarForJar using Curve logic
- Step 3: Call withdrawAll on strategy to transfer all cDAI to controller (bug #3, bug #1)
- Step 4: Have controller deposit newly retrieved cDAI to fake jar
- For full exploit, see https://github.com/banteg/evil-jar/blob/master/reference/samczsun.sol

Input Validation

- Duck typing doesn’t work on the blockchain
- Everything is untrusted until validated against a chain of trust

```
function logMarketTransferred(IUniverse _universe, address _from, address _to) public returns (bool) {
    require(isKnownUniverse(_universe));
    IMarket _market = IMarket(msg.sender);
    require(_universe.isContainerForMarket(_market));
    emit MarketTransferred(address(_universe), address(_market), _from, _to);
    return true;
}
```

Greater Than The Sum Of The Parts

- Complexity breeds insecurity
- Multiple low severity bugs can come together in disastrous ways

Next Up

How To Optimize Responsibly

How To Optimize Responsibly

Optimizations

- The cost of unoptimized code stacks up
  - Every user calling the function will pay that price, for ever call
  - Users don’t like it when they spend $30 to transfer some tokens
- It makes sense to want to optimize
  - Heavily used contracts, which will be called extremely often
  - Extremely complex contracts, which use millions of gas per call

When All You Have Is A Hammer

- Very tempting to drop down to assembly when optimizing
  - No need for all that compiler-generated boilerplate!
- Important to keep security in mind when performing optimizations
  - Compilers will do things that seem odd but address specific edge cases

0x Exchange v2

- Popular orderbook-based DEX
- Makers sign orders and publish them off-chain
  - Makers approve exchange contract to spend tokens beforehand
- Takers broadcast signed order as well as their offer
  - Exchange contract validates signature then swaps assets
- Exchange contract must not allow fake orders, or taker can lie about what maker's terms are
  - https://github.com/0xProject/0x-monorepo/tree/965d60/packages

0x Exchange v2

- Supports 7 types of signatures
  - 2 are always invalid (0x00, 0x01)
  - 1 is “pre-signed” (0x06)
  - 2 are signed by users off-chain (0x02, 0x03)
  - 2 are approved by wallets on-chain (0x04, 0x05)
- Wallet signatures must make external call to smart wallet
  - To avoid reentrancy, 0x made use of the staticcall instruction

Signature Validation

```
function isValidWalletSignature(
    bytes32 hash,
    address walletAddress,
    bytes signature
)
    internal
    view
    returns (bool isValid)
{
    bytes memory calldata = abi.encodeWithSelector(
        IWallet(walletAddress).isValidSignature.selector,
        hash,
        signature
    );
    assembly {
        let cdStart := add(calldata, 32)
        let success := staticcall(
            gas,                  // forward all gas
            walletAddress,        // address of Wallet contract
            cdStart,              // pointer to start of input
            mload(calldata),      // length of input
            cdStart,              // write output over input
            32                    // output size is 32 bytes
        )
```

Signature Validation

```
            switch success
            case 0 {
                // Revert with `Error("WALLET_ERROR")`
                mstore(0, 0x08c379a000000000000000000000000000000000000000000000000000000000)
                mstore(32, 0x0000002000000000000000000000000000000000000000000000000000000000)
                mstore(64, 0x0000000c57414c4c45545f4552524f5200000000000000000000000000000000)
                mstore(96, 0)
                revert(0, 100)
            }
            case 1 {
                // Signature is valid if call did not revert and returned true
                isValid := mload(cdStart)
            }
        }
        return isValid;
    }
```

Memory in Ethereum

- EVM has no concept of pages or malloc
- When you try to read or write outside of your memory size, your memory is expanded
  - You pay a gas cost proportional to the amount of newly allocated memory
- One common optimization is to reuse memory

CALL Opcode Semantics

- Today, read return data with the RETURNDATACOPY opcode
  - Allows for copying dynamic amounts of returndata
- When the EVM was first designed, only static return sizes were allowed
- The CALL opcodes all require:
  - Target contract address
  - Call data memory address
  - Call data length
  - Return output memory address
  - Return data length

CALL Opcode Semantics

- What happens if contract returns more than specified length?
  - All memory up to length is overwritten with return data
  - Extra data is truncated
- What happens if contract returns *less* than specified length?
  - All data returned is written to memory
  - Extra memory is *left as-is*

CALL Opcode Semantics

- What happens if the contract reverts?
  - CALL opcode pushes 0 onto stack
- What happens if the contract doesn't revert?
  - CALL opcode pushes 1 onto stack
- What happens if the contract has no code in the first place?
  - CALL pretends like it has a single STOP opcode
  - Push 1 onto stack
  - No return data

Signature Validation

```
function isValidWalletSignature(
    bytes32 hash,
    address walletAddress,
    bytes signature
)
    internal
    view
    returns (bool isValid)
{
    bytes memory calldata = abi.encodeWithSelector(
        IWallet(walletAddress).isValidSignature.selector,
        hash,
        signature
    );
    assembly {
        let cdStart := add(calldata, 32)
        let success := staticcall(
            gas,                  // forward all gas
            walletAddress,        // address of Wallet contract
            cdStart,              // pointer to start of input
            mload(calldata),      // length of input
            cdStart,              // write output over input
            32                    // output size is 32 bytes
        )
```

Signature Validation

```
        switch success
        case 0 {
            // Revert with `Error("WALLET_ERROR")`
            mstore(0, 0x08c379a000000000000000000000000000000000000000000000000000000000)
            mstore(32, 0x0000002000000000000000000000000000000000000000000000000000000000)
            mstore(64, 0x0000000c57414c4c45545f4552524f5200000000000000000000000000000000)
            mstore(96, 0)
            revert(0, 100)
        }
        case 1 {
            // Signature is valid if call did not revert and returned true
            isValid := mload(cdStart)
        }
    }
    return isValid;
}
```

Signature Validation

- Set signature to 0x04 (wallet type), automatically valid for all EOAs
- Compiler (if it supported STATICCALL) would have prevented this
  - Check address has code
  - Check there’s enough return data

