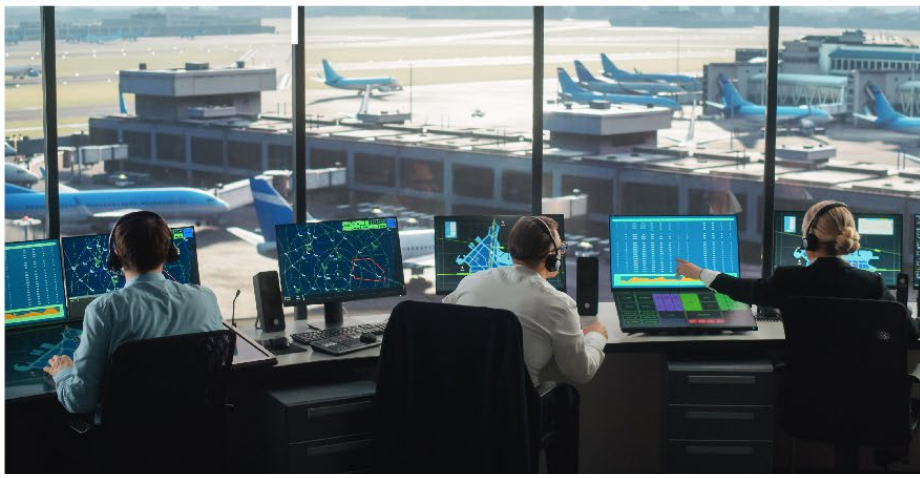




A Security Analysis of the Data Distribution Service (DDS) Protocol

Federico Maggi, Rainer Vosseler
Trend Micro Research

Mars Cheng, Patrick Kuo, Chizuru Toyama, Ta-Lun Yen
TXOne Networks

Erik Boasson
ADLINK

Víctor Mayoral Vilches
Alias Robotics

TREND MICRO™ | research

Published by

Trend Micro Research

Written by

Federico Maggi,
Rainer Vosseler
Trend Micro Research

Mars Cheng, Patrick Kuo,
Chizuru Toyama, Ta-Lun Yen
TXOne Networks

Erik Boasson
ADLINK

Víctor Mayoral Vilches
Alias Robotics

Stock images used under license from Shutterstock.com

For Raimund Genes (1963-2017)

Contents

We analyzed the Data Distribution Service (DDS) standard, a middleware technology that drives billions of devices and mechanisms such as railways, autonomous cars, airports, spacecraft, diagnostic imaging machines, luggage handling systems, industrial robots, military tanks, and frigates. We discovered and reported multiple security vulnerabilities in DDS: 13 were given new CVE IDs (in November 2021) from the six most common DDS implementations, plus one vulnerability in the standard specifications.

We spent one month scanning and found hundreds of distinct public-facing DDS services from over one hundred organizations from industries like telecommunications, cloud, software, and research from different countries. Some were identified and affected by the newly identified CVEs. Others were identifiable via nearly 90 internet service providers (ISPs) through hundreds of leaked private IP addresses and other internal network architecture details. These are considered significantly dangerous because DDS is not supposed to be deployed on public-facing endpoints. Vulnerabilities can serve as openings for initial access to a targeted system via supply chain compromise or service discovery. Furthermore, our scanning results show that the post-exploitation impact can range from denial of service (DoS) to loss of control or safety.

We advocate for the continuous security testing of DDS and other related critical technology. We also provide actionable recommendations to use for a secure DDS integration. In the short term, we recommend mitigation procedures such as network service scanning, network intrusion prevention, network segmentation (such as avoiding DDS exposure to public-facing networks), network traffic filtering, execution prevention, and periodic auditing. In the long run, we recommend implementing supply-chain management processes to ensure that critical software components such as DDS are properly tracked in derived software, as well as implementing continuous security testing (for example, continuous fuzzing).

This research was accomplished through the collaboration of Trend Micro Research and TXOne Networks (Federico Maggi, Mars Cheng, Patrick Kuo, Chizuru Toyama, Rainer Vosseler, and Ta-Lun Yen), ADLINK Labs (with Erik Boasson, one of the inventors and core developers of DDS), Alias Robotics (with Víctor Mayoral Vilches, Robotics Architect), and Trend Micro Zero Day Initiative (ZDI).

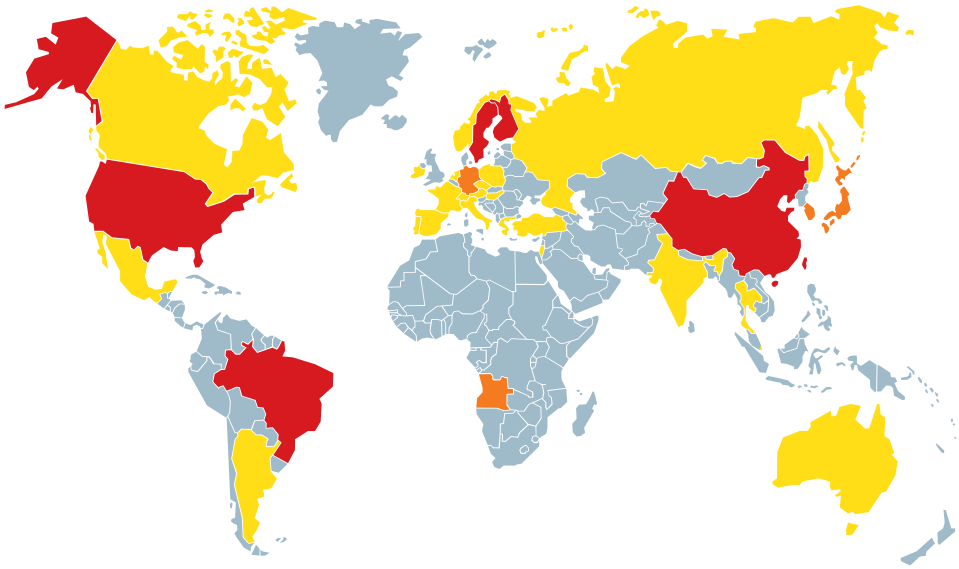
1 Introduction

Even within the industry, a big percentage of practitioners are unaware that the Data Distribution Service (DDS) drives systems such as railways, autonomous cars, airports, spacecraft, diagnostic imaging machines, luggage handling, industrial robots, military tanks, and frigates, among others. It has been in use for about a decade, and its adoption continues to steadily increase.[1]

We discovered and reported vulnerabilities in DDS that warranted new CVE IDs: Five with a score of ≥ 7.0, four with a score of > 8.5, one vulnerability in the standard specifications, and other deployment issues in the DDS software ecosystem (including a fully open production system).

Successful exploitation of these vulnerabilities can facilitate initial access (MITRE ATT&CK Technique ID TA0108) via exploitation of remote services (T0866, T0886) or supply chain compromise (T0862), and allow the attacker to perform discovery (TA0102, T0856) by abusing the discovery protocol. The consequences of successful exploitation, in any of the critical sectors where DDS is used, range from denial of service (T0814) via brute forcing (T0806), to loss of control (T0827), or loss of safety (T0880). The DDS protocol itself can also be abused to create an efficient command and control channel (T0869). Based on our analysis, we recommend mitigations such as vulnerability scanning (MITRE ATT&CK Mitigation ID M1016), network intrusion prevention (M1031), network segmentation (M1030), filter network traffic (M1037), execution prevention (M1038), and auditing (M1047).

1.1 Findings in Brief



China	333	France	3
Finland	118	Austria	3
United States	47	Argentina	3
Brazil	24	Poland	2
Taiwan	22	Norway	2
Sweden	16	Mexico	2
South Korea	9	India	2
Japan	9	Turkey	1
Germany	8	Thailand	1
Angola	7	Switzerland	1
Portugal	4	Spain	1
Canada	4	Netherlands	1
Singapore	3	Israel	1
Russia	3	Hungary	1
Italy	3	Hong Kong	1
Ireland	3	Czechia	1
Greece	3	Australia	1

Figure 1. We found exposed DDS systems in 34 countries, including vulnerable ones, identified via distinct IPs leaking data

Given this technology's versatility, we analyzed and discovered multiple security vulnerabilities, resulting in 13 new CVE IDs[2] for the six most common DDS implementations.[3] This includes one vulnerability in the standard specifications and other deployment issues in the DDS software ecosystem (including a fully open production system). These vulnerabilities have been patched or mitigated by the vendors since we reported them.[4]

By measuring the exposure of DDS services, in one month we found over 600 distinct public-facing DDS services in 34 countries affecting 100 organizations via 89 internet service providers (ISPs). Of the DDS implementations by seven distinct vendors (one of which we were initially unaware of), 202 leaked private IP addresses (referring to internal network architecture details), and seven supposedly secret URLs. Some of these IP addresses expose unpatched or outdated DDS implementations, which are affected by some of the vulnerabilities that we've discovered and disclosed in November.

During our research, we interviewed key DDS users and system integrators to collect their feedback on our findings and the importance of DDS for innovation in their respective sectors. In this research paper, we analyze and discuss the specifications of DDS and the six most actively developed implementations maintained by certified vendors and with millions of deployments worldwide. We also released an open-source software: a Scapy-based dissector[5] and several fuzzing harnesses for three open DDS implementations.

1.2 Background, Scope, and DDS Applications

DDS is a standardized middleware software based on the publish-subscribe paradigm, helping the development of middleware layers for machine-to-machine communication. This software is integral especially to embedded systems or applications with real-time[6] requirements. Maintained by the Object Management Group (OMG),[7] DDS is used in all classes of critical applications to implement a reliable communication layer between sensors, controllers, and actuators.

DDS is at the beginning of the software supply chain, making it easy to lose track of and is an attractive target for attackers. Between 2020 and 2021, 66% of attacks focused on the suppliers' codes.[8] While we were in the process of doing this research, we encountered an exposed source-code repository hosting a proprietary implementation of DDS. Left open, this would have let an attacker infect the source code (MITRE ATT&CK T0873, T0839).

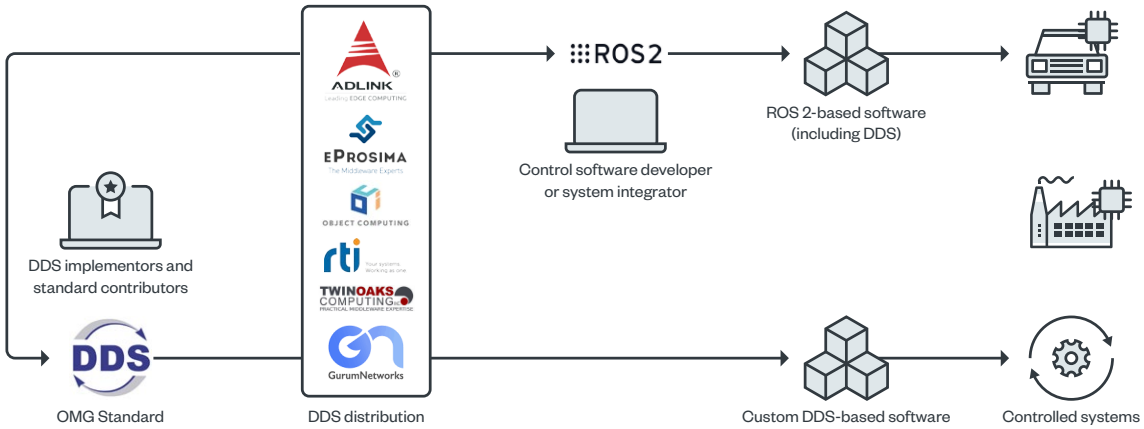


Figure 2. DDS is a standardized software library used for software-based controlled systems, directly or via ROS 2

Notably, the following companies and agencies use DDS (note that this is not an exhaustive list of currently using the technology):

- National Aeronautics and Space Administration (NASA) at the Kennedy Space Center
- Siemens in wind power plants
- Volkswagen and Bosch for autonomous valet parking systems
- Nav Canada and European CoFlight for air-traffic control

DDS is the foundation of other industry standards such as OpenFMB[9] for smart-grid applications and Adaptive AUTOSAR,[10] among other sectors that we identify in the next section. The Robot Operating System 2 (ROS 2), which is the de facto standard operating system for robotics and automation, uses DDS as the default middleware. We also noted that, according to a confidential document leaked online, NVIDIA has listed DDS as a tool for system-virtualization and cloud-computing applications, mainly for exchanging data within and across virtual machines.

2 DDS and Real-Time Publish-Subscribe (RTPS) Packets

There are many software-based controlled systems in the world that connect sensors, actuators, and controlling and monitoring applications. DDS was invented for such systems, with a strong focus on interoperability and fault tolerance. It is optimized for publish-subscribe and peer-to-peer applications as most applications can't afford a single point of failure. The middleware relies on multicast (group communication or data transmission to multiple recipients) for discovery, allowing everything to run without needing initial configurations.

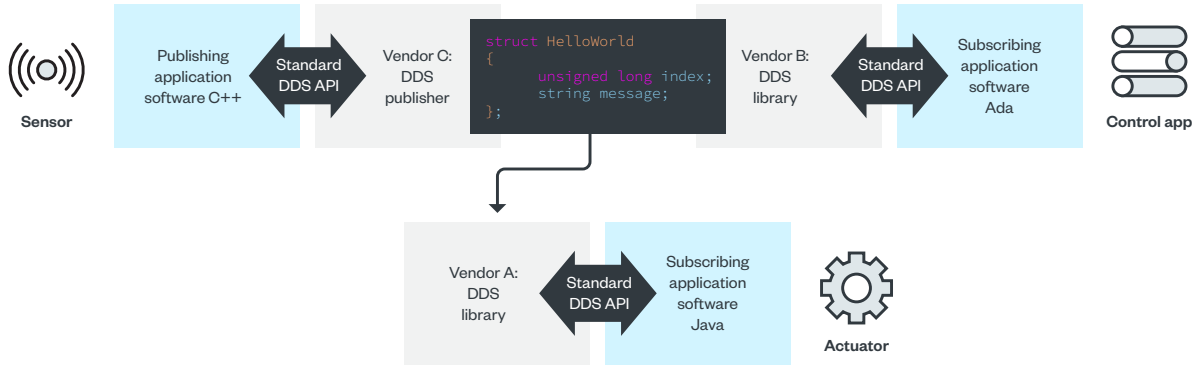


Figure 3. Simplified software-based control system with actuators, controller, sensors, communicating by exchanging data over DDS.

From a software developer standpoint, DDS is a communication middleware that facilitates interoperability of processes across machines in all main programming languages. From another viewpoint, DDS is a data-centric, publish-subscribe communication protocol that allows developers to build a flexible shared data "space" or "bus" for virtually any application that requires two or more nodes to exchange typed data.

```
using namespace org::eclipse::cyclonedds;

int main()
{
  dds::domain::DomainParticipant participant(0);
  dds::pub::Publisher publisher(participant));
  dds::topic::Topic<HelloWorld> topic(participant, "HelloWorld");
  dds::pub::DataWriter<HelloWorld> writer(publisher, topic);

  unsigned i = 0;
  while (true)
  {
    HelloWorld msg(i++, "Hello, world!");
    writer << msg;
    std::this_thread::sleep_for(std::chrono::seconds(1));
  }

  return 0;
}
```

Figure 4. A DDS minimal working example in C++ with a participant, which writes a message on the "HelloWorld" topic using Cyclone DDS

From a programmer's perspective, DDS is a powerful application programming interface (API), as exemplified by the minimal working example in Figure 4. On top of the plain byte-streams and C-strings, DDS supports serialization and de-serialization of any built-in or custom data type through a dedicated interface definition language (IDL). This function allows developers to create any complex type system, similar to but more powerful than Protobuf.[11] It also features all the usual data values like integers, floating points, and structures, among others. Recently, the concept of type evolution has been added, so the existing types can be extended and evolved[12] instead of creating new types only. As a result, the data types can become as complex as the application or developers need it to be.

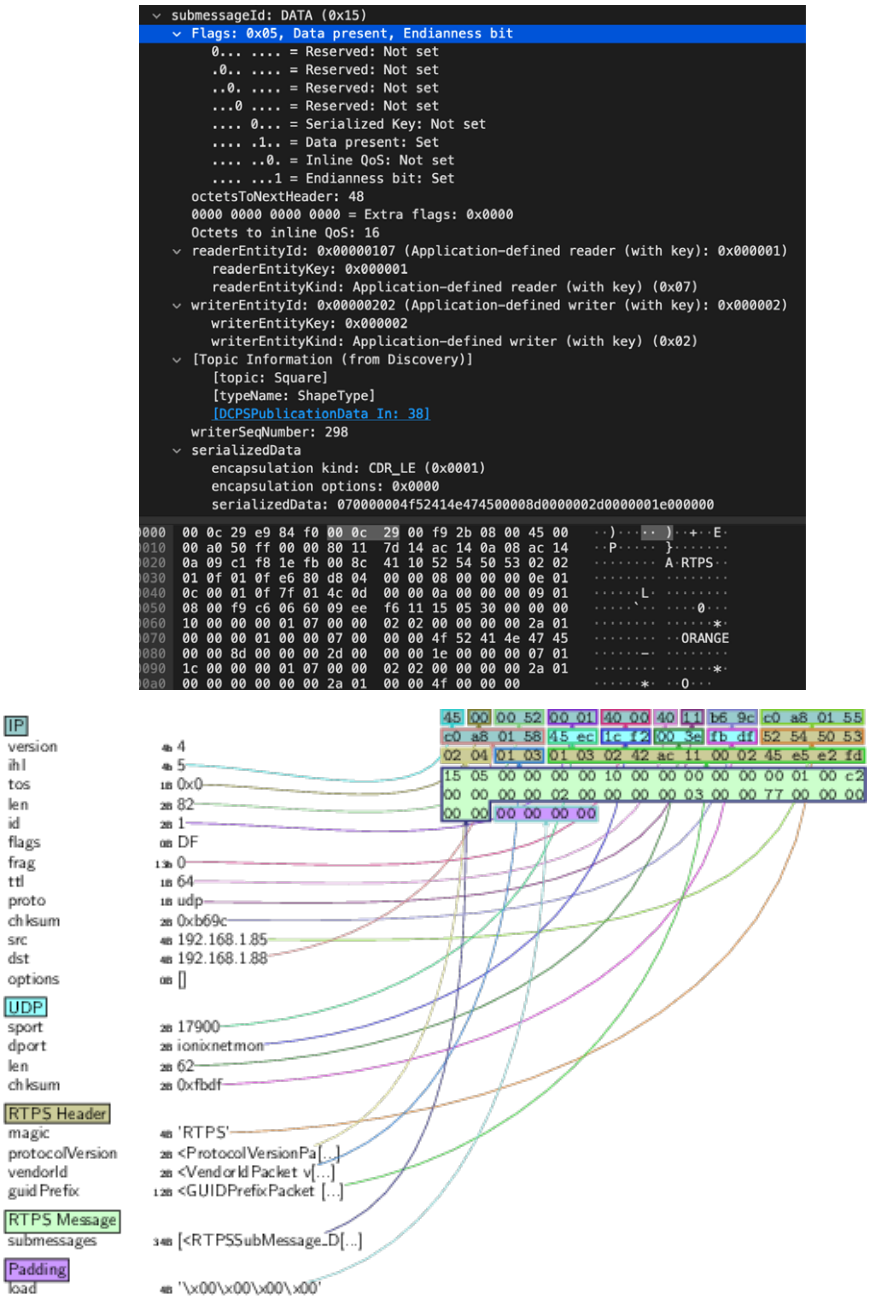


Figure 5. An example of a packet structure of an RTPS message with a DATA submessage

The DDS layer is encapsulated into real-time publish-subscribe (RTPS) packets, which for now can be considered as a collection of sub-messages (such as timestamp, discovery, data, and security metadata), as shown in Figure 5. Given the strong dependency between DDS and RTPS, we focused our research on RTPS for increased generality.

Because of its flexibility, DDS and its underlying layers do not come as a ready-to-use, off-the-shelf product like other middleware tools (such as Message Queuing Telemetry Transport or MQTT). Rather, DDS is a programming library that developers use to build custom middleware protocols with advanced features such as custom data types, quality of service (QoS) policies, network partitioning, authentication, and encryption.

2.1 Research Scope: RTPS, DDS, and Robot Operating System 2 (ROS 2)

In addition to the DDS standard specifications, we focused our investigation on the six DDS implementations listed in Table 1. These implementations are used globally and have customers and users in the critical sectors identified in the same table. Because DDS depends on RTPS as a lower-layer standard protocol, each DDS implementation ships with its own RTPS implementation. In other words, DDS data is contained as a sub-message within RTPS, thereby focusing on both protocols.

