ReDMArk: Bypassing RDMA Security Mechanisms

Usenix Security 2021
Collaborators for this work

Benjamin Rothenberger  
PhD Student  
Netsec

Konstantin Taranov  
PhD Student  
SPCL

Adrian Perrig  
Professor  
Netsec

Torsten Hoefler  
Professor  
SPCL
Data Processing in Modern RDMA Networks

Remote Nodes (via network)

Local Node
Core i7 Haswell

RDMA NIC
DMA Unit

PCIe bus
~ 250ns

arriving packets

RDMA Processing

Local Node

Regs
4 cycles ~ 1.3ns
L1
11 cycles ~ 3.6ns
L2
34 cycles ~ 11.3ns
L3
125 cycles ~ 41.6ns

Input buffer
Main Memory

~ 250ns

RDMA is a Trending Topic in HPC and Cloud Systems

Vulnerabilities?  Exploits?  Mitigations?

designed for performance – lower latency, higher bandwidth, lower CPU utilization etc.
ReDMArk Overview

<table>
<thead>
<tr>
<th>4 adversary models</th>
<th>10 vulnerabilities</th>
<th>8 mitigations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ● enables attack
- ○ facilitates attack
- □ does not affect attack
- □ mitigates attack
- + increases attack complexity
- ✗ does not mitigate attack

Implemented 6 attacks
Adversary Model

(T1) An attacker with a normal end host
   - can connect to RDMA services
   - issue messages over these connections

(T2) An attacker with a compromised end host
   - fabricate and inject messages

(T3) An in-network attacker (e.g., malicious switch)
   - on-path
   - eavesdrop, modify

(T4) A malware-based attack
   - use RDMA for data exfiltration
     (e.g., as covert channel)
RDMA Write Packet Format and Packet Processing

<table>
<thead>
<tr>
<th>Routing Header (RH)</th>
<th>Base Transport Header (BTH)</th>
<th>RDMA Header</th>
<th>Payload</th>
<th>Checksums</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfiniBand: IB Routing Header</td>
<td>Queue Pair Number (QPN)</td>
<td>Target virtual address</td>
<td>Two integrity CRC checksums</td>
<td></td>
</tr>
<tr>
<td>RoCEv1: Ethernet + IB RH</td>
<td>Packet Sequence Number (PSN)</td>
<td>Memory key (rkey)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RoCEv2: Ethernet + UDP/IP</td>
<td>Type of RDMA request</td>
<td>Data Length</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- InfiniBand: IB Routing Header
- RoCEv1: Ethernet + IB RH
- RoCEv2: Ethernet + UDP/IP
RDMA Write Packet Format and Packet Processing

Routing Header (RH) | Base Transport Header (BTH) | RDMA Header | Payload | Checksums
---|---|---|---|---
InfiniBand: IB Routing Header | Queue Pair Number (QPN) | Target virtual address | Two integrity CRC checksums
RoCEv1: Ethernet + IB RH | Packet Sequence Number (PSN) | Memory key (rkey) | checksums
RoCEv2: Ethernet + UDP/IP | Type of RDMA request | Data Length

arriving packets

CRC check | QPN matching | PSN matching | rkey matching | virtual address translation

RDMA NIC

PCIe bus

Main Memory
RDMA Write Packet Format and Packet Processing

Routing Header (RH) | Base Transport Header (BTH) | RDMA Header | Payload | Checksums
---|---|---|---|---
InfiniBand: IB Routing Header | Queue Pair Number (QPN) | Target virtual address | Two integrity CRC checksums
RoCEv1: Ethernet + IB RH | Packet Sequence Number (PSN) | Memory key (rkey) |
RoCEv2: Ethernet + UDP/IP | Type of RDMA request | Data Length |

Silent to the application
- Corrupted packet
- Connection does not exist
- Out-of-order packet or duplicate

Connection error
- rkey does not exist
- Address does not belong to rkey

InfiniBand: IB Routing Header
RoCEv1: Ethernet + IB RH
RoCEv2: Ethernet + UDP/IP

Main Memory
PCle bus

arriving packets
Bypassing RDMA Processing Checks

Packet injection (e.g., using RDMA send)

Unauthorized access (e.g., using RDMA read)

 CRC check  QPN matching  PSN matching  rkey matching  virtual address translation

arriving packets

PCIe bus

Main Memory

RDMA NIC
Towards Packet Injection -- CRC Check

- **Observations**
  - Neither encryption nor authentication are used in today’s RDMA protocols
  - CRC checksums are used for packet integrity checks
    - *but have known seeds and polynomials*
    - *and can easily be computed by an adversary*
Towards Packet Injection -- QPN Matching

- **Observations**
  - Queue pair numbers are 24 bits (< 17M possible QPNs!)
  - In practice: they are allocated sequentially!
    - predicting preceding or subsequent QPNs is trivial

- **Device analysis**

<table>
<thead>
<tr>
<th>Model</th>
<th>Driver</th>
<th>Arch.</th>
<th>QPNs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcom NetXtreme-E BCM57414</td>
<td>bnxt_re</td>
<td>RoCEv2</td>
<td>sequential</td>
</tr>
<tr>
<td>Broadcom Stingray PS225 BCM58802</td>
<td>bnxt_re</td>
<td>RoCEv2</td>
<td>sequential</td>
</tr>
<tr>
<td>Mellanox ConnectX-3 MT27500</td>
<td>mlx_4</td>
<td>IB/RoCEv1</td>
<td>sequential</td>
</tr>
<tr>
<td>Mellanox ConnectX-4 MT27700</td>
<td>mlx_5</td>
<td>IB/RoCEv2</td>
<td>sequential</td>
</tr>
<tr>
<td>Mellanox ConnectX-5 MT27800</td>
<td>mlx_5</td>
<td>IB/RoCEv2</td>
<td>sequential</td>
</tr>
<tr>
<td>softRoCE</td>
<td>rxe</td>
<td>RoCEv2</td>
<td>sequential</td>
</tr>
</tbody>
</table>
Towards Packet Injection – PSN matching