ENS Registry

- ENS (Ethereum Name Service), like DNS but on Ethereum
  - Supports .eth (and other TLDs)
- Registry stores owner, resolver, TTL, for each ENS entry
- Low-level component, heavily optimized for gas, written in LLL
  - https://github.com/ensdomains/ens/blob/master/contracts/ENS.lll

ENS Registry

```
;; Struct: Record
 (def 'resolver 0x00) ; address
 (def 'owner    0x20) ; address
 (def 'ttl      0x40) ; uint64
```

ENS Registry

```
;; -------------------------------------------------------------------------
;; @notice Checks that the caller is the node owner.
;; @param node Check owner of this node.
(def 'only-node-owner (node)
  (when (!= (caller) (get-owner node))
    (jump invalid-location)))

;; -------------------------------------------------------------------------
;; @notice Retrieves owner from node record.
;; @param node Get owner of this node.
(def 'get-owner (node)
  (sload (+ node owner)))
```

ENS Registry

```
;; ----------------------------------------------------------------------
     ;; @notice Transfers ownership of a node to a new address. May only be
     ;;         called by the current owner of the node.
     ;; @dev Signature: setOwner(bytes32,address)
     ;; @param node The node to transfer ownership of.
     ;; @param new-owner The address of the new owner.

     (def 'node (calldataload 0x04))
     (def 'new-owner (calldataload 0x24))

     (function set-node-owner
       (seq (only-node-owner node)

         ;; Transfer ownership by storing passed-in address.
         (set-owner node new-owner)

         ;; Emit an event about the transfer.
         ;; Transfer(bytes32 indexed node, address owner);
         (mstore call-result new-owner)
         (log2 call-result 32
             (sha3 0x00 (lit 0x00 "Transfer(bytes32,address)")) node)

         ;; Nothing to return.
         (stop)))
```

ENS Registry

```
namehash([]) = 0x0000000000000000000000000000000000000000000000000000000000000000
namehash([label, …]) = keccak256(namehash(…), keccak256(label))
```

Storage Layout

```
---------------------------
| +0x00 |    resolver     |
---------------------------
| +0x20 |     owner       |
---------------------------
| +0x40 |      ttl        |
---------------------------
```

Storage Layout

```
--------------------------
| +0x00 |    resolver    |
---------------------------------------------------
| +0x20 |     owner      | +0x00 |    resolver    |
---------------------------------------------------
| +0x40 |      ttl       | +0x20 |     owner      |
---------------------------------------------------
                         | +0x40 |      ttl       |
                         --------------------------
```

Storage Layout

- Setting the resolver for one node sets the owner for the one before
- Setting the ttl for one node sets the owner for the one after
- Breaks the invariant wherein changing the value of one entry should not change the value of another
- Allows users to set backdoors by claiming ownership of nodes going backwards, then activate backdoor by claiming ownership of nodes going forwards

Storage Layout

- Compiler would have addressed this
  - Mappings are implemented by computing the storage slot as keccak256(key . slot)
  - Changing the value of one value can’t affect the value of another (without breaking keccak256)

Optimizations

- Hand-rolling optimizations is fine, but know the EVM first
- Compilers do things for a reason

Next Up

Cross-Chain Complications

Cross-Chain Complications

More Chains, More Problems

- Securing individual chains is hard
- Securing interactions between chains is harder
  - Especially if the two chains can’t natively communicate with each other

Atomic Loans

- Decentralized loan platform built on BTC and ETH
- Lock BTC, get stablecoins
- Complex state machine due to lack of native cross-chain communication
    - Agent is provided to automate loan origination

Atomic Loans

- Step 1: Alice (borrower) and Bob (lender) agree to terms
- Step 2: Alice and Bob initialize loan on ETH and commit secrets
  - A1: Allows Bob to withdraw collateral if loan expires
  - B1: Allows Alice to reclaim collateral after repaying loan
  - A2/B2: For liquidations, unimportant to us
- Step 3: Bob locks loan on ETH by sending to smart contract
- Step 4: Alice locks collateral on BTC by sending to P2SH address
  - Script allows funds to be spent if
    - Alice signs the tx and provides B1
    - Bob signs the tx, provides A1, and liquidation period is over

Atomic Loans

- Step 5: Bob confirms BTC was locked and unlocks loan
- Step 6: Alice reveals A1 in order to withdraw loan
- Step 7: Alice repays loan
- Step 8: Bob reveals B1 in order to claim payment

Cross-Chain Bugs

- What are some goals?
  - Stealing locked loans on ETH as a third party
  - Stealing locked collateral on BTC as a third party
  - Taking a loan on ETH without locking BTC
  - Liquidating collateral on BTC without providing loan

Cross-Chain Bugs

- Focus on third option, take a loan without locking BTC
  - Never lock BTC in the first place
  - Lock but somehow obtain B1 secret

BTC Transactions

- BTC tracks balances using transaction outputs
- Each transaction spends some inputs and generates some outputs

BTC Transactions

```
const refundableBalance = await loan.collateralClient().chain.getBalance([collateralRefundableP2SHAddress])
const seizableBalance = await loan.collateralClient().chain.getBalance([collateralSeizableP2SHAddress])

const refundableUnspent = await loan.collateralClient().getMethod('getUnspentTransactions')([collateralRefundableP2SHAddress])
const seizableUnspent = await loan.collateralClient().getMethod('getUnspentTransactions')([collateralSeizableP2SHAddress])

const collateralRequirementsMet = (refundableBalance.toNumber() >= refundableCollateralAmount &&
    seizableBalance.toNumber() >= seizableCollateralAmount)
const refundableConfirmationRequirementsMet = refundableUnspent.length === 0 ? false : refundableUnspent[0].confirmations > 0
const seizableConfirmationRequirementsMet = seizableUnspent.length === 0 ? false : seizableUnspent[0].confirmations > 0

if (collateralRequirementsMet && refundableConfirmationRequirementsMet && seizableConfirmationRequirementsMet) {
  await agenda.now('approve-loan', { loanModelId: loan.id })

  res.json({ message: 'Approving Loan', status: 0 })
} else {
  res.json({ message: 'Collateral has not be locked', status: 2 })
}
```

Secret Extraction

- How can we get B1 secret from agent?
- Look through agent code for when B1 is published
  - When loan repayment is accepted
  - When loan is cancelled
- Don’t care about first, implies we just repaid the loan
- What about loan cancellation?

