ReDMArk: Bypassing RDMA Security Mechanisms

Benjamin Rothenberger[*], Konstantin Taranov[*], Adrian Perrig, and Torsten Hoefler
Department of Computer Science, ETH Zurich

Abstract

State-of-the-art remote direct memory access (RDMA) technologies such as InfiniBand (IB) or RDMA over Converged Ethernet (RoCE) are becoming widely used in data center applications and are gaining traction in cloud environments. Hence, the security of RDMA architectures is crucial, yet potential security implications of using RDMA communication remain largely unstudied. ReDMArk shows that current security mechanisms of IB-based architectures are insufficient against both in-network attackers and attackers located on end hosts, thus affecting not only secrecy, but also integrity of RDMA applications. We demonstrate multiple vulnerabilities in the design of IB-based architectures and implementations of RDMA-capable network interface cards (RNICs) and exploit those vulnerabilities to enable powerful attacks such as packet injection using impersonation, unauthorized memory access, and Denial-of-Service (DoS) attacks. To thwart the discovered attacks we propose multiple mitigation mechanisms that are deployable in current RDMA networks.

1 Introduction

In recent years, numerous state-of-the-art systems started to leverage remote direct memory access (RDMA) primitives as a communication mechanism that enables high performance guarantees and resource utilization. Deployments in public clouds, such as Microsoft Azure and IBM Cloud, are becoming available and an increasing number of systems make use of RDMA for high-performance communication [8,11,18,28]. However, the design of RDMA architectures is mainly focused on performance rather than security. Despite the trend of using RDMA, potential security implications and dangers that might be involved with using RDMA communication in upper layer protocols remain largely unstudied. For example, RFC 5042 [30] analyzes basic security issues and potential attacks in RDMA-based implementations, but lacks an in-depth analysis of state-of-the-art RDMA architectures and implementations.

Current RDMA technologies include multiple plaintext access tokens to enforce isolation and prevent unauthorized access to system memory. As these tokens are transmitted in plaintext, any entity that obtains or guesses them can read and write memory locations that have been exposed by using RDMA on *any machine* in the network, compromising not only secrecy but also integrity of applications. To avoid compromise of these access tokens, RDMA architectures rely on isolation and the assumption that the underlying network is a well-protected resource. Otherwise, an attacker that is located on the path between two communicating parties (e.g., bugged wire or malicious switch) can eavesdrop on access tokens of bypassing packets.

Unfortunately, encryption and authentication of RDMA packets (e.g., as proposed by Taranov et al. [36]) is not part of current RDMA specifications. While IPsec transport recently became available for RoCE traffic, the IPsec standard does not support InfiniBand traffic. Furthermore, application-level encryption (e.g., based on TLS) is not possible since RDMA operations can be handled without involvement of the CPU. As TLS cannot support purely one-sided communication routines, the applications would need to store packets in a buffer before decryption, completely negating RDMA's performance advantages. We discuss these potential mitigation techniques to secure RDMA in more detail in §7.3.

In this work, we analyze current security mechanisms of RDMA architectures based on InfiniBand (IB) such as native InfiniBand and RDMA over converged Ethernet (RoCE) versions 1 and 2. ReDMArk reveals multiple vulnerabilities and flaws in the design of InfiniBand, but also in implementations of several RDMA-capable network interface cards (RNICs) by Mellanox and Broadcom. These vulnerabilities enable powerful attacks on RDMA networks, such as unauthorized memory access or breaking of existing connections based on packet injection. To show the feasibility of the discovered attacks in practice, we implemented an attack framework, that is able to inject bogus packets into the network and impersonate other endpoints to corrupt the memory state of remote endpoints. For each of the discovered attacks, we discuss potential

[*]These authors contributed equally to this work.

long-term mitigation mechanisms. In addition, we propose short-term mitigations that can be deployed in today's RDMA networks before the long-term mitigations become available. Finally, we assess the vulnerability of open-source systems that rely on RDMA for high-performance communication against the discovered attacks.

2 Remote Direct Memory Access

RDMA enables direct data access on remote machines across a network. Memory accesses are offloaded to dedicated hardware and can be processed without involvement of the CPU (and context switches). Using RDMA read and write requests application data is read from/written to a remote memory address and directly delivered to the network, reducing latency and enabling fast message transfer. RDMA can also enable one-sided operations, where the CPU at the target node is not notified of incoming RDMA requests.

Even though several network architectures support RDMA, in this work we focus on the most widely used interconnects for RDMA: InfiniBand (IB) [3] and RDMA over Converged Ethernet (RoCE) [4]. InfiniBand is a network architecture specifically designed to enable reliable RDMA and defines its own hardware and protocol specification. RoCE is an extension to Ethernet to enable RDMA over an Ethernet network and exists in two versions. RoCEv1 uses the IB routing header, whereas RoCEv2 uses UDP/IP for routing. Even though this work focuses on IBA and RoCE, the proposed attacks could also be extended to other RDMA architectures.

2.1 RDMA packet format

The RDMA packet header consists of a routing header and a base transport header (see Figure 1). The routing header contains the source and destination ports, that identify link layer endpoints. The IB protocol uses the IB link layer protocol as a data link, whereas RoCE relies on Ethernet. RoCEv1 encapsulates an IB packet, including its IB routing header, into an Ethernet frame. RoCEv2 is designed as an Internet layer protocol and uses a UDP/IP header for routing.

All data communication in RDMA is based on *queue pair (QP)* connections between the two communicating parties. QPs are a bi-directional message transport mechanism used to send and receive data in InfiniBand. Endpoints in RDMA are identified by the combination of an adapter port address and a queue pair number (QPN), a unique identifier of a QP connection within destination port. For all QP endpoints at a destination port, the RNIC generates a unique QPN.

2.2 InfiniBand Architecture Security Model

Processing of incoming packets is based on the base transport header that contains the destination QPN and also a packet sequence number (PSN). The PSN is used to enforce in-order delivery and detect duplicate or lost packets. Packets with invalid QPN or PSN are dropped without any notification to the receiving application.

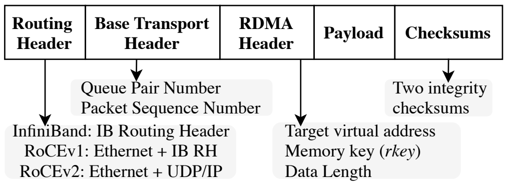

Figure 1: General format of an RDMA packet.

To detect errors that may have been introduced during the transmission, each packet contains two checksums that are checked by the receiving node. The checksum algorithms are defined in the IB specification and use pre-defined seeds.

In addition to packet integrity checks, IBA defines three memory protection mechanisms to restrict unauthorized access to local memory by remote entities: Memory Regions, Memory Windows, and Protection Domains (PD) [37]. These mechanisms allow enforcing memory access restrictions (e.g., the application allows reads, but no writes to a memory region).

Memory Regions. To access host memory, the RNIC first allocates the memory region, which involves copying page table entries of the corresponding memory to the memory management unit of the RNIC. Then, the RNIC creates a memory region to enforce access restrictions to the memory such as read-only, write-only, or local-only. Memory regions can also be reregistered to change its properties or deregistered to destroy its memory mappings.

For each memory region RNIC generates two keys for local and remote access, namely *lkey* and *rkey*. To remotely access a memory location using RDMA read or write operations, each packet must include a virtual address and its associated *rkey* as depicted in Figure 1. The *rkey*s are not used in any form of cryptographic computation, but used as access tokens that are transmitted in plaintext. The *lkey*s are not part of the transport protocol, but used as a local authorization token allowing the channel adapter to access local memory of an application.

Memory Windows. To allow different access rights among remote QPs within a memory region or grant access to a part of the region, IBA makes use of *Memory windows type 1*. *Memory windows type 2* further extend this protection mechanism by assigning a single QP to a memory window and enforcing that only the assigned QP can access it.

Protection Domain. IBA protection domains (PD) group IB resources such as QP connections and memory regions, such that QP connections within a PD can only access memory regions allocated in the same PD, providing protection from unauthorized or inadvertent use of a memory area. All QPs and memory regions are always assigned to a specific PD and can only be a member of one PD.

3 Adversary Model

In our adversary model we consider three parties (see Figure 2): an RDMA service which hosts one or several RDMA applications, a client who interacts with the service through RDMA, and an adversary who can legitimately connect to the RDMA service, but tries to violate RDMA's security mechanisms (e.g., access memory of other clients using RDMA).

We assume that the adversary is located within the same network as the other parties and consider four different attacker models.

Model T1. First, we consider an adversary that is located at a different end host than the victim (off-path) and have rightfully obtained these hosts (e.g., by renting an instance in a public cloud). This attacker cannot conduct any network-based attacks such as packet injection, but can connect to RDMA services and issue RDMA messages over these connections.