In addition, the Robot Operating System 2 (ROS 2), which is the default standard operating system (OS) for robotics and automation, has DDS as its default middleware. For this reason, the impact of each vulnerability extends beyond DDS alone, and includes all ROS 2 instances.

2.2 DDS Applications and Impacted Sectors

DDS and RTPS are used to implement industry-grade middleware layers as they are designed for mission-critical applications. For example, when the artificial intelligence (AI) of an autonomous car needs to issue a "turn left" command, DDS is used to transport that command from the electronic control unit (ECU) (the car's "brain") down to the steering servomotors. The same instance also happens when speed sensors send information from the motor up to the ECU. We verified that the DDS runs successfully on starter kit ECUs,[13] making any autonomous vehicle based on this hardware and software stack susceptible to our findings.

Another example is when an airport operator inside the air traffic control tower needs to illuminate the runway. In modern airports, these specific signals[14] are transmitted via software, and DDS is used to ensure timely delivery of those commands.

Table 1 lists examples of where DDS is used in critical industries, including external resources that offer estimates on how many devices in each sector exist or are expected to exist in the near future.

Sector	Example Use Cases	Notable Users
Telecommunications and networks	– Software-defined networking (SDNs) technologies – Appliance Life Cycle Management (LCM) tools, including 5G	– Fujitsu
Defense Industry	– Command and control (C&C) systems – Navigation and radar systems – Launch systems	– National Aeronautics and Space Administration (NASA) – NATO Generic Vehicle Architecture (NGVA)[15] – Spanish Army
Virtualization & Cloud	– Inter- and intra-communications of security operations centers (SOC)	– NVIDIA
Energy	– Power generation and distribution – Research	– Siemens – Sunrise Wind
Healthcare	– Medical devices' interoperability – Magnetic resonance imaging (MRI), computerized tomography scans (CT scans)	– GE Healthcare – Medical Device Plug-and-Play interoperability program (MD PnP)
Mining	– Precision mining – Mining system automation – Geological modeling	– Komatsu – Plotlogic – Atlas Copco
Industrial internet of things (IIoT) and robotics	– Universal middleware	– Robot Operating System (ROS 2) – AWS RoboMaker – iRobot
Public and private transportation	– Autonomous vehicles – Air traffic control (ATC) – Railway management and control	– Volkswagen and Bosch[16] – Coflight Consortium (Thales, Selex-SI)[17] – Nav Canada

Table 1. Overview of the most notable DDS use cases.

2.2.1 Telecommunications and Networks

Optical transponders,[18] such as Fujitsu's 1FINITY T600 series, provide the foundation for 5G[19] mobile transport, and these use DDS to function and optimize communication between components.[20] Likewise, the system for configuring and monitoring this networking equipment is built using DDS. Configuration settings and updates are distributed via DDS to all the blades, then a DDS application configures the hardware accordingly.

DDS is also being tested for software-defined networking (SDN) technologies[21] so it can be integrated in the next-generation of networking control planes. In particular, a comparison of DDS versus other solutions showed significant performance improvements in SDN applications.

2.2.2 Defense Industry

The typical usage of middleware technology in the defense industry include systems for navigation, weaponry management, mapping, radar, and power management. From a threat perspective, there is interest from adversaries at the state and non-state level. One recent example is the ThreatNeedle malware, which was documented to be used by the Lazarus group to target defense companies.[22]

The most notable use of DDS in this sector is NASA's launch control system at Kennedy Space Center (KSC) using one of the world's largest supervisory control and data acquisition (SCADA) systems with over 400,000 control points.[23] Other use cases are for C&C services (such as for bridging Ethernet networks to tactical radio equipment on the field), which requires efficient and reliable data transport in challenging conditions. Some system integrators such as MilSoft and KONGSBERG specialize in using DDS for defense applications. The Spanish Army uses Fast-DDS for C&C applications,[24] while an unidentified defense technology company uses OpenDDS for its connectivity framework.[25]

2.2.3 Data Centers, Virtualization, and Cloud Computing

There are over 7.2 million data centers worldwide,[26] with each containing thousands of servers and together form the hardware foundation required for cloud and traditional computing.

Middleware technology in virtualization and data-centers span from high-level software (for managing virtual and bare-metal machines in data centers and efficient protocols for data exchange) to low-level enhancements (for improving how computing cores and virtual machines (VMs) communicate between each other). In all these applications, DDS is a natural choice as DDS implementations (such as those from Object Computing,[27] ADLINK Technology,[28] eProsima,[29] RTI[30] and Gurum Networks[31]) come with built-in or add-on integration services. It can be used across distributed networks to create the appearance of one uniform DDS-based network. This can also be one of the reasons why we found exposed DDS endpoints on the internet.

For example, according to a public document marked "confidential", NVIDIA is exploring the application of OpenDDS for inter-VM/inter-SoC communication "to transfer data from one VM to another in a multi-virtual machine environment."[32]

2.2.4 Energy

Driven by changing climate challenges and goals, the energy sector is undergoing significant changes and innovation. Owing to emerging and available IIoT solutions, an entire spectrum of issues are covered, such as clean and traditional power generation, storage and management, and distribution from companies themselves to the end users being adopted by states.[33]

DDS is being used in these applications and is predicted to become more popular in the future. For instance, OpenSplice Vortex (now ADLINK Cyclone DDS) is used at the large scale fusion reactor system in the Culham Centre for Fusion Energy,[34] RTI Connext DDS is used in distributed power generation by Siemens,[35] and LocalGrid uses DDS for smart-grid distribution and control[36] and for research.[37]

2.2.5 Healthcare

DDS enables interoperable data connectivity for medical devices and clinical systems. For example, it is used by the MD PnP interoperability program,[38] which facilitates the adoption of open standards and interoperable technologies in integrated clinical environments (ICE).[39] Companies such as GE Healthcare use this connectivity to work with over 200 hospitals with a command center software in different countries with its applications to process real-time needs.[40] RTI Connext DDS is used in MRI and CT scanning equipment and for hospital patient safety and integration of clinical decision systems.[41] ADLINK's OpenSplice DDS is used to integrate medical tablets and in medical panel computers.

2.2.6 Mining Industry

While less visible to the public, the mining industry is fertile ground for innovation in information and communication technologies (ICT). For example, OpenDDS is used by Plotlogic for precision mining through geological modeling, which helps reducing waste in the process.[42] RTI Connext DDS used by Komatsu for mining machinery integration and control,[43] while Atlas Copco is using OpenSplice Vortex to create a common platform for mining system automation.[44]

2.2.7 IIoT and Robotics

In 2020, the International Federation of Robotics (IFR) estimated that 373,000 industrial robots were installed globally, which was a huge jump compared to 2011, when they estimated 89,000. The market for professional service robots (such as for transportation, logistics, cleaning, and hospitality) grew in 2020 by 12%, with 1,067 companies specializing in service robots worldwide (a 17% increase from 2019).[45]

DDS plays a fundamental role to the robotics sector because it is the default middleware of ROS 2, which is the rapidly growing, de facto standard OS for consumer, service, and industrial robots, as well as for autonomous systems in general.[46] Some would say that ROS is to robotics what Ubuntu and Linux is for computing. Particularly, Eclipse Cyclone DDS has been chosen to be the default DDS implementation in ROS 2.

From 2019, the adoption rate of ROS 2 went from less than 5% to more than 50%. According to ROS Metrics, 55% of the downloads of ROS are ROS 2, which points to a DDS layer.[47] The ROS official docker image has been downloaded more than 10 million times. AWS RoboMaker, a simulation service used by robotics developers such as iRobot,[48] is based on ROS 2.[49, 50] While these numbers do not directly indicate the number of robots running ROS 2, it implies a growing trend and interest in the sector.

As more sectors make use of robots running ROS 2 for operations, more devices and machines can become vulnerable to attacks that abuse gaps in DDS. As noted by the Rochester Institute of Technology, differently than with ROS, "network vulnerabilities are directly tied to the functionality in ROS2 with their new DDS standard,"[51] and their conclusion is that most of the security issues found during the research are brought in by DDS, also confirmed by other researchers.[52]

2.2.8 Public and Private Transportation

Public transportation is an immense industry and another use case for DDS. There are 1.1 million railway lines worldwide, transporting 4,150B passenger-kilometers (pkm).[53] ProRail uses OpenSplice Vortex for distributed railway network management, in a system chosen by the Dutch railway network.[54]

There are also more than 10,000 (expected to become 44,000) airports in the world,[55] each with an average of 2.5 runways (up to 36).[56] An airport runway has thousands of control points, and even if only 1% of them were using DDS (conservatively), this would make roughly 250,000 DDS nodes (up to approximately 1.1 million). The ATC towers in Spain, the United Kingdom, and Germany use eProsima Fast-DDS, while Coflight Consortium (Thales in France, and Selex-SI in Italy) in European Air Traffic Management (EATM) use OpenSplice Vortex for flight data processing.[57] Nav Canada uses RTI Connext DDS for the same purposes.[58]

In the private transportation sector, the adoption of ICT is also growing significantly, with more than 54 million autonomous cars expected in 2024 (almost twice if compared to 2019 estimates).[59] The number of sensors (from 60 to 100 per car) and the real-time management needed for autonomous vehicles' control applications call for efficient middleware technology. DDS has been chosen by several autonomous driving solution developers to respond to these requirements.[60] As an example, Volkswagen's Driver-Assistive and Integrated Safety system uses DDS to combine radars, laser range finders, and video to assist safe operation. It tracks the driver's eyes to detect drowsiness, detects lane departures, avoids collisions, and helps keep the car within the lane. Apex.AI,[61] Clearpath, and Auterion's PX4 autopilot are just some examples of the new players in the autonomous driving landscape. We had a conversation with the engineers of one of the Indy Autonomous Challenge teams, who confirmed the use of DDS as their in-vehicle data plane.

3 Findings: Vulnerabilities and Exposure

The complexity of parsing dynamic and custom-defined data types (known to be prone to bugs) makes DDS a security-critical building block. A single vulnerability will impact the rest of the software stack. Aside from software vulnerabilities, we found DDS hosts being incidentally exposed on public-facing networks such as the internet.

Product name	Developer	HQ region	Open source	Core language	Developed Since
Fast-DDS	eProsima	EMEA	Apache License 2.0	C++	2014
Cyclone DDS	Eclipse Foundation project, driven by ADLINK	EMEA	Eclipse Public License 2.0 and Eclipse Development License 1.0	C	2011
OpenDDS	OCI	NABU	Custom	C++	2005
Connext DDS	RTI	NABU	Extensions are open source	C++	2005 (NDDS – 1995)
CoreDX DDS	TwinOaks	NABU	Not open source	C	2009
Gurum DDS	GurumNetworks	APAC	Not open source	C	

Table 2. A list of DDS implementations that we analyzed in this research.

Note: ADLINK confirmed that for the purpose of analyzing the RTPS layer, OpenSplice and Cyclone DDS are essentially the same. We decided to focus on Cyclone DDS because it is the most accessible and most actively developed of the two.

This section summarizes previous security work on DDS and covers the most relevant findings on the network, as well as the configuration attack surfaces of DDS and RTPS, concluding with other findings on the DDS software ecosystem. We disclosed our findings between April and December 2021 through Trend Micro's Zero Day Initiative (ZDI) program with the support of the Cybersecurity and Infrastructure Security Agency (CISA) given the importance of the applications for which DDS is used. As a result of our disclosure, the CISA released the ICS Advisory (ICSA-21-315-02 [4]).[62] We also detail our research methodology in the subsequent section, "Research Methodology and Technical Details."

3.1 Known DDS Vulnerabilities

As shown in Table 3, other researchers before us have analyzed DDS from a security standpoint and showed now-patched vulnerabilities that could allow local or remote attackers to compromise a system or conduct DoS-based attacks.

ATT&CK ICS	Surface	Reference	Scope	Weaknesses (CWE)
T0866: Exploitation of Remote Services	Network	2017-A-0097.NASL	Connext DDS	CWE-122: Heap-based buffer overflow
T0866: Exploitation of Remote Services		2017-A-0097.NASL		CWE-190: Integer overflow
T0814: DoS T0866: Exploitation of Remote Services		2017-A-0097.NASL		CWE-502: Deserialization of untrusted data
T0866: Exploitation of Remote Services		2017-A-0097.NASL		CWE-120: Buffer overflow
TA0043: Reconnaissance T0518: Remote System Discovery		CVE-2019-15135 [53]	All	CWE-319: Cleartext transmission of sensitive information
T0814: Denial of Service		CVE-2020-18734	Cyclone DDS	CWE-787: Out-of-bounds write
		CVE-2020-18735	Cyclone DDS	CWE-787: Out-of-bounds write

Table 3. Known DDS-related vulnerabilities with attacker pre-requisites and consequences highlighted (according to the MITRE ATT&CK® matrix)

We noted that four out of seven publicly known vulnerabilities have yet to be assigned a CVE ID, specifically for reconnaissance Nessus scripts exist. The lack of a CVE ID prevents tracking patches, exploits, and network signatures, making identification and monitoring also difficult for security teams and researchers.

We also noted that CVE-2019-15135 affects all DDS Security extensions, which adds confidentiality, integrity, and authentication to DDS.[63] When abused, CVE-2019-15135 allows an attacker to collect information about the DDS nodes in a network due to the verbosity of the DDS security layer. The layer sends cleartext metadata such as endpoint identifiers, internal IP addresses, vendor, and product version. The vulnerabilities that we categorize as CWE-406 in Table 4 also target the discovery protocol, but this time we found that it can be exploited to reflect and amplify network traffic.

3.2 New DDS Vulnerabilities

As DDS is the default middleware of ROS 2, all the vulnerabilities we discovered also affect ROS 2, as highlighted in Table 4.

ATT&CK ICS	Surface	Vector	CVE	Scope	CVSS	Weaknesses (CWE)
T0804: Brute Force I/O T0814: DoS T0827: Loss of Control T0880: Loss of Safety T0802: Automated Collection T0846: Remote System Discovery T0856: Spoof of Reporting Message	Network	RTPS discovery packet	CVE-2021-38425	Fast-DDS, ROS 2	7.5	CWE-406: Network amplification
			CVE-2021-38429	OpenDDS, ROS 2	7.5	
			CVE-2021-38487	Connext DDS, ROS 2	7.5	
			CVE-2021-43547	CoreDX DDS, ROS 2	7.5	
		Malformed RTPS packet	CVE-2021-38447	OpenDDS, ROS 2	8.6	CWE-405: Network amplification
			CVE-2021-38445	OpenDDS, ROS 2	7.0	CWE-130: Improper handling of length
			CVE-2021-38423	Gurum DDS, ROS 2	8.6	CWE-131: Incorrect calculation of buffer size
			CVE-2021-38435	Connext DDS, ROS 2	8.6	
			CVE-2021-38439	GurumDDS, ROS 2	8.6	CWE-122: Heap-based buffer overflow
T0862: Supply Chain Compromise T0839: Module Firmware T0873: Project File Infection	Config.	XML file	CVE-2021-38427	Connext DDS, ROS 2	6.6	CWE-121: Stack-based buffer overflow
			CVE-2021-38433	Connext DDS, ROS 2	6.6	
			CVE-2021-38443	Cyclone DDS, ROS 2	6.6	CWE-228: Improper handling of syntactically invalid structure
			CVE-2021-38441	Cyclone DDS, ROS 2	6.6	CWE-123: Write-what-where condition

Table 4. A summary of our findings across the main DDS implementations and standard specification

The vulnerabilities affecting the network attack surface allow an attacker to perform spoofing, reconnaissance, automated data collection, and denial of service (DoS), affecting the control of an exposed system. The vulnerabilities affecting the configuration attack surface can affect the developer or system integrator, potentially compromising the integrity of the software supply chain (which means an attacker targets a DDS developer or system integrator when exploiting one of these vulnerabilities).

3.2.1 Vulnerabilities in the DDS Standard Specification

The built-in RTPS discovery protocol is used in peer-to-peer networks to discover the locator of each participant (such as IP address and UDP/TCP port or offset in shared memory). The "chatty" nature of this discovery protocol and the fact that it expects a reply from each contacted participant, paired with easy-to-spoof transport protocols such as the User Datagram Protocol (UDP), make RTPS vulnerable to network reflection and amplification. Confidentiality and authenticity for this data is not protected even with DDS Security, making it possible for an attacker to spoof the information.

CVE	Scope	Partially mitigated*	BAF	% of attack duration (Total experiment duration = 139s)
CVE-2021-38425	Fast-DDS, ROS 2	master branch	9.875	100.0
CVE-2021-38429	OpenDDS, ROS 2	>= 3.18.1	18.68	24.17
CVE-2021-38487	Connext DDS, ROS 2	>= 6.1.0	2.011	84.17
CVE-2021-43547	CoreDX DDS, ROS 2	> 5.9.1	32.82	18.14

Table 5. The network reflection and amplification vulnerability with bandwidth amplification factor (BAF) is calculated as the ratio between outbound and reflected traffic

Note: Implementations not reaching 100% attack duration likely have a timeout mechanism.
(*) A full mitigation will require relevant changes in the RTPS specification.

To measure the amount of reflected traffic, we created a setup similar to the situation depicted in Figure 6 and let the DDS nodes run for as long as they would keep running. The longest running node was based on Connext DDS (at 139 seconds), which we kept as a reference. Table 5 shows that the BAF is greater than one, meaning there is asymmetric network flows although the values are at the order of magnitude lower than modern amplification attacks (note that Memcached can reach 10,000 to 51,000 BAF).[64] However, the network bandwidth in embedded systems is also lower than, for example, what can be afforded by internet nodes.

This built-in discovery feature can be abused by an attacker for remote discovery and fingerprinting. As described in our methodology ("Source Code and Binary Fuzzing"), we sent RPTS discovery probes to the entire IPv4 space (except for the no-scan subnets) and received answers from 643 hosts (excluding obvious honeypots). Notably, tens of hosts never stopped sending traffic to us, even if we only sent them a single 288-byte packet.

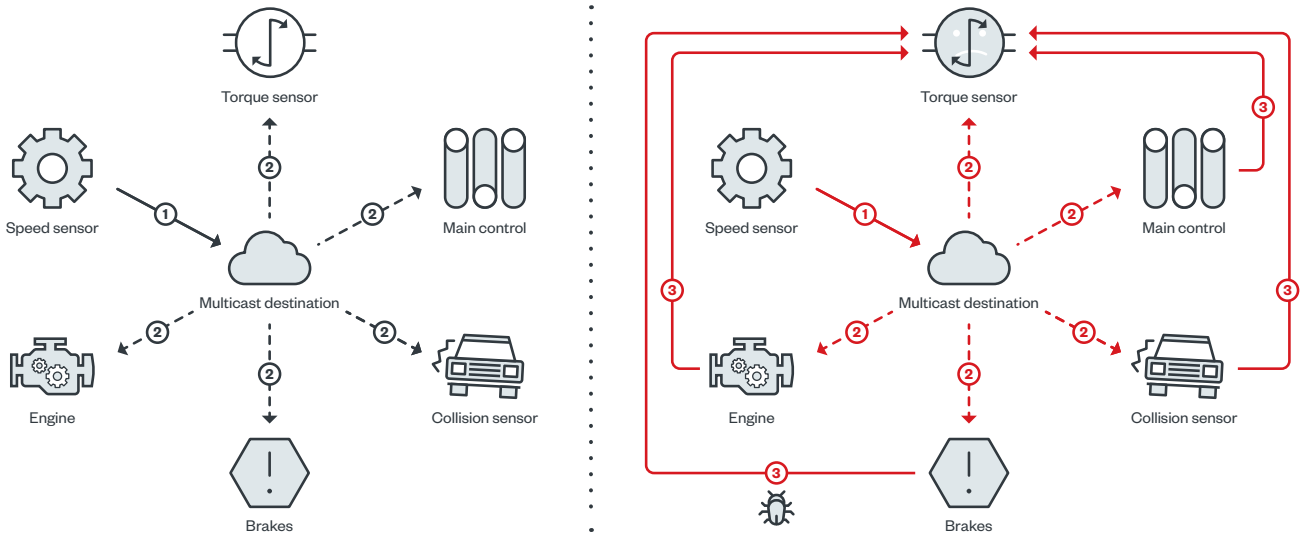


Figure 6. By spoofing the participant locator, any participant can pretend to be anyone else, and the receiver is forced (as per specification requirement) to answer back until a valid acknowledgement is received

This new network-reflection vulnerability that we found is not the only instance of a specification-level vulnerability. In 2015, researchers analyzed the DDS Security 1.0 specifications and theorized a scenario where unauthorized nodes are able to inject data into or read data from the DDS network.[65] This possibility was confirmed in 2017 through practical experiments that found that an attacker could be allowed to perform network reconnaissance.[66]

In 2018, another demonstration showed that the default and most common settings of the DDS security extension do not prevent a malicious DDS entity to act as a man-in-the-middle within a DDS network. By anonymously subscribing to existing data streams (called "topics") and republishing multiple (even altered) copies of such data, the authors demonstrated four other attacks in an air traffic control scenario, leveraging only corner cases in the DDS specifications.[67]

In 2019, researchers demonstrated that the most recent revision of the DDS security specifications (version 1.1) allows an attacker outside the DDS network to perform reconnaissance through passive sniffing.[68] This was possible because the cryptographic parameters are exchanged in cleartext during the handshake phase, thus exposing identifying information about the nodes in a network (CVE-2019-15135). The authors used this leaked information to show that an attacker can reconstruct the network topology, which facilitates subsequent selective DoS or exploitation of vendor-specific vulnerabilities.