Secret Extraction

```
// Cancel loan if not withdrawn within 22 hours after approveExpiration
if ((currentTime > (parseInt(approveExpiration) + 79200)) && !withdrawn) {
    log('info', `Check Loan Statuses and Update Job | ${principal} Loan #${loanId} was not withdrawn within 22 hours | Cancelling loan`)
    await agenda.schedule(getInterval('ACTION_INTERVAL'), 'accept-or-cancel-loan', { loanModelId: loan.id })
}
```

Secret Extraction

```
/**
 * @notice Borrower withdraws loan
 * @param loan The Id of the Loan
 * @param secretA1 Secret A1 provided by the borrower
 */
function withdraw(bytes32 loan, bytes32 secretA1) external {
    require(!off(loan), "Loans.withdraw: Loan cannot be inactive");
    require(bools[loan].funded == true, "Loans.withdraw: Loan must be funded");
    require(bools[loan].approved == true, "Loans.withdraw: Loan must be approved");
    require(bools[loan].withdrawn == false, "Loans.withdraw: Loan principal has already been withdrawn");
    require(sha256(abi.encodePacked(secretA1)) == secretHashes[loan].secretHashA1, "Loans.withdraw: Secret does not match");
    bools[loan].withdrawn = true;
    require(token.transfer(loans[loan].borrower, principal(loan)), "Loans.withdraw: Failed to transfer tokens");

    secretHashes[loan].withdrawSecret = secretA1;
    if (address(col.onDemandSpv()) != address(0)) {col.requestSpv(loan);}

    emit Withdraw(loan, secretA1);
}
```

Cross-Chain Complications

- Every chain has its own nuances that aren’t immediately obvious
  - Bitcoin: 0-conf transactions
  - Ethereum: frontrunning
- Building a cross-chain state machine is hard
  - Ensure well-defined transitions

Next Up

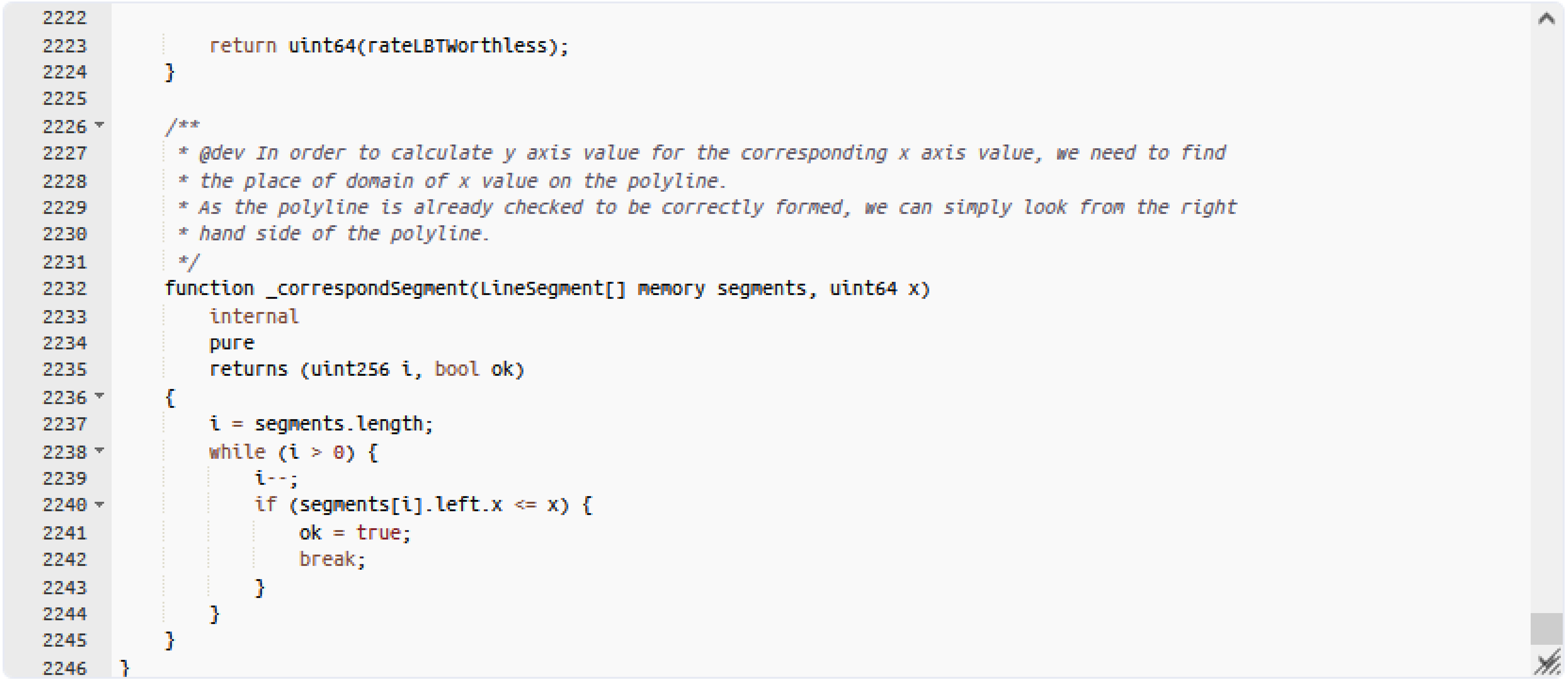
Escaping The Dark Forest

Escaping The Dark Forest

Escaping The Dark Forest

- Unofficial sequel to Dan Robinson’s and Georgios Konstantopoulos’s *Ethereum Is A Dark Forest*
- Follow along
  - https://etherscan.io/address/0x8b24f5c764ab741bc8a2426505bda458c30df010
  - Funds at the time: ~25,000 ETH, ~9,600,000 USD