Model T2. Second, we consider attackers (potentially off-path) that can actively compromise end hosts and *fabricate* and *inject* messages. To successfully inject an arbitrary RDMA request the adversary must have root administrative access. The adversary is required to know the host's address (local identifier for IBA / IP address for RoCE), QP numbers, and the PSN to forge a valid RDMA packet. Additionally, to read or write a memory location on the remote host, the adversary needs to include a valid virtual memory address and the corresponding memory protection key *rkey*.

Model T3. Third, we consider network-based attackers where the attacker is located on the path between the victim and the service. On-path attacks require the attacker to control routers or links between the victims (e.g., rogue cloud provider, rogue administrator, malicious bump-in-the-wire device). A network-based attacker can passively *eavesdrop* on messages, but also actively tamper with the communication between hosts by injecting, dropping, delaying, replaying, or altering messages. This includes *altering* and forging any information in *any* packet header, including all IB and Ethernet headers. Since RDMA communication is in plaintext and the IB protocol does not provide any mechanisms for authenticating a message to prevent on-path packet alteration, this only requires recalculation of packet checksums, whose algorithms and seeds are publicly available in the IBA specification.

Model T4. Finally, we consider an adversary that makes use of RDMA as a covert channel for exfiltrating data. For this purpose, the adversary manipulates code or libraries executed by the victim (e.g., using malware) such that it establishes an RDMA connection to an RDMA-capable attacker machine in the same network as the victim (e.g., by renting an instance in a public cluster). This allows the adversary to exploit one-sided RDMA operation to "silently" access memory of the victim process.

Since both the network and control over a machine are well protected resources in cloud datacenters, we assume that these potential attack locations are much harder to achieve than obtaining or compromising an arbitrary end-host.

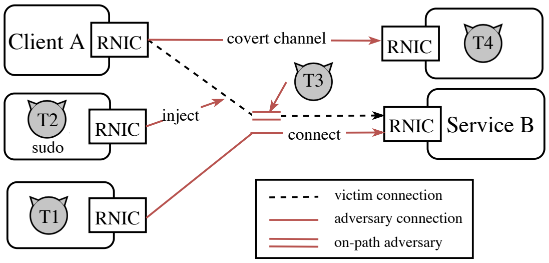

Figure 2: Illustration of the adversary model including potential adversary locations.

4 Security Analysis of IB Architectures

Given the aforementioned adversary model, we analyse existing security mechanisms in IB-based architectures including memory protection key generation, QP number generation, memory regions, memory windows, and protection domains. We identify 10 vulnerabilities, labeled V1 – V10 .

4.1 Analysis Setup

Our analysis setup includes multiple IB-based architectures such as native InfiniBand (IBA), RoCEv1, and RoCEv2. To execute and evaluate our tests we use a server cluster with RNICs from Broadcom, Mellanox, and also run tests on Microsoft Azure HPC instances that support RDMA (A8, A9, H16r). Additionally, we consider software-based RoCE (softRoCE), a software implementation of RoCE that has been integrated into the Linux kernel [21]. Table 1 lists the analyzed devices and summarizes the discovered memory protection issues.

4.2 Memory Protection Keys

V1 Memory Protection Key Randomness. To protect remote memory against unauthorized memory access, IBA requires that RDMA read/write requests include a remote memory access key *rkey*, which is negotiated between communicating peers and is checked at the remote RNIC. Packets with an invalid *rkey* cause a connection error leading to disconnection. The requirement of including an *rkey* is built into the silicon and the driver code cannot be disabled by an attacker. Thus, to successfully circumvent this protection mechanism against unauthorized memory access, an attacker needs to include a valid *rkey* in his requests.

We analyze the randomness of the *rkey* generation process for different RNIC models and drivers. For all tested devices, *rkey* generation is independent of the address and length of the buffer to be registered. Changes in access flags have no influence on the generation of an *rkey*. The generated *rkey*s

Table 1: Summary of Memory protection issues across different IBA drivers.

Model	Driver	Arch.	Static Init.	Shared Gen.	Key Step	QPNs	QP limit[d]
Broadcom NetXtreme-E BCM57414	bnxt_re	RoCEv2	✓	✓	`0x100`	sequential	32,707
Broadcom Stingray PS225 BCM58802	bnxt_re	RoCEv2	✓	✓	`0x100`	sequential	61,438
Mellanox ConnectX-3 MT27500	mlx4	IB/RoCEv1	✓	✓	`0x100`[a]	sequential	261,359
Mellanox ConnectX-4 MT27700	mlx5	IB/RoCEv2	✗	✓	random[b]	sequential	64,443
Mellanox ConnectX-5 MT27800	mlx5	IB/RoCEv2	✗	✓	random[b]	sequential	65,449
Mellanox ConnectX-6 Dx MT28841	mlx5	RoCEv2	✗	✓	random[b]	sequential	262,100
softRoCE	rxe	RoCEv2	✓	✓	`0x100` + lfsr-8bit[c]	sequential	32,707

[a] for a subsequent registrations [b] has low entropy [c] seed and states are known [d] bound by the OS limit on active file descriptors

only depend on previous registration/deregistration operations. Further, we investigate how registration/deregistration affects memory registration.

RNIC models from Broadcom (using the bnxt_re driver) always increase the *rkey* value by `0x100` independent of the previously mentioned factors. The exact algorithm can be found in Appendix A. Thus, assuming the attacker is able to obtain an *rkey* that is part of this series of increasing key values, predicting preceding or subsequent *rkey*s is trivial.

For devices based on the mlx4 driver, the sequence of *rkey*s depends on registration/deregistration operations. For consecutive registration operations each *rkey* gets incremented by `0x100`. However, after a deregistration operation, the next *rkey* gets incremented by `0x80000` based on the *rkey* for the memory region that has been deregistered. In case of multiple consecutive deregistration operations, the *rkey*s of the deregistered memory regions are queued and for each upcoming registration operation a key gets dequeued. The algorithm for key generation can be found in Appendix A. All tested Azure HPC instances (A8, A9, H16r) use the mlx4 driver and allocate *rkey*s with the previously described algorithm.

The software implementation of RoCE, SoftRoCE, also increases the *rkey* by `0x100` for each registration operation, but additionally randomizes the last 8 bit using a linear-feedback shift register (LFSR). However, since LFSRs are deterministic and the initial seed is known, all subsequent states are easily computable. Moreover, the LFSR implementation used by SoftRoCE generates only 15 distinct numbers, which does not increase the randomness of *rkey*s. The full algorithm for key generation can be found in Appendix A.

Devices based on the mlx5 driver do not use a fixed increase between subsequent registrations, but still strictly increase the values with a random value (modulo 2^{32}). An analysis of these values shows that with more than 60% probability either the value `0x101` or `0x102` (see Figure 3) is chosen. Thus, even though the key generation process of devices based on mlx5 driver contains higher entropy than other drivers, the sequence of generated keys is still predictable by an adversary with moderate effort.

Key Entropy Analysis. Given that the *rkey* generation process for all tested drivers seems predictable, we further quantify the randomness of the key generation process by calculating the *min-entropy* [5, 24], which denotes a measure to describe the uncertainty associated with a random variable by guessing the key until a correct key is found. Thus, the min-entropy measures the difficulty of guessing the most likely output of an entropy source. Following, the optimal strategy for successive guessing is to try all possible values in order of decreasing probability.

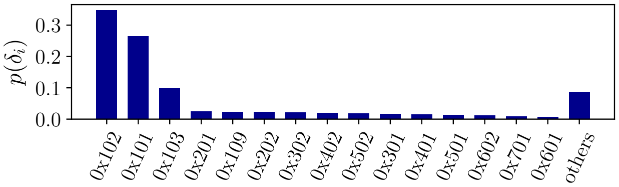

Figure 3: Probabilities of differences for random *rkey* value generated using mlx5 device.

If we consider this problem of guessing a discrete random variable X on $x_1, x_2, \ldots, x_m$ with probability distribution $P = (p_1, p_2, \ldots, p_m)$, where $p_i = P(X = x_i)$, $1 \leq i \leq m$. We assume that $p_1 \geq p_2 \geq \ldots \geq p_m$. Then the min-entropy of X is defined as

$$H_\infty(X) = \min_{1 \leq i \leq m} (-\log_2 \; p_i) = -\log_2 \max_{1 \leq i \leq m} p_i = -log_2 \; p_1 \tag{1}$$

Our observations showed that for all tested drivers the sequence of generated *rkey*s was strictly increasing (modulo 2^{32}). Thus, we define the dependency of a newly generated key on the previous key as follows:

$$x_{i+1} = x_i + \delta_i \tag{2}$$

where δ_i denotes the difference between key x_i and x_{i+1} and is further described using a discrete random variable Δ with probability density function $P(\Delta = \delta_i)$. Thus, given key x_i the generation of key x_{i+1} is dependent on the randomness of Δ and quantified by the min-entropy of Δ (see Table 2).