- **Observations**
  - Packet Sequence Number (PSN) is also 24 bits
  - PSN can be selected by the entity that creates an RDMA connection

- **Connection establishment via IB verbs versus RDMA connection manager**
  - RDMA connection manager assigns a random PSN to the connection
  - Establishing a connection using InfiniBand verbs leaves the option to the developer
  - Most analysed open-source RDMA systems tend to use IB verbs with a static PSN (simplicity?)

<table>
<thead>
<tr>
<th>System</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octopus [21]</td>
<td>Native</td>
</tr>
<tr>
<td>HERD [12]</td>
<td>Native</td>
</tr>
<tr>
<td>RamCloud [25]</td>
<td>Native</td>
</tr>
<tr>
<td>Dare [28]</td>
<td>Native</td>
</tr>
<tr>
<td>Crail [30, 31]</td>
<td>Manager</td>
</tr>
</tbody>
</table>
Towards Packet Injection

**Approach**
- Bypassing the first three checks allows us to inject RDMA send packets (no RDMA header)
- Our PoC injection tool can inject up to 1.6 Mpps (Mellanox ConnectX-5)
  - takes roughly 11s to enumerate the full 24 bit PSN

**Observations**
- Injecting RDMA packets with invalid PSN does not break the connection
- Duplicate packets are dropped (and acknowledged)
  - “silent” packet replacement is possible!
- Injecting $2^{24}$ packets makes PSN counter wrap and can hide the attack from the application
Misuse Packet Injection for Denial-of-Service

- **Approach**
  - Packets that passed the first three checks but contain protocol errors can force the QP into an error state \(\rightarrow\) breaks the QP connection!
  - Our tool can inject up 1.6 Mpps
    - *known PSN:* we can scan 1.6 M connections per second (and disconnect!)
    - *unknown PSN:* enumerate a full PSN in ~ 11s (QPN is sequential)

- **Observations**
  - QPN randomness is crucial to increase the attack complexity for packet injection
  - Example: victim with 1,000 RDMA connections with a random QPN, our tool is expected to break one of the connections in 48h
Towards Unauthorized Access – Guessing rkeys

- **Observations**
  - rkeys are used as 32 bit access control tokens
  - but: the rkey generation is highly predictable (less than 3 bits of entropy!)

- **Other problems**
  - Static initialization for key generation: the NIC generates the same keys after a reboot
  - Same constant protection domain for all QPs: allows to access memory even without impersonation using any QP connection
  - Shared key generator state: applications use the same network interface even if they use different protection domains
Towards Unauthorized Access – Guessing Addresses

- **Virtual addresses are 64 bits**
  - Linux typically only uses 48 bits
  - Developers tend to use page-aligned memory for performance → 36 bits!

- **Consecutive allocation of memory regions**
  - Subsequent objects in memory are allocated in consecutive addresses with respect to a random address base

- Example: InfiniSwap*
  
  Infiniswap is a remote swapping device for Linux
  
  Uses posix_memalign in a loop to allocate register buffers of 1GB
  
  Allows an attacker to predict the position of a newly allocated buffer

Implementing Unauthorized Memory Access

**Approach**

- An attacker (M) can connect to a RDMA service to get an address and a rkey of its communication buffer
- Assuming the buffers are allocated sequentially, the attacker can guess addresses and rkeys of other buffers on the service
- All 6 analysed open-source RDMA systems were vulnerable to this attack
- Attack is even simpler for an in-network attacker (eavesdrop rkey and buffer addresses)

Figure 4: Unauthorized memory access on the same host.
Mitigation Mechanisms
Prevent Packet Injection and Packet Alteration

- Use a secure transport with authentication
  - IPSec for RoCE (e.g., Mellanox Connect-X 6 DX)
  - sRDMA* for InfiniBand and RoCE

![Diagram of IPSec and sRDMA headers]

QPN & PSN Randomization

- **We propose a software-based algorithm for QPN randomization in the paper**
  - Approach: RDMA allows creation of connection stubs that get a QPN assigned without actually connecting to a RDMA service

- **PSN randomization**
  - Use RDMA CM → randomly generates a PSN
  - But: RDMA CM exchanges connection parameters in plaintext
  - Solution: Native IB verbs interface with a random PSN
Rkey Randomization

- We propose a software-based Rkey randomization algorithm for short-term mitigation

- Use multiple Protection Domains (PDs)
  - share PDs between trusted connections

- Use Memory Windows Type 2
  - can be pinned to a specific QPN

- sRDMA proposes crypto-based memory protection
Additional Content in the Paper

- **Additional attacks**
  - QP exhaustion
  - Performance degradation
  - Using RDMA for data exfiltration

- **Mitigations**
  - Short-term and long-term mitigation mechanisms
  - Example: software-based algorithms for QPN and rkey randomization
Thank you for your attention!

- ReDMArk provides an in-depth analysis of current RDMA security
- We discovered 10 vulnerabilities / design flaws
- We implemented 6 attacks under 4 different threat models
- We tested 6 open-source systems
- We propose 8 mitigation techniques

ReDMArk implementation: