



ISC 互联网安全大会



360 互联网安全中心

漫谈以太坊公链安全

罗元琮 PeckShield 漏洞研究總监

ISC 互联网安全大会 中国·北京
Internet Security Conference 2018 Beijing·China



360 技术

IT大咖说
知识共享平台

罗元琮 (Edward)

- Director of Vulnerability Research at PeckShield
- Has extensive experiences in OS kernel layer with deep knowledge in advanced vulnerability discovery and exploitation
- Before joining PeckShield, I worked for Qihoo 360 as the team lead of CORE Team, which was recognized by Google as the top research team in 2017. I am now focusing on the security of blockchain infrastructure
- Submitted several vulnerabilities to the Ethereum Foundation

ETHEREUM IN 2017

1,090 Dapps & 700+ Tokens

100,000 New Users Per Day

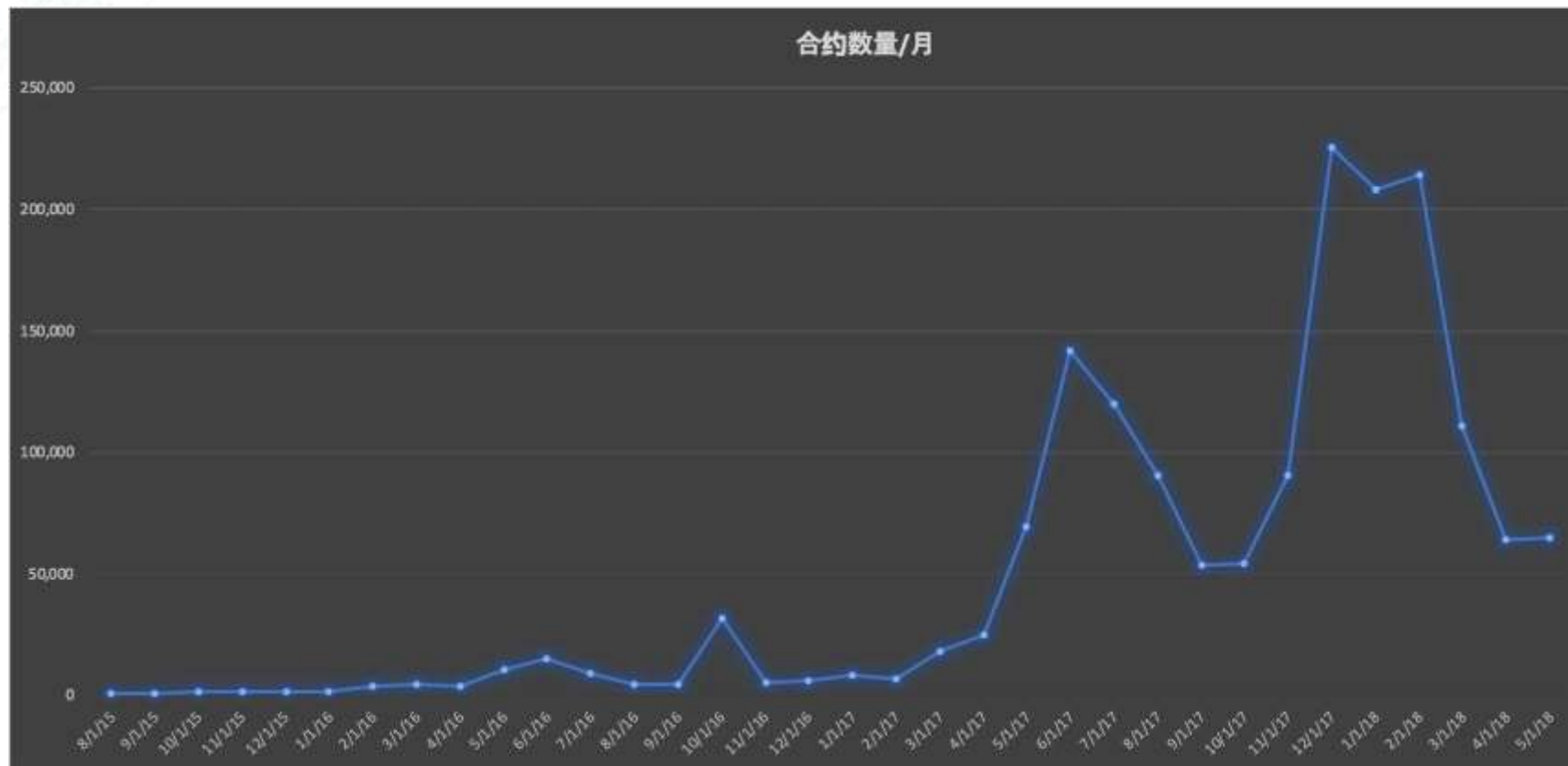
Daily Trading > 1,000,000

MARKET CAP IN 2018

1,845 Cryptocurrencies

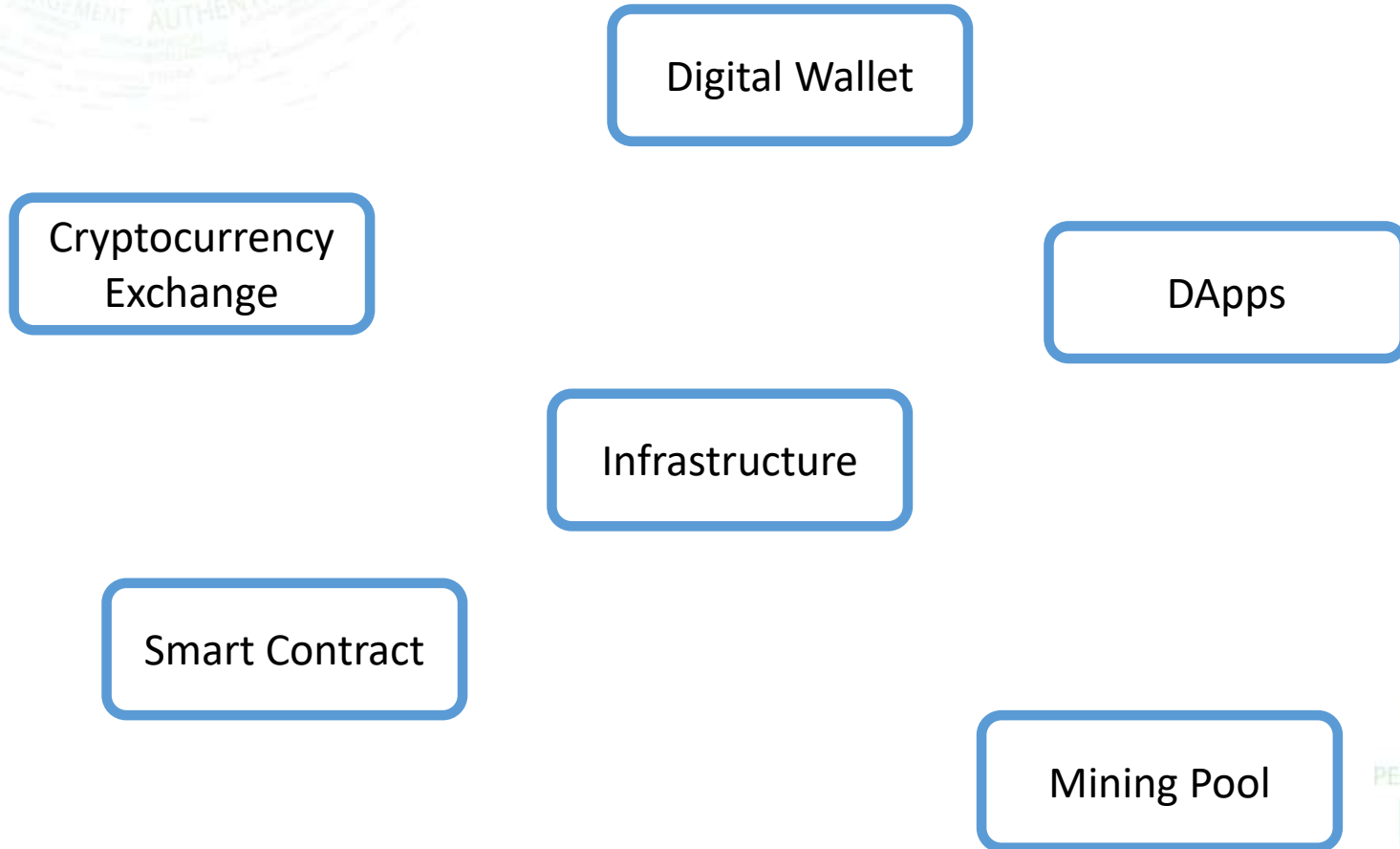
Market Cap > 200 Billion

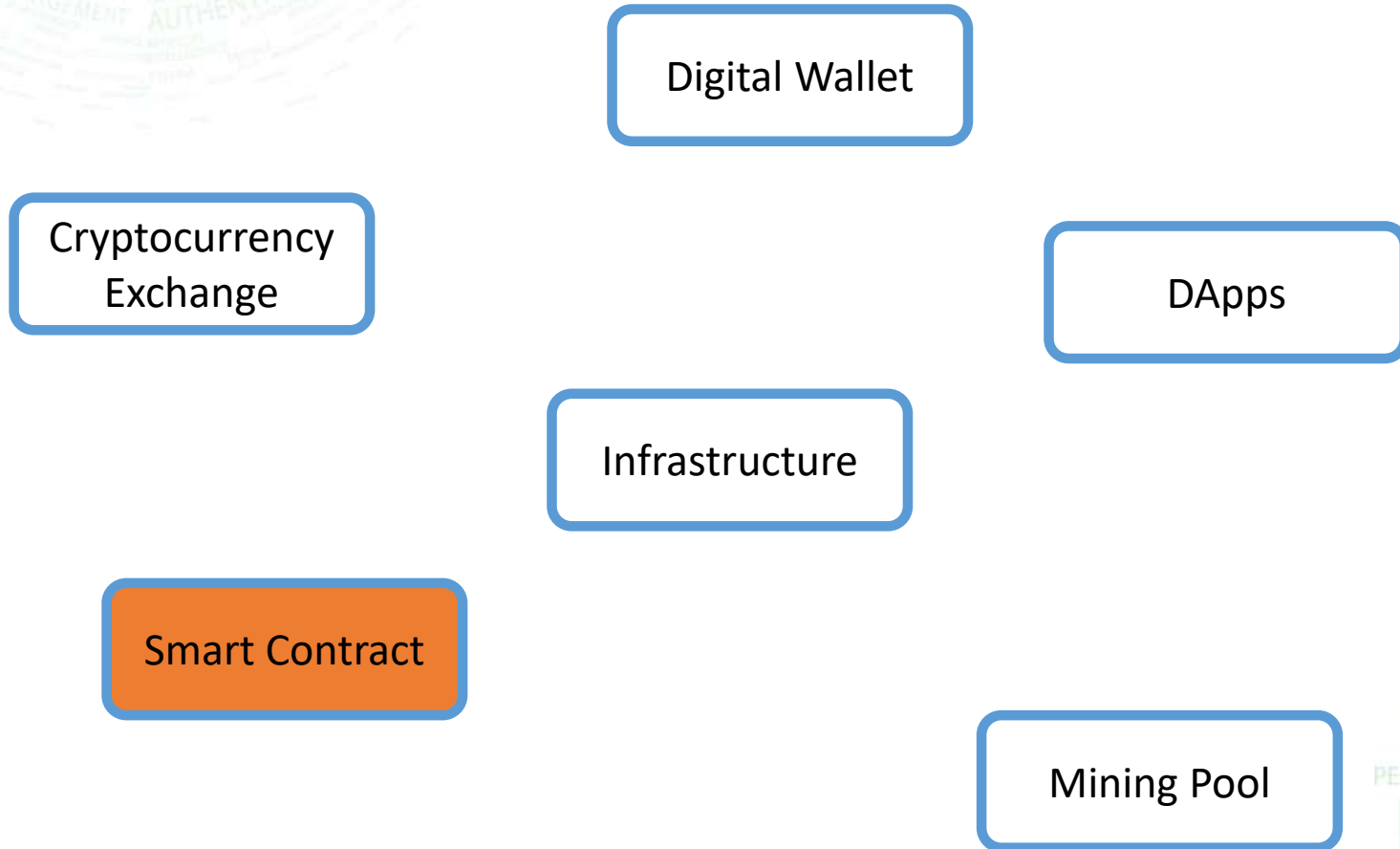
Global GDP Rank: ~50th





2018/06	Bithumb Hacks with \$31 Million Dollars Stolen
2018/05	EDU, BAIC Smart Contracts Bugs
2018/04	BEC, SMT Smart Contracts Bugs
2018/04	Myetherwallet Suffer from DNS Hijacking
2018/02	BitGrail Hacks with Stolen Nano Tokens of 170 Million Dollars
2018/01	Coincheck Hacks with 530 Million Dollars Stolen
2017/12	Nicehash Hacks with 4700 BTC Missing with 62 Million Dollars
2017/06	Bithumb Hacks with 1 Billion Korean Yuan Loss and 30 Thousand User Info. Leaked
2016/08	Bitfinex Hacks with 120,000 BTC Stolen of 75Million Dollars
2016/01	Cryptsy Hacks with 13,000 BTC and 300,000 LTC Stolen
2015/01	Bitstamp Hacks with 19,000 BTC Stolen
2014/03	Poloniex Hacks with 12.3% BTC Lost
2014/02	Mt.Gox Hacks with Followed Bankruptcy





Disclosed by PeckShield	
codename	CVE-ID
batchOverflow	CVE-2018-10299
proxyOverflow	CVE-2018-10376
transferFlaw	CVE-2018-10468
ownerAnyone	CVE-2018-10705
multiOverflow	CVE-2018-10706
burnOverflow	CVE-2018-11239
ceoAnyone	CVE-2018-11329
allowAnyone1	CVE-2018-11397
allowAnyone2	CVE-2018-11398
tradeTrap1	CVE-2018-12017
tradeTrap2	CVE-2018-12062
tradeTrap3	CVE-2018-12079
...	...

Secure | <https://cointelegraph.com/news/multiple-exchanges-suspend-erc20-token-trading-due-to-potential-batchoverflow-bug>

By Molly Jane Zuckerman APR 25, 2018

Multiple Exchanges Suspend ERC20 Token Trading Due To Potential BatchOverflow Bug

29984 Total views 385 Total shares



Blog

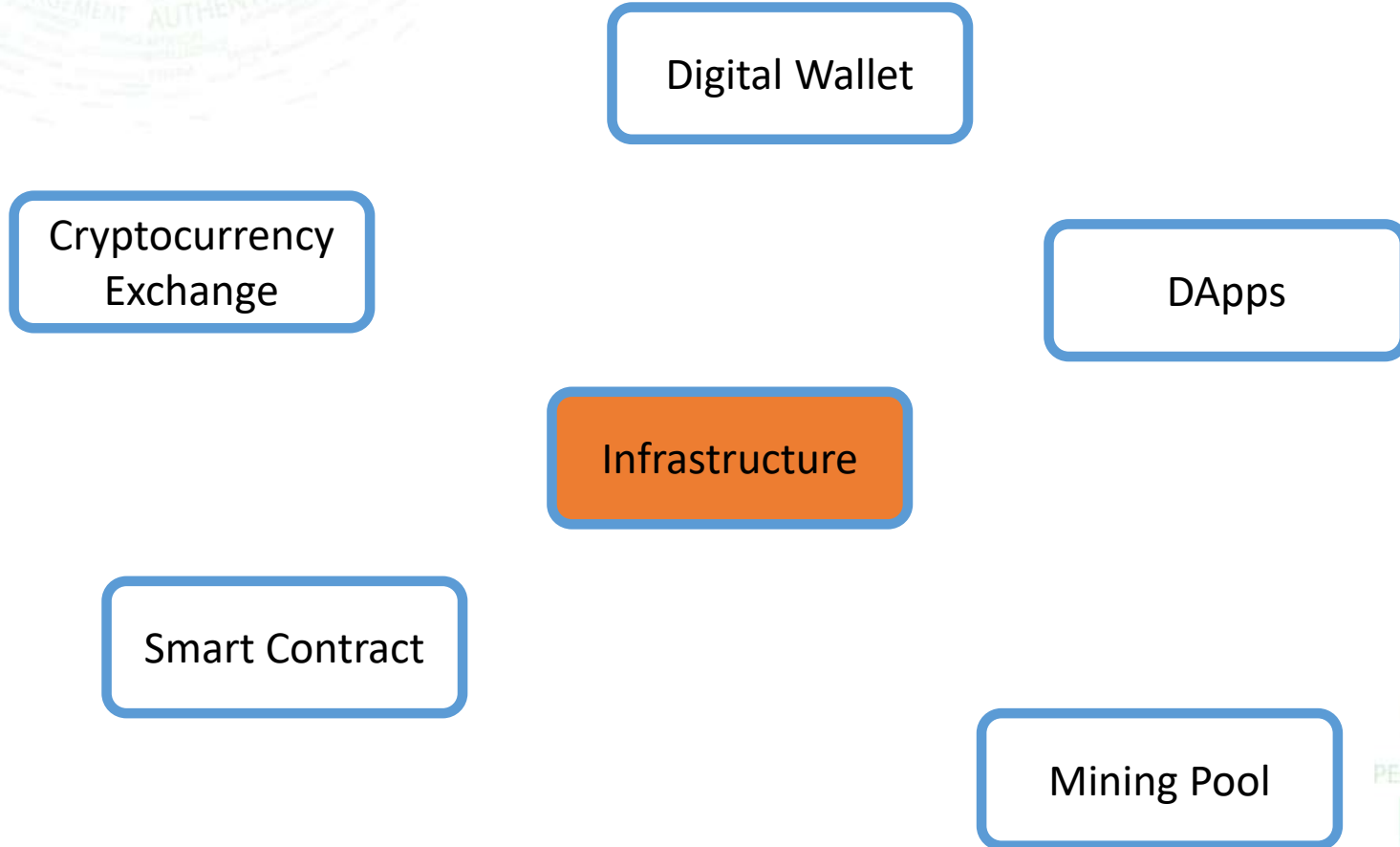
Research

Advisories

About



- [CVE-2018-12105](#) - New transferFlaw Bug Identified in an ERC20-Based Smart Contract
- [CVE-2018-12084](#) - New tradeTrap Bug Identified in BitAsean (BAS) Smart Contract
- [CVE-2018-12083](#) - New tradeTrap Bug Identified in GOAL Bonanza (GOAL) Smart Contract
- [CVE-2018-12082](#) - New tradeTrap Bug Identified in Fujinto (NTO) Smart Contract
- [CVE-2018-12081](#) - New tradeTrap Bug Identified in Target Coin (TGT) Smart Contract
- [CVE-2018-12080](#) - New tradeTrap Bug Identified in Internet Node Token (INT) Smart Contract
- [CVE-2018-12079](#) - New tradeTrap Bug Identified in Substratum (SUB) Smart Contract
- [CVE-2018-12078](#) - New tradeTrap Bug Identified in PolyAI (AI) Smart Contract
- [CVE-2018-12063](#) - New tradeTrap (Overflow) Bug Identified in Internet Node Token (INT) Smart Contract