Finding The Bug



```
        return uint64(rateLBTWorthless);
    }

    /**
     * @dev In order to calculate y axis value for the corresponding x axis value, we need to find
     * the place of domain of x value on the polyline.
     * As the polyline is already checked to be correctly formed, we can simply look from the right
     * hand side of the polyline.
     */
    function _correspondSegment(LineSegment[] memory segments, uint64 x)
        internal
        pure
        returns (uint256 i, bool ok)
    {
        i = segments.length;
        while (i > 0) {
            i--;
            if (segments[i].left.x <= x) {
                ok = true;
                break;
            }
        }
    }
}
```

Finding The Bug

```
function _transferETH(
    address payable recipient,
    uint256 amount,
    string memory errorMessage
) internal {
    require(_hasSufficientBalance(amount), errorMessage);
    (bool success, ) = recipient.call{value: amount}("");
    require(success, "transferring Ether failed");
    emit LogTransferETH(address(this), recipient, amount);
}
```

Finding The Bug

```
function issueNewBonds(uint256 bondGroupID)
    public
    override
    payable
    returns (uint256)
{
    BondGroup storage bondGroup = _bondGroupList[bondGroupID];
    bytes32[] storage bondIDs = bondGroup.bondIDs;
    require(
        _getBlockTimestampSec() < bondGroup.maturity,
        "the maturity has already expired"
    );

    uint256 fee = msg.value.mul(2).div(1002);

    // [snip]

    _transferETH(payable(LIEN_TOKEN_ADDRESS), fee);

    emit LogIssueNewBonds(bondGroupID, msg.sender, amount);

    return amount;
}
```

Finding The Bug

```
function _distributeETH2BondTokenContract(
    uint256 bondGroupID,
    uint256 oracleHintID
) internal {
    // [snip]

    for (uint256 i = 0; i < bondGroup.bondIDs.length; i++) {
        bytes32 bondID = bondGroup.bondIDs[i];
        BondToken bondTokenContract = _bonds[bondID].contractInstance;
        require(
            address(bondTokenContract) != address(0),
            "the bond is not registered"
        );

        // [snip]

        uint256 totalSupply = bondTokenContract.totalSupply();
        bool expiredFlag = bondTokenContract.expire(n, d);

        if (expiredFlag) {
            uint256 payment = totalSupply.mul(10**(18 - 8)).mul(n).div(d);
            _transferETH(
                address(),
                payment, bondTokenContract
                "system error: BondMaker's balance is less than payment"
            );
        }
    }
}
```

Finding The Bug

```solidity
/**
 * @notice redeems ETH from the total set of bonds in the bondGroupID before maturity date.
 */
function reverseBondToETH(uint256 bondGroupID, uint256 amountE8)
    public
    override
    returns (bool)
{
    BondGroup storage bondGroup = _bondGroupList[bondGroupID];
    bytes32[] storage bondIDs = bondGroup.bondIDs;
    require(
        _getBlockTimestampSec() < bondGroup.maturity,
        "the maturity has already expired"
    );
    bytes32 bondID;
    for (
        uint256 bondFnMapIndex = 0;
        bondFnMapIndex < bondIDs.length;
        bondFnMapIndex++
    ) {
        bondID = bondIDs[bondFnMapIndex];
        _burnBond(bondID, msg.sender, amountE8);
    }

    _transferETH(
        msg.sender,
        amountE8.mul(10**10),
        "system error: insufficient Ether balance"
    );

    emit LogReverseBondToETH(bondGroupID, msg.sender, amountE8.mul(10**10));

    return true;
}
```

Finding The Bug

```
/**
 * @notice Collect bondIDs that regenerate the original ETH, and group them as a bond group.
 * Any bond is described as a set of linear functions(i.e. polyline),
 * so we can easily check if the set of bondIDs are well-formed by looking at all the end
 * points of the lines.
 */
function registerNewBondGroup(bytes32[] memory bondIDs, uint256 maturity)
    public
    override
    returns (uint256 bondGroupID)
{
    _assertBondGroup(bondIDs, maturity);

    // Get and increment next bond group ID
    bondGroupID = nextBondGroupID;
    nextBondGroupID = nextBondGroupID.add(1);

    _bondGroupList[bondGroupID] = BondGroup(bondIDs, maturity);

    emit LogNewBondGroup(bondGroupID);

    return bondGroupID;
}
```

Finding The Bug

```
function _assertBondGroup(bytes32[] memory bondIDs, uint256 maturity)
    internal
    view
{
    uint256 numOfBreakPoints = 0;
    for (uint256 i = 0; i < bondIDs.length; i++) {
        BondInfo storage bond = _bonds[bondIDs[i]];
        require(
            bond.maturity == maturity,
            "the maturity of the bonds must be same"
        );
        LineSegment[] storage polyline = _registeredFnMap[bond.fnMapID];
        numOfBreakPoints = numOfBreakPoints.add(polyline.length);
    }

    uint256 nextBreakPointIndex = 0;
    uint64[] memory rateBreakPoints = new uint64[](numOfBreakPoints);
    for (uint256 i = 0; i < bondIDs.length; i++) {
        // [snip]
    }

    for (uint256 k = 0; k < rateBreakPoints.length; k++) {
        // [snip]
    }
}
```

Reporting The Bug

- Whose contract is it?
  - No comments about owner
- Google contract address
  - No results
- Google contract name
  - Find blog post: https://medium.com/lien-finance/lien-version-2-overview-8ecd0bdeb51e

Reporting The Bug

- Team is anonymous
  - Unsure if admin on Telegram is core dev or not
- Notice audit reports by ConsenSys and CertiK
  - Try contacting ConsenSys

Reporting The Bug

- Brief Alex Wade on the situation
- Discuss possible solutions
  - Publish announcement asking users to withdraw
  - Use the exploit to rescue funds
- Both solutions are bad

Ethereum Is A Dark Forest

- Advanced frontrunning bots monitor the mempool and look for opportunities
  - Handles generic transactions
- Trying to exploit the bug would likely result in a bot frontrunning us