Table 2: Entropy of rkey generation for key differences

Driver	$H_\infty(\Delta)$	$H(Y\|X)$
bnxt_re	0	0
mlx4	0.14	1
mlx5	2.16	2.85
rxe	2.04	0

This dependency can further be generalized by calculating the *conditional entropy* [23] of subsequent key generations, which quantifies the amount of information needed to predict a newly generated key x_{i+1} given the previous key x_i (e.g., if an attacker obtains a key legitimately by registering a memory region) and is defined as:

$$H(Y|X) = -\sum_{x\in X, y\in Y} p(x,y)\ \log_2 \frac{p(x,y)}{p(x)} \tag{3}$$

where $p(x,y)$ denotes the joint probability of x and y.

For bnxt_re and rxe we can always predict which *rkey* will be generated next, making the value of x_{i+1} completely determined by x_i, which results in the conditional entropy being equal to 0 [23]. For mlx4, the value of Y only depends on whether a region has been deregistered before the next *rkey* is generated. Assuming that this occurs with probability 0.5, the conditional entropy is 1 bit. Finally, for mlx5 the distribution of key differences Δ is illustrated in Figure 3. If an attacker guesses that the next key is incremented by the difference with the highest probability, his guess would be correct in one out of three guesses on average. The computation of the conditional entropy of mlx5 results in 2.85 bits of entropy, which enables an attacker to guess the *rkey* of future registrations with high probability.

V2 **Static Initialization State for Key Generation.** In addition to the limited number of *rkey*s, the RNICs based on the bnxt_re and mlx4 drivers are initialized using static state and the same set of keys persists across different physical reboots of the machine. Assuming that an adversary has observed the entire key set, the same keys will be reused even after the physical machine rebooted.

Since the IB network adapter on Azure instances is virtualized and a reboot of the instance does not lead to physical reboot of the machine, the tested Azure instances were *not* affected by static initialization.

V3 **Shared Key Generator.** On all tested devices the key generator is *fully shared* between applications using the same network interface even if they use different protection domains. Thus, if multiple RDMA applications are running on the same service the prediction of *rkey*s of other applications based on own *rkey*s is trivial as they have been generated using the same key generator.

In addition to enabling memory key prediction across multiple application, this vulnerability can also be exploited to open a side-channel between applications sharing the same RNIC, e.g., by encoding a bit-stream in the number of registrations they perform per a time unit. This is especially critical if an adversary is located on the same physical host as the victim (e.g., two VMs on the same physical host in a public cloud environment).

4.3 Memory Allocation Randomness

V4 **Consecutive Allocation of Memory Regions.** In addition to the *rkey* associated to a memory location, the adversary is also required to predict the corresponding memory address. Typically, techniques such as address space layout randomization (ASLR) randomly arrange the address space positions of a process. This prevents an attackers from directly referring to other objects in memory by randomizing their locations. However, subsequent objects in memory are allocated in consecutive addresses with respect to a random address base [40]. For example, all objects allocated via the `mmap()` Linux system call are placed side by side in the mmap area.

Since RDMA-based applications run in a single process on the target host, they are not protected by ASLR, but instead objects in memory are allocated side-by-side. Assuming an attacker knows the address of a memory object on a target host, predicting the memory address of other objects is possible. Even though consecutive allocation of memory regions is not caused by RDMA protocols, it still affects the security of RDMA applications.

4.4 QP Number Identifiers & Packet Sequence Numbers

V5 **Linearly Increasing QP Numbers.** Our evaluation (see Table 1) shows that for all tested devices and drivers the QP numbers are allocated sequentially. Assuming that an adversary registers a QP himself or observes a QP registration request, predicting preceding or subsequent QP numbers is trivial. Furthermore, as IBA uses 24 bit QP numbers, it is not possible to establish more than 2^{24} QP connections within an RNIC.

V6 **Fixed Starting Packet Sequence Number (PSN).** The implementation of RDMA offers two ways of establishing RDMA connections: a native RDMA connection interface or using the RDMA connection manager [20] to establish connections. Using the native connection interface, the connection parameters, such as destination QP number, local and remote starting PSNs, are set by the application developers. The RDMA connection manager moves this burden away from application developers and randomly generates a starting PSN (using a cryptographic pseudorandom number generator), thereby making the process of RDMA connection establishment similar to TCP sockets. Our analysis (see §6) shows that many RDMA-based open-source applications opt for using the native interface and manually set the starting

PSNs. In case the starting PSNs are not randomized on a per QP connection basis, predicting PSNs of established connections becomes much simpler.

4.5 Other Security Weaknesses in IBA/RoCE

Furthermore, we describe four security weaknesses that greatly enable or facilitate attacks on RDMA applications.

V7 **Limited Attack Detection Capabilities.** RDMA allows one machine to directly access data on remote machines across the network. Due to network offloading of one-sided RDMA operations, all memory accesses are performed using dedicated hardware on RNIC without any CPU interaction. This makes memory accesses completely invisible to applications and limits their capabilities of detecting attacks.

V8 **Missing Encryption and Authentication in RDMA Protocols.** Existing RDMA network protocols do not provide any mechanisms for authentication nor encryption of the header and the payload of RDMA packets. An adversary can spoof any field in the packet header or alter any byte in the packet payload of RDMA messages. In-network packet alteration only requires recalculation of packet checksums, whose algorithms and seeds are known and specified by the IBA. Potential solutions for encryption and authentication in IBA are further discussed in §7.3.

V9 **Single Protection Domain for all QPs.** To reduce the state overhead on RNICs the RDMA connection manager by default uses a single protection domain for all established QPs and memory registrations within a single process. As a result, all QPs of a single process can access memory of each other. Nonetheless, even developers using the native connection interface seem to opt for using a single PD for all its QPs and memory registrations (see §6).

V10 **Implicit On-Demand Paging (ODP).** Implicit On-Demand Paging (ODP) enables a process to register its complete memory address space for I/O accesses. This feature is used for high-performance communication settings, where the overhead of frequently registering communication buffers leads to performance degradation. ODP removes the need to register memory as any memory address can be registered by RNIC on demand. If ODP is enabled, an attacker can remotely access the entire memory space of a process resulting in high attack potential. While this feature is disabled by default, recent advances in high-performance communication systems lead to this feature gaining traction in IB deployments [19].

5 Attacks on IBA

Using the discovered vulnerabilities, an adversary could launch attacks in RDMA networks, e.g., by using unauthorized access to memory regions or by disrupting communication using DoS. Furthermore, vulnerabilities in RDMA could also be misused as an attack vector for application-level attacks (e.g., malware). In this section, we describe six potential attacks on RDMA networks, labeled A1–A6. For each of the attacks we explain the experiments we conducted and discuss potential mitigation mechanisms, which are discussed in greater detail in §7. Table 3 outlines the dependency on attacker locations and vulnerabilities, and potential mitigation mechanisms for each of the attacks.

5.1 A1: Packet Injection using Impersonation

As current RDMA systems enforce no source authentication V8, an adversary can impersonate any other endpoint and inject packets that seem to belong to an established connection by another client. To inject an RDMA packet that is considered valid by the receiving endpoint, the adversary needs to know the QPN of the victim and the current PSN.

Apart from obtaining these parameters by on-path eavesdropping or impersonation of end hosts, an attacker could try to predict them. Given that QP numbers are generated sequentially for each new client V5, an attacker can obtain expected QPNs of clients by simply connecting to the RDMA-enabled service and decrement the QPN that gets assigned to the attacker. Thus, a valid PSN remains the only protection mechanism that prevents an attacker from injecting a packet. If an application also does not generate starting PSNs randomly, the attacker can start bruteforcing PSN by exploiting the fixed starting PSN issue V6, which significantly reduces the search space. Otherwise, even with random PSNs, the attacker can bruteforce the PSN within a reasonable amount of time as only 2^{23} packets are required on average to generate a valid PSN. Bruteforcing the current PSN of a QP connection is related to enumerating the sequence number of a TCP connection [41], with the main difference that packets with invalid PSN simply get discarded by the RNIC and do not affect the established QP connection.

In addition to regular RDMA packets, injection of RDMA read and write packets additionally requires the attacker to know a valid memory address and its corresponding memory protection key *rkey*. This attack is further discussed in §5.3.

In the remainder of this work, attacks based on impersonation are referenced to as A1.

Experiments. To verify the feasibility of the attacks on RDMA protocols, we implemented a spoofing tool for RoCE[*]. The tool can fabricate any custom RoCEv1 and RoCEv2 packet including RDMA read and write operations and is fully compatible with the IBA specification [3]. Our RoCEv1 implementation uses Linux raw Ethernet sockets, and RoCEv2 uses IPv4 raw sockets and UDP as a transport layer. The tool can mimic any RDMA request initiator and inject custom RDMA packets over any Ethernet links. RDMA over IB link cannot be fabricated in software, as the IB protocol is implemented fully in hardware. The tool has been tested on all RoCE devices listed in the Table 1.

[*] https://github.com/spcl/redmark

Table 3: Overview of dependency on attacker location and vulnerabilities for attacks on RDMA combined with an overview of mitigation mechanisms that thwart the attack.