- [CVE-2018-12062](#) - New tradeTrap Bug Identified in SwiftCoin (SWFTC) Smart Contract
- [CVE-2018-12018](#) - New Out-of-Bound Access Bug Identified in an Ethereum Client
- [CVE-2018-12017](#) - New tradeTrap Bug Identified in hotchain.io (HCH) Smart Contract
- [CVE-2018-11673](#) - New Memory Leak Bug Identified in an Ethereum Client
- [CVE-2018-11585](#) - New ctorMismatch Bug Identified in MORPH (MORPH) Smart Contract
- [CVE-2018-11582](#) - New balanceAnyone Bug Identified in an ERC20-Based Smart Contract
- [CVE-2018-11561](#) - New distributeFlaw Bug Identified in Multiple ERC20-Based Smart Contracts
- [CVE-2018-11446](#) - New tradeTrap Bug Identified in an ERC20-Based Smart Contract
- [CVE-2018-11441](#) - New tradeTrap Bug Identified in an ERC20-Based Smart Contract



ETHEREUM CLIENTS

geth (golang)

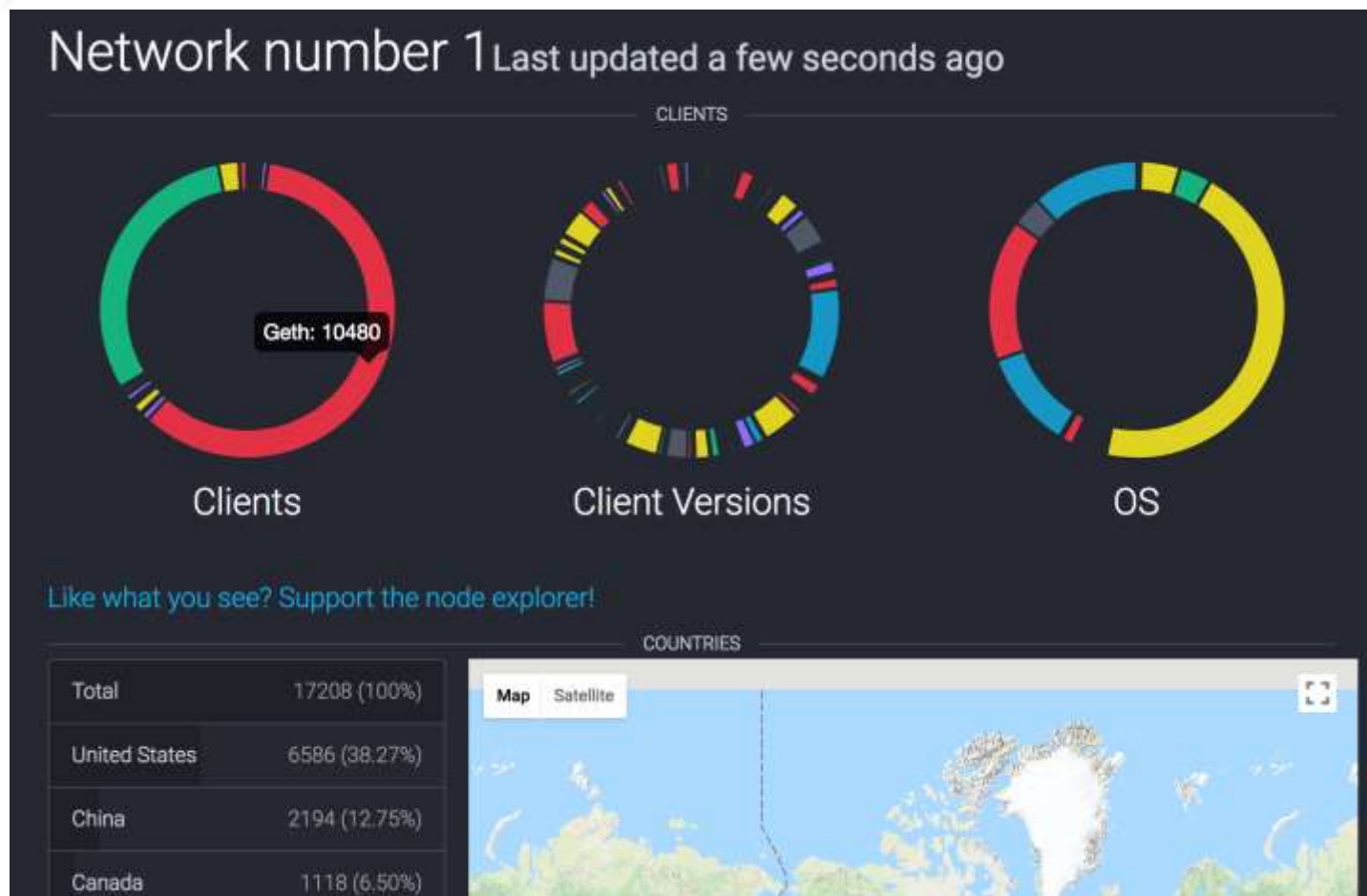
aleth (c++)

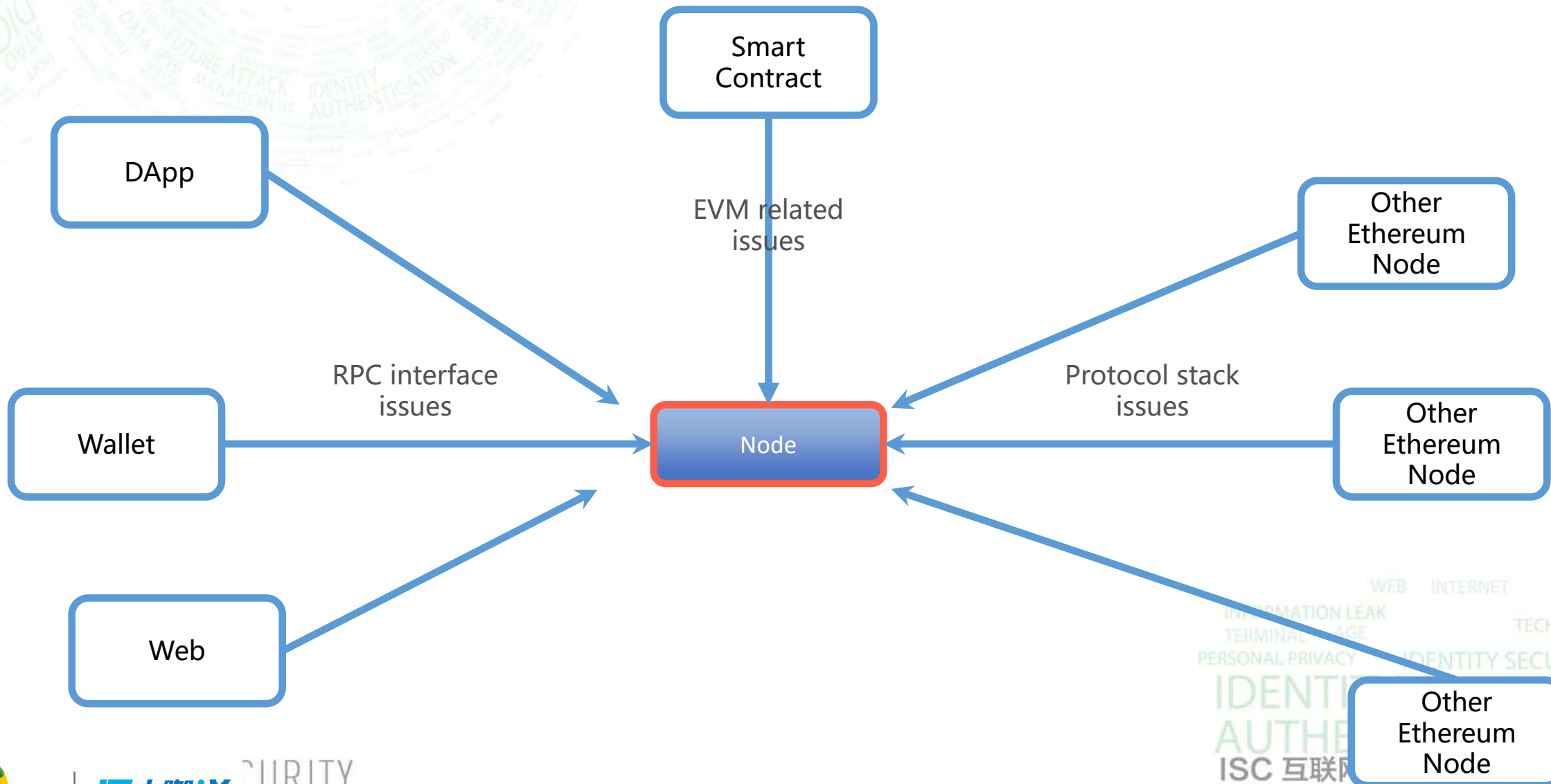
Parity (Rust) – by Parity Technologies

MARKET SHARE

geth ~2/3

parity ~1/3





ETH

LES

Whisper

...