3.2.2 Vulnerabilities in the Main DDS Implementations

Much like any application layer, data serialization and deserialization are the most critical functions because they handle external (and therefore untrustworthy) data. This is where “data” and “instructions” intermix. Malformed data can turn into unwanted application behavior, ranging from crashes in memory read-or-write primitives. This means that an attacker can send data to the UDP socket of a running DDS/RTPS node, and has the opportunity to “play” with RTPS (de)serialization functions, thereby potentially triggering a vulnerability. The same reasoning applies to parsing XML files, with the difference that these are not delivered via network sockets but via static files.

By focusing on RTPS (de)serialization and XML parsing functions, we discovered nine vulnerabilities allowing an attacker read-or-write access to the stack or the heap, and up to 6 bytes into the instruction pointer. We summarize this finding in Table 6.

CVE ID	Scope	Patched	Description
CVE-2021-38447	OpenDDS, ROS 2	>= 3.18.1	Slowloris behavior by forcing the allocator to allocate 1-byte chunks in a loop via malformed RTPS payload.
CVE-2021-38445	OpenDDS, ROS 2	>= 3.18.1	Failed assertion condition causing runtime to exit abruptly via malformed RTPS payload.
CVE-2021-38423	GurumDDS, ROS 2	Unpatched	Segmentation fault by forcing deserialization of a malformed RTPS packet.
CVE-2021-38435	Connext DDS, ROS 2	>= 6.1.0	Segmentation fault by forcing deserialization of a malformed RTPS packet.
CVE-2021-38439	GurumDDS, ROS 2	Unpatched	Heap overflow causing segmentation fault by forcing deserialization of malformed RTPS data.
CVE-2021-38427	Connext DDS, ROS 2	>= 6.1.0	Stack overflow via malformed XML file with up to 6 bytes write access to instruction pointer.
CVE-2021-38433	Connext DDS, ROS 2	>= 6.1.0	Stack overflow via malformed XML file with up to 6 bytes write access to instruction pointer.
CVE-2021-38443	Cyclone DDS, ROS 2	>= 0.8.1 master branch	Multi-byte heap-write via malformed XML file.
CVE-2021-38441	Cyclone DDS, ROS 2	>= 0.8.1 master branch	Null dereference and heap-write primitive (up to 8-bytes) via malformed XML file.

Table 6. Vulnerabilities in the network and configuration surface of the six target DDS implementations

3.2.2.1 Network Attack Surface

The main focus of our technical analysis has been on using coverage-guided fuzz-testing to find vulnerabilities in the RTPS-parsing routines of all DDS implementations. This led to the discovery of various memory errors (CVE-2021-38423, CVE-2021-38439, CVE-2021-38435) that an attacker in the network could abuse to abruptly interrupt normal operations and, in some cases, gain code-execution

capabilities (recall that not all embedded systems can afford memory protections such as W^X or address space layout randomization (ASLR), which are enabled by default on server-grade hardware and software).

We show concrete examples in the subsequent sections on how to find good fuzz targets for RTPS implementations and prepare them for popular frameworks like OSS-Fuzz and UnicornAFL. As part of our research, we release the fuzz targets packaged in a format compatible with the OSS-Fuzz repository. As of November 2021, all three open-source implementations of DDS are integrated into the OSS-Fuzz repository and are being continuously fuzzed.

3.2.2.2 Configuration Attack Surface

In addition to focusing on the network, we noticed that most DDS implementations are highly dependent on XML files for configuration. XML files can represent a stealthy attack vector because these are text-based. By first scripting a RADAMSA-based file fuzzer — later converted into an AFL-based fuzz harness — we found XML-parsing vulnerabilities in almost all DDS implementations (CVE-2021-38441, CVE-2021-38443, CVE-2021-38427, and CVE-2021-38433), and one implementation using an XML library unmaintained since 2010 (CVE-2021-38437). An attacker could trigger such vulnerabilities with a simple, malformed XML file.

Details about the impact of these vulnerabilities and how we used fuzzing to find them are in the subsequent sections (“Research Methodology and Technical Details”).

3.3 DevOps Failures in the DDS World

We looked at the auxiliary tools used by DDS users, developers, and system integrators such as Docker images, development environment, and continuous integration systems and found that DevOps flows are not always built with security in mind. Sometimes, projects are created with outdated Docker images in official repositories, cloud-integration services are built with a fragile threat model, and a development backend has been left fully exposed. This leaves a wider vulnerability and attack surface that threat actors can exploit.

3.3.1 An exposed CI/CD pipeline

While monitoring for exposed continuous-integration/continuous-deployment (CI/CD) systems via Shodan, we found that one of the DDS developers left their custom CI/CD environment fully exposed to the internet with default credentials.

Unfortunately, we did not receive a response from this vendor despite our numerous attempts to inform them of this gap, including our attempts through brokers and CERTs. Fortunately, the exposed system was properly locked down after a few months. If left exposed, a malicious actor could have wiped, stolen, or trojanized their most valuable intellectual property (the source code).

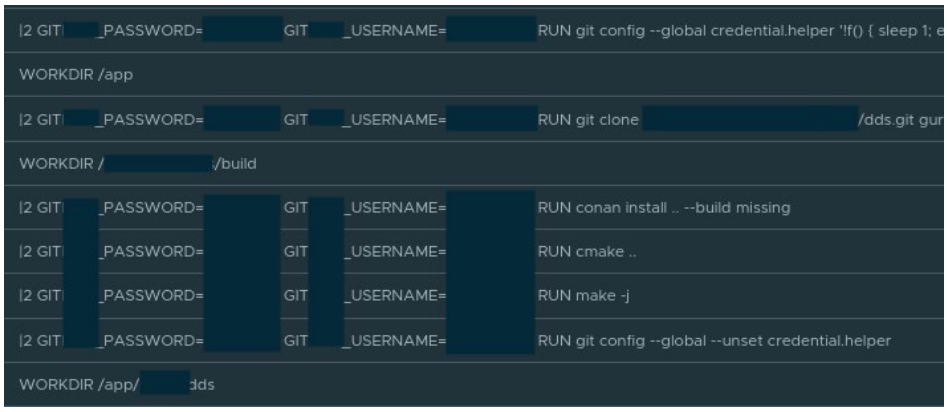


Figure 7. An exposed CI system used by a DDS developer with default access credentials in plaintext

3.3.2 Outdated Docker Images with Vulnerable Packages

To complement our understanding of the security posture of DDS vendors, we briefly looked at the available Docker images related to or based on DDS implementations.

Table 6 shows that most of the Docker images related to DDS products are outdated or, with a few exceptions, contain software packages affected by known vulnerabilities. Figure 8 shows an example output for two of the images.

			Automatically Detected Vulnerabilities (docker scan) - CVSS		
Docker Image	**Downloads**	**Last Updated**	**Low**	**Medium**	**High**
objectcomputing/opendds	10K+	Nov. 2021	48	30	2
objectcomputing/opendds_ros2	3.5K	Jul. 2021	40	30	2
objectcomputing/rmw_ros2_depends	2.5K	Nov. 2020	49	75	5
eprosima/micro-xrce-dds	114	Jul. 2021	19	6	1
eprosima/fast-dds:2.4.0,2.4.1	8K+	Nov. 2021	0	0	0
ros:foxy	10M+	Nov. 2021	31	11	0

Table 7. Known vulnerabilities found in the Docker images related to DDS

Figure 8. Screenshots of Docker scan (via Snyk) on DDS-related Docker images

3.4 RTPS and DDS Hosts Exposed on Public-facing Networks

We discovered hundreds of distinct IPs reflecting packets to our collector, with some of them still continuing to send us data from day zero. We received data from all six DDS "flavors," plus one (namely ETRI Technology) that we were initially unaware of.

How we post-processed the data:

- We extracted printable strings, and used regular expressions to extract URLs, IP addresses, and version numbers from any payload after each RTPS submessage header.
- We enriched each IP with metadata from the Maxmind GeoIP[69] database, which also contains information about country, ISP, and organization (sometimes the latter two are the same).
- We pivoted the data along various axes to obtain breakdowns per country, vendor, ISP, organization, and presence of leaked information (such as private IPs).

Table 8 shows the data classified according to DDS vendor, confirming that our initial selection of the six DDS implementations matches the popularity of these platforms. We used the version information (when available) to estimate how many services are running outdated versions of DDS. Note that "N/A" means that we were unable to find any version information, making the estimation a lower bound of the real numbers. We decided to avoid any deeper scanning that could potentially show the said information for legal and ethical matters: Since DDS is a protocol used in a controlled system, deeper scanning may trigger unwanted behavior, and the risks of affecting systems connected to the physical world are higher.

Vendor	# Distinct IPs	# of IPs with outdated DDS	# of distinct RTPS payloads	Total data received [bytes]	Average RTPS payload size [bytes]
eProsima - Fast-RTPS	426	N/A	1,838	264,008	1,436,387
PrismTech Inc. - OpenSplice DDS	126	39	1,163	452,668	3,892,244
Real-Time Innovations, Inc. - Connext DDS	65	19	473	100,764	2,130,317
ADLINK - Cyclone DDS	14	10	102	23,652	2,318,824
ETRI Electronics and Telecommunication Research Institute	8	N/A	85	62,968	7,408
TwinOaks Computing, Inc. - CoreDX DDS	4	N/A	26	3,424	1,316,923
Object Computing Incorporated, Inc. (OCI) - OpenDDS	1	N/A	3	252	84

Table 8. Exposed DDS endpoints by vendor

Note: We were not aware that ETRI Technology was a DDS vendor when we selected the six DDS implementations of the study. The estimated number of outdated instances is a conservative guess based on the limited version information when present in the RTPS responses.

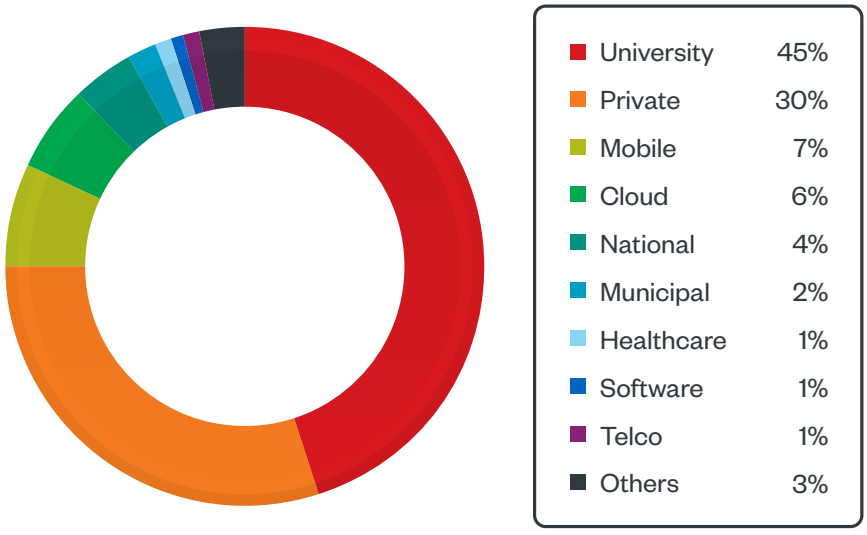


Figure 9. According to publicly available IP metadata, the organization types or verticals most affected by the exposure include universities and research centers, followed by a private internet service provider (ISP), which is an umbrella category that hides many other business types. Notably, we found exposed DDS instances hosted on all major cloud providers.

3.4.1 Leaked Private Network Details

Without engaging any in-depth scanning, the RTPS packets that we received sometimes contained data that is supposed to be confidential, which may give an advantage to an external attacker willing to learn about the internal details of a network.

Almost 63% of the publicly accessible endpoints exposed at least one private IP (for example, 172.16.0.8 and 192.168.3.10), a total of 202 private IPs. In addition, we found seven Rebus[70] URLs, which reference internal endpoints. All the URLs contained a keyword that uniquely identified a leading manufacturer of telco equipment. All the URLs were also leaked by DDS endpoints found in the network of the same Swedish ISP. We believe that the said ISP uses the DDS to manage broadband equipment via a lifecycle management (LCM) API.

URL	# of distinct leaking IPs	Sample leaking IP
rebus://189e7bcfd0d57544/*/com.[REDACTED].lcm/0.6.0	4	5.REDACTED.223
rebus://bc9e87ae06b3ff7d/*/com. [REDACTED].lcm/0.6.0	1	80.REDACTED.182
rebus://bab9fb616d333a5b/*/com. [REDACTED].lcm/0.6.0	1	78.REDACTED.219
rebus://38a05b31a1be9bfd/*/com. [REDACTED].lcm/0.6.0	1	83.REDACTED.232
rebus://271e3b4b87cf5fa1/*/com. [REDACTED].lcm/0.6.0	1	84.REDACTED.123
rebus://bc9e87ae06b3ff7d/b5a0002/com. [REDACTED].pfs.deviceInfo/1.0.0	1	80.REDACTED.182
rebus://189e7bcfd0d57544/cd40002/com. [REDACTED].pfs.deviceInfo/1.0.0	1	155.REDACTED.59

Table 9. Rebus URLs and a sample of the corresponding IPs being leaked

Following the zero trust principle, every component of a software supply chain should at least be analyzed for the presence of known security vulnerabilities. It is also a common best practice to continuously update software versions. Despite these common security practices, related security incidents[71] keep reminding everyone that Docker images and Docker Compose files are mistakenly used “as is,” even in critical deployments.

4 Attack Scenarios: Autonomous Driving Proof of Concept

The impact of any vulnerability in the DDS can be fully appreciated only by considering how it's embedded in a final product, considering that the middleware is at the very beginning of the software supply chain. However, showcasing the effects of an exploit is not as direct. First, because each vertical will have different requirements, priorities, and operational conditions, making it difficult to create representative attack scenarios. Second, the importance of the systems where DDS is used puts barriers on accessing testbed devices for offensive research purposes.

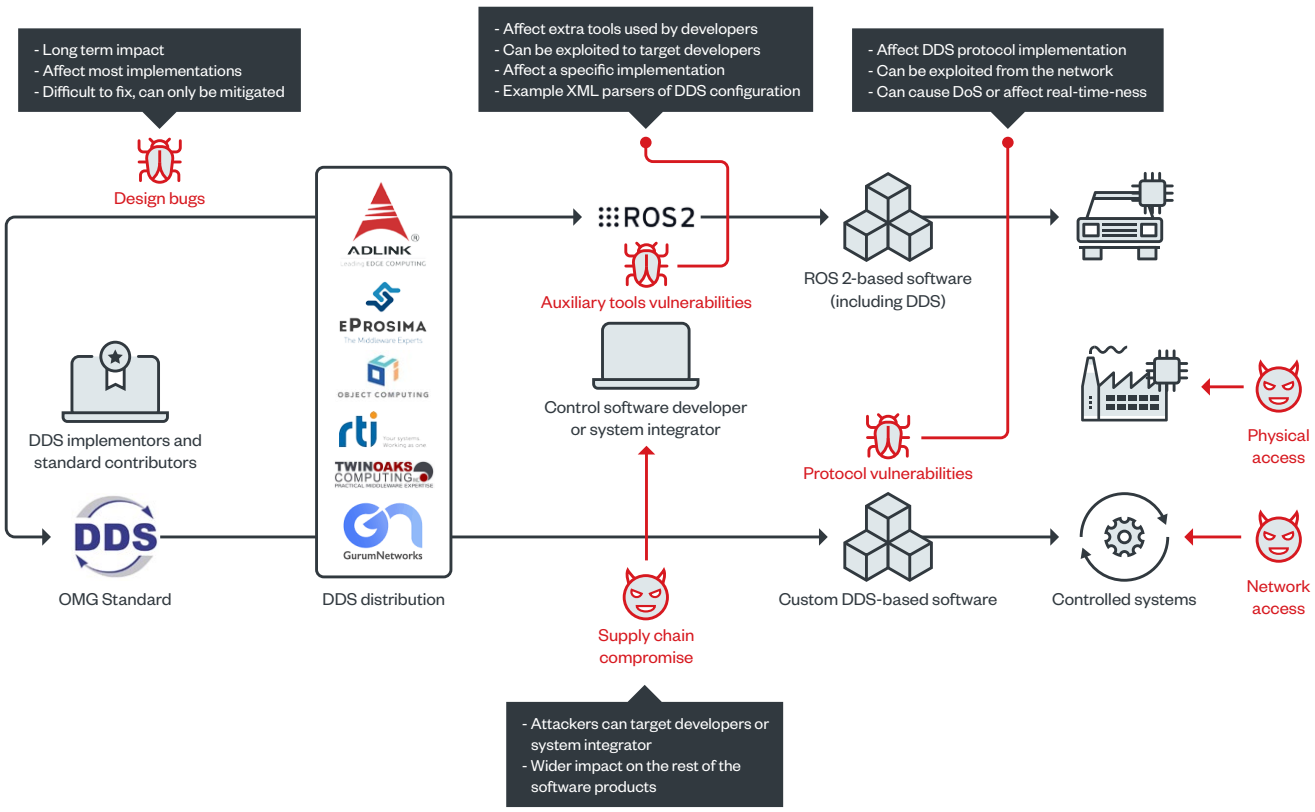


Figure 10. We discovered vulnerabilities affecting the design and implementations of DDS

For those two reasons, we followed three approaches:

1. We launched an attack against a software-based simulation of an autonomous-driving mobile robot based on the ROS 2 stack, which uses DDS as its default middleware.
2. We reproduced the same attack against a small, physical autonomous-driving mobile robot prototype (again based on ROS 2) in a controlled environment, the same prototype used to develop a full-fledged autonomous-driving control software.
3. To cover the other sectors, we used the MITRE ATT&CK Industrial Control Systems (ICS) framework to describe an abstract attack scenario by visualizing where and how a hypothetical attacker could take advantage of security gaps in DDS.

4.1 Simulation: Attacking an Autonomous Driving Platform

Before using DDS to control a real, physical machine, we created a simulated environment using Gazebo, a simulator used by roboticists to test algorithms, design robots, perform regression testing, and train AI systems using realistic scenarios. The software stack that runs on the simulated vehicle is the same that runs on the real TurtleBot3 (see the next section), and the simulator guarantees fidelity of the virtual world thanks to the physics engines.

CVE ID	Description	Scope	CVSS	Root Cause
CVE-2021-38447	An attacker sends a specially crafted packet to flood target devices with unwanted traffic, resulting in a DoS condition	OpenDDS, ROS 2	8.6	Resource exhaustion
CVE-2021-38445	Do not handle a length parameter consistent with the actual length of the associated data	OpenDDS, ROS 2	7.0	Failed assertion

Table 10. Vulnerabilities used to showcase the effect of a DoS attack against a simulated autonomous-driving platform

The teleoperated autonomous-driving mobile robot runs a ROS 2 graph that moves it in a 3D-simulated world, avoiding obstacles thanks to a Lidar sensor, exactly like the sensor used by TurtleBot3.

Both CVE-2021-38447 and CVE-2021-38445 affects OpenDDS, leading ROS 2 nodes to either crash or execute arbitrary code due to DDS not handling the length of the PID_BUILTIN_ENDPOINT_QOS parameter within RTPS's RTPSSubMessage_DATA submessage properly. With the Scapy RTPS layer (), creating an exploit for these vulnerabilities is as easy as setting the parameterLength to 4 null bytes, as exemplified in Figure 10.

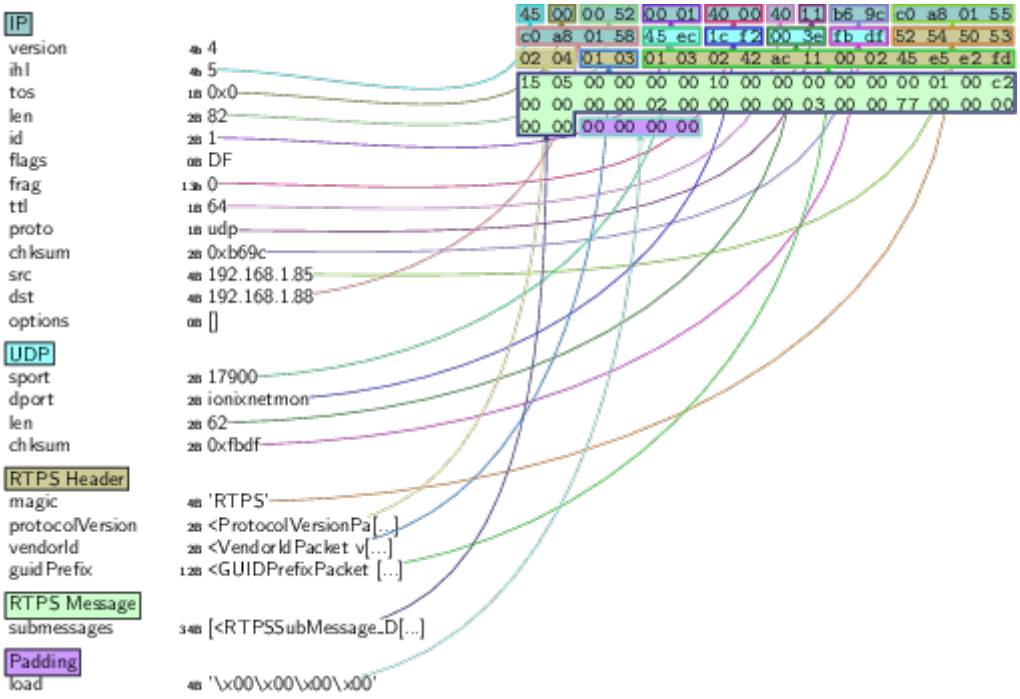


Figure 11. Sample exploit payload for CVE-2021-38447 and CVE-2021-38445

Figure 11 shows the mobile robot moving in a 3D environment with nine obstacles (seen as white circles), with the Lidar sensor “seeing” clear ways (blue areas) that the robot can take. On the left side, the controlling loop printing debug information about the velocity along each axis can be seen.

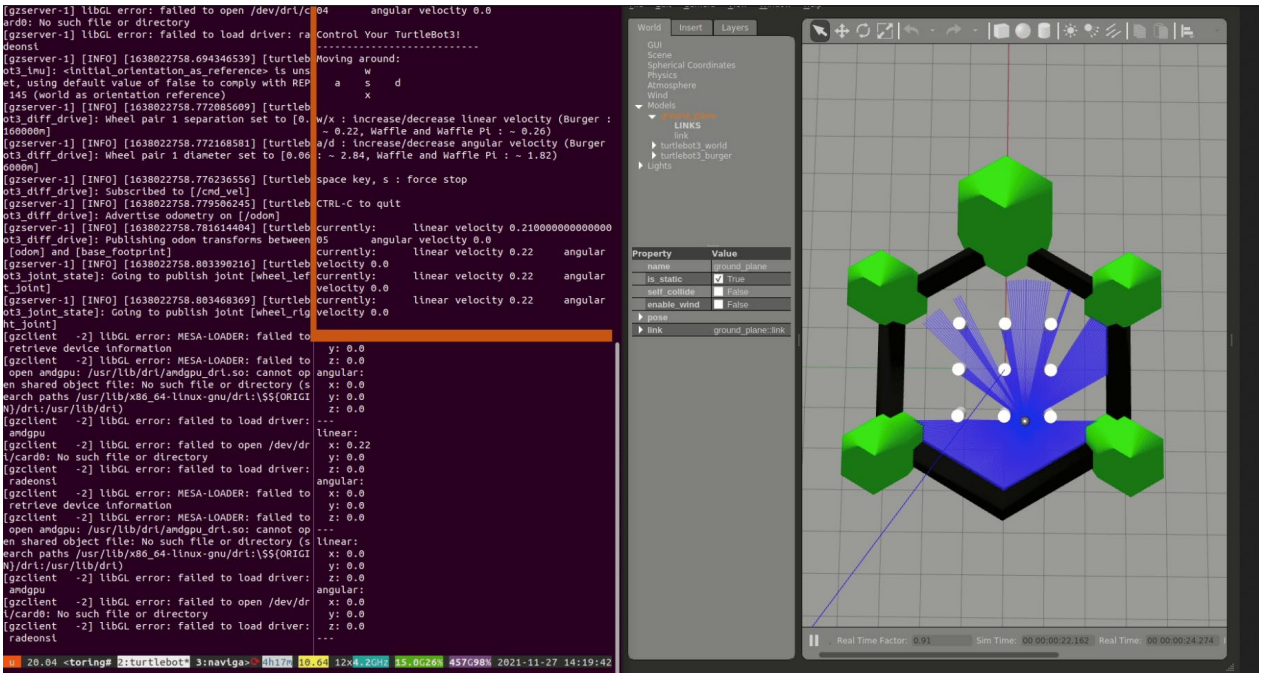


Figure 12. The output of the operating node (see the velocity) that sends a command to the robot via DDS (left) and the tele-operated autonomous-driving mobile robot moving in a 3D-simulated world and avoiding obstacles “seen” by a Lidar sensor (right)

We created an attacker on the network that sends an RTPS payload with parameterLength set to 4 null bytes, causing the DDS layer underneath the ROS 2 node to crash abruptly. The Lidar sensor is still sending information about obstacles, but this is not delivered in time (if at all), causing the control loop to miss deadlines. Consequently, the robot will be blind to obstacles or won't see them in time.

4.2 Experiment: Crashing a Miniature Autonomous-Driving Mobile Robot

After successfully testing CVE-2021-38447 and CVE-2021-38445 on a simulated robot, we replicated the same setup in the physical world using a TurtleBot3.[72] This time we used CVE-2021-38435 against RTI Connext DDS, which causes a segmentation fault. The TurtleBot3 is a small but powerful mobile robot used to prototype autonomous-driving control algorithms. As shown in Figure 13 and as described on the manufacturer's website, the TurtleBot3 is used to design and test autonomous vehicles driving in miniature smart cities. We installed on the TurtleBot3 the same software stack we used to control the virtual robot in the 3D simulated world, and used stuffed toy animals as obstacles, as shown in Figure 14.

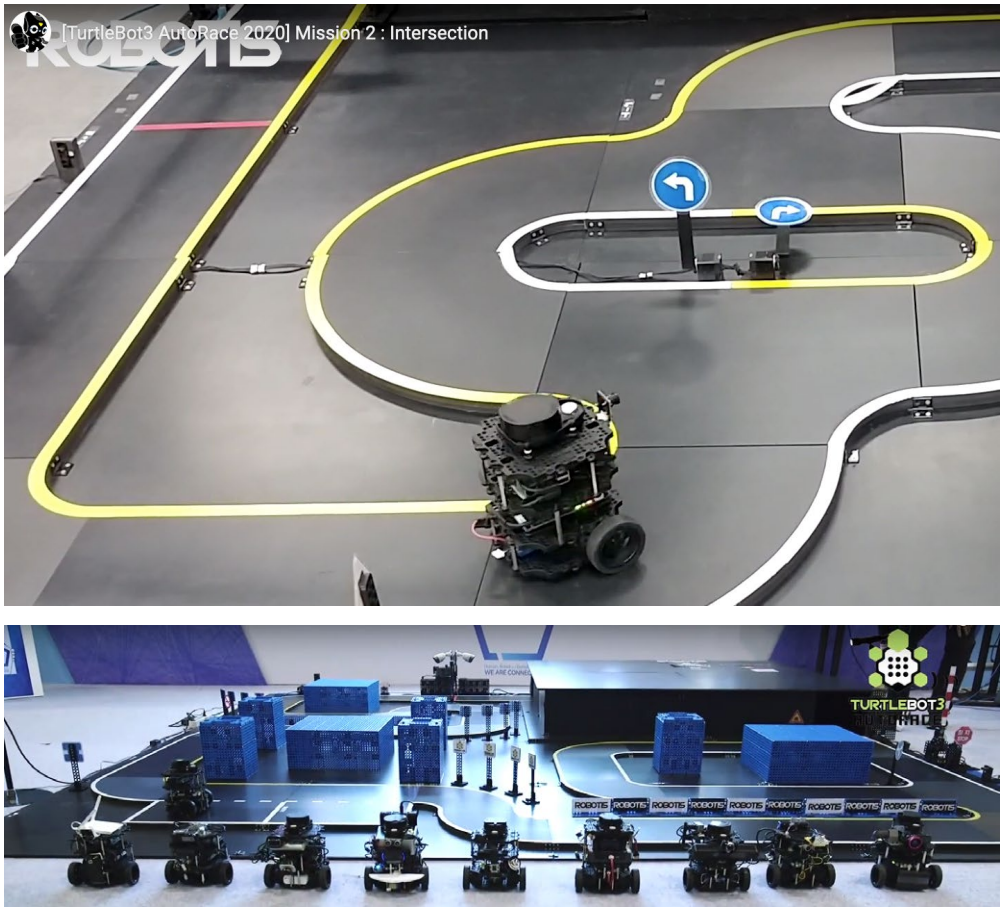


Figure 13. The TurtleBot3 autonomous vehicle prototype in a testing lab by ROBOTIS. We used the same device in a home-based setting to showcase the effect of a DDS node crashing while operating the mobile robot. Images courtesy of ROBOTIS.[73]

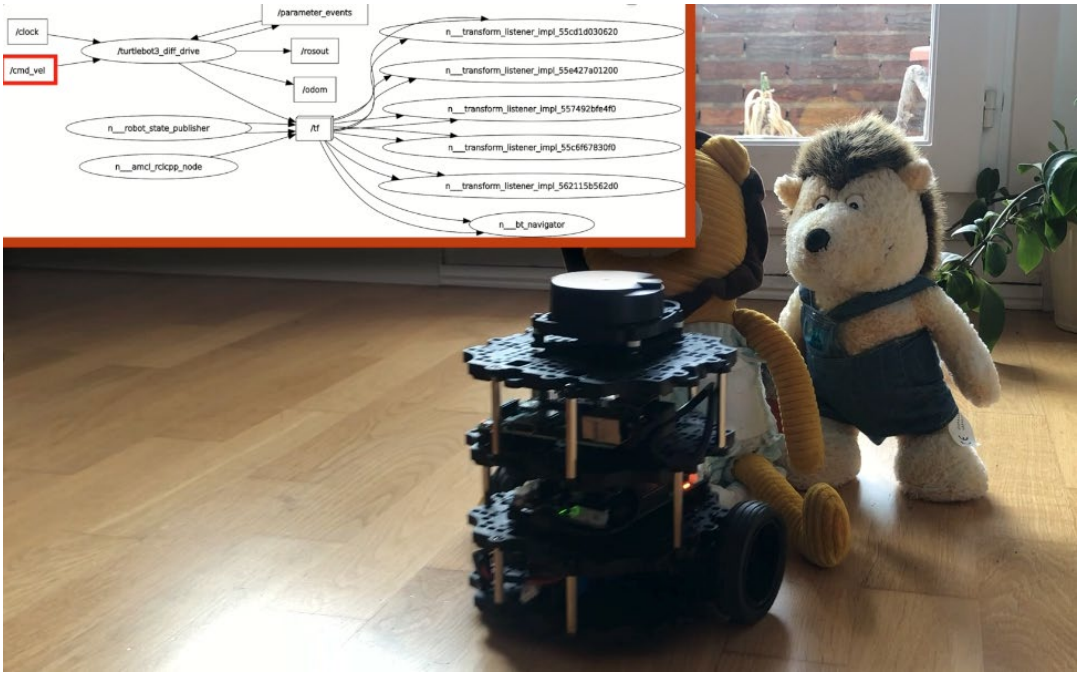


Figure 14. The ROS 2 computational graph (top left, with velocity command "/cmd_vel") of the teleoperated autonomous-driving mobile robot. The CVE-2021-38435 exploit against the RTI Connext DDS layer of ROS 2 prevents the robot from seeing the obstacle in time to stop.

In normal conditions, the information about obstacles sent by the Lidar sensor is received and processed before the deadline. The process ensures there is enough time for the control signal to be sent, received, and actuated by the motors to avoid the obstacles. When we exploit CVE-2021-38435, the RTI Connext DDS node crashes and causes the ROS 2 node to crash as well. This experiment confirms the exploitability of CVE-2021-38435 from the network.

4.3 DDS Vulnerabilities Through the MITRE ATT&CK ICS Lens

We conclude this section by contextualizing the CVEs affecting DDS products within the MITRE ATT&CK ICS framework. The result is intended for the use of security engineers for threat modeling or to prioritize vulnerabilities, even future ones. Table 11 highlights the techniques and tactics that are more likely associated with the CVEs that we discovered.

Initial Access	Persistence	Evasion	Discovery	Lateral Movement
T0886 Remote Services	**T0839** Module Firmware	**T0856** Spoof Reporting Message	**T0846** Remote System Discovery	**T0886** Remote Services
T0862 Supply Chain Compromise	**T0862** Project File Infection			

Collection	Command and Control	Inhibit Response Function	Impair Process Control	Impact
T0802 Automated Collection	**T0869** Standard Application Layer Protocol	**T0814** Denial of Service	**T0806** Brute Force I/O	**T0827** Loss of Control
			T0839 Module Firmware	**T0880** Loss of Safety
			T0856 Spoof Reporting Message	

Table 11. The MITRE ATT&CK ICS matrix contextualizes the vulnerabilities that affect the DDS implementations and specifications

Successful exploitation of these vulnerabilities can:

- Allow an attacker to perform discovery (TA0102) by abusing the discovery protocol (T0846): discovery must be possible even if DDS Security extension is present, which makes DDS easily discoverable. This empowers the attacker with reconnaissance capabilities, which we confirmed by conducting an internet-wide scanning campaign, identifying hundreds of endpoints.
- Facilitate initial access (TA0108) via
    - exploitation of remote services (T0866, T0886), as shown in the previous section by crashing a mobile autonomous vehicle, or
    - supply chain compromise (T0862), which attackers have been increasingly leveraging in many critical software stacks. To corroborate this hypothesis, we found fully exposed CI/CD pipelines from one DDS vendor, which could have allowed an attacker to modify the source code of that implementation at their will.

The consequences of successful exploitation, in any of the critical sectors where DDS is used, range from:

- inhibiting response function via denial of service (T0814),
- impairing control processes via brute force (T0806),
- attacking impact range from loss of control (T0827) or availability (T0826), to loss of safety (T0880).

The DDS protocol itself can also be abused to create an efficient C&C channel (T0869).

5 Research Methodology and Technical Details

Figure 15 provides an overview of the research methodology we followed. We analyzed each DDS implementation from two main angles: network and source — or binary — code. We wanted to have a deep understanding of the low-level details of the RTPS network layer to craft arbitrary test packets, and prove that it is possible for an attacker to mass scan a network and map the attack surface. In a parallel investigation, we manually studied the original or decompiled source code, with the main goal to find good fuzz targets (i.e., functions that receive and process untrustworthy data like network packets).

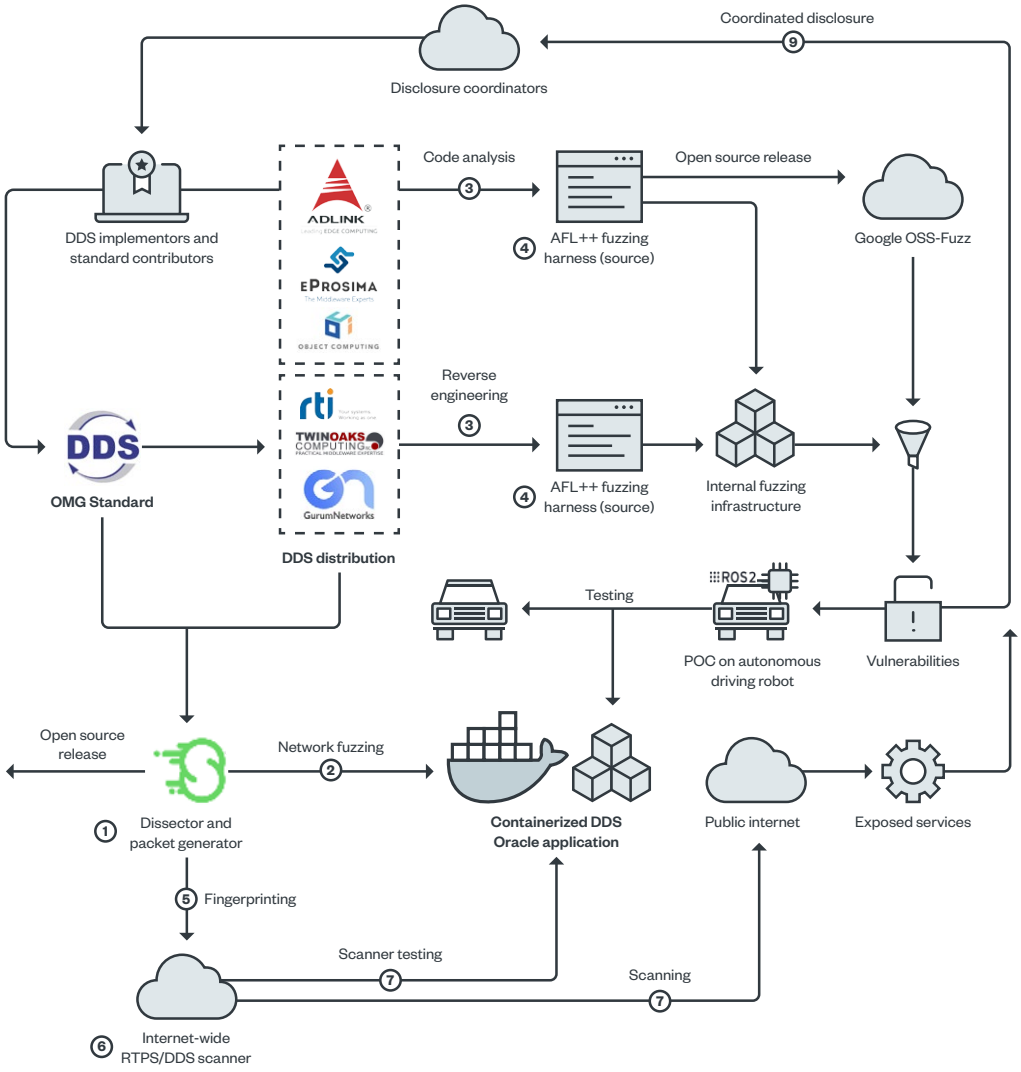


Figure 15. Diagram of the research methodology and workflow we used: The numbered items indicate the sequence of steps we followed throughout this research, beginning with the development of a packet dissector

5.1 A New Scapy Layer to Dissect and Forge RTPS and DDS Data

Although Wireshark already includes an RTPS dissection plugin, we needed something more scriptable. Since we spent some time manually crafting RTPS packets at the beginning, we decided to develop a RTPS Scapy-based dissector. We released the resulting Scapy layer as open-source code under the GNU General Public License v2.0.[74]

5.1.1 Crafting RTPS probes with Scapy

Without going into the details of our Scapy RTPS implementation, note that it can be used to programmatically create RTPS packets by writing Python code (as shown in the Figure 14 computational graph), like any other Scapy layer. In practice, that's seldom what a researcher would do, especially for "thick" protocols with lots of options.

Figure 16. With the Scapy RTPS layer, a developer can create arbitrarily complex (and unexpected but valid) RTPS packets

Instead, the developer's typical workflow can be:

1. **Intercept traffic.** Use Tcpdump or Wireshark to collect the traffic generated by the "hello world" example typically provided with a DDS distribution.
2. **Extract UDP payload.** Use Scapy (or manually via Wireshark) to select the packet of interest and extract the UDP payload (which contains the RTPS layer, as shown in Figure 15).
3. **Dissect with the RTPS class.** Pass the extracted payload to the Scapy RTPS class, which will automatically dissect it.
4. **Generate Python code automatically.** Use Scapy's built-in .command() method to output the Python code that will declaratively generate the packet that has just been dissected. If necessary, modify the packet so obtained (as exemplified in Figure 21).

5. **Test modified packet against oracle.** Either use Scapy's built-in send()/sendp() functions or Python's socket module to send the packet to a target oracle DDS endpoint and check if it triggers the desired behavior.