	T1	T2	T3	T4	V1 weak rkey	V2 static init.	V3 shared key gen.	V4 weak mem. rand.	V5 lin. inc. QPN	V6 fixed starting PSN	V7 lim. attack detect.	V8 no enc./auth.	V9 single PD	V10 ODP enabled	M1 rand. QPN	M2 rand. rkey	M3 HW counters	M4 mem. win. type 2	M5 multiple PDs	M6 enc./auth. in IB	M7 resource const.	M8 in-network filt.
A1	◑	●	●	○	○	○	○	○	◑	◑	○	●	○	○	✛	✗	✗	✗	✗	✓	✗	✓
A2	◑	●	●	○	○	○	○	○	◑	◑	○	●	○	○	✛	✗	✗	✗	✗	✓	✗	✓
A3	●	◑	●	○	●	◑	◑	●	○	○	●	◑	●	◑	✗	✓	✗	✓	✓	✗	✗	✗
A4	●	○	○	○	○	○	○	○	●	○	○	○	○	○	✗	✗	✗	✗	✗	✗	✛	✗
A5	●	◑	●	○	○	○	○	○	○	○	●	○	○	○	✗	✗	✛	✗	✗	✗	✛	✗
A6	○	○	○	●	○	○	○	○	○	○	●	○	○	◑	✗	✗	✛	✗	✗	✗	✗	✗

● enables attack ◑ facilitates attack ○ does not affect attack
✓ mitigates attack ✛ increases attack complexity ✗ does not mitigate attack

In these experiments (see Table 4), we measure the injection throughput of our RoCE spoofing tool, which allows an estimation of the time required to bruteforce a random PSN and a PSN that has been generated based on a known initialization value. The injection tool for RoCEv1 was able generate 1.30 millions packets per second (Mpps), whereas the tool for RoCEv2 was able to generate 0.74 Mpps for Broadcom and 1.57 Mpps for ConnectX-5. Full enumeration of a random PSN thus took 13 s for RoCEv1. RoCEv2 enumeration tool takes 10.60 s for Mellanox and 23 s for Broadcom.

The performance of ReDMArk's spoofing and packet injection framework could not achieve line-rate for the tested devices as: 1) the packet checksums are calculated by the CPU and not offloaded to the NIC. 2) our framework does not bypass the OS, whereas native RDMA messages do. To improve the performance of packet injection and thus exhausting this bruteforce search even faster, a hardware appliance could be used. Furthermore, specific Mellanox NICs support raw Ethernet programming with a kernel bypass [2].

Table 4: Injection throughput of the RoCE spoofing tool.

Model	Link speed	Protocol	Throughput
Mellanox ConnectX-3	40 Gbps	RoCEv1	1.31 Mpps
Mellanox ConnectX-5	100 Gbps	RoCEv2	1.57 Mpps
Broadcom NetXtreme-E	25 Gbps	RoCEv2	0.74 Mpps
Broadcom Stingray	25 Gbps	RoCEv2	0.74 Mpps

Interestingly, injection of a single correct packet does not cause a victim's connection to break. A mismatch in the PSN counter by 1 packet between sender and receiver is resolved by the protocol. The protocol treats the victim's packet as a repeated packet and always acknowledges it without processing. The sender receives the acknowledgment about successful transmission, even though the packet has not been processed by the remote RNIC. As a result, the attacker is able to replace the victim's packet with a forged one.

Injection of multiple valid packets causes a connection loss on the victim side. The QP of the victim who has been impersonated by the adversary experiences a "Transport Retry Counter Exceeded" error and transitions its connection to an error state, when it tries to send packets. However, the other endpoint of the QP only transitions into error state if it tries to reply to the disconnected victim QP side. Otherwise, the connection remains open. Moreover, since packet injection increases the PSN counter on the receiver, the victim's packets will get discarded due to a PSN mismatch. This effectively prevents the victim from closing the connection and allows the attacker to inject messages over an extended period.

Furthermore, our experiments showed that if the attacker injects exactly 2^{24} packets (i.e., the size of the PSN counter), then the injection remains completely unnoticed by the victim and it can continue communication due to matching PSN counters. Our injection tool requires approximately 13 s to inject 2^{24} packets. Therefore, if the victim does not use its connection for this amount of time, the attacker can successfully inject 2^{24} packets without disrupting the connection of the victim.

Practicality. Packet injection based on impersonation can be performed under the assumption of the T2 model, i.e., the attacker requires root access to any machine in the same network as the victim. Similarly, an on-path attacker T3 (e.g., bump-in-the-wire) also has packet injection capabilities. Since root access to machines in public cloud environments is a well protected resource (and would require a sandbox escape [27]), this attack is more realistic in the setting of small cloud providers or private RDMA cluster as used by companies and research groups.

Mitigation. To effectively mitigate attacks based on impersonation, source authentication could be deployed. However, since this is on-going research and not yet available for IB-based RDMA deployments, we suggest the following mitigations to increase the complexity of packet injection by an off-path attacker: each QP connection should be initialized with a random starting PSN instead of using a per-device starting PSN. As this only marginally increases the attack complexity also QPN should be randomly assigned to QPs. We suggest a mechanism for randomizing QPNs for existing RDMA deployments and that can be deployed for all vendors of RNICs in §7.1. As modern RNICs provide hardware counters that are accessible by the application (see §7.2), these counters should be used to detect bruteforcing attempts. Furthermore, operators of RDMA networks could also perform ingress filtering for all end hosts (see §7.4).

5.2 A2: DoS Attack by Transiting QPs to an Error State

In IB-based architectures, connections based on the RC QPs are sensitive to content of the header of requests. Protocol errors, such as inconsistencies in the sequence number or QP number, are recoverable errors and resolved by the protocol. However, memory errors, such as incorrect operation numbers or an inconsistency between payload length and DMA length immediately leads to unrecoverable errors, which will cause the RNIC to transit the QP to the error state and the QP to disconnect [3]. We refer to this attack as A2.

An on-path attacker T3 can trivially modify the operation numbers or payload lengths to drop connections. However, even an off-path adversary can inject incorrect packets towards a victim QP endpoint and effectively disrupt communication of other entities (see §5.1).

Experiments. To conduct an attack that transits a QP to an error state, we use the packet injection tool with the goal to inject invalid packets into the victim's QP connection such that it triggers an unrecoverable error, which results in the QP being forced to disconnect. These experiments showed that a single fabricated packet injected into the connection was sufficient to effectively break the victim's connection.

Given these insights, an attacker could try to repeatedly drop the connection of a specific client or drop all connections of clients that are trying to connect to a service. The fact that QPNs are generated sequentially V6 highly facilitates the realization of such attacks. In addition, if an application uses non-random starting PSNs V7, the attacker needs to guess only small number of PSNs to break a connection. For example, our injection tool for RoCEv2 can drop one connection every 10.60 s by fully enumerating all possible 2^{24} PSNs for a single QP. Then it transits to the next QPN and thus sequentially breaks all QP connections.

Practicality. As this attack relies on packet injection, it assumes the same threat models as A1.

Mitigation. As this attack relies on packet injection to successfully break a victim's connection, similar mitigation techniques should be applied to thwart such DoS attacks. In addition, our QPN randomization technique can significantly reduce performance of the attack, as the attacker will be unaware of the QPN of other clients, making enumeration infeasible. If QPNs are randomly generated, the attacker needs to probe approximately $\frac{2^{24}}{o}$ QPNs, where o is the number of open connections on the victim's machine. For example, if the victim has 1024 open QPs, the attack tool can only break one connection per 48 h on average.

5.3 A3: Unauthorized Memory Access

Unauthorized memory access effectively breaks secrecy of applications running on a victim host, but might also influence their behavior. Even worse, since RDMA operations can be performed purely one-sided V7, the victim is unable to detect such attacks. Attacks based on unauthorized memory access are referred to as A3.

As illustrated in Figure 4, an attacker establishes an RDMA connection with a service and tries to access memory of other clients connected to this service. RDMA applications typically share a PD for all RDMA resources V9 and allocate private RDMA-accessible buffers for each new user. These buffers are allocated in close proximity to each other, e.g., as chunks of a larger continuous memory region, allowing an attacker to predict the virtual memory address of buffers belonging to other clients V4. In the example, the attacker tries to access the memory region adjacent to its own memory region. To gain access, the attacker is also required to guess the corresponding memory protection key *rkey* for a memory region. Given that *rkey*s are highly predictable V1, the attacker can guess the keys of adjacent memory regions based on its *rkey* that he obtained after registering a memory region.

The attacker can also exploit other vulnerabilities of *rkey* generator such as static initialization of memory key generator V2 and shared key generator V3. V2 can be used to guess *rkey* after a reboot of the machine. V3 can be employed by an attacker sitting on the same physical machine. Finally, ODP allows accessing any virtual address by having a single *rkey* V10 . If the attacker can successfully guess the *rkey* of ODP registration, it can access the whole memory space of the process.

