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 Deliverers
What is XRP?

- XRP means XRP Raid Protector
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.
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.

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

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.

The Bug (CVE-2022-29077)

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

```cpp
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;
```
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
The Bug (CVE-2022-29077)

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

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

```
// LivecacheAllocator::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];
}
```
The Bug (CVE-2022-29077)

- Memory layout

```cpp
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

```c
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

```cpp
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

```
[+] 0x4d360f8, off 5, 0x7f71b42b2970, 0x4d360f8
[+] 0x4d35fe0, off 23, 0x4d35e18, 0x4d35e0
[-] 0x4d35fb8, off 25, 0x4d35f8a, 0x4d35fa8
[-] 0x4d35ea9, off 43, 0x4d35d40, 0x4d3e130
[-] 0x4d35d30, off 65, 0x4d35d28, 0x4d35d28
[-] 0x4d35c0, off 69, 0x4c129b0, 0x4c116e0
[-] 0x4d35cc0, off 71, 0x4d35ce0, 0x4d35d0
[-] 0x4d35ce8, off 72, 0x4d35b8, 0x4d35d0
[-] 0x4d35bf8, off 85, 0x4d35be8, 0x4d35be8
[-] 0x4d35bd8, off 87, 0x4d35d8, 0x4d35d0
[-] 0x4d35e0, off 94, 0x4d38450, 0x4d38450
[-] 0x4d35a40, off 112, 0x7f71b41c15e0, 0x7f71b002e00
[-] 0x4d3570, off 221, 0x4d35b80, 0x4d35b80
[-] 0x4d35c8, off 222, 0x4c3b70, 0x4c3b80
[-] 0x4d35c8, off 227, 0x4d3d9e0, 0x4d3d9e0
[-] 0x4d35250, off 239, 0x4c127d8, 0x4c127d8
[-] 0x4d35230, off 241, 0x4d3c80, 0x4d3c80
[-] 0x4d35220, off 242, 0x4c12780, 0x4c12780
[-] 0x4d351e0, off 243, 0x48e2be0, 0x4c18770
[-] 0x4d351e0, off 255, 0x4d35b0, 0x4d35b0
[-] 0x4d351e0, off 257, 0x4c26d8, 0x4c26d86
```

```
x x x
next
prev
x x x
next
prev
x x x
m_lists[0]
m_lists[1]
...
```

Endpoint
Endpoint
Endpoint
Endpoint
[Bonus] DoS Exploit

- 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]

## Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Hops</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x010</td>
<td>ip_address</td>
<td></td>
</tr>
<tr>
<td>0x020</td>
<td>-</td>
<td>port</td>
</tr>
<tr>
<td>0x030</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

## Diagram
Heap Spraying

Challenges

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

Long-term memory preallocation

- **Endpoint**
  - “ipv6 address” field
  - must follow the validation verifications
  - only last for 30 seconds

- **Transaction**
  - “Condition” field
  - 250 transactions in queue at most
  - will be broadcast into the whole network
Heap Spraying

- Manifest
- Go deeper into Protobuf

```cpp
// Provides the current ephemeral key for a validator. */
message TMManifest
{
  // A Manifest object in the Ripple serialization format.
  required bytes sobject = 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::Internal
  typedef void InternalArenaConstrucatable;
  typedef void DestructorSkippable;
  ::PROTOBUF_NAMESPACE_ID::internal::HasBits<T> _has_bits;
  mutable ::PROTOBUF_NAMESPACE_ID::internal::CachedSize cached_size;
};

// 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
Heap Spraying

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

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.
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

```
0x8000000000003620 0x8000000000009c0  
0x8000000000000001 0x8000000000000800  
0x80000000000027b5c1 0x8000000000000a23  
0x8000000000009c0 0x8000000000000000  
0x8000000000000000 0x8000000000000000  
```

Endpoint Object

```
0x8000000000003620 0x8000000000009c0  
0x8000000000000001 0x8000000000000800  
0x80000000000027b5c1 0x8000000000000a23  
0x8000000000009c0 0x8000000000000000  
0x8000000000000000 0x8000000000000000  
```

Faked vtable

```
0x8000000000003620 0x8000000000009c0  
0x8000000000000001 0x8000000000000800  
0x80000000000027b5c1 0x8000000000000a23  
0x8000000000009c0 0x8000000000000000  
0x8000000000000000 0x8000000000000000  
```

ROP Payload
(dispensed as serialized Manifest object)

---

Thread 2 "IO svc #0" hit Breakpoint 1, 0x8000000001604151 in zipline::PeerImp::onMessage(std::shared_ptr<IMPpeer::SharedInfoV2> const&) (gdb) x/4 rip
rax => 0x80483d3c - __KTask7::KTask7::PeerImp::onMessage(EMSHaredInfoV2::ptr_t)IMPpeer::SharedInfoV2EE14:
```callq*0x8640(rax)``

(gdb) x/16g rax
```0x77bce12400 0x77bce118ff 0x77bce119fd 0x77bce117de 0x77bce11461 0x77bce11445 0x77bce11465 0x77bce1170b 0x77bce117ab 0x77bce119e2 0x77bce1180f 0x77bce11e7b 0x77bce11e4a 0x77bce11e2e 0x77bce11e01 0x77bce11d8a 0x77bce11d5e 0x77bce11d2a 0x77bce11c98 0x77bce11c64 0x77bce11c30 0x77bce11b96 0x77bce11bf0 0x77bce11c74 0x77bce11c3c 0x77bce11b9c 0x77bce11b68 0x77bce11b34 0x77bce11b9a 0x77bce11bf8 0x77bce11c14 0x77bce11c70 0x77bce11c36 0x77bce11b9e 0x77bce11b6e 0x77bce11b3c 0x77bce11b08 0x77bce11bf4 0x77bce11c10 0x77bce11c76 0x77bce11c32 0x77bce11b9c 0x77bce11b68 0x77bce11b34 0x77bce11b9a 0x77bce11bf8 0x77bce11c14 0x77bce11c70 0x77bce11c36 0x77bce11b9e 0x77bce11b6e 0x77bce11b3c 0x77bce11b08 0x77bce11bf4 0x77bce11c10 0x77bce11c76 0x77bce11c32 0x77bce11b9c 0x77bce11b68 0x77bce11b34 0x77bce11b9a 0x77bce11bf8 0x77bce11c14 0x77bce11c70 0x77bce11c36 0x77bce11b9e 0x77bce11b6e 0x77bce11b3c 0x77bce11b08 0x77bce11bf4 0x77bce11c10 0x77bce11c76 0x77bce11c32 0x77bce11b9c 0x77bce11b68 0x77bce11b34 0x77bce11b9a 0x77bce11bf8 0x77bce11c14 0x77bce11c70 0x77bce11c36 0x77bce11b9e 0x77bce11b6e 0x77bce11b3c 0x77bce11b08 0x77bce11bf4 0x77bce11c10 0x77bce11c76 0x77bce11c32 0x77bce11b9c 0x77bce11b68 0xb774d77a4b50ff -echo Hacked > /opt/zipple/bin/hacked
```
### Exploiting Estimation

<table>
<thead>
<tr>
<th>For exploiting one victim node</th>
<th>Network traffic</th>
<th>1220MB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time cost</td>
<td>12minutes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>For exploiting the entire network (1000 victims)</th>
<th>Network traffic</th>
<th>1191GB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time cost</td>
<td>9 Days</td>
</tr>
</tbody>
</table>
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
The Ending

- 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

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