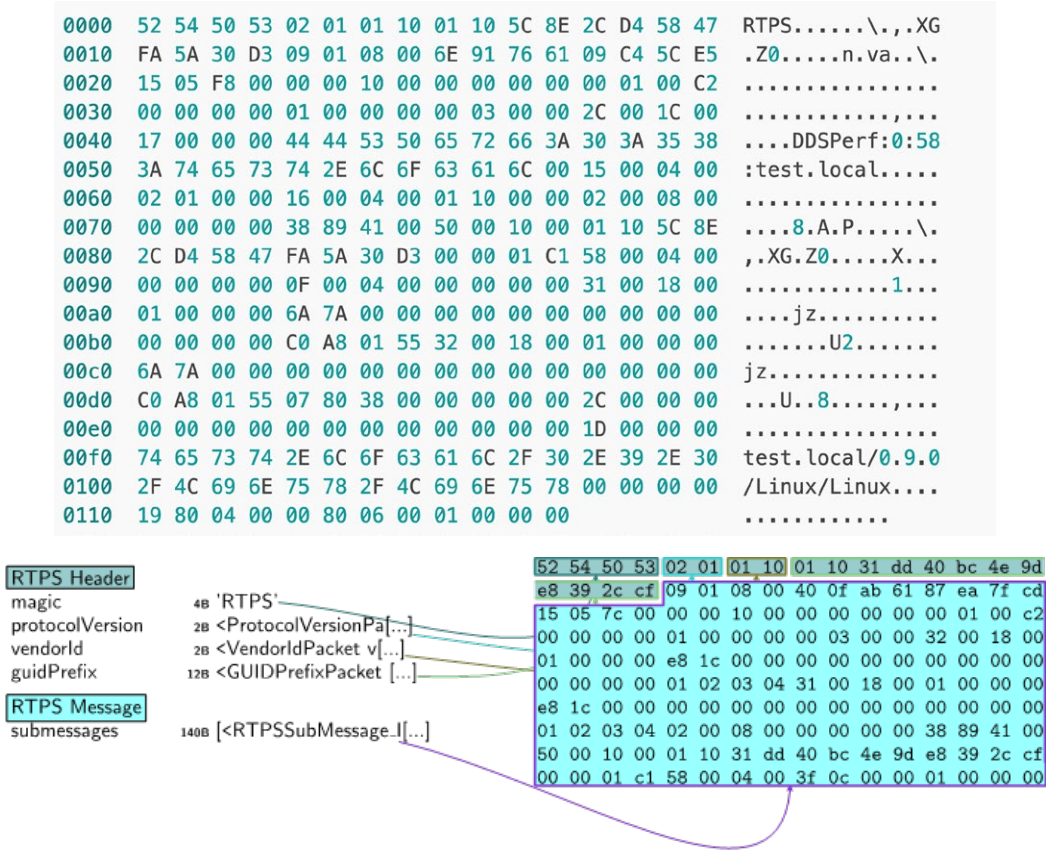


Figure 17. The UDP payload contains the RTPS header and subsequent data

The following section briefly describes how, almost by accident, we discovered the amplification vulnerability while dissecting and modifying packets during the early stages of the development of our Scapy RTPS layer.

5.1.2 Finding the Amplification Vulnerability

Although network fuzzing via Scapy was not directly effective in our research, creating a Scapy layer helped and motivated us to investigate all the RTPS packets' fields in depth. The activity led us to find the amplification vulnerability (CVE-2021-38425, CVE-2021-38429, CVE-2021-38487, CVE-2021-43547). In the long run, we recommend that developers and users leverage our Scapy RTPS layer — or similar libraries — as a building block for building continuous network fuzzers for RTPS and DDS.

The goal of the RTPS discovery phase is to send "probe" packets (e.g., to multicast addresses) and wait for responses from new locators. Locators could be IP-port pairs (see the PID_DEFAULT_UNICAST_LOCATOR in Figure 16, right side of the screenshot) or memory offsets in a shared-memory transport. Before reading the specifications in depth, we assumed that an RTPS discovery packet would allow us to restrict the locator to the IP addresses within the network the machine is connected to, and would not blindly send RTPS data to any IP-port found in the locator field. On a second read, however, this is exactly how discovery works by design.

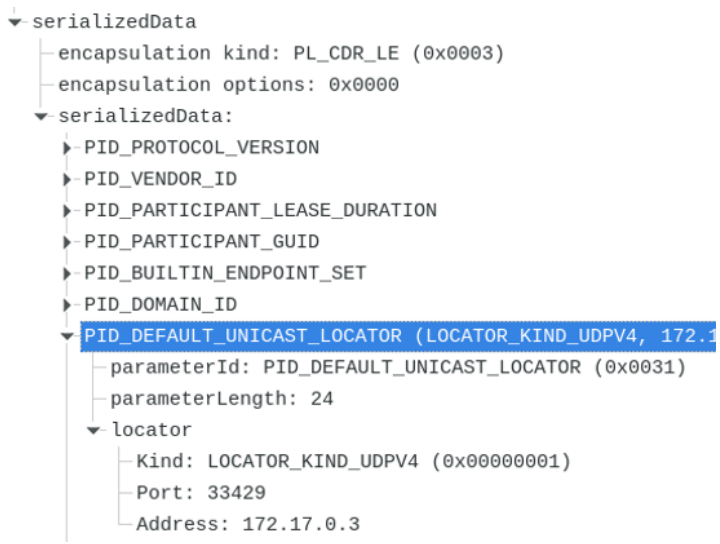


Figure 18. In case of UDP or TCP transport, the locator is the IP-port pair

We discovered this by setting the PID_DEFAULT_UNICAST_LOCATOR to the first IP address that came to mind (in this case, the Google DNS, because it's easy to type at "8.8.8.8"). Almost immediately, a flow of outbound packets came from the DDS node, as shown in Figure 17 and 19.

172.17.0.4	172.17.0.3	RTPS	350 INFO_TS, DATA(p)
172.17.0.3	8.8.8.8	RTPS	366 INFO_DST, INFO_TS, DATA(p)
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTE
172.17.0.3	8.8.8.8	RTPS	162 INFO_DST, ACKNACK, ACKNACK, ACKNACK
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTE
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTE
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTE
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTE
172.17.0.1	172.17.255.255	UDP	86 57621 → 57621 Len=44
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTE
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTE
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTE
172.17.0.3	8.8.8.8	RTPS	366 INFO_DST, INFO_TS, DATA(p)
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTE
172.17.0.3	8.8.8.8	RTPS	162 INFO_DST, ACKNACK, ACKNACK, ACKNACK

Figure 19. We found the amplification vulnerability almost by accident, by setting the PID_DEFAULT_UNICAST_LOCATOR to the first IP address that came to mind and easy to type

3.2 Source-code and Binary Fuzzing

Of all the implementation vulnerabilities that we disclosed for this research, all but three have been found through source-code or binary fuzzing, and three through scripting a file-format input mutator (RADAMSA). There are many fuzzing tools freely available to researchers, and we chose one based on what has been used successfully for years by the largest public fuzzing platform (Google OSS-Fuss), which uses a combination of AFL++, libFuzzer, and Honggfuzz. Although the choice of the specific tool can influence the efficiency of a fuzzing campaign, we focused our attention on the most important piece: finding good fuzz targets and writing good fuzzing harnesses.

5.2.1 Source-code Fuzzing with AFL++ and libFuzzer

We used AFL++ for fuzzing with multiple sanitizers in LLVM. AFL++ requires the project compile with the latest version of LLVM and the build system of some DDS implementations required some work. Aside from this, most of the effort in this phase went into finding the right fuzz target and implementing a harness while keeping the code deterministic (for example, no threading).

From the high-level viewpoint depicted in Figure 20 and 24, we were interested in finding the most self-contained function in charge of processing data coming from the network. We found a repeating pattern in all the DDS implementations: upon receiving network data (i.e., recv() or some abstraction on top of it), there are one or more deserialization functions in which we likely find a switch-case control structure, used to dispatch the RTPS sub-message IDs to the right routine.

Given the importance of finding the right fuzz target, we dedicate the remainder of this section to showcase some examples of fuzz targets.

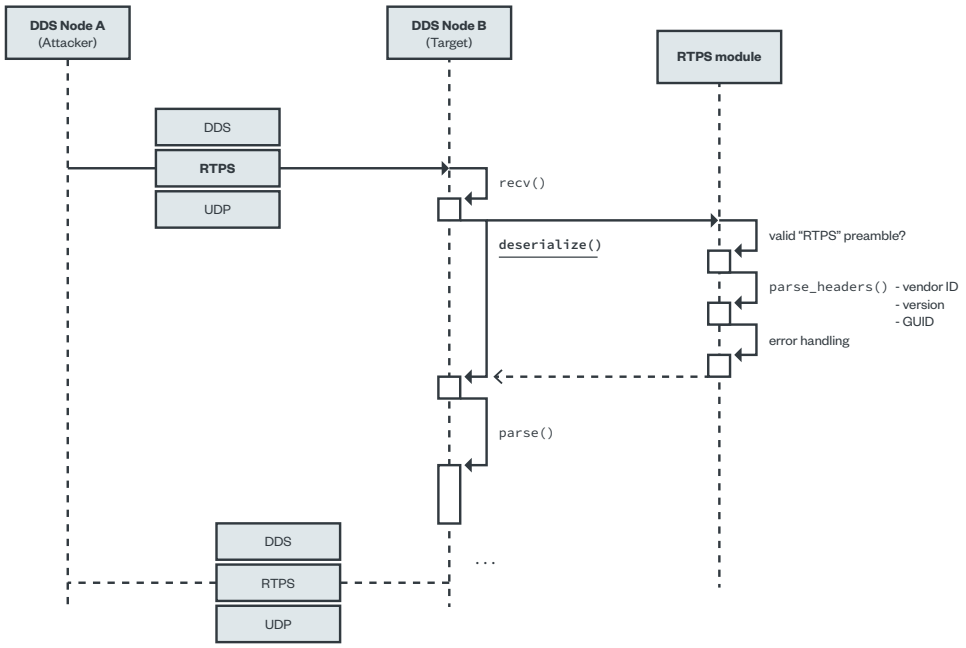


Figure 20. Abstract representation of the data flow in a typical DDS/RTPS message exchange. From a fuzzing perspective, the deserialize() step is the fuzz target.

We began with the supplied Docker images or make files to compile example programs, run them under GDB, and inspect debug traces, like exemplified for OpenDDS in Figure 21.

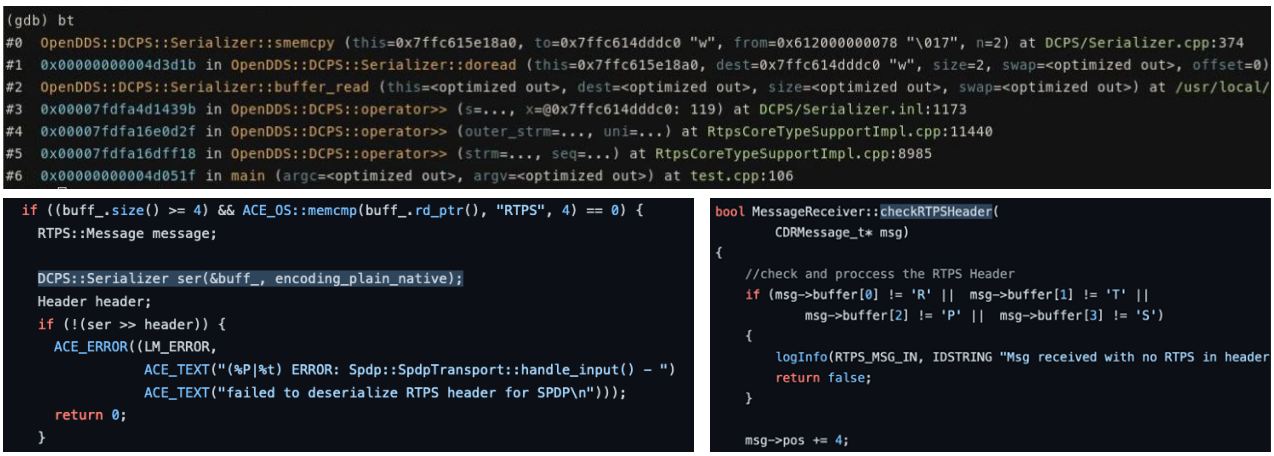


Figure 21. Starting from a debug trace, we found interesting functions and explored further by manually looking into the source code with the aid of Visual Studio Code engine

By following the function calls with the aid of the code analyzer part of Visual Studio Code, we were able to see that all three DDS implementations were using very similar procedures for deserializing network payloads. In particular, we found that they all had a switch-case to handle the RTPS sub-message types, as exemplified for OpenDDS in Figure 25.

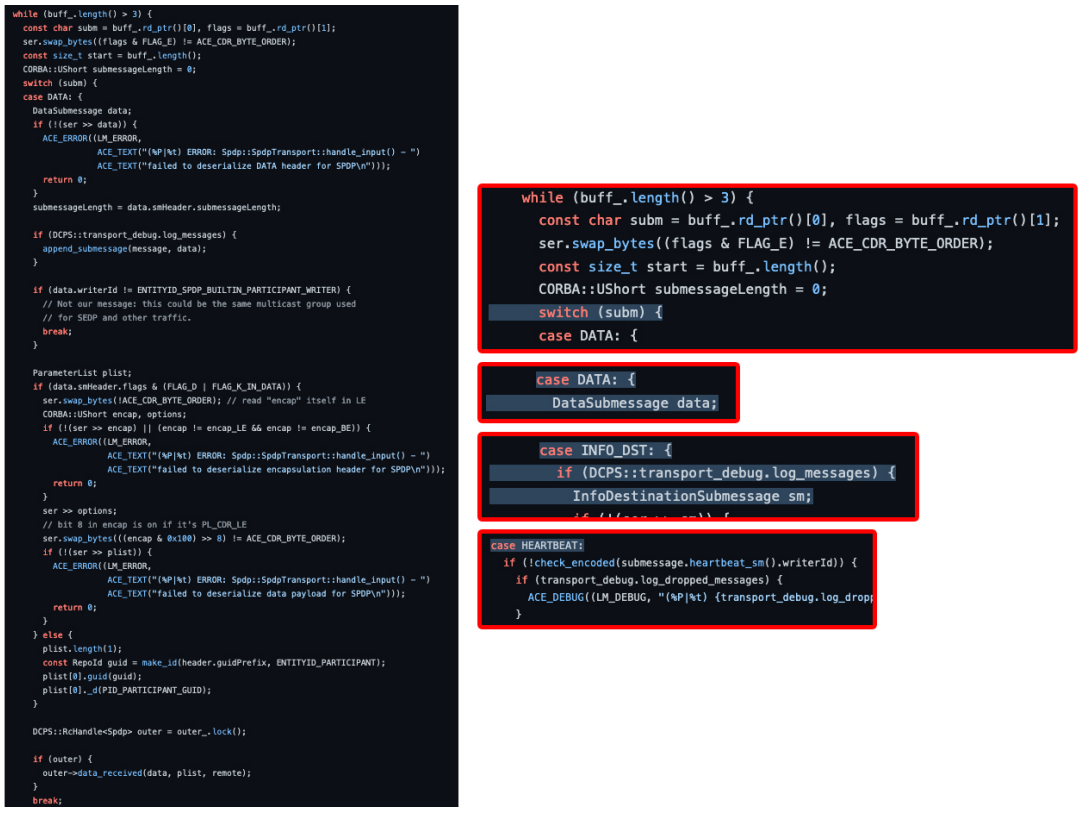


Figure 22. Typical switch-case control structure found similar in all DDS implementations. Each of the branches takes care of one RTPS sub-message type (e.g., DATA, INFO_DST, HEARTBEAT).

In some cases, we adjusted the source code right before the beginning of the de-serialization to dump the binary data being passed to the first function. This was useful to confirm that it was the network payload that we expected the function to receive, as exemplified for Cyclone DDS in Figure 26.

```
    sz = ddsi_conn_read (conn, buff, stream_hdr_size, true, &srcloc);
#ifdef FUZZING_BUILD_MODE_UNSAFE_FOR_PRODUCTION
    memcpy(buff, fuzz_data, fuzz_size);
    sz = fuzz_size;
#endif
```

```
#ifdef FUZZING_BUILD_MODE_UNSAFE_FOR_PRODUCTION_INPUTGEN
  FILE * fp = NULL;
  char fname[256];
  sprintf(fname, "buff.%d", cnt++);
  fp = fopen(fname, "wb");
  fwrite(buff, sz, 1, fp);
  fclose(fp);
#endif
```

Figure 23. (Top screenshot) In some cases, we inserted a memcpy() to directly fuzz the input in the right place when it was not possible to further decompose the function into a smaller, self-contained fuzz target. In other cases, we used the original code to dump the data received by the de-serialization routine to verify that we found the right fuzz target (bottom screenshot).

5.2.1.1 CVE-2021-38445 (OpenDDS): Failed Assertion Check in RTPS Handshake

```
unsigned char *s = __AFL_FUZZ_TESTCASE_BUF;
size_t sz = (size_t) __AFL_FUZZ_TESTCASE_LEN;
while (__AFL_LOOP(10000))
{
ACE_Message_Block *mb = new ACE_Message_Block (sz);

// do our own write to mb
ACE_OS::memcpy(mb->wr_ptr(), s, sz);
mb->wr_ptr(sz);

// most code below stolen from Spdp.cpp

OpenDDS::DCPS::Serializer ser (mb, encoding_plain_native);
OpenDDS::RTPS::Header header;
if (!(ser >> header)) {
  return 0; // this might be mutated by afl
  // ACE_OS::printf("%s\n", "fail deserialize");
}
```

Figure 24. Example harness for OpenDDS RTPS deserialization routine written for AFL++ using persistent mode

Using the harness (shown in Figure 27) we found out that, in OpenDDS ≤ v3.17, while receiving a RTPS packet with valid headers, with DATA sub-message, any attached serialized sub-data segment with a parameterLength of 0 will cause an assertion to fail in Serializer::doread, which subsequently called Serializer::smemcpy with a const char* from of zero.

This vulnerability can be exploited via the network even without authorization and can cause the DDS node to crash. It cannot be developed into a buffer overflow so it does not grant any code-execution primitives.

More specifically, Serializer::doread does not check for segments of 0 length and continues to handle the messages. This is passed by RtpsCoreTypeSupportImpl.cpp near bool operator >>(Serializer& outer_strm, ::OpenDDS::RTPS::Parameter& uni), which extracts size information from parameterLength but does not check if it is a valid value. It only makes sure extracting values from the serializer is successful.

5.2.1.2 CVE-2021-38445 (OpenDDS): Memory exhaustion

The opposite occurs with CVE-2021-38445 explained in the previous section, wherein the serializer is tricked into allocating very large chunks of memory. AFL++ found a crash in OpenDDS's serializer: While deserializing data with parameter ID type 0x55, it does not sanitize the value in its length field. This allows attackers to exhaust a server's memory by crafting a packet with a very large number in that field.

In bool operator>>(Serializer& strm, ::OpenDDS::RTPS::FilterResult_t& seq) (RtpsCoreTypeSupportImpl.cpp:1977), a check should be made to make sure it never allocates more memory than it's allowed, or a hard limit should be implemented.

5.2.1.3 CVE-2021-38441 and CVE-2021-38443 (Cyclone DDS): XML Parsing to Heap-write

Some DDS implementations had networking functionalities plugged deep into the application code, which required some mock functions in the harness, as exemplified in Figure 28 for Cyclone DDS.

```
static ssize_t fakeconn_write(
    ddsi_tran_conn_t conn,
    const ddsi_locator_t *dst,
    size_t niov,
    const ddsrt_iovec_t *iov, uint32_t flags)
{
    return (ssize_t)niov;
}

static ssize_t fakeconn_read(
    ddsi_tran_conn_t conn,
    unsigned char *buf,
    size_t len,
    bool allow_spurious,
    ddsi_locator_t *srcloc)
{
    return (ssize_t)len;
}

static struct cfgst *cfgst;
static struct ddsi_domaingv gv;
static ddsi_tran_conn_t fakeconn;
static ddsi_tran_factory_t fakenet;
static struct thread_state1 *ts1;
static struct nn_rbufpool *rbpool;
static bool initialized = false;
```

```
static struct cfgst *cfgst;
static struct ddsi_domaingv gv;
static ddsi_tran_conn_t fakeconn;
static ddsi_tran_factory_t fakenet;
static struct thread_state1 *ts1;
static struct nn_rbufpool *rbpool;
static bool initialized = false;

int LLVMFuzzerTestOneInput(
    const uint8_t *data,
    size_t size)
{
    if (!initialized)
    {
        ddsi_iid_init();

        memset(&dds_global, 0, sizeof(dds_global));
        ddsrt_mutex_init(&dds_global.m_mutex);

        ddsi_config_init_default(&gv.config);

        memset(&gv, 0, sizeof(gv));
        initialized = true;
    }

    const char *filename = buf_to_file(data, size);

    if (filename == NULL)
        return EXIT_FAILURE;

    cfgst = config_init(
        filename,
        &gv.config,
        DDS_DOMAIN_DEFAULT);

    if (delete_file(filename) != 0)
        return EXIT_FAILURE;

    rtps_config_prep(&gv, cfgst);
    return 0;
}
```

Figure 25. Cyclone DDS harness required a mock network subsystem. The actual fuzzing is happening at line 86, where we pass the configuration initializer a pointer memory-mapped XML file.