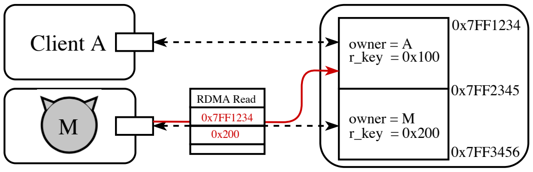

Figure 4: Unauthorized memory access on the same host.

Due to the missing integrity protection V8, an on-path attacker T3 would even be able to alter remote memory by spoofing valid RDMA write packets. Similarly, an off-path attacker T2 can perform this attack via impersonation by tricking the victim host into processing fabricated packets.

Experiments. Using our attack framework, an attacker connects to an RDMA-enabled system to obtain a memory access key and memory address. Then, it tries to gain unauthorized access by trying combinations of *rkey*s and memory locations. The attacker polls for RDMA completion events to receive acknowledgment on the success of the unauthorized access. If the guess is incorrect, the attack framework reconnects and retries the attack.

Successful unauthorized access without sniffing requires knowing the code of the system under attack to see the patterns in memory allocation and registration. This is required to reduce the search space of potential virtual addresses. We further analyzed open-source RDMA applications to see whether they are vulnerable to these type of attacks (see §6). Unfortunately, almost all of the tested application were vulnerable to unauthorized memory access (see Table 5).

Practicality. Unauthorized memory access is possible under the assumption of the T1 model, as it can be performed by any client located on an RDMA-enabled service without requiring any special capabilities, and even in a trusted network. If an RDMA-based system (e.g., see Table 5) would be deployed in a public cloud environment with RDMA support, any client could perform unauthorized memory access.

Mitigation. To mitigate unauthorized memory access, each new RDMA client could be assigned to a different PD. However, this would increase the resource usage per client on RNICs. Additionally, more modern RDMA devices can employ memory windows type 2 to pin a memory region to a specific QP, which prevents other clients from accessing it. Memory windows type 2 further allow applications to choose the 8 least significant bits of the *rkey* randomly.

Another measure to prevent unauthorized memory access would be the randomization of memory addresses chosen for buffers (similar to ASLR [40] / PIC [29] for regular applications).

Finally, RDMA applications should randomize *rkey* generation, especially, if the RDMA devices with low entropy are used. We propose a mechanism, working on all RDMA devices, that randomizes the *rkey* generation process (see §7.1).

5.4 A4: DoS Attack based on Queue Pair Allocation Resource Exhaustion

Another exhaustion attack focuses on the number of QPs a device can handle. Theoretically, up to 2^{24} connections could be opened on a device V5. In reality, the tested devices were able to handle much lower numbers. Thus, an attacker could try to open as many QP connections as possible and keep them open with minimal effort. Thus, if the attacker is able to saturate the limit for QP allocations of the victim service, he could effectively deny other benign clients from opening a QP connection, which is further referenced as A4.

Experiments. According to our findings, tested devices had different limit on the number of active QP connections per application: the results varied from 32,707 for Broadcom to 261,359 for Mellanox. Thus, the attacker needs to keep alive a much smaller number of active connections than 2^{24}. The variation in the numbers comes from default settings of the drivers and the OS. Drivers put a limit on the number of open QP connections per application.

In addition, if an application uses the RDMA connection manager to establish connections, each RDMA connection gets a file descriptor assigned for receiving link events. The underlying operating system usually enforces strict limits on the number of concurrently open file descriptors. Thus, by opening QP connections the attacker can exhaust the number of available file descriptors (instead of QPs), which might be much smaller. Experiments in our testbeds showed a file descriptor limit of 4096, whereas for instances deployed on Microsoft Azure we were able to open 65,535 file descriptors.

Practicality. Resource exhaustion of QP allocations is possible under the assumption of the T1 model and does not require any special capabilities.

Mitigation. RDMA-capable devices should limit the number of open QP connections from the same remote endpoint. This could be realized based on the IB endpoint identifiers or the IP addresses for RoCE.

5.5 A5: Performance Degradation using Resource Exhaustion

Since RDMA allows an attacker to target offloading resources to an RNIC V7, an attacker might try to exhaust these resources by issuing a large number of RDMA reads or writes. For example, an attacker might target computational resources of the RNIC's packet processing units. This will cause an increased latency for other entities accessing the same end host, but might eventually lead to disruption of service access. Interestingly, due to the one-sided nature of RDMA reads and writes, resource exhaustion attacks can be executed *"silently"*, i.e., with minor detection possibilities on the victim host. We refer to performance degradation attacks based on resource exhaustion as A5.

Experiments. We analyze the influence of resource exhaustion using a varying number of attackers on the latency and bandwidth of RDMA read or write operations. Each attacker is located on a dedicated machine equipped with a Mellanox ConnectX-3 RNIC and connected to the victim service through a switch. Each attacker floods the victim service with RDMA write requests of maximum transmission unit size, (4 KB in the testing environment) with the intention of exhausting packet processing resources of the victim's RNIC. Since RDMA write are enabled by default, but RDMA read operations must be explicitly enabled during connection establishment, exhaustion attacks based on RDMA writes are more likely to occur.

To observe the effect of this attack, we measure the latency and available bandwidth as observed by a client of the vic-

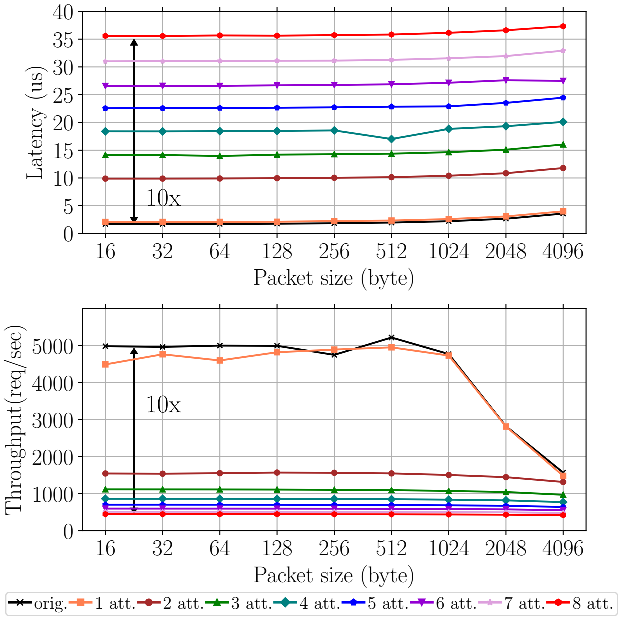

Figure 5: Effect of an exhaustion attack using RDMA write on latency and bandwidth of *RDMA read*.

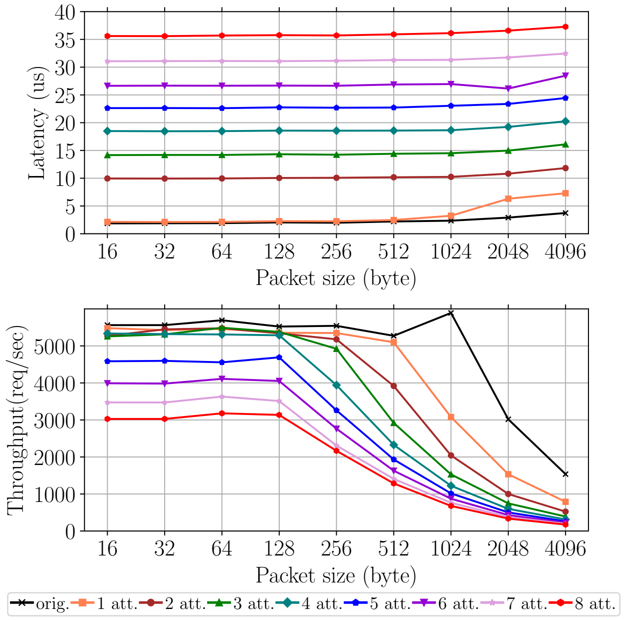

Figure 6: Effect of an exhaustion attack using RDMA write on latency and bandwidth of *RDMA write*.

tim service. Figures 5 and 6 illustrate the results of these experiments. During normal operation, the latencies for both RDMA read and write operations as observed by the client remain largely unaffected by the size of the requests as the requests fit in a single packet. However, given the presence of only two attackers that flood the victim service with requests, the latency for regular RDMA requests increases by factor 3. For each additional attacker the latency further increases by 4.20 µs. In terms of throughput, the resource exhaustion attack is even more severe. For two or more attackers, the throughput of a victim reduces by factor 8 – 10 for RDMA read requests. For RDMA writes, the attack should be performed by at least five attackers to notably affect the write throughput of legitimate applications.

Practicality. This resource exhaustion attack does not require any special capabilities and is achievable under the assumption of the T1 model. However, collusion of several attackers is required to effectively disrupt a public service.