- Various Sub-protocols

ΞVp2p

- Support arbitrary sub-protocols (aka capabilities) over the basic wire protocol
- Connection management

RLPx

- Encrypted Handshake/Authentication
- Peer Persistence
- UDP Node Discovery Protocol

ETH

LES

Whisper

...

- Various Sub-protocols

⊕EVP2p

- Support arbitrary sub-protocols (aka capabilities) over the basic wire protocol
- Connection management

RLPx

- Encrypted Handshake/Authentication
- Peer Persistence
- UDP Node Discovery Protocol

LIGHT ETHEREUM SUBPROTOCOL

- used by "light" clients, which only download block headers as they appear and fetch other parts of the on-demand
- do not mine and therefore do not take part in the consensus process
- Several message handlers for different types of messages
 - StatusMsg, AnnounceMsg, GetBlockBodiesMsg, GetBlockHeadersMsg

```
// getBlockHeadersData represents a block header query.
type getBlockHeadersData struct {
    Origin hashOrNumber // Block from which to retrieve headers
    Amount uint64        // Maximum number of headers to retrieve
    Skip    uint64        // Blocks to skip between consecutive headers
    Reverse bool           // Query direction (false = rising towards latest)
}
```

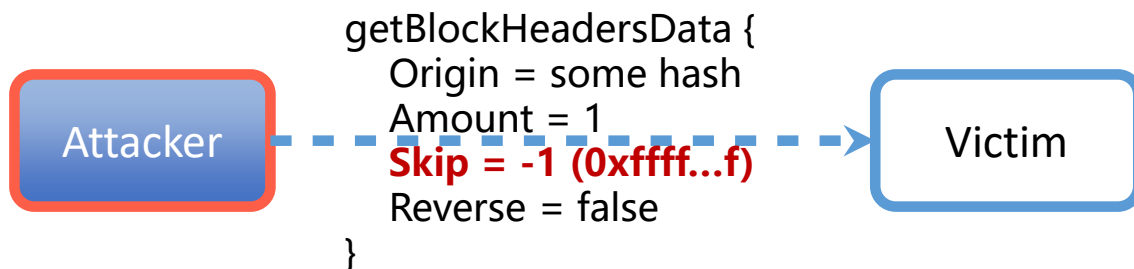
```
// hashOrNumber is a combined field for specifying an origin block.
type hashOrNumber struct {
    Hash    common.Hash // Block hash from which to retrieve headers (excludes Number)
    Number  uint64       // Block hash from which to retrieve headers (excludes Hash)
}
```



```
// Retrieve the next header satisfying the query
var origin *types.Header
if hashMode {
    origin = pm.blockchain.GetHeaderByHash(query.Origin.Hash)
} else {
    origin = pm.blockchain.GetHeaderByNumber(query.Origin.Number)
}
if origin == nil {
    break
}
```

```
case query.Origin.Hash != (common.Hash{}) && !query.Reverse:
    // Hash based traversal towards the leaf block
    if header := pm.blockchain.GetHeaderByNumber(origin.Number.Uint64() + query.Skip + 1); header != nil {
        if pm.blockchain.GetBlockHashesFromHash(header.Hash(), query.Skip+1)[query.Skip] == query.Origin.Hash {
            query.Origin.Hash = header.Hash()
        } else {
            unknown = true
        }
    } else {
        unknown = true
    }
}
```

```
//getBlockHeadersData represents a block header query.  
type getBlockHeadersData struct {  
    Origin hashOrNumber // Block from which to retrieve headers  
    Amount uint64        // Maximum number of headers to retrieve  
    Skip    uint64        // Blocks to skip between consecutive headers  
    Reverse bool          // Query direction (false = rising towards latest  
}
```