Reporting The Bug

- Contact Scott Bigelow, collaborated on frontrunners in the past
- Then contact Tina, who had been reaching out to miners
- Lien still hadn’t responded, so contact CertiK too
  - Introduced to Georgios Delkos

Identity Verification

- How do you verify someone is who they say they are?
  - Just ask them? Must trust their word
- If verifying professional identity, send a code to work email
  - Possible to spoof email sender, but harder to intercept
  - Proves that they own the domain and inbox
  - Optionally, ask for a reply over email too
- If verifying contract ownership, get a signature from deployer
  - Make sure to sign a message including all relevant information
  - Proves they deployed the contract (even if they're not the current owner)

Reporting The Bug

- Finally, got in touch with anonymous developer
  - Identity verification! Alex and Georgios validate the anonymous developer has access to the email used during the audit
- Proposed solutions to Lien
  - Urge people to withdraw
  - Try the exploit ourselves
  - Contact a mining pool and do a private transaction
- Lien agreed to go with option #3

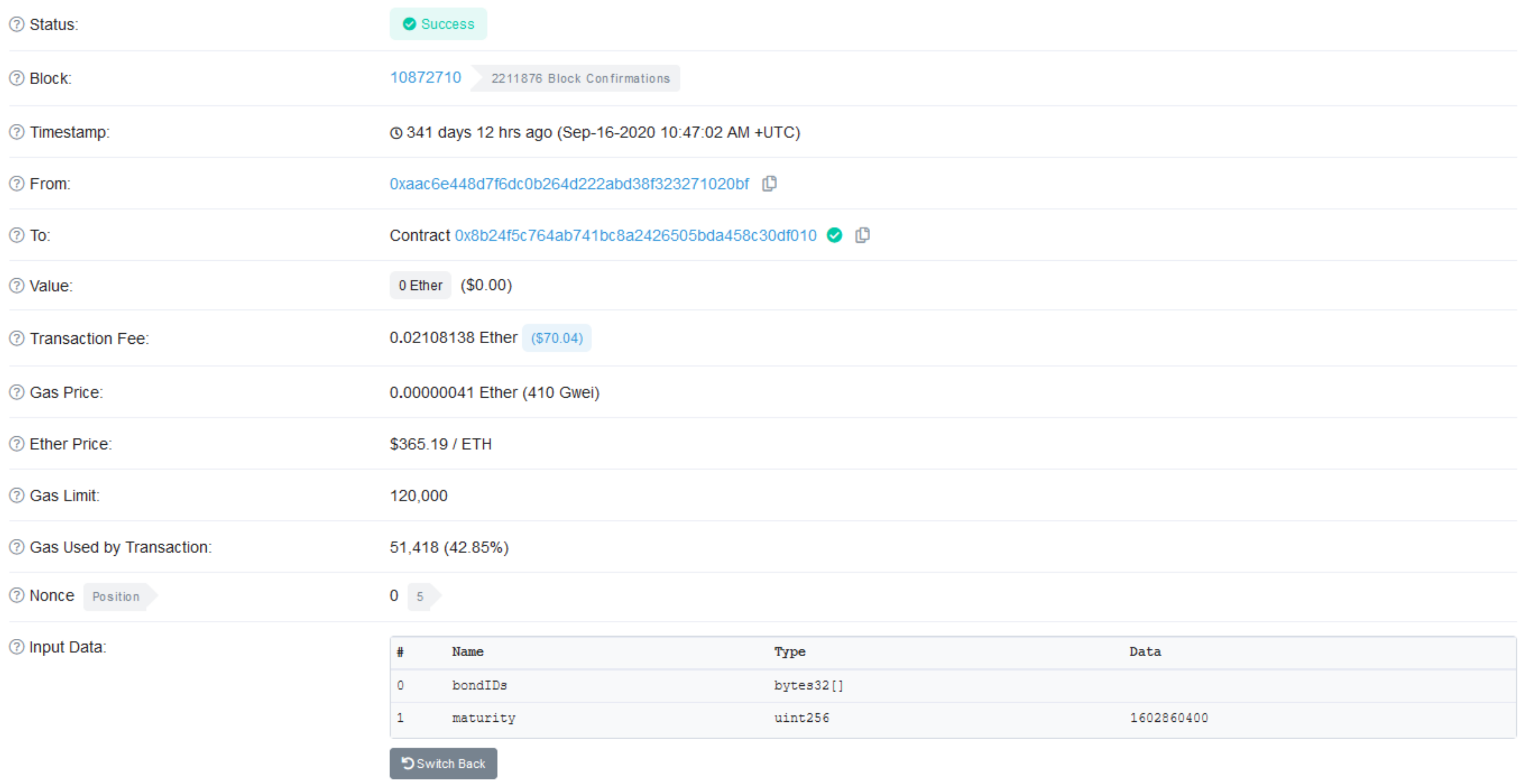
Fixing The Bug

- Tina contacted SparkPool’s co-founder, Shaoping, who offered to help
  - More identity verification!
- Fortunately, SparkPool had been in the middle of a private relay service already
- SparkPool finished development in 2 hours
- Meanwhile, Scott and I were working on the rescue payload

Liabilities

- What happens if a rescue goes wrong? Who’s at fault?
  - Solution: Have Lien perform the withdrawal
- What sort of tax obligations does temporarily receiving 10 million USD create?
  - Solution: Send tokens directly to Lien