Mitigation. Due to the nature of one-sided RDMA operations, the misuse of RDMA for performance degradation is almost undetectable. However, modern RNICs (e.g., based on mlx5) support hardware counters on the device which are accessible by the host. Thus, a host would be able to detect resource exhaustion attacks based on excessive issuance of requests. Using this detection based on HW counters would allow a host to mitigate these attacks.

5.6 A6: Facilitating Attacks using RDMA

In addition to attacks that use RDMA as an attack vector, RDMA can also facilitate attacks (e.g., for data extraction). As RDMA read and write operations do not require any interaction by a remote host's CPU V7, they allow an attacker to "silently" read and write data.

For example, if an attacker has the privilege to preload a library to a victim's application, the attacker can misuse this ability to *inject code* that establishes an RDMA connection to the attacker's application. This RDMA connection can then be used by the attacker to read memory from the victim without involvement of the victim CPU or intervention with applications executed by the victim.

Since the memory registered by an application must also be readable by the RNIC, the attacker can either preregister a large chunk of memory or enable ODP access which grants access to any valid virtual address without memory registration V10 . Then, by continuously sweeping the readable memory, the attacker can eavesdrop on sensitive data of applications.

Experiments. To illustrate the feasibility of using RDMA as an attack vector, we implemented a proof-of-concept application that preloads a malware library to a binary and allows an attacker to intercept a secret passphrase entered by the victim by reading memory using RDMA read operations. The malware (see Listing 1) preallocates a memory space and register it for RDMA Read access. Then it sends the *rkey* and the memory address of the memory region to the attacker, and deallocates the memory. Freed memory is still RDMA

```
//Initialization
rdma_connection* con = connect("Attacker's IP and PORT")
//Size of adversarial memory
const uint32_t length = 4096;
//Pre-allocate the adversarial memory
void* buf = malloc(length);
//Register the pre-allocated buffer
// with RDMA READ access
ibv_mr * mr = ibv_reg_mr(PD,buf,size,RDMA_READ);
//Send the memory region to the attacker
con->send(mr->address,mr->rkey,mr->length);
//Free the buffer so that
// the victim can use it.
free(buf);
```

Listing 1: Pseudocode of RDMA malware library

accessible, as it is not deregistered. Since the memory is deallocated, the victim is able to use it for storing its passphrase. The remote attacker can continuously read the memory using RDMA to gain the passphrase. ODP strengthens the attack by allowing the attacker to read the whole memory space, and not just the pre-registered region.

Practicality. The attack is achievable under T4 model. Both attacker and the victim needs to be in the same network and be equipped with RDMA-capable NICs. In addition, the attacker should be capable of replacing or modifying the execution binaries of the victim. Note that this attack can not only be applied to RDMA applications, but can also be used to obtain sensitive data from other applications.

Mitigation. A mitigation mechanism that prevents an adversary from misusing RDMA as an attack vector would be check to what libraries are preloaded with a binary. However, to more generally prevent attacks that include RDMA operations in code, the system should rely on remote code attestation (e.g., based on Intel SGX [6]).

6 Vulnerability Assessment of Open-Source RDMA Systems

We analyse whether recent open-source applications and systems that use RDMA as a communication mechanism are vulnerable to the aforementioned attacks. Table 5 lists the analyzed systems and their security issues.

Infiniswap [11] is a remote memory paging system that uses remote memory as a swap block device and is specifically designed to be used in an RDMA network. To "swap out" memory pages a local block is sent over RDMA to a remote block using an RDMA write operation. Similarly, to "swap in" memory pages RDMA read operations are used. Using A1, an attacker can inject a packet and modify the content of swapped pages. Infiniswap is also vulnerable to DoS attacks using A2 which breaks connections of other clients. Furthermore, the Infiniswap daemon uses `posix_memalign` in a loop to allocate *and* register buffers of 1 GB, allowing an attacker to predict the position and *rkey* of newly allocated buffers (difference of `0x40002000` bytes). Using A3, an attacker can connect to the Infiniswap service and get access to the memory of other clients in the same PD. Since Infiniswap does not limit the number of connections and resources per client, a single client can occupy all connections using A4 or execute performance degradation attacks using A5.

Table 5: Summary on vulnerabilities of open-source RDMA-enabled systems, and how they establish connections: using native interface or connection manager.

System	Connection	A1	A2	A3	A4	A5
Infiniswap [11]	Manager	✓[a]	✓[a]	✓	✓	✓
Octopus [22]	Native	✓[a]	✓[a]	✓	✓	✓
HERD [13]	Native	✓[a]	✓[a]	✓	✓	✓
RamCloud [28]	Native	✓[a]	✓[a]	✓	✓	✗
Dare [31]	Native	✓[a]	✓[a]	✓	✗	✓
Crail [34, 35]	Manager	✓[a]	✓[a]	✓	✓	✓

[a] if deployed over RoCE

✓ vulnerable to attack ✗ resilient to attack

Octopus [22] is an RDMA-enabled distributed persistent memory file system. As Octopus uses a hard-coded fixed starting PSN, an attacker can trivially predict subsequent PSNs and a perform packet injection attack A1 and A2. Furthermore, all clients share a single buffer that can be accessed using a single *rkey* A3. Thus, the Octopus system relies on strict trust in all participating parties, i.e., clients must write remote procedure call (RPC) requests to predefined offsets, as otherwise the system would fail. Additionally, even though Octopus does not use RDMA reads, all buffers are registered with read permissions enabled. Thus, a misbehaving client can read and *change* RPCs of all other clients and force the system to execute a wrong RPC. Finally, Octopus also does not limit the number connections and resources per client A4, and clients are able to obtain RDMA writable memory regions after establishing a connection A5.

The HERD [13] system implements an RDMA-enabled key-value store. Similar to Octopus, HERD uses a single memory buffer with a single registration for all RPC requests by clients, does not limit the number of connections and resources per client and is thus vulnerable to A3, A4, and A5. However, unlike Octopus HERD generates PSNs randomly. Unfortunately, systems such as Hermes [14] and cc-NUMA [10] that are implemented using HERD inherit all its vulnerabilities.

RamCloud [28] is a distributed key-value store based on two-sided RDMA. As one-sided RDMA operations are not enabled, performance degradation using A5 is not possible. However, unauthorized memory access A3 is still possible because RamCloud registers memory with enabled remote accesses and the memory allocation is static. RamCloud starts memory allocation at address `0x40000000` and all subsequent allocations are offset by adding 1 GB to the previous address. Furthermore, the number of clients is also not limited, and a single client can exhaust all QP resources A5.

Dare [31] specifies an RDMA-accelerated consensus protocol that is based on trust in all participating entities. Unfortunately, a static initial PSN is used and all control data is registered using a single registration. Thus, a misbehaving participant can forge the votes of other participants using packet injection A1 or unauthorized memory access A3, and manipulate the consensus decision to his benefit. However, A4 is not possible as the the number of clients is fixed and defined by the consensus quorum size.

Crail [34, 35] is a high-performance distributed data store designed for fast sharing of ephemeral data in distributed data processing workloads. Crail has similar vulnerabilities as Infiniswap as it maps and registers 1 GB fixed-size files in a loop, making the memory addresses and corresponding *rkeys* highly predictable. Finally, the number of connections is not limited and all memory buffers are accessible using RDMA.

7 Mitigation Mechanisms

7.1 Software-based Security Mechanisms

In the following, we propose two mitigation mechanisms that are readily deployable by RDMA applications without requiring changes to hardware or the IB protocol. While these mechanisms introduce some computational overhead on the application, they could be deployed until other mitigations mechanisms become available.

M1 **Randomization of QPNs.** We propose the following mechanism for randomization of QPNs (see Listing 2). An RDMA application creates a pool of unconnected QP descriptors with random QPNs. As soon as QP connection is required, one of the QP descriptors is fetched and registered. This measure will introduce some overhead on the RDMA host, but can be deployed *without* modification of existing RDMA protocols and will increase the number of packets that an adversary needs to inject to 2^{24}. Given that modern RNICs provide hardware counters that are accessible by applications, such bruteforce attempts could be detected.

M2 **Randomization of *rkeys*.** In addition to randomizing QPNs, we propose a mechanism to randomize memory protection keys *rkeys*. Similar to the first mitigation mechanism, the application preregisters a pool of empty memory regions with different *rkeys* (see Listing 3). When a new buffer needs to be registered, the application can randomly get a memory descriptor and remap it to the specified buffer using an *ibv_rereg_mr* call.

```
//Initialization
RandomPool pool;
//Create a pool of QP connections
// by skipping a random number of QP connections
for(int i=0; i<POOL_SIZE; i++){
 int random_value = secure_prng();
//Create and destroy QPs to get a random QP number
 for(int j=0; j<random_value; j++){
//Each new QP has a predictable QPN
  ibv_qp * qp = ibv_create_qp(params);
  ibv_destroy_qp(qp);
 }
// Only random QPs are stored
 pool.add(qp);
}

//On connection request, a random QP is taken
ibv_qp * create_qp( ){
//Take random QP, which has a random QP number
 struct ibv_qp * qp = Pool.get_random();
 return qp;
}
```

Listing 2: Algorithm for randomization of QP numbers

7.2 Leveraging Existing IB Security Mechanisms

M3 **Hardware Counters in RNICs.** Recent RNICs from Mellanox (based on the mlx5 driver) support port and hardware counters that are accessible by RDMA applications [26]. These counters enable precise monitoring of requests for debugging, load estimation and error detection. For example, `resp_remote_access_errors` could be used to monitor invalid requests that resulted in access errors. Attacks based on flooding a victim with malicious RDMA traffic could be detected using these counters. Even though attack detection does not directly prevent attacks, it is an important countermeasure to A3, A5, and A6.