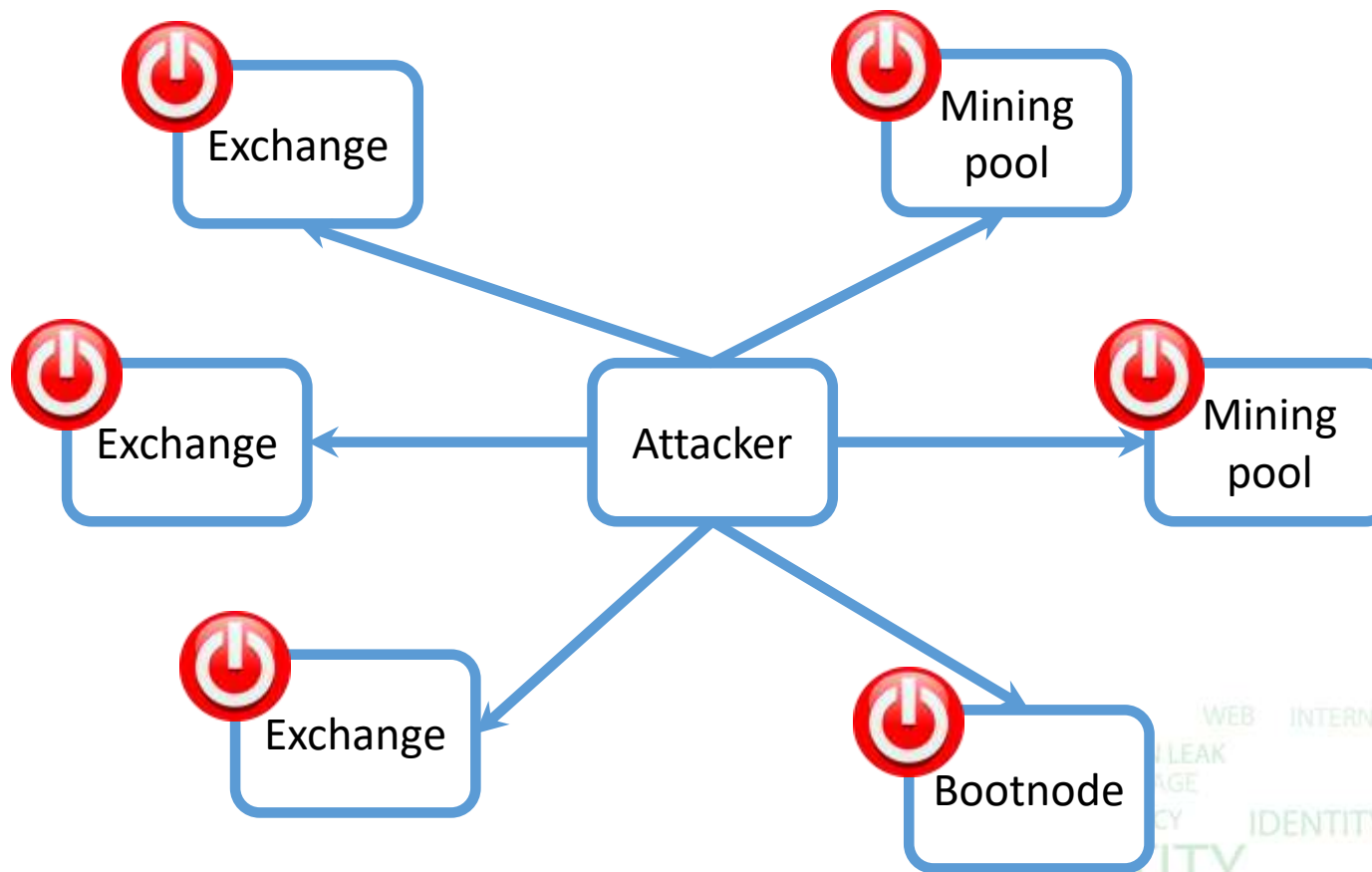


1. Allocate array for max # of blocks
 - allocate (Skip+1):
Zero-Size Array Allocated
2. Query from returned array[Skip]
 - Access array[-1]:
Out-of-Bound Read

DEMO

POSSIBLE VICTIMS

- Cryptocurrency exchanges
- Mining pools
- Bootnodes



```
var (  
    current = origin.Number.Uint64()  
    next    = current + query.Skip + 1  
)  
if next <= current {  
    infos, _ := json.MarshalIndent(p.Peer.Info(), "", " ")  
    p.Log().Warn("GetBlockHeaders skip overflow attack", "c"  
    unknown = true
```

```
case AnnounceMsg:
    ...

    var req announceData
    if err := msg.Decode(&req); err != nil {
        return errResp(ErrDecode, "%v: %v", msg, err)
    }

    ...

    if pm.fetcher != nil {
        pm.fetcher.announce(p, &req)
    }
}
```

```
// announceData is the network packet for the block announcements.
type announceData struct {
    Hash      common.Hash // Hash of one particular block being announced
    Number    uint64      // Number of one particular block being announced
    Td        *big.Int    // Total difficulty of one particular block being announced
    ReorgDepth uint64
    Update    keyValueList
}
```

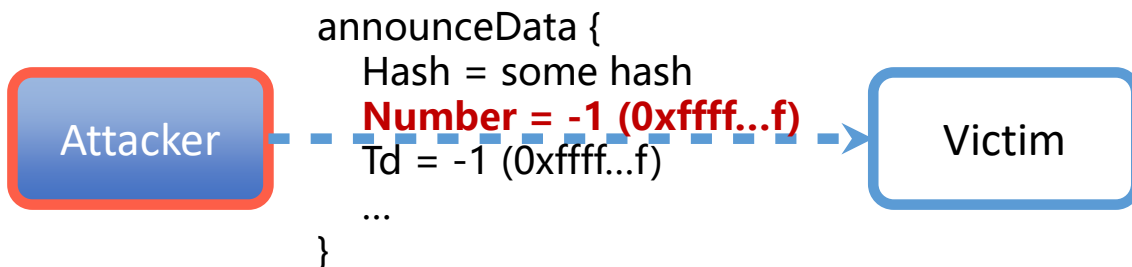


```
func (f *lightFetcher) announce(p *peer, head *announceData) {
    ...

    fp := f.peers[p]
    ...

    n := fp.lastAnnounced
    ...
    if n != nil {
        // n is now the reorg common ancestor, add a new branch of nodes
        // check if the node count is too high to add new nodes
        locked := false
        for uint64(fp.nodeCnt)+head.Number-n.number > maxNodeCount && fp.root != nil {
            ...
        }
        if n != nil {
            for n.number < head.Number {
                nn := &fetcherTreeNode{number: n.number + 1, parent: n}
                n.children = append(n.children, nn)
                n = nn
                fp.nodeCnt++
            }
            n.hash = head.Hash
            n.td = head.Td
            fp.nodeByHash[n.hash] = n
        }
    }
}
```

```
// announceData is the network packet for the block announcements.
type announceData struct {
    Hash      common.Hash // Hash of one particular block being announced
    Number    uint64       // Number of one particular block being announced
    Td        *big.Int     // Total difficulty of one particular block being announced
    ReorgDepth uint64
    Update    keyValueList
}
```



- for $n < \text{head.Number}$
 - Allocate `fetcherTreeNode`
 - Append to `n.children`
 - **Out-of-memory**

DEMO


```
+ // n is now the reorg common ancestor, add a new branch of nodes
+ if n != nil && (head.Number >= n.number+maxNodeCount || head.Number <= n.number) {
+     // if announced head block height is lower or same as n or too far from it to add
+     // intermediate nodes then discard previous announcement info and trigger a resync
+     n = nil
+     fp.nodeCnt = 0
+     fp.nodeByHash = make(map[common.Hash]*fetcherTreeNode)
+ }
+ if n != nil {
```

ETH

LES

Whisper

...