The harness in Figure 28 found several crashes, which led to two vulnerabilities. One is exemplified in Figure 29, a multi-byte heap-write primitive. Upon checking the source code, we noticed that there were multiple inputs that can lead to a heap overflow in the XML parsing routines. This causes at least a crash and can be exploited to write in the heap, potentially overflowing into the stack. Without heap protections, this vulnerability is exploitable as it is a write primitive of at least 8 bytes, and certainly causes the program to crash in the best case.

```
==14==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x612000000148 at pc 0x0000004d1461 bp
WRITE of size 8 at 0x612000000148 thread T0
SCARINESS: 42 (8-byte-write-heap-buffer-overflow)
    #0 0x4d1460 in cfgst_push /src/cyclonedds/src/core/ddsi/src/q_config.c:365:36
    #1 0x4d1460 in proc_elem_open /src/cyclonedds/src/core/ddsi/src/q_config.c:1906:5
    #2 0x55566a in parse_element /src/cyclonedds/src/ddsrt/src/xmlparser.c:627:16
    #3 0x555d28 in parse_element /src/cyclonedds/src/ddsrt/src/xmlparser.c:663:32
    #4 0x555d28 in parse_element /src/cyclonedds/src/ddsrt/src/xmlparser.c:663:32
    #5 0x555d28 in parse_element /src/cyclonedds/src/ddsrt/src/xmlparser.c:663:32
    #6 0x555d28 in parse_element /src/cyclonedds/src/ddsrt/src/xmlparser.c:663:32
    #7 0x555d28 in parse_element /src/cyclonedds/src/ddsrt/src/xmlparser.c:663:32
    #8 0x555d28 in parse_element /src/cyclonedds/src/ddsrt/src/xmlparser.c:663:32
    #9 0x555d28 in parse_element /src/cyclonedds/src/ddsrt/src/xmlparser.c:663:32
    #10 0x555d28 in parse_element /src/cyclonedds/src/ddsrt/src/xmlparser.c:663:32
    #11 0x555d28 in parse_element /src/cyclonedds/src/ddsrt/src/xmlparser.c:663:32
    #12 0x55505b in ddsrt_xmlp_parse /src/cyclonedds/src/ddsrt/src/xmlparser.c:754:19
    #13 0x4cc4bb in config_init /src/cyclonedds/src/core/ddsi/src/q_config.c:2242:11
    #14 0x4ca425 in LLVMFuzzerTestOneInput /src/cyclonedds/fuzz/fuzz_config_init.c:100:13
    #15 0x886b00 in ExecuteFilesOnyByOne /src/aflplusplus/utils/aflpp_driver/aflpp_driver.c:191:7
    #16 0x8868d0 in main /src/aflplusplus/utils/aflpp_driver/aflpp_driver.c
    #17 0x7fbc3823683f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2083f)
    #18 0x41e308 in _start (/out/fuzz_config_init+0x41e308)

DEDUP_TOKEN: cfgst_push--proc_elem_open--parse_element
0x612000000148 is located 0 bytes to the right of 264-byte region [0x612000000040,0x612000000148)
allocated by thread T0 here:
    #0 0x498e4d in malloc /src/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.cpp:145:3
    #1 0x5668ba in ddsrt_malloc_s /src/cyclonedds/src/ddsrt/src/heap/posix/heap.c:20:10
    #2 0x5668ba in ddsrt_malloc /src/cyclonedds/src/ddsrt/src/heap/posix/heap.c:26:15
    #3 0x4cbab9 in config_init /src/cyclonedds/src/core/ddsi/src/q_config.c:2183:11
    #4 0x4ca425 in LLVMFuzzerTestOneInput /src/cyclonedds/fuzz/fuzz_config_init.c:100:13
    #5 0x886b00 in ExecuteFilesOnyByOne /src/aflplusplus/utils/aflpp_driver/aflpp_driver.c:191:7
```

Figure 26. Backtrace of a crash found by libFuzzer on Cyclone DDS, which led us to CVE-2021-38441, a multi-byte heap-write primitive.

5.2.2 Binary Fuzzing with UnicornAFL

The trial licenses for RTI Connext DDS, CoreDX DDS, and Gurum DDS grant access only to binary distributions of the libraries. After compiling the example programs that ship with the original software distribution, we used GDB to inspect run traces. This turned out to be quite verbose given the presence of several debug symbols. We filled the missing information by inspecting the listing via Ghidra and IDA Pro. This allowed us to find interesting fuzz targets, as seen in Figure 30.

For coverage-guided fuzzing we used UnicornAFL, which is a fork of AFL++ that uses the Unicorn emulation engine to “execute” the target and employs block-edge instrumentation in a similar fashion to AFL’s QEMU mode.

In practice, we dumped the context of a running process with GDB and prepared a harness (see Figure 31) that lets UnicornAFL restore that context, set registers and memory state, the RIP register, and start emulation. Like AFL, UnicornAFL will take care of mutating the input, passing it to the fuzz target, and keep track of the coverage. The main shortcoming is that we had to re-implement some memory management functions (e.g., malloc, memset).

This approach is inherently slow due to emulation, but was good enough for initial vulnerability research. It costed us a few hours of AWS EC2 computation (c5a.8xlarge), and we found that AMD EPYC machines were three times faster than Intel Xeons while fuzzing using UnicornAFL.

```
Thread 9 "FilterBug" received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0x7fffe67fc700 (LWP 15429)]
0x00005555559690ca in DDS_DynamicData2TypePlugin_return_sample (endpoint_data=0x555556d8e130,
1195    DynamicData2TypePlugin.c: No such file or directory.
(gdb) bt
#0  0x00005555559690ca in DDS_DynamicData2TypePlugin_return_sample (endpoint_data=0x555556d8e1
#1  0x000055555605ab44 in PRESPsReaderQueue_returnQueueSample (me=0x555556da1140, entry=0x555
#2  0x000055555605f4ac in PRESPsReaderQueue_addQueueEntryToPolled (me=0x555556da1140, lostCou
    entry=0x555556da55d0, receptionTsIn=0x7fffe67fb8a0, now=0x7fffe67fb8a0, remoteWriterQueue=
    at PsReaderQueue.c:4266
#3  0x0000555556068437 in PRESPsReaderQueue_newData (me=0x555556da1140, dataAvailable=0x555556
    rejectedReason=0x555556da1198, receivedInlineQosBitmap=0x7fffe67fb5bc, remoteWriterQueue=0
    localData=0x0, decodingKeyHandle=0x0, strength=0, reservedCount=-1, timestamp=0x7fffe67fb8
    worker=0x555556a3a630) at PsReaderQueue.c:6596
#4  0x0000555555cd0f97 in PRESPsService_readerSampleListenerOnNewData (listener=0x55555685d310
    timestamp=0x7fffe67fb8a0, storage=0x555556a9abb8, worker=0x555556a3a630) at PsServiceImpl.
#5  0x0000555555e2bf8c in COMMENDBeReaderService_onSubmessage (listener=0x555556a96b30, contex
    at BeReaderService.c:1350
#6  0x0000555555edef4d in MIGInterpreter_parse (me=0x5555568d2830, context=0x7fffcc000be0, ms
#7  0x0000555555e22770 in COMMENDActiveReadeReceiver_loop (param=0x555556d3a000) at ActiveRea
#8  0x000055555601209c in RTIOsapiThreadChild_onSpawned (param=0x555556716b00) at Thread.c:141
#9  0x00007ffff79b96db in start_thread (arg=0x7fffe67fc700) at pthread_create.c:463
#10 0x00007ffff6f3988f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95
```

```
134   }
135   else {
136     if ((*(int *)bufferToParse.pointer == 0x53505452) ||
137        (*(int *)bufferToParse.pointer == 0x58505452)) {
138       data = (char *)((int *)bufferToParse.pointer + 1);
139       iVar11 = MIGInterpreter_parseHeader(context,&data);
140       if (iVar11 == 0) {
141         (me->_stat).versionMismatchCount = (me->_stat).versionMismatchCount + 1;
142       }
143       else {
144         keyToMatch.guid.prefix._0_8_ = *(undefined8 *)&context->sourceHostId;
145         keyToMatch.guid.prefix.instanceId = context->sourceInstanceId;
146         pMVar3 = (me->_property).forwarder;
147         pRVar14 = context->_decodeBuffers;
148         pRVar15 = context->_decodeBuffers + 1;
149         uVar17 = 0x637e93;
150         iVar12 = RFDACursor_startEnc(context-> cursor.(int *)0x0);
```

Figure 27. Finding fuzz targets in RTI Connext DDS, CoreDX DDS, and Gurum DDS required us to reverse engineer the binary libraries, which was easy as the vendor did not use any anti-reverse engineering measures

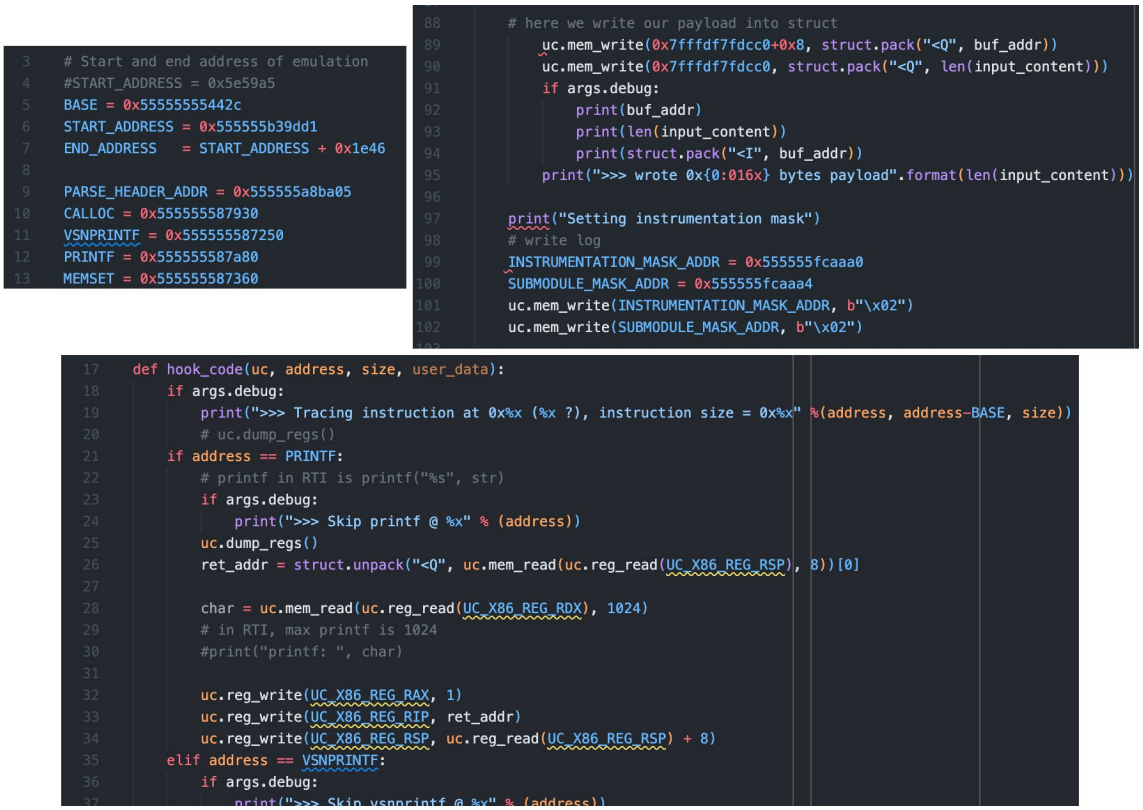


Figure 28. We used the debug trace and the decompiled code (see Figure 30) to create a harness for UnicornAFL based on the template provided by Nathan Voss[75]

5.2.2.1 CVE-2021-38435 (RTI Connext DDS): Segmentation Fault on Malformed RTPS Packet

The UnicornAFL instrumentation that we prepared found a segmentation-fault in the RTPS deserializer in RTI Connext DDS when receiving a malformed packet. This would cause runtimes to exit immediately and a DoS. In particular, the RTICdrStream_skipStringAndGetLength() function does not properly check inputs, using the result straight from RTICdrStream_align(), thus triggering a segmentation fault. Both publisher and subscriber are affected.

5.2.2.2 CVE-2021-38439 and CVE-2021-38423 (Gurum DDS): Heap Overflow and Segmentation Fault

While using UnicornAFL on Gurum DDS fuzz targets, we discovered that there is a heap overflow in the RTPS routine that handles payload parsing. This causes a segmentation fault leading to DoS.

More specifically, the crash is triggered in rtps_read_AckNackMessage() function when called in read_Submessage(), which creates a multi-byte heap overflow condition. We found this crash by using a harness that passes RTPS payload directly to the rtps_read_Data(…, buf, len, …) function through the buf argument.

Another case we found is in the rtps_Parameter_load2() function, which does a type conversion from a buffer and does a check to exclude specific IDs. During this conversion, we found some inputs causing a segmentation fault. We verified that this is exploitable via network by crafting a packet based on the crash dump provided by the fuzzing engine.

5.2.3 Scripting RADAMSA to Mutate XML Files

At the beginning of this research and before using AFL++ and UnicornAFL, we used RADAMSA directly, with some simple shell scripting (see Figure 32). Without any prior knowledge on the target software, this simple technique can be surprisingly effective at finding crashes, which can also lead to the discovery of vulnerabilities.

```
#!/bin/bash
i=1
while true; do
  radamsa ShapeExample.xml > ShapeExample1.xml
  timeout 120 python simple/writer1.py > /dev/null 2>&1
  if [ $? -gt 127 ]; then
    cp ShapeExample1.xml crash_`date +%Y%m%d.%H%M%S`.xml
    echo "Crash found!"
  fi
  echo $((i++))
done
```

Figure 29. A simple scripting of RADAMSA can lead to surprising results

5.2.3.1 CVE-2021-38427 and CVE-2021-38433 (RTI Connext DDS): Stack-based Buffer Overflows Python Bindings

The simple “harness” shown in Figure 32 allowed us to find two vulnerabilities; one could be exploited beyond just a crash to control a pointer using a malformed XML file.

When the length of an attribute value in a configuration XML file is longer than a certain limit, RTIXMLObject_lookUpRef() would trigger a buffer overflow. If the length is exactly 894 characters, we could overwrite RIP register (see Figure 33). We have not investigated further, but we see the possibility of preparing a ROP chain for this target. However, the XML parser does not accept arbitrary hexadecimal characters, so we’re limited within the Unicode range.

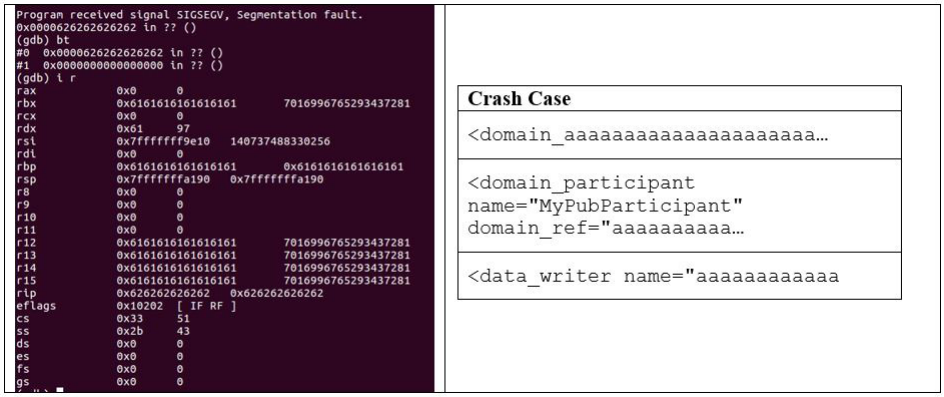


Figure 30. Register state and sample crash cases in RTI Connext DDS Connector (Python bindings) found with RADAMSA from the original XML configuration file.

A variant of this vulnerability is in RTIXMLDtdParser_getElement(), which does not properly validate the length of an element prior to copying it to a fixed-length stack buffer.

5.2.3.2 Unmaintained XML Parsing Libraries

Unfortunately, we discovered that Gurum DDS uses ezXML, an open-source XML library that has been in beta status since 2006 that has never been updated. The mailing list of the project has been silent since 2010, showing that no users are actively discussing it. The latest version was 0.8.6, but an inspection of the binary code revealed that the developer changed the version number to 1.0.0, which was the only change. EzXML currently has 16 known vulnerabilities (eight in 2021),[76] all with medium to high severity ratings and have never been fixed.

Probably because of its small footprint, we discovered that ezXML is also used in many embedded applications like router firmware, and has hundreds of downloads per week. We have reached out to Gurum DDS several times — about this and other vulnerabilities — since the summer of 2021 and have received no response.

5.2.4 Integration with Google OSS-Fuzz

One-shot fuzzing campaigns are useful but can only reveal bugs in the current version. Fuzzing should be considered as a security-oriented regression testing process and, like unit testing, should run continuously as part of the CI/CD pipeline. For this reason, we decided to contribute to the public OSS-Fuzz initiative by Google, which encourages open-source developers to integrate their projects in their continuous-fuzzing infrastructure. This is of particular importance for critical open-source projects, which form the foundation of larger and widely used software distributions (for example, server software).

DDS Implementation	Google OSS-Fuzz Integration Status		Fuzz Harnesses Implemented
	Previous	Current	
eProsima Fast-DDS	Integrated	Integrated	fuzz_XMLProfiles fuzz_processCDRMsg[77]
Eclipse Cyclone DDS	Not integrated	Integrated	fuzz_config_init
OCI OpenDDS	Not integrated	Integrated	None

Table 12. Before we started this research, only Fast-DDS started integrating fuzz harnesses as part of their code base

As shown in Table 12 when we started this research, there was only one DDS implementation that had integrated a fuzzing harness into the Google OSS-Fuzz. While developing a fuzzing harness on a local computer requires deep understanding of the software's internals, integrating it into a public fuzzing infrastructure like Google OSS-Fuzz requires even more effort. This is because the security researcher needs to develop the harness and package it by following the infrastructure's conventions, ensure that the target project's pipeline is not impacted, talk to the target project's lead developer so that they understand what the initiative is about, obtain their permission, send a pull request, and wait for the OSS-Fuzz maintainers to process the integration request. This is easier said than done, although we have found fertile ground while talking to all three open-source DDS developers.

We encourage other researchers to contribute with new fuzzing harnesses, hopefully including the open-source parts of the closed-source DDS distributions.

5.3 Internet-wide Scanning for RTPS Endpoints

We wanted to demonstrate how an attacker could leverage the RTPS built-in discovery protocol for automated, large-scale reconnaissance of RTPS/DDS endpoints. We found hundreds of exposed services as a byproduct, which was unexpected. Understanding that RTPS/DDS was designed for local-network applications, we did not expect to find more than a couple of endpoints exposed by mistake. Not only did we find several hundreds, but 35 of them have never stopped sending responses to our scanner despite the fact that we only sent them one single RTPS packet.

After trying to use readily available internet scanning services (such as Shodan, Censys, and LeakIX), we ended up implementing our own scanning prototype because of the intricacies of the RTPS discovery phase. This makes it a bit convoluted to correctly fingerprint a service.