M4 **Type 2 Memory Windows.** IBA offers *type 2 memory windows* which bind a memory region to a specified QP and prevent unauthorized memory access by other QPs. However, since IBA has no means of source authentication, an attacker can mimic any RDMA request initiator and inject RDMA write packets to corrupt memory of the victim host by spoofing an RDMA packet which contains the memory address and its *rkey*. Additionally, type 2 memory windows have the disadvantage that the RNIC is required to store the QP number and the corresponding *rkey* for each QP which is allowed to access the window.

M5 **Protection Domains.** Memory regions can also be protected using PDs, which prevents accesses to the memory across different memory domains by enforcing that each memory region must be part of a single PD and that all QPs can be only member of a single PD. Thus, only QPs within the same PDs can access these memory regions. However, in practice many RDMA applications use only a single PD

```
//Initialization
RandomPool pool;
//Allocate an anchor buffer
void *anchor=mmap(0, PAGESIZE, PROT_READ|PROT_WRITE,
               MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
//Create a pool of memory registration
// register the anchor buffer many times.
for(int i=0; i<POOL_SIZE; i++){
//Register a buffer
//Each registration can have a predictable rkey
 ibv_mr * mr = ibv_reg_mr(PD,anchor,PAGESIZE,0);
 pool.add(mr);
}
//On registration, a random registration is taken
// and reregistered to the requested address
// with requested length and access permissions.
ibv_mr * reg_mr(void* addr, size_t len, int access){
//Take random mr, which has a random rkey
 struct ibv_mr * mr = Pool.get_random();
 int flags = IBV_REREG_MR_CHANGE_TRANSLATION |
           IBV_REREG_MR_CHANGE_ACCESS;
//After reregistration rkey will not change.
 ibv_rereg_mr(mr,flags,PD,addr,len,access);
 return mr;
}
```

Listing 3: Algorithm for randomization of *rkey*

for all connections and memory regions to reduce memory overhead on the RNIC. For example, the RDMA connection manager (`librdmacm`) by default uses a single PD for all RDMA related operations such as connection establishment and memory registration within a process.

7.3 Encryption and Authentication in RDMA Protocols

Since the existing IBA security mechanism can be circumvented due to the lack of endpoint and packet authentication, IBA could make use of encryption and authentication at any layer in the protocol stack M6 . In the following, we discuss the most relevant options.

RDMA-over-IPsec. RDMA protocols based on IP (e.g., RoCE) would allow the use of IPsec [7] for packet encryption and authentication of end points. Recent NICs support IPsec for RoCE traffic (e.g., Mellanox ConnectX-6 DX [25]) by providing IPsec tunnels as a transport and simply encapsulating RoCE packets inside IPsec. However, since IPsec does not directly support RDMA traffic and encapsulates the RDMA headers, it authenticates traffic based on IP address and UDP port. To prevent the injection of RoCE packets into IPsec-enabled QPs from other end hosts, the NIC stores QP context information and enforces an association the QP number and the source IP address. Using an IPsec tunnel between two end points would be able to prevent A1 and A2 for Ethernet networks, but cannot be applied to the InfiniBand protocol.

Application-layer Encryption and Authentication. Application-level encryption and authentication (e.g., based on TLS with client authentication [32]) of RDMA applications is not possible, because RDMA read and write operations can operate as purely one-sided communication routines (without involvement of the other parties CPU). An approach based on application-layer encryption would require a temporal buffer for the incoming encrypted messages. These would then be decrypted by the CPU and then copied to the destined location completely negating RDMA's advantages.

Encryption and Authentication Integrated in IBA. In contrary to application-layer cryptography, encryption and authentication of RDMA messages could also be integrated into the design of IBA. Lee et al. [16,17] suggested to replace the Invariant CRC field with a MAC to achieve packet authentication. Recently, Taranov et al. [36] proposed sRDMA, a protocol that extends IBA by designing a connection mode that provides authentication and encryption for RDMA based on symmetric cryptography.

Encryption and authentication integrated into IBA can prevent information leakage to on-path attackers and also prevent message tampering as the RDMA message header is authenticated. Thus, it becomes impossible for an attacker to spoof RDMA header fields, prevents him conducting all attacks based on packet injection.

7.4 Other Mitigation Mechanisms

M7 **Per-Client Resource Constraints.** RDMA-capable devices should limit the number of concurrently open QP connections and allocated resources on a per-client basis. Otherwise, attacks based on resource exhaustion cannot be prevented. With per-client resource constraints in place an attacker would need to collude with a large number of endpoints to successfully execute resource exhaustion attacks. Allocating resources per-client could be realized based on the InfiniBand adapter identifiers for native IB connections and using the IP address for RoCE connections.

M8 **In-Network Filtering.** Apart from modifying IB-based architectures, packet injection could also be prevented using in-network filtering. In datacenter deployments, operators could deploy a filtering mechanism at the ingress of the network to effectively prevent an attacker from injecting spoofed packets (e.g., similar to [9]).

8 Related Work on RDMA attacks

RFC 5042 [30] analyzes the security issues around uses of RDMA protocols. It defines an architectural model for RDMA-based implementations and reviews various basic attacks including spoofing, tampering, information disclosure, and exhaustion of shared resources. The authors suggest the use of IPsec encryption and authentication to mitigate attacks that target end-to-end security, which unfortunately fails to solve the problem of endpoint authentication. RFC 5042 aims to provide a guideline for designing protocols based on RDMA, but is completely implementation agnostic and only mentions potential vulnerabilities specific to RDMA protocols. ReDMArk tests the applicability of vulnerabilities to specific implementations of RDMA (such as InfiniBand) and shows that the security pitfalls of using RDMA remain misunderstood.

Tsai et al. [39] discuss the threats and opportunities of one-sided communication. They raise concerns about the predictability of hardware-managed memory protection key and the potential misuse of one-sided RDMA communication for DoS. Compared to previous work, ReDMArk provides an in-depth security analysis of RDMA networking (e.g., investigates the algorithms behind *rkey* generation in detail) covering not only vulnerabilities, but discussing the full chain of vulnerabilities, proposes specific attacks based on the discovered vulnerabilities, and mitigations for these attacks.

Kornfeld Simpson et al. [33] summarize the security flaws in RDMA protocols (e.g., missing authentication and encryption) and discuss security challenges of designing RDMA-enabled storage systems. In addition to the attacks discussed by ReDMArk, they suggest to exploit priority flow control (PFC) pause frames in RoCE [12] to flood buffers on switches. However, they mention that the most recent version of RoCE is not subject to this attack as it does not require PFC.

Furthermore, Tsai et al. [38] discovered that RNICs could be exploited for side-channel attacks. They implemented an RDMA-based side channel attack that allows an attacker on one client machine to learn how victims on other client machines. The attacker uses RDMA access latency and a trained classifier to statistically predict victim accesses.

Kurth et al. [15] have shown that the Intel DDIO [1] and RDMA features facilitate a side-channel attack named NetCAT. Intel DDIO technology allows RDMA read and write accesses not only to a pinned memory region, but also parts of the lowest CPU cache. NetCAT remotely measures cache activity caused by a victim's SSH connection to perform a keystroke timing analysis and recovers words typed in the SSH session. Using this analysis, an attacker can recover words typed in the SSH session on another computer. These works based on side-channel attacks using RDMA are complementary to ReDMArk.

9 Conclusion

RDMA architectures such as RoCE and InfiniBand were designed for HPC and private networks, and have neglected security in their design in favor of focusing on high performance. As illustrated by ReDMArk, the design of IBA and the implementation of IB-capable NICs contain multiple vulnerabilities and design flaws. These weaknesses allow an adversary to inject packets, gain unauthorized access to memory regions of other clients connected to an RDMA-based service with potentially drastic consequences, and effectively disrupt communication in RDMA networks. Given that InfiniBand is deployed in public infrastructure and more providers plan to adopt RDMA networking, weak RDMA security creates real-world vulnerabilities in RDMA-enabled systems. This work shows the security implications of RDMA on cloud systems and demonstrates the critical importance of security in the design of upcoming versions of InfiniBand and RoCE (e.g., by fully integrating header authentication and payload encryption). In addition, developers of RDMA-enabled systems must be aware of the threats introduced by RDMA networking and should employ mitigations such as using type 2 memory windows, a separate PD for each connection, and our proposed algorithms to randomize the QPN and the *rkey* generation.

