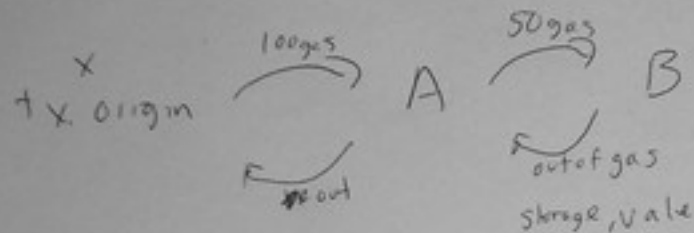


EVIL CALLS

4 params: ~~gas~~ ; gas, value, in (buf, size), out (hd, size)
 + to, from



X → A → B

Context

msg. sender

CALL

B

A

CALL CODE

A

A

DELEGATECALL

A

X

address x;

MyContract c = MyContract(x);

	target	gas	errors?
x.send(v)	fallback	minimum (2300)	return false
x.transfer(v)	fallback	minimum	throw
x.call, callcode, delegatecall	pass by high offset	all *	returns false
c.foo()	foo	all *	-throw

```

function claim Prize () {
    if (! claimed) {
        winner, send ( prizeVal );
        claimed = true;
    }
}

```

how can send fail?

- winner's fallback function can run out of gas
- hit call stack limit

```
if (! claimed) {  
    winner, send (prizeVal);  
    claimed = true;  
}
```

how can send fail?

- winner's fallback function can run out of gas
- hit call stack limit

Contract Nullify {

```
function nullify (uint count) {  
    if (count > 0)  
        set this.nullify (count-1);  
    else  
        target.claim Prize ();  
}
```

BUG: unchecked send

```
function claim Prize () {  
    if (! claimed) {  
        winner, send (prizeVal);  
        claimed = true;  
    }  
}
```

how can send fail?

- winner's fallback function can run out of gas
- hit call stack limit

Contract Nullify {

Fixing unchecked send

- use `transfer()` instead of `send()`
- check return value

Bug #2 : Reentrancy

```
function withdraw (uint amount) {  
    if (accounts[msg.sender] ≥ amount) {  
        msg.sender.call.value(amount)();  
        accounts[msg.sender] -= amount;  
    }  
}
```

}

```
Contract Reentrant {  
    Lottery target;  
    function () payable {  
        target.withdraw(100);  
    }  
}
```

}

}

3

Contract Reentrant &
 Lottery target;
 function () payable &
 target.withdraw(100);

L. balance	R. balance	L. accounts[R]	call stack
1000	0	100	R
900	100	100	R → L
800	200	100	R → L → R → L
700	300	100	R
0	1000	100	(R → L) × 10
0	1000	- 900	


```

- function withdraw (amount) {
    if (accounts[msg.sender] >= amount) {           CHECKS
        accounts[msg.sender] -= amount;             EFFECTS
        if (!msg.sender.call.value(amount)) {       INTERNAL
            accounts[msg.sender] += amount;
        }
    }
}

```

- use send(), transfer()

- use a lock

```

bool _rLock;

```

```

modifier noReentrancy() {
    require(!_rLock);
    _rLock = true;
    _;
    _rLock = false;
}

```

1) Verification / ~~capturing~~ capturing developer intent

2) Proof - ~~of~~ ^{of} stake

- Snow White
- Casper
- Ouroboros
- Algorand

3) Privacy (Smart contracts + zk)

- Zerocash / (Project Alchemy)
zcash
- Hawk

4) Scalability

- sharding
- state channels (off-chain channels)

Security issues in Ethereum



* mostly

Recap: why is secure programming hard for smart contracts?

- no bug fixes (code is fixed)
- concurrency
- no regulation (anonymity)
- system is immature
- money (no undo* / no chargeback)
- requirements hard to specify

Alice

Bob



$$\text{Commit}(s) = H(r, s)$$

Solution

- deposit
- timeout



$$\text{Commit}(s) = H(r, s)$$

Solution

- deposit
- timeout



Vickrey Auction

(sealed bid second price)

Bid phase

Commit (bid), deposit

Commit (bid), deposit

Reveal phase

reveal, \$bid

reveal, \$bid