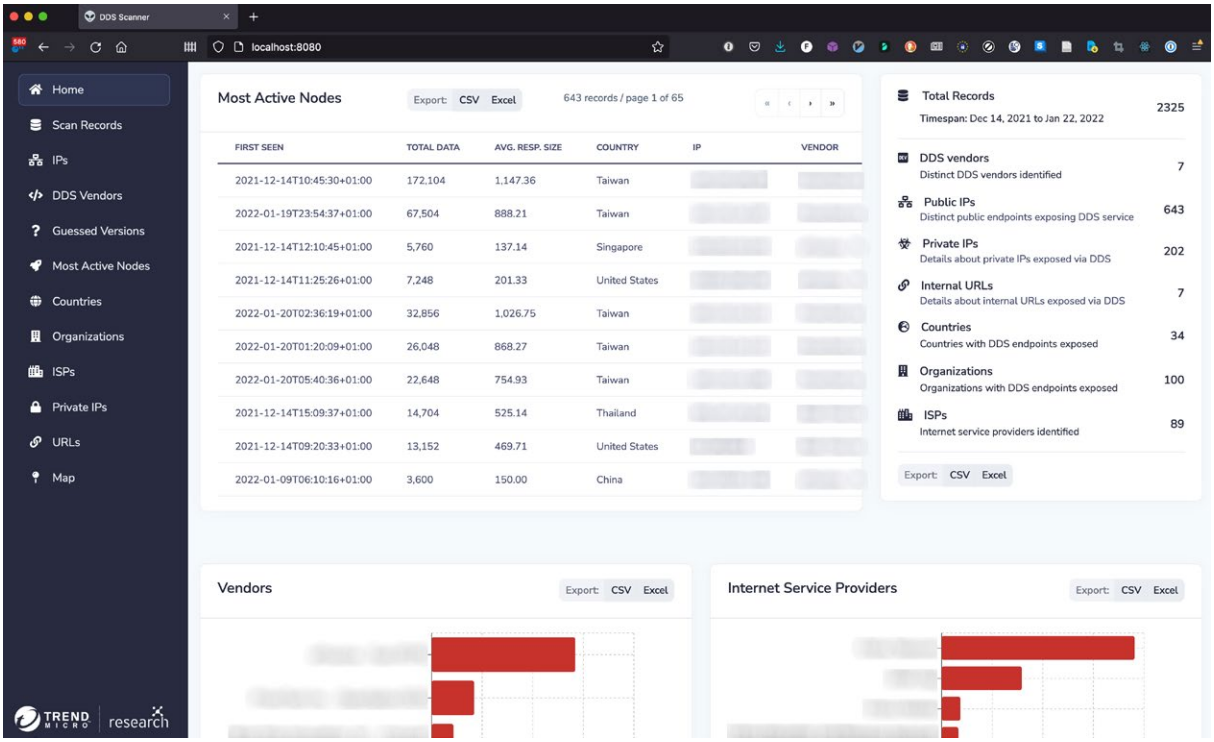


Figure 31. The dashboard of our DDS-scanning system allows analysts to explore the data

5.3.1 Challenges of RTPS/DDS Reconnaissance

In summary, the main challenges of RTPS/DDS reconnaissance are:

- **Dynamic and arbitrarily large port range.** Depending on the number of participants in a RTPS/DDS network, there can be tens of thousands of ports to check. The formula to calculate the port is defined in the specifications,[78] and each implementation has distinct defaults, as shown in Table 13.
- **Latency and connectionless nature.** Although RTPS/DDS are transport-agnostic, the de facto standard is to use UDP, which makes efficient scanning techniques useless. To verify if there is a valid RTPS/DDS endpoint bound to a given address (IP and UDP port), we need to wait for an answer, which may or may come within a few seconds. Given the size of the public IPv4 space, it's impractical to wait for answers upon each request.
- **Addressing information at application layer.** Addressing information is exchanged at the application layer. Sending a valid RTPS packet to the correct UDP port (for example, the default 7400 discovery port) does not guarantee a response, even if there is an RTPS service running. To trigger a response, the RTPS discovery packet must include correct locator information (for instance, IP and UDP port), which will receive a response.

Vendor	UDP Discovery Port							
	7200	7399	7400	7401	7410	7411	7412	7413
Core DX					√	√		
Cyclone DDS					√	√	√	√
Fast-DDS			√				√	
GurumDDS	√	√	√	√	√	√		
Connext DDS			√		√	√		
OpenDDS					√	√		
Min. Scan Set			**√**		**√**	**√**	**√**	

Table 13. Assuming up to one DDS domain and at least one participant, we tested the open UDP ports of each of the six reference implementations. All those marked with checks could detect any of the identified ports just by scanning for four ports listed on the last row of the table.

5.3.2 Scanning Approach

Given the challenges mentioned in the previous section, we implemented a distributed scanning system (see Figure 19) that we first validated in a private network against all six DDS implementations, under the simplifying assumption that the developer would not go too far from the "default" set of ports listed in Table 13.

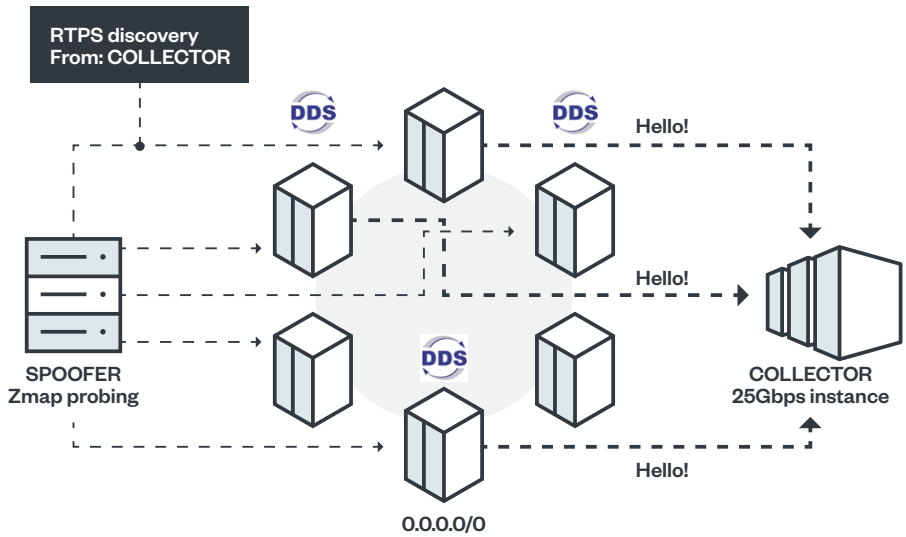


Figure 32. We used spoofed RTPS discovery messages sent via ZMap to collect answers from valid DDS endpoints and filtered echoed and invalid responses

As shown in Figure 20, we created a template RTPS packet (using our Scapy RTPS layer, as shown in Figure 21) with a parametric locator IP and port number. We then generated an actual RTPS discovery packet by fixing the locator IP and port numbers according to the collector that we set up to receive the (reflected) packet. The collector will know what packet to expect given its IP and port number. Since there are several honeypots that simply reply to every request by echoing traffic they receive, the collector filters these "echoed" packets and keeps only valid responses. The collector checks if a received RTPS packet is valid by using the Scapy layer to dissect it and checks whether the globally unique identifier field (GUID) is new.

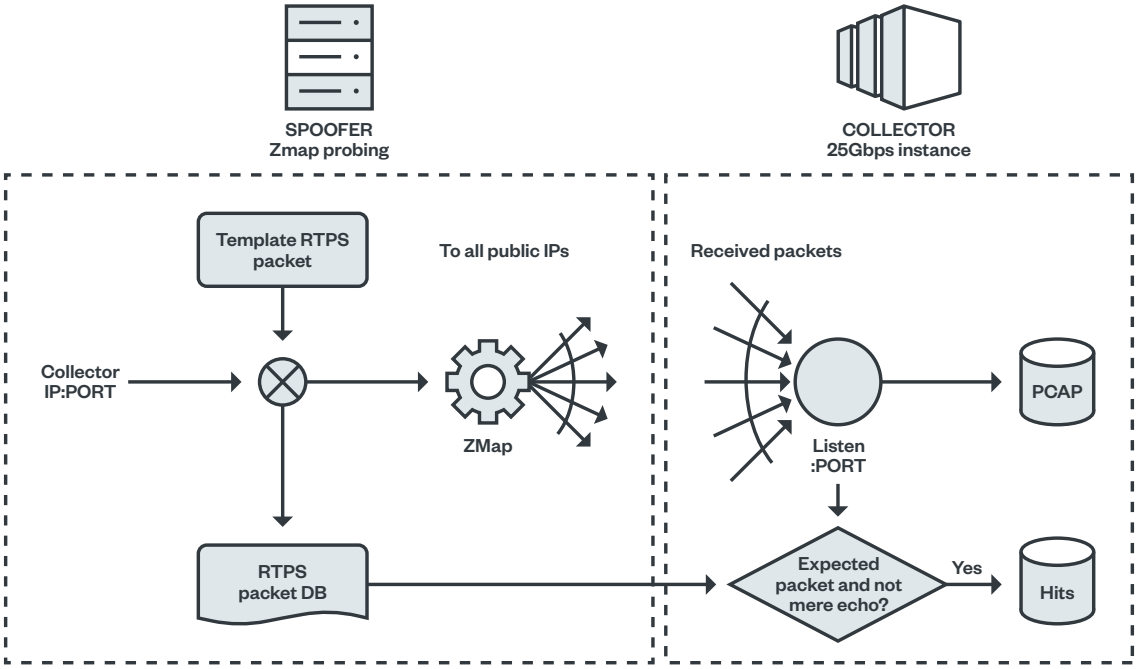
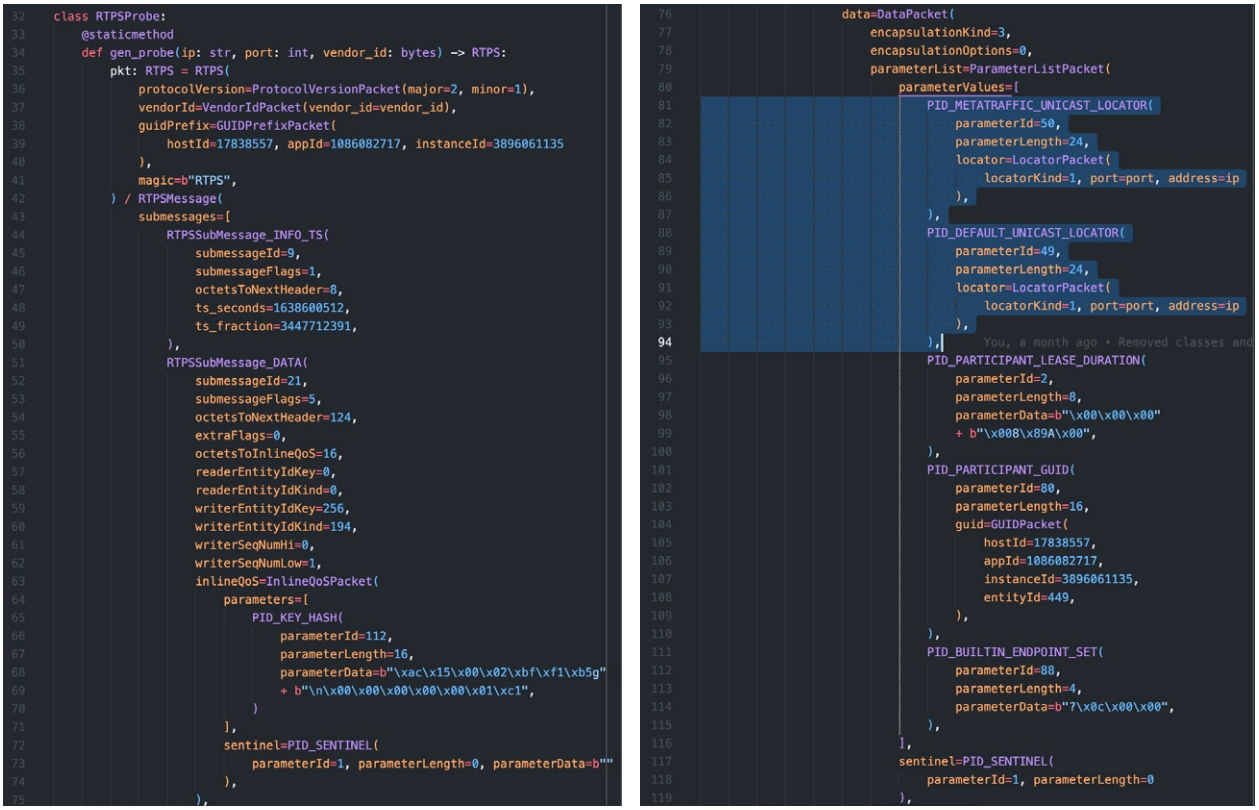


Figure 33. Starting from a template RTPS packet, the spoofer instantiates it for a given collector (IP and port) and sends it out via ZMap. The collector uses the RTPS packets sent out to decide whether the received packets are valid and not simply duplicates of what was sent out.

The following section briefly outlines how we analyzed the data that we received from the collector.



```
class RTPSProbe:
    @staticmethod
    def gen_probe(ip: str, port: int, vendor_id: bytes) -> RTPS:
        pkt: RTPS = RTPS(
            protocolVersion=ProtocolVersionPacket(major=2, minor=1),
            vendorId=VendorIdPacket(vendor_id=vendor_id),
            guidPrefix=GUIDPrefixPacket(
                hostId=17838557, appId=1086082717, instanceId=3896061135
            ),
            magic=b"RTPS",
        ) / RTPSMessage(
            submessages=[
                RTPSSubMessage_INFO_TS(
                    submessageId=9,
                    submessageFlags=1,
                    octetsToNextHeader=8,
                    ts_seconds=1638600512,
                    ts_fraction=3447712391,
                ),
                RTPSSubMessage_DATA(
                    submessageId=21,
                    submessageFlags=5,
                    octetsToNextHeader=124,
                    extraFlags=0,
                    octetsToInlineQoS=16,
                    readerEntityIdKey=0,
                    readerEntityIdKind=0,
                    writerEntityIdKey=256,
                    writerEntityIdKind=194,
                    writerSeqNumHi=0,
                    writerSeqNumLow=1,
                    inlineQoS=InlineQoSPacket(
                        parameters=[
                            PID_KEY_HASH(
                                parameterId=112,
                                parameterLength=16,
                                parameterData=b"\xac\x15\x00\x02\xbf\xf1\xb5g"
                                + b"\n\x00\x00\x00\x00\x00\x01\xc1",
                            )
                        ],
                        sentinel=PID_SENTINEL(
                            parameterId=1, parameterLength=0, parameterData=b""
                        ),
                    ),
```

```
                    data=DataPacket(
                        encapsulationKind=3,
                        encapsulationOptions=0,
                        parameterList=ParameterListPacket(
                            parameterValues=[
                                PID_METATRAFFIC_UNICAST_LOCATOR(
                                    parameterId=50,
                                    parameterLength=24,
                                    locator=LocatorPacket(
                                        locatorKind=1, port=port, address=ip
                                    ),
                                ),
                                PID_DEFAULT_UNICAST_LOCATOR(
                                    parameterId=49,
                                    parameterLength=24,
                                    locator=LocatorPacket(
                                        locatorKind=1, port=port, address=ip
                                    ),
                                ),
                                PID_PARTICIPANT_LEASE_DURATION(
                                    parameterId=2,
                                    parameterLength=8,
                                    parameterData=b"\x00\x00\x00"
                                    + b"\x008\x89A\x00",
                                ),
                                PID_PARTICIPANT_GUID(
                                    parameterId=80,
                                    parameterLength=16,
                                    guid=GUIDPacket(
                                        hostId=17838557,
                                        appId=1086082717,
                                        instanceId=3896061135,
                                        entityId=449,
                                    ),
                                ),
                                PID_BUILTIN_ENDPOINT_SET(
                                    parameterId=88,
                                    parameterLength=4,
                                    parameterData=b"?\x0c\x00\x00",
                                ),
                            ],
                            sentinel=PID_SENTINEL(
                                parameterId=1, parameterLength=0
                            ),
```

Figure 34. Template RTPS discovery packet with parametric locator information highlighted

6 Recommendations and Conclusion

Following the MITRE ATT&CK® nomenclature, we recommend the implementation of mitigation best practices such as:

- Periodic vulnerability scanning (M1016) to detect the presence of unpatched services
- Deploying network intrusion prevention (M1031)
- Network segmentation (M1030)
- Filtering of network traffic (M1037) to detect spoofed DDS messages and prevent the exploitation of the reflection vulnerability
- Execution prevention (M1038) to reduce the exploitation of memory errors
- Periodic auditing (M1047).

6.1 Short-term Mitigations

While typically found in all modern hardware and software environments, we understand that protections such as address-space layout randomization (ASLR), executable space protection (ESP), data execution prevention (DEP), no-execute (NX), and write XOR execute (W^X) are not always applicable to the use cases where DDS runs. The most notable example is a low-power embedded system with minimal resources such as a field sensor. However, when applicable, such protections eliminate exploitability, leaving DoS as the only viable tactic for an attacker.

Our internet-wide scanning revealed hundreds of exposed DDS endpoints, which we hope the maintainers are aware of. Unfortunately, some of those instances were running unpatched or otherwise outdated versions of DDS, thus vulnerable to some of the weaknesses that we discovered. We recommend never exposing a DDS endpoint unless necessary and securing it with DDS Security in those cases. Despite all the optimization, time-critical applications require stripped-down versions of DDS to meet deadlines, as confirmed by developers and technologists working in the air-traffic-control and autonomous-driving fields. We are aware that adding a security layer like a simple VPN (not recommended) or the DDS Security extension (recommended) implies even higher overheads, which are unacceptable in certain scenarios.[79]

If patching is not possible, or in cases where a patch is not available (such as the amplification vulnerability), we recommend deploying IPS rules to spot forged or malicious packets. Trend Micro™ TXOne™ Networks EdgeIPS Pro™, EdgeIPS™, and EdgeFire™ customers are protected under rule 1137699 ICS DDS RTPS-mode Amplification Attack (CVE-2021-38429). Similarly, there are endpoint protection rules that can be configured to detect anomalous XML files.

6.2 Supply Chain Management of Critical Libraries

When dealing with supply chain security of critical software such as DDS, proper supply chain management processes allow immediate contextualization of a new vulnerability within the myriad of downstream software utilizing a certain library. DDS is just one of the many critical libraries used in embedded applications, and considerably easy to lose track of. We're surprised to discover that MITRE has a specific CWE number to track the use of unmaintained third-party components (CWE-1104), but found it unfortunate that this tool is not used in cases like ezXML. We believe that using CWE-1104 is a simple but practical and effective way to pinpoint security-sensitive components in the software supply chain. The mere use of an unmaintained component is a weakness, especially if that component carries known vulnerabilities.

6.3 Shift-left Approach and Continuous Fuzzing

The second most pressing need after managing current active assets is to make the code base more amenable to the integration of automated security-testing tools. Taking fuzz-testing as a representative example, we advocate that all critical software libraries such as DDS should be developed with a strong orientation to security testing, on top of traditional unit testing. The situation has improved greatly, thanks to initiatives such as OSS-Fuzz. But there's still a significant gap between security engineers and software engineers, resulting in tedious manual code reviews, unwanted modifications in the code to integrate security checks, and so on. Viewed as a whole, these gaps cause delays in the wide scale adoption of available security tools. The positive and engaging response by ADLINK, which went to the point of assisting Trend Micro researchers in creating good fuzz targets against their own code base,[80] should serve as an example to the entire software-engineering industry.

References

1 Object Management Group. (Apr. 2015). “OMG Data Distribution Service (DDS) - 1.4, formal/2015-04-10.” Accessed on Feb. 15, 2022 at https://www.omg.org/spec/DDS-SECURITY/1.1/PDF.

2 Cybersecurity & Infrastructure Security Agency. (Feb. 1, 2022). “ICS Advisory (ICSA-21-315-02): Multiple Data Distribution Service (DDS) Implementations (Update A).” Accessed on Feb. 15, 2022 at https://www.cisa.gov/uscert/ics/advisories/icsa-21-315-02.

3 Object Management Group. (2019). “DDS Vendor directory Listing.” Accessed on Feb. 15, 2022 at https://www.omg.org/dds-directory/vendor/list.htm.

4 Ta-Lun Yen, Federico Maggi, Erik Boasson, Victor Mayoral-vilches, Mars Cheng, Patrick Kuo, and Chizuru Toyama. (Nov. 11, 2021). “The Data Distribution Service (DDS) Protocol is Critical: Let’s Use it Securely!” presented at the Black Hat Europe Briefings, London. Accessed on Feb. 15, 2022. at https://www.blackhat.com/eu-21/briefings/schedule/index.html#the-data-distribution-service-dds-protocol-is-critical-lets-use-it-securely-24934.

5 Federico Maggi and Victor Mayoral-Vilches. (Nov. 29, 2021). *GitHub*. “RTPS contrib layer · Pull Request #3403 · secdev/scapy.” Accessed on Feb 16, 2022 at https://github.com/secdev/scapy/pull/3403.

6 “Real-time” is used in a generic and non-strict (hard real-time) manner in DDS specifications. We were unable to find timing guarantees (hard real-time, firm real-time, or soft real-time) provided in any of the reviewed documents. For the purposes of this research, we conclude that DDS targets remote soft real-time communications at best, leaving firm and hard real-time interactions to other technologies.

7 Object Management Group. (Apr. 2015). “OMG Data Distribution Service (DDS) - 1.4, formal/2015-04-10.” Accessed on Feb. 15, 2022 at https://www.omg.org/spec/DDS-SECURITY/1.1/PDF.

8 ENISA. (2021). *ENISA*. “ENISA Threat Landscape 2021.” Accessed on Oct. 2021 at https://www.enisa.europa.eu/publications/enisa-threat-landscape-2021.