Responsible Disclosure

We have notified and responsibly disclosed the weaknesses to Mellanox, Broadcom, and Microsoft prior to the submission of this work.

Acknowledgments

We would like to thank our shepherd, Haya Shulman, and the anonymous reviewers for their constructive feedback. We thank Mellanox Technologies and Broadcom Inc. for the hardware donations as well as their feedback during the disclosure of this work. In addition, we thank Igor Zablotchi for assisting with the evaluation of this work. We gratefully acknowledge support from ETH Zurich, and from the Zurich Information Security and Privacy Center (ZISC). Furthermore, we thank the Microsoft Swiss Joint Research Centre for their support.

References

[1] Intel® Data Direct I/O Technology Overview. https://www.intel.co.jp/content/dam/www/public/us/en/documents/white-papers/data-direct-i-o-technology-overview-paper.pdf, 2019. [Online; accessed 19-Sep-2020].

[2] Raw Ethernet Programming: Basic Introduction - Code Example. https://community.mellanox.com/s/article/raw-ethernet-programming--basic-introduction---code-example, 2019. [Online; accessed 19-Sep-2020].

[3] InfiniBand Trade Association. The InfiniBand architecture specification. *https://www.infinibandta.org/ibta-specifications-download/*, 2000.

[4] Infiniband Trade Association. Supplement to InfiniBand architecture specification volume 1, release 1.2. 1: Annex a16: RDMA over Converged Ethernet (RoCE), 2010.

[5] Christian Cachin. *Entropy measures and unconditional security in cryptography*. PhD thesis, ETH Zurich, 1997.

[6] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, (086), 2016.

[7] Naganand Doraswamy and Dan Harkins. *IPSec: the new security standard for the Internet, intranets, and virtual private networks*. Prentice Hall Professional, 2003.

[8] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. Farm: Fast remote memory. In *Proceedings of USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 401–414, 2014.

[9] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. BCP 38, 2000.

[10] Vasilis Gavrielatos, Antonios Katsarakis, Arpit Joshi, Nicolai Oswald, Boris Grot, and Vijay Nagarajan. Scale-out ccnuma: Exploiting skew with strongly consistent caching. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–15, 2018.

[11] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient memory disaggregation with INFINISWAP. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 649–667, 2017.

[12] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the ACM SIGCOMM Conference*, pages 202–215, 2016.

[13] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Using RDMA efficiently for key-value services. In *Proceedings of ACM SIGCOMM*, pages 295–306, 2014.

[14] Antonios Katsarakis, Vasilis Gavrielatos, MR Siavash Katebzadeh, Arpit Joshi, Aleksandar Dragojevic, Boris Grot, and Vijay Nagarajan. Hermes: a fast, fault-tolerant and linearizable replication protocol. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 201–217, 2020.

[15] Michael Kurth, Ben Gras, Dennis Andriesse, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. NetCAT: Practical cache attacks from the network. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[16] Manhee Lee and Eun Jung Kim. A comprehensive framework for enhancing security in InfiniBand architecture. *IEEE Transactions on Parallel and Distributed Systems*, 18, 2007.

[17] Manhee Lee, Eun Jung Kim, and Mazin Yousif. Security enhancement in InfiniBand architecture. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 2005.

[18] Bojie Li, Tianyi Cui, Zibo Wang, Wei Bai, and Lintao Zhang. Socksdirect: Datacenter sockets can be fast and compatible. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 90–103. 2019.

[19] Mingzhe Li, Xiaoyi Lu, Hari Subramoni, and Dhabaleswar K Panda. Designing registration caching free high-performance MPI library with implicit on-demand paging (ODP) of InfiniBand. In *IEEE International Conference on High Performance Computing (HiPC)*, pages 62–71, 2017.

[20] Linux RDMA. RDMA core userspace libraries and daemons. https://github.com/linux-rdma/rdma-core/, 2020. [Online; accessed 19-Sept-2020].

[21] Linux RDMA. Software RDMA over Converged Ethernet. https://github.com/SoftRoCE/rxe-dev/, 2020. [Online; accessed 19-Sept-2020].

[22] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. Octopus: an RDMA-enabled distributed persistent memory file system. In *USENIX Annual Technical Conference (ATC)*, pages 773–785, July 2017.

[23] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[24] James L Massey. Guessing and entropy. In *Proceedings of 1994 IEEE International Symposium on Information Theory*, page 204. IEEE, 1994.

[25] Mellanox. NVidia Mellanox ConnectX-6 DX. https://www.mellanox.com/files/doc-2020/pb-connectx-6-dx-en-card.pdf, 2020. [Online; accessed 19-Sept-2020].

[26] Mellanox. Understanding mlx5 Linux Counters and Status Parameters. https://community.mellanox.com/s/article/understanding-mlx5-linux-counters-and-status-parameters, 2020. [Online; accessed 19-Sept-2020].

[27] Microsoft. Cve-2019-1372, azure stack remote code execution vulnerability. https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2019-1372, 2020. [Online; accessed 19-Sept-2020].

[28] John Ousterhout, Arjun Gopalan, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnam Montazeri, Diego Ongaro, Seo Jin Park, Henry Qin, Mendel Rosenblum, Stephen

Rumble, Ryan Stutsman, and Stephen Yang. The RAMCloud storage system. *ACM Trans. Comput. Syst.*, 33(3):7:1–7:55, August 2015.

[29] R Kim Peterson. Position independent code location system, 1996. US Patent 5,504,901.

[30] J. Pinkerton and E. Deleganes. Direct Data Placement Protocol (DDP) / Remote Direct Memory Access Protocol (RDMAP) Security. RFC 5042, October 2007.

[31] Marius Poke and Torsten Hoefler. Dare: High-performance state machine replication on rdma networks. In *Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pages 107–118, 2015.

[32] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, 2018.

[33] Anna Kornfeld Simpson, Adriana Szekeres, Jacob Nelson, and Irene Zhang. Securing RDMA for high-performance datacenter storage systems. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2020.

[34] Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, Ana Klimovic, Adrian Schuepbach, and Bernard Metzler. Unification of temporary storage in the nodekernel architecture. In *Proceedings of USENIX Conference on Usenix Annual Technical Conference (ATC)*, page 767–781, 2019.

[35] Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, Radu Stoica, Bernard Metzler, Nikolas Ioannou, and Ioannis Koltsidas. Crail: A high-performance I/O architecture for distributed data processing. *IEEE Data Eng. Bull.*, 40(1):38–49, 2017.

[36] Konstantin Taranov, Benjamin Rothenberger, Adrian Perrig, and Torsten Hoefler. sRDMA: Efficient nic-based authentication and encryption for remote direct memory access. In *USENIX Annual Technical Conference (ATC)*, 2020.

[37] Mellanox Technologies. RDMA Aware Networks Programming User Manual, Rev 1.7. https://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf, 2015.

[38] Shin-Yeh Tsai, Mathias Payer, and Yiying Zhang. Pythia: Remote oracles for the masses. In *USENIX Security*, pages 693–710, 2019.

[39] Shin-Yeh Tsai and Yiying Zhang. A double-edged sword: Security threats and opportunities in one-sided network communication. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2019.

[40] Fernando Vano-Garcia and Hector Marco-Gisbert. KASLR-MT: Kernel address space layout randomization for multi-tenant cloud systems. *Journal of Parallel and Distributed Computing*, 137:77–90, 2020.

[41] Michal Zalewski. Strange attractors and tcp/ip sequence number analysis. *RAZOR/Bindview Corporation*, 2001.

A Algorithms of *rkey* generators

```
static uint32_t bnxt_get_key(void){
        static uint32_t key = 0x100;
        key += 0x100
        return key;
}
```

Listing 4: *rkey* generation of bnxt_re

```
static uint32_t rxe_get_key(void){
        static uint32_t base = 0x100;
        static unsigned key = 1;
        base +=  0x100;
        key = key << 1;
        key |= (0 != (key&0x100))^(0 != (key&0x10))
            ^(0 != (key&0x80))^(0 != (key&0x40));
        key &= 0xff;
        return base + ((uint8_t)key);
}
```

Listing 5: *rkey* generation of SoftRoCE

```
static Queue key_queue;//queue for deregistered keys
static uint32_t base = 0x100; // is device-specific
static uint32_t MASK = 0xFFFFFFFF; // 24bit mask
static uint32_t mlx4_get_key(void){
        static uint32_t key = 0x100;
        if(key_queue.is_empty()){
                key+= 0x100;
                return base + (key & MASK);
        }
        uint32_t old_key = key_queue.pop();
        return base + (old_key & MASK);
}
static void mlx4_dereg_key(uint32_t old_key){
        base += 0x8000000;
        key_queue.push(old_key);
}
```

Listing 6: *rkey* generation of mlx4