Escaping The Dark Forest

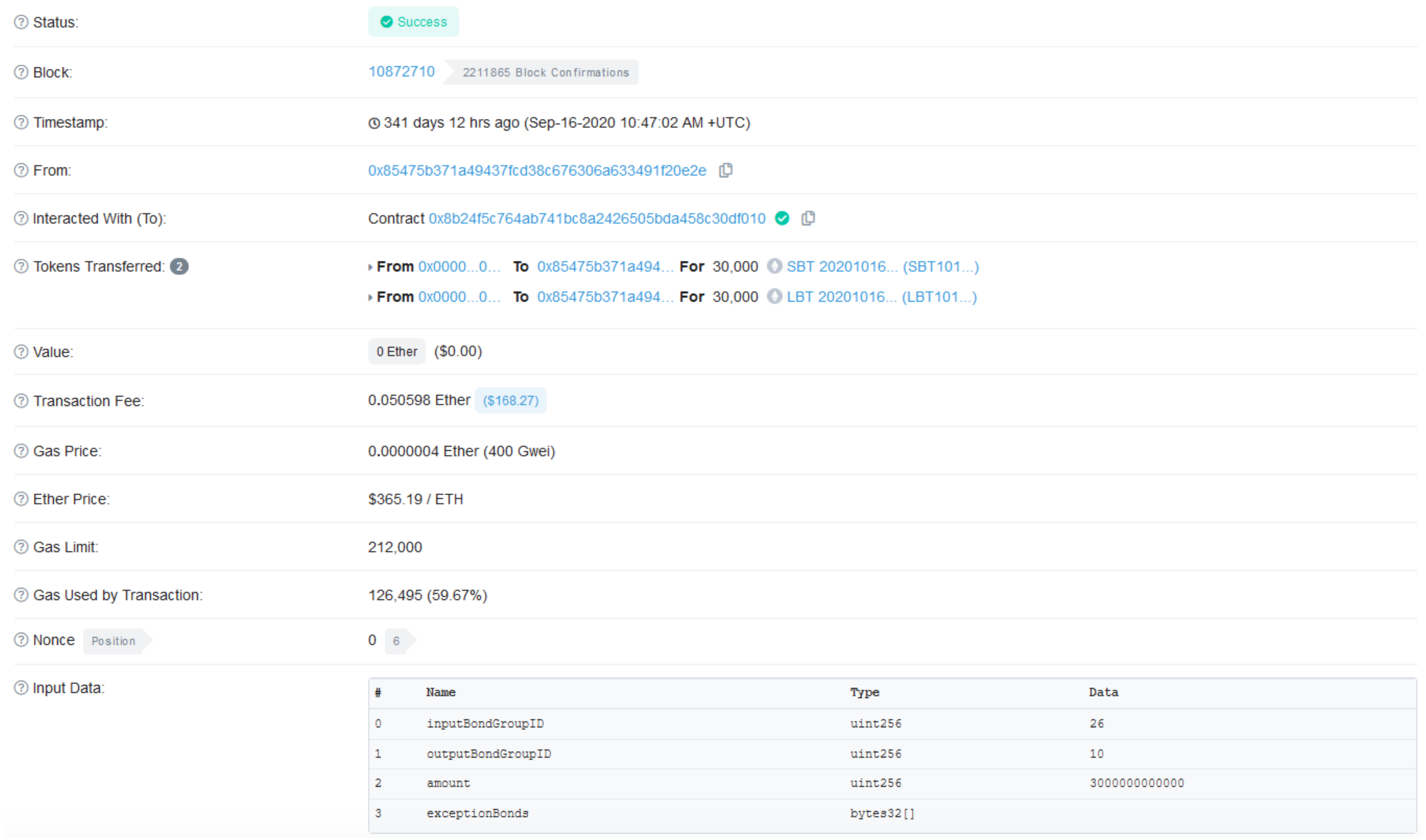


Status:	Success
Block:	10872710 2211876 Block Confirmations
Timestamp:	341 days 12 hrs ago (Sep-16-2020 10:47:02 AM +UTC)
From:	0xaac6e448d7f6dc0b264d222abd38f323271020bf
To:	Contract 0x8b24f5c764ab741bc8a2426505bda458c30df010
Value:	0 Ether ($0.00)
Transaction Fee:	0.02108138 Ether ($70.04)
Gas Price:	0.00000041 Ether (410 Gwei)
Ether Price:	$365.19 / ETH
Gas Limit:	120,000
Gas Used by Transaction:	51,418 (42.85%)
Nonce Position	0 5
Input Data:	