9 UCA International Users Group. (2022). *OpenFMB Users*. “Open Field Message Bus (OpenFMB).” Accessed on Nov. 29, 2021 at https://openfmb.ucaiug.org/.

10 Kai Richter and Emilio Guijarro Cameros, “AUTOSAR and DDS: A Fresh Approach to Enabling Flexible Vehicle Architectures,” Mar. 02, 2021. https://www.rti.com/blog/fresh-approach-to-enabling-flexible-vehicle-architectures (accessed Nov. 29, 2021).

11 Google. (n.d.). *Google*. “Protocol Buffers.” Accessed on Feb. 15, 2022 at https://developers.google.com/protocol-buffers.

12 For future security research, recall that complex type systems can be used for type-confusion attacks.

13 Renesas. (Nov. 30, 2021). *Renesas*. “R-Car H3e-2G & H3 & M3 Starter Kit.” Accessed on Nov. 29, 2021 at https://www.renesas.com/us/en/products/automotive-products/automotive-system-chips-socs/r-car-h3-m3-starter-kit.

14 Aeronautical Information Manual. (n.d.). *FAA*. “Section 1. Airport Lighting Aids.” Accessed on January 2021 at https://www.faa.gov/air_traffic/publications/atpubs/aim_html/chap2_section_1.html.

15 Real-Time Innovations (RTI). (Oct. 22, 2015). *Real Time Innovations*. “Generic Vehicle Architecture – DDS at the Core.” Accessed on Nov. 29, 2021 at https://www.slideshare.net/RealTimeInnovations/generic-vehicle-architecture-dds-at-the-core.

16 Johan Scholliers, Pasi Pyykonen, Ari Virtanen, Alina Aittoniemi, Fanny Malin, Maija Federley, and Stella Nikolaou. (2020). *TRA2020 for* 8th Transport Research Arena, TRA 2020 - Conference cancelled” in Proceedings of TRA2020, the 8th Transport Research Arena, Rethinking transport – towards clean and inclusive mobility. “Automated Valet Parking using IoT: Design, user experience and business opportunities.” Accessed on Nov. 29, 2021 at https://www.traficom.fi/sites/default/files/media/publication/TRA2020-Book-of-Abstract-Traficom-research-publication.pdf (p.69).

17 ADLINK Tech. (2010). *ADLINK Technology*. “Coflight Consortium Selects Vortex OpenSplice DDS Middleware for Next Generation European Flight Data Processor.” Accessed Feb. 01, 2022 at https://www.adlinktech.com/en/Coflight.

18 Fujitsu. (Mar. 12, 2018). “Fujitsu Accelerates Path to 5G and Conscious Networks with Next-Generation Variable Optical Transport - Fujitsu United States.” Accessed Dec. 02, 2021 at https://www.fujitsu.com/us/about/resources/news/press-releases/2018/fnc-20180312.html.

19 UK 5G Innovation Network. (n.d.). “ADLINK Technology.” Accessed on Dec. 2021 at https://uk5g.org/5g-supplier-directory/adlink-technology/.

20 ADLINK Technology. (2020). *ADLINK Technology*. “Vortex OpenSplice Selected by Fujitsu for 1FINITY Networking Platform.” Accessed on Dec. 13, 2021 https://www.adlinktech.com/en/Fujitsu.

21 A. Llorens-Carrodeguas, C. Cervello-Pastor, I. Leyva-Pupo, J. M. Lopez-Soler, J. Navarro-Ortiz, and J. A. Exposito-Arenas. (Apr. 2018). “An architecture for the 5G control plane based on SDN and data distribution service,” in 2018 Fifth International Conference on Software Defined Systems (SDS), doi: 10/gnn4gf, pp. 105–111.

22 Elizabeth Montalbano. (Feb. 26, 2021). *Threatpost*. “Lazarus Targets Defense Companies with ThreatNeedle Malware.” Accessed Feb. 01, 2022 at https://threatpost.com/lazarus-targets-defense-threatneedle-malware/164321/.

23 Object Management Group. (June 05, 2019). *DDS Foundation*. “Case Study: NASA Launch and Control Systems.” Accessed Nov. 29, 2021 at https://www.omgwiki.org/ddsf/doku.php?id=ddsf:public:applications:aerospace_and_defense:nasa_launch_and_control_systems.

24 Jaime Martin Losa. (April 17, 2012). *Real-time, Embedded and Enterprise-Scale Time-Critical Systems*. “DDS in Low-Bandwidth Environments.” Accessed on February 01, 2022 at https://www.omg.org/news/meetings/workshops/RT-2012-presentations/03-S9-02_Losa.pdf.

25 Object Computing. (n.d.). *Object Computing*. “Case Study: The Autonomous Future Calls for Stringent Safety & Security Measures.” Accessed on Feb. 01, 2022 at https://objectcomputing.com/case-studies/vx-works-opendds-case-study.

26 India Berry. (Sep. 20, 2021). *Data Center*. “Top 10 Countries with the most data centres.” Accessed on Feb. 1, 2022 at https://datacentremagazine.com/top10/top-10-countries-most-data-centres.

27 Justin Wilson. (Mar. 2019). *Object Computing*. “Bringing Multicast to the Cloud for Interoperable DDS Applications.” Accessed on Feb. 1, 2022 at https://objectcomputing.com/resources/publications/sett/march-2019-multicast-to-cloud-for-dds-applications.

28 ADLINK Technology. (2021). *ADLINK Technology*. “Vortex Link.” Accessed on Feb 1, 2022 at https://www.adlinktech.com/en/vortex-link-data-distribution-service.

29 eProsima. (2021). *eProsima*. “eProsima Integration Service.” Accessed on Feb. 1, 2022 at https://integration-service.docs.eprosima.com/en/latest/.

30 angelrti. (Nov. 19, 2021). *GitHub*. “rticonnextdds-gateway.” Accessed on Feb. 1, 2022 at https://github.com/rticommunity/rticonnextdds-gateway.

31 Gurum Networks. (2021). *Gurum Networks*. “Gurum Networks, the DDS company.” Accessed on Feb. 1, 2022 at https://www.gurum.cc/index_eng.

32 NVIDIA. (Feb. 2019). *NVIDIA*. “NVIDIA Open Data Distribution Service (Linux).” Accessed on November 01, 2021 at https://docs.nvidia.com/drive/archive/5.1.6.0L/nvvib_docs/DRIVE_OS_Linux_SDK_Development_Guide/baggage/Open_Data_Distribution_for_LNX_User_Guide.pdf.

33 Siemens Energy. (Oct. 27, 2021). “Green energy for New York: Siemens Energy will connect state’s first utility scale offshore wind farm to the grid.” Accessed on Dec. 02, 2021 at https://press.siemens-energy.com/global/en/pressrelease/green-energy-new-york-siemens-energy-will-connect-states-first-utility-scale-offshore.

34 Businesswire. (Jun. 05, 2014). “Culham Centre for Fusion Energy Selects PrismTech’s OpenSplice DDS for Remote Handling System.” Accessed on Feb. 01, 2022 at https://www.businesswire.com/news/home/20140605005591/en/Culham-Centre-for-Fusion-Energy-Selects-PrismTech%E2%80%99s-OpenSplice-DDS-for-Remote-Handling-System.

35 Real Time Innovations. (Feb. 14, 2012). *RTI*. “RTI Selected for Wind Power Plant Design by Siemens,” Accessed on Nov. 29, 2021 at https://www.rti.com/news/siemens-wind-power.

36 Real Time Innovations. (Jan. 27, 2014). *RTI*. “LocalGrid Technologies and RTI Partner to Deliver Secure MicroGrid Solutions Using Advanced DDS Protocol.” Accessed on Feb. 01, 2022 at https://www.rti.com/news/localgrid-secure-microgrid-using-dds.

37 Real-Time Innovations. (Nov. 07, 2017). *RTI*. “RTI Advances Management of Smart Grid IoT Devices with $1M of Department of Energy Funding.” Accessed on Feb. 01, 2022 at https://www.rti.com/news/rti-advances-management-of-smart-grid-iot-devices-with-1m-of-department-of-energy-funding.

38 MD PnP Program. (2013). *MdPnP*. “MD PnP Program | OpenICE.” Accessed on Nov. 29, 2021 at http://mdpnp.org/MD_PnP_Program___OpenICE.html.

39 Real-Time Innovations. (Dec. 11, 2015). “On-demand Webinar: GE Healthcare’s Industrial Internet of Things (IoT) Architecture.” Accessed on Nov. 29, 2021 at https://www.rti.com/ge2015dec.

40 Canadian Healthcare Technology. (Feb. 03, 2021). *Canadian Healthcare Technology*. “Frost & Sullivan highlights GE’s command centres.” Accessed on Dec. 02, 2021 at https://www.canhealth.com/2021/02/03/frost-sullivan-highlights-ges-command-centres/.

41 Real-Time Innovations. (2020). “RTI Customer Snapshot: DocBox.” Accessed on Feb. 01, 2022 at https://info.rti.com/hubfs/Collateral_2017/Customer_Snapshots/RTI_Customer-Snapshot_60005_DocBox_V4_0718.pdf.

42 Object Computing. (Apr. 13, 2021). *Object Computing*. “Case Study: Plotlogic Reimagines Precision Mining With OpenDDS.” Accessed on Feb. 01, 2022 at https://objectcomputing.com/case-studies/plotlogic-reimagines-precision-mining-with-opendds.

43 Real-Time Innovations. (2020). *Real-Time Innovations*. “RTI Customer Snapshot: Komatsu.” Accessed on Feb. 01, 2022 at https://info.rti.com/hubfs/Collateral_2017/Customer_Snapshots/RTI_Customer-Snapshot_60007_Komatsu_V7_0718.pdf.

44 ADLINK Technology. (2014). *ADLINK Technology*. “Atlas Copco Selects Vortex OpenSplice DDS for its Latest Construction and Mining Products.” Accessed on Feb. 1, 2022 at https://www.adlinktech.com/en/vortex-dds-client-Atlas.

45 International Federation of Robotics. (2021). “Executive Summary World Robotics 2021 Service Robots.” Accessed on Feb. 1, 2022 at https://ifr.org/img/worldrobotics/Executive_Summary_WR_Industrial_Robots_2021.pdf.

46 Dirk Thomas. (Sep. 2017). *ROS 2 Design*. “ROS 2 middleware interface,” Accessed on Nov. 29, 2021 at https://design.ros2.org/articles/ros_middleware_interface.html.

47 Robot Operating System Consortium. (Jan. 02, 2022). “ROS Metrics.” Accessed on Feb. 1, 2022 at https://metrics.ros.org/packages_top.html.

48 Alberto Soragna, Juan Oxoby, and Goel Dhiraj. (Nov. 01, 2019). “ROS 2 for Consumer Robotics - The iRobot use-case.” Accessed on Feb. 01, 2022 at https://roscon.ros.org/2019/talks/roscon2019_irobot_usecase.pdf.

49 Matt Hansen. (May 03, 2021). *Amazon Web Services*. “AWS RoboMaker now supports ROS2 Foxy Fitzroy featuring Navigation2.” Accessed on Feb. 01, 2022 at https://aws.amazon.com/blogs/robotics/aws-robomaker-now-supports-ros2-foxy-fitzroy-featuring-navigation2/.

50 Alberto Soragna, Juan Oxoby, and Goel Dhiraj. (Nov. 01, 2019). *ROSCON*. “ROS 2 for Consumer Robotics - The iRobot use-case.” Accessed on Feb. 01, 2022 at https://roscon.ros.org/2019/talks/roscon2019_irobot_usecase.pdf.

51 John Lawrence. (May 2020). *Rochester Institute of Technology*. “ROS 2 Prevalence and Security Risks.” Accessed on Feb. 01, 2022 at https://www.johnlawrencecsec.com/ROS2_Prevalance_and_Security_Risks.pdf.

52 Kim, J. M. Smereka, C. Cheung, S. Nepal, and M. Grobler. (Sep. 2018). *Cornell University*.“Security and Performance Considerations in ROS 2: A Balancing Act.” Accessed on Nov. 17, 2021 at http://arxiv.org/abs/1809.09566.

53 UIC. (Feb. 01, 2022). *International Union of Railways*. “RAILISA Statistics.” Accessed on Feb. 01, 2022 at https://uic.org/support-activities/statistics/.

54 ADLINK Technology. (2014). *ADLINK Technology*. “ProRail Deploys Vortex OpenSplice.” Accessed Nov. 29, 2021 at https://www.adlinktech.com/en/ProRail.

55 Scott Welsch. (Jan. 04, 2021). *EFC*. “How Many Airports Are There in the World?” Accessed on Nov. 29, 2021 at https://euflightcompensation.com/how-many-airports-are-there-in-the-world/.

56 Wikipedia. (Nov. 27, 2021). *Wikipedia*. “Runway.” Accessed on Nov. 29, 2021 at https://en.wikipedia.org/w/index.php?title=Runway&oldid=1057372302.

57 ADLINK Technology. (2010). *ADLINK Technology*. “Coflight Consortium Selects Vortex OpenSplice DDS Middleware for Next Generation European Flight Data Processor.” Accessed on Feb. 01, 2022 at https://www.adlinktech.com/en/Coflight.

58 Airport Technology. (Sep. 08, 2013). *Airport Technology*. “NAV CANADA improves air traffic management with RTI platform.” Accessed on Nov. 29, 2021 at https://www.airport-technology.com/uncategorised/newsnav-canada-improves-air-traffic-management-rti-platform.

59 Statista. (2019). *Statista*. “Projected number of autonomous cars worldwide.” Accessed on Feb. 01, 2022 at https://www.statista.com/statistics/1230664/projected-number-autonomous-cars-worldwide/.

60 Kai Richter and Emilio Guijarro Cameros. (Mar. 02, 2021). *Real-Time Innovations*. “AUTOSAR and DDS: A Fresh Approach to Enabling Flexible Vehicle Architectures.” Accessed on Nov. 29, 2021 at https://www.rti.com/blog/fresh-approach-to-enabling-flexible-vehicle-architectures.

61 Erik Boasson. (Nov. 15, 2021). *GitHub*. “Added QNX OS support.” Accessed on Nov. 29, 2021 at https://github.com/eclipse-cyclonedds/cyclonedds/pull/847.

62 CISA. (Nov. 11, 2021). *CISA*. “Multiple Data Distribution Service (DDS) Implementations.” Accessed on Nov. 13, 2021 at https://us-cert.cisa.gov/ics/advisories/icsa-21-315-02.

63 Ruffin White, Gianluca Caiazza, Chenxu Jiang, Xinyue Ou, Zhiyue Yang, Agostino Cortesi, and Henrik Christensen. (Aug. 2019). *Cornell University*. “Network Reconnaissance and Vulnerability Excavation of Secure DDS Systems.” Accessed on Nov. 17, 2021 at http://arxiv.org/abs/1908.05310.

64 CISA. (Jan. 17, 2014). “UDP-Based Amplification Attacks.” Accessed on Nov. 30, 2021 at https://us-cert.cisa.gov/ncas/alerts/TA14-017A..

65 Ruffin White, Gianluca Caiazza, Chenxu Jiang, Xinyue Ou, Zhiyue Yang, Agostino Cortesi, and Henrik Christensen. (Aug. 2019). *Cornell University*. “Network Reconnaissance and Vulnerability Excavation of Secure DDS Systems.” Accessed on Nov. 17, 2021 at http://arxiv.org/abs/1908.05310.

66 Thomas White, Michael N. Johnstone, and Matthew Peacock. (2017). *Edith Cowan University*. “An investigation into some security issues in the DDS messaging protocol,” Australian Information Security Management. Accessed on December 2021 at https://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1215&context=ism.

67 Michael Michaud, Thomas Dean, and Sylvain Leblanc.(2018). “Attacking OMG Data Distribution Service (DDS) Based Real-Time Mission Critical Distributed Systems.” Accessed on Dec. 2021 at https://ieeexplore.ieee.org/document/8659368.

68 Ruffin White, Gianluca Caiazza, Chenxu Jiang, Xinyue Ou, Zhiyue Yang, Agostino Cortesi, and Henrik Christensen. (Aug. 2019). *Cornell University*. “Network Reconnaissance and Vulnerability Excavation of Secure DDS Systems.” Accessed on Nov. 17, 2021 at http://arxiv.org/abs/1908.05310.

69 We used the full database from https://www.maxmind.com/en/geoip2-services-and-databases.

70 Rebus is a .NET abstraction layer between an application a queueing or database system: https://rebus.fm/what-is-rebus/.

71 Schengen Visa Info. (Nov. 01, 2021). *Schengen Visa Info*. “EU Investigates Hacking of COVID Digital Certificates Gateway, After ‘Adolf Hitler Certificate’ Was Generated.” accessed on Dec. 01, 2021 at https://www.schengenvisainfo.com/news/eu-investigates-hacking-of-covid-digital-certificates-gateway-after-adolf-hitler-certificate-was-generated/.

72 Trend Micro. (Jan. 27, 2022). *Youtube*. “A Security Analysis of the Data Distribution Service Protocol.” Accessed on Jan. 27, 2022 at https://www.youtube.com/watch?v=fUvggLRMq1M.

73 Robotis. (2022). *Robotis*. “Robotis e-Manual.” Accessed on Dec. 01, 2021 at https://emanual.robotis.com/docs/en/platform/turtlebot3/autonomous_driving/.

74 Federico Maggi and Victor Mayoral-Vilches. (Nov. 29, 2021). *GitHub*. “RTPS contrib layer · Pull Request #3403 · secdev/scapy.” Accessed on Feb 16, 2022 at https://github.com/secdev/scapy/pull/3403.

75 njv299. (Nov. 15, 2017). *GitHub*. “afl-unicorn.” Accessed on Nov. 2021 at https://github.com/Battelle/afl-unicorn/blob/master/unicorn_mode/helper_scripts/template_test_harness.py.

76 National Vulnerability Database. (n.d.). *National Vulnerability Database*. “Search Results.” Accessed on Dec. 2021 at https://nvd.nist.gov/vuln/search/results?query=ezxml.

77 This harness had been already contributed when we started this project.

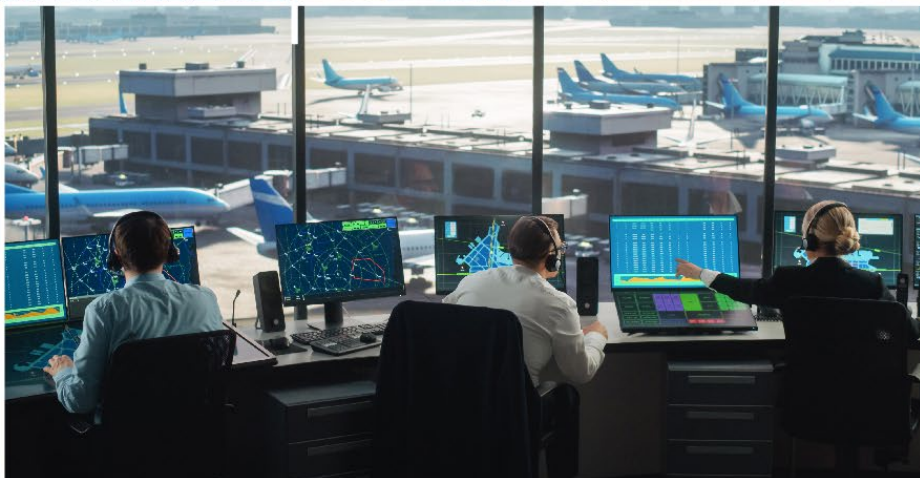
78 Object Management Group. (Mar. 2021). *Object Management Group*. “The Real-time Publish-Subscribe Protocol DDS Interoperability Wire Protocol (DDSI-RTPS) Specification 2.5.” Accessed on dec. 2021 at https://www.omg.org/spec/DDSI-RTPS/2.5/PDF.

79 Jongkil Kim, Jonathon Smereka, Calvin Cheung, Surya Nepal, and Marthie Grobler. (2018). *Cornell University*. “Security and Performance Considerations in ROS 2: A Balancing Act.” Accessed on Nov. 17, 2021 at http://arxiv.org/abs/1809.09566.

80 Erik Boasson. (Feb. 16, 2022). *GitHub*. “cyclonedds.” Accessed on Feb. 2022 at https://github.com/eclipse-cyclonedds/cyclonedds/tree/master/fuzz.





TREND MICRO™ RESEARCH

Trend Micro, a global leader in cybersecurity, helps to make the world safe for exchanging digital information.

Trend Micro Research is powered by experts who are passionate about discovering new threats, sharing key insights, and supporting efforts to stop cybercriminals. Our global team helps identify millions of threats daily, leads the industry in vulnerability disclosures, and publishes innovative research on new threat techniques. We continually work to anticipate new threats and deliver thought-provoking research.

www.trendmicro.com

TREND MICRO™ | research