- Various Sub-protocols

⊕EVP2p

- Support arbitrary sub-protocols (aka capabilities) over the basic wire protocol
- Connection management

RLPx

- Encrypted Handshake/Authentication
- Peer Persistence
- UDP Node Discovery Protocol

NODE DISCOVERY PROTOCOL

- Aimed at discovering RLPx nodes to connect to
- UDP-based RPC protocol (kademlia-like)
- Defines 4 packet types: ping, pong, findnode and neighbors

```
func (t *udp) handlePacket(from *net.UDPAddr, buf []byte) error {
    packet, fromID, hash, err := decodePacket(buf)
    if err != nil {
        log.Debug("Bad discv4 packet", "addr", from, "err", err)
        return err
    }
    err = packet.handle(t, from, fromID, hash)
    log.Trace("<< "+packet.name(), "addr", from, "err", err)
    return err
}
```



```
ping struct {
    Version      uint
    From, To     rpcEndpoint
    Expiration   uint64
    // Ignore additional fields (for forward compatibility).
    Rest []rlp.RawValue `rlp:"tail"`
}
```

```
func (req *ping) handle(t *udp, from *net.UDPAddr, fromID NodeID, mac []byte) error {
    if expired(req.Expiration) {
        return errExpired
    }
    t.send(from, pongPacket, &pong{
        To:          makeEndpoint(from, req.From.TCP),
        ReplyTok:    mac,
        Expiration:  uint64(time.Now().Add(expiration).Unix()),
    })
    if !t.handleReply(fromID, pingPacket, req) {
        // Note: we're ignoring the provided IP address right now
        go t.bond(true, fromID, from, req.From.TCP)
    }
}
```

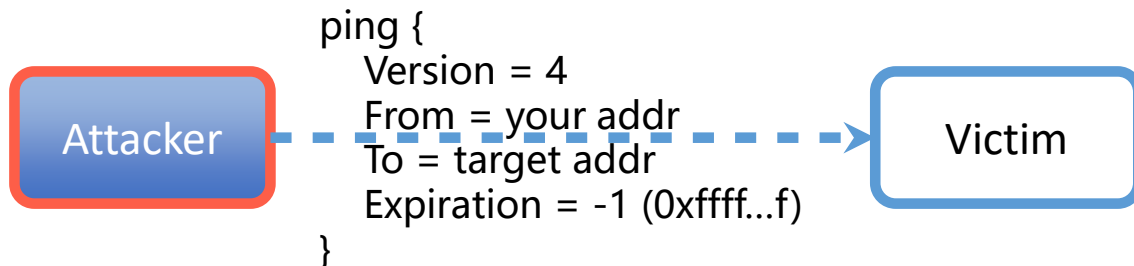
```
node, fails := tab.db.node(id), tab.db.findFails(id)
age := time.Since(tab.db.bondTime(id))
var result error
if fails > 0 || age > nodeDBNodeExpiration {

    tab.bondmu.Lock()
    w := tab.bonding[id]
    if w != nil {
        // Wait for an existing bonding process to complete.
        tab.bondmu.Unlock()
        <-w.done
    } else {
        // Register a new bonding process.
        w = &bondproc{done: make(chan struct{})}
        tab.bonding[id] = w
        tab.bondmu.Unlock()
        // Do the ping/pong. The result goes into w.
        tab.pingpong(w, pinged, id, addr, tcpPort)
        // Unregister the process after it's done.
        tab.bondmu.Lock()
        delete(tab.bonding, id)
        tab.bondmu.Unlock()
    }
    // Retrieve the bonding results
    result = w.err
    if result == nil {
        node = w.n
    }
}
```



```
// Request a bonding slot to limit network usage  
<- tab.bondslots  
defer func() { tab.bondslots <- struct{}{} }()
```

```
ping struct {  
    Version      uint  
    From, To     rpcEndpoint  
    Expiration   uint64  
    // Ignore additional fields (for forward compatibility).  
    Rest []rlp.RawValue `rlp:"tail"`  
}
```



1. Generate many key pairs
2. Sign the ping packets with each private key
3. Flood the victim with many ping packets
4. Each ping will consume a goroutine resource **after 16 pending ping requests**

DEMO

```
+ // Add the node to the table. Before doing so, ensure that we have a recent enough pong
+ // recorded in the database so their findnode requests will be accepted later.
+ n := NewNode(fromID, from.IP, uint16(from.Port), req.From.TCP)
+ if time.Since(t.db.lastPongReceived(fromID)) > nodeDBNodeExpiration {
+     t.sendPing(fromID, from, func() { t.addThroughPing(n) })
+ } else {
+     t.addThroughPing(n)
+ }
+ t.db.updateLastPingReceived(fromID, time.Now())
```

- Blockchain can't function without the fundamental components
 - Infrastructure (nodes)
 - Mining pool
 - ...
- Vulnerability could exist in any aspects of the blockchain ecosystem
 - EPoD / EPoD2
 - Freether
- Some suggestions
 - Smart contract audit before going online
 - Security response after going online
 - Community / Bounty Program



ISC 互联网安全大会



360互联网安全中心

谢谢!

ISC 互联网安全大会 中国·北京
Internet Security Conference 2018 Beijing·China



360技术

IT大咖说
知识共享平台