#	Name	Type	Data
0	bondIDs	bytes32[]	
1	maturity	uint256	1602860400

Switch Back

Escaping The Dark Forest



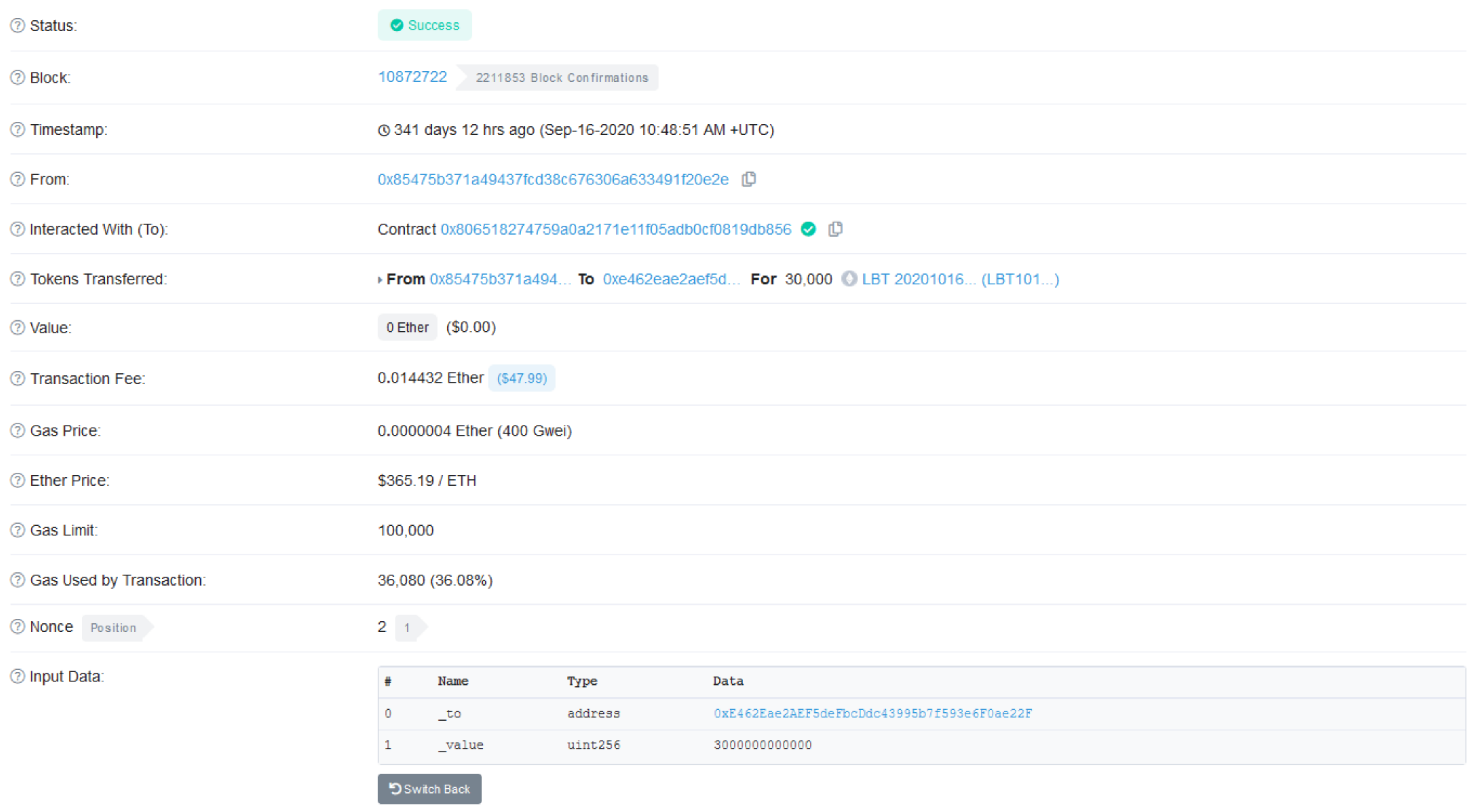
Status:	Success
Block:	10872710 2211865 Block Confirmations
Timestamp:	341 days 12 hrs ago (Sep-16-2020 10:47:02 AM +UTC)
From:	0x85475b371a49437fcd38c676306a633491f20e2e
Interacted With (To):	Contract 0x8b24f5c764ab741bc8a2426505bda458c30df010
Tokens Transferred: 2	From 0x0000...0... To 0x85475b371a494... For 30,000 SBT 20201016... (SBT101...) From 0x0000...0... To 0x85475b371a494... For 30,000 LBT 20201016... (LBT101...)
Value:	0 Ether ($0.00)
Transaction Fee:	0.050598 Ether ($168.27)
Gas Price:	0.0000004 Ether (400 Gwei)
Ether Price:	$365.19 / ETH
Gas Limit:	212,000
Gas Used by Transaction:	126,495 (59.67%)
Nonce Position	0 6

Input Data:

#	Name	Type	Data
0	inputBondGroupID	uint256	26
1	outputBondGroupID	uint256	10
2	amount	uint256	3000000000000
3	exceptionBonds	bytes32[]	

DeFi MOOC

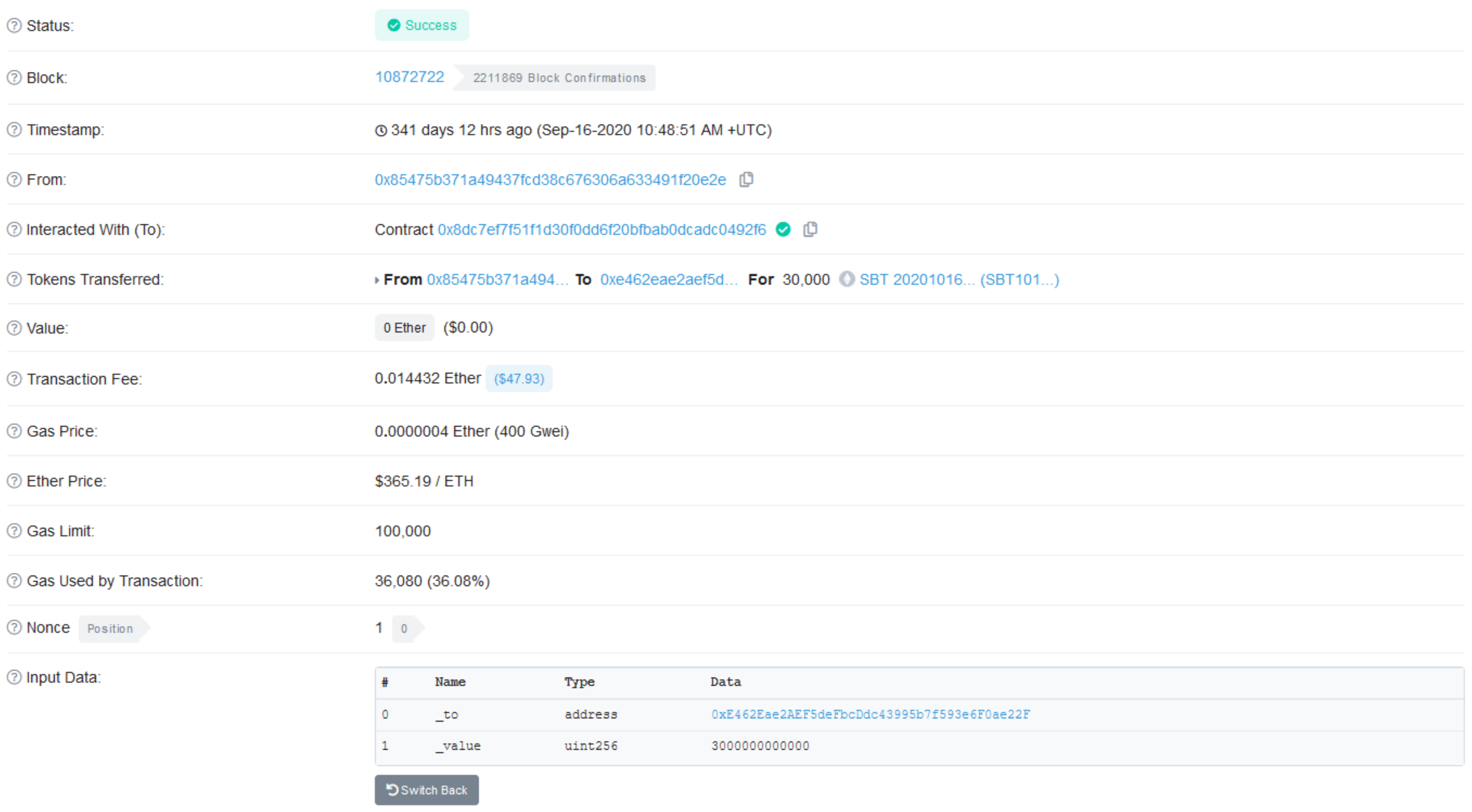
Escaping The Dark Forest



Status:	Success
Block:	10872722 2211853 Block Confirmations
Timestamp:	341 days 12 hrs ago (Sep-16-2020 10:48:51 AM +UTC)
From:	0x85475b371a49437fcd38c676306a633491f20e2e
Interacted With (To):	Contract 0x806518274759a0a2171e11f05adb0cf0819db856
Tokens Transferred:	**From** 0x85475b371a494... **To** 0xe462eae2aef5d... **For** 30,000 LBT 20201016... (LBT101...)
Value:	0 Ether ($0.00)
Transaction Fee:	0.014432 Ether ($47.99)
Gas Price:	0.0000004 Ether (400 Gwei)
Ether Price:	$365.19 / ETH
Gas Limit:	100,000
Gas Used by Transaction:	36,080 (36.08%)
Nonce Position	2 1
Input Data:	

#	Name	Type	Data
0	_to	address	0xE462Eae2AEF5deFbcDdc43995b7f593e6F0ae22F
1	_value	uint256	3000000000000

Escaping The Dark Forest

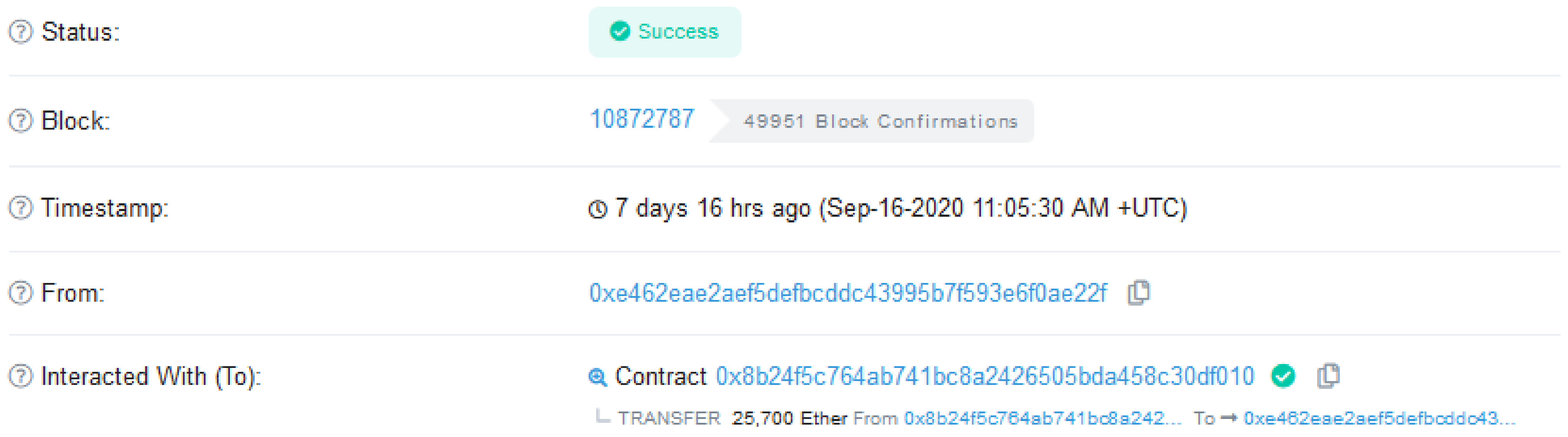


Status:	Success
Block:	10872722 2211869 Block Confirmations
Timestamp:	341 days 12 hrs ago (Sep-16-2020 10:48:51 AM +UTC)
From:	0x85475b371a49437fcd38c676306a633491f20e2e
Interacted With (To):	Contract 0x8dc7ef7f51f1d30f0dd6f20bfbab0dcadc0492f6
Tokens Transferred:	**From** 0x85475b371a494... **To** 0xe462eae2aef5d... **For** 30,000 SBT 20201016... (SBT101...)
Value:	0 Ether ($0.00)
Transaction Fee:	0.014432 Ether ($47.93)
Gas Price:	0.0000004 Ether (400 Gwei)
Ether Price:	$365.19 / ETH
Gas Limit:	100,000
Gas Used by Transaction:	36,080 (36.08%)
Nonce Position	1 0
Input Data:	

#	Name	Type	Data
0	_to	address	0xE462Eae2AEF5deFbcDdc43995b7f593e6F0ae22F
1	_value	uint256	3000000000000

Switch Back

Escaping The Dark Forest



Status:	Success
Block:	10872787 49951 Block Confirmations
Timestamp:	7 days 16 hrs ago (Sep-16-2020 11:05:30 AM +UTC)
From:	0xe462eae2aef5defbcddc43995b7f593e6f0ae22f
Interacted With (To):	Contract 0x8b24f5c764ab741bc8a2426505bda458c30df010 └ TRANSFER 25,700 Ether From 0x8b24f5c764ab741bc8a242... To → 0xe462eae2aef5defbcddc43...

Escaping The Dark Forest

- Lots to consider when rescuing 7+ figure funds
  - Identity verification
  - Liabilities
  - Taxes?
- Private relays for everyone

Next Up

Conclusions

Conclusions

Conclusions

- DeFi security is hard, but we're learning from past mistakes
  - Don't trust names of methods
  - How to methodically review code
  - Exploit chaining
  - EVM quirks
  - Cross-chain state machines
- Finding the bug is only the first step
  - Coordination with project
  - Identity verification
  - Taxes/liability

Conclusions

- All this sound interesting? Exciting? Feel free to reach out
  - Email: sam@samczsun.com
  - Telegram: @samczsun