Network Reconnaissance and Vulnerability Excavation of Secure DDS Systems

Abstract—**Data Distribution Service (DDS) is a realtime peer-to-peer protocol that serves as a scalable middleware between distributed networked systems found in many Industrial IoT domains such as automotive, medical, energy, and defense. Since the initial ratification of the standard, specifications have introduced a Security Model and Service Plugin Interface (SPI) architecture, facilitating authenticated encryption and data centric access control while preserving interoperable data exchange. However, as Secure DDS v1.1, the default plugin specifications presently exchanges digitally signed capability lists of both participants in the clear during the crypto handshake for permission attestation; this breaches confidentiality of the context of the connection. In this work, we present an attacker model that makes use of network reconnaissance afforded by this leaked context in conjunction with formal verification and model checking to arbitrarily reason about the underlying topology and reachability of information flow, enabling targeted attacks such as selective denial of service, adversarial partitioning of the data bus, or vulnerability excavation of vendor implementations.**

Index Terms—**Data Distribution Service, Protocols, Security, Network Reconnaissance, Formal Verification**

I. Introduction

The ubiquity of connected and autonomous devices defined as Internet of Things (IoT) and Industrial IoT (IIoT), has uncovered how the limited resources and the weak security design choices that have been made in the past represent a source of concerns in terms of safety and security in deployment. To address those problems we can distinguish between two lines of research, either studying the security and hardening solutions for the devices or focusing on the communication infrastructures.

This work regards the latter; in particular we discuss Data Distribution Service (DDS) [1] from the Object Management Group (OMG), a widely used[1] real-time middleware communication mechanism based on a publish-subscribe model. This standard, used in several industries including Automotive, Transportation, Healthcare, Energy systems, Aerospace, Defense, etc., permits one to build large scaled distributed network without relying on a centralized server. However, such applications require a rigid security mechanism since any potential vulnerability can possibly lead to millions in economic losses or damages. In order to cope with requests, this design for Secure DDS Plugins [2] has been developed. This enhanced version of the original DDS protocol adds Authentication, Access Control, Domain protection and Cryptographic support to the standard.

In further detail, the security model adopted is meant to provide: Confidentiality of the data samples, data and messages

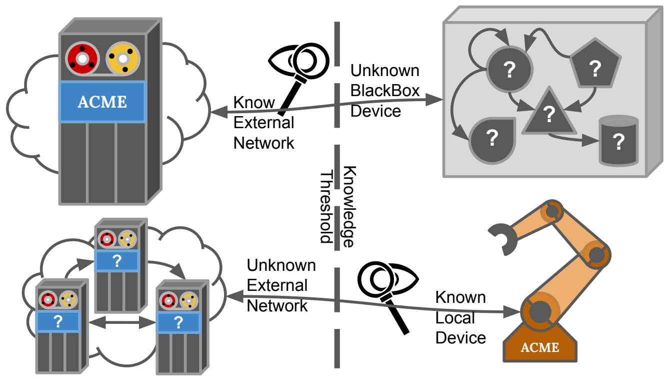


Fig. 1: An example scenario where a external observer wishes to internally analyze either a closed off device or an external restricted network by monitoring traffic between know connected systems. Such an adversary may be hunting for vulnerabilities to later exploit or to attempt to recover a hidden state or purpose of the system.

Integrity, Authentication and Authorization of DDS writers and readers, message-origin and data-origin Authentication, and optionally non-repudiation. By enforcing those properties, threats such as unauthorized subscriber and publisher creation, tampering and replay messages, and unauthorized access to data, are blocked. Nevertheless, the proposed threat model doesn't cover permission confidentiality [2]. In fact, by analyzing the plugin, we can see how the nodes' handshakes are performed by exchanging a plain text permission file. Although digitally signed to preserve Integrity and block an unauthorized node from accessing resources via forged permissions, its transmission plain text voids its Confidentiality. Permission files define a node's capability to read and or write data in a certain domain on the databus. By leaking such information, an attacker can infer the application layer topology by comparing the capabilities of each node and deducing possible connections without having to decrypt ciphered data messages. A case example, where the topic names themselves may remain sensitive, could include when the topics offer some clue as to the private number of resources, e.g. the number of safety sensors or alarms armed in a network. If such topics are indexed sequentially, then an attacker may use a classical statistical theory of estimation, similar to that applied during WW2 to solve the 'German tank problem'.

Unlike traditional network reconnaissance methods like us-

[1]https://www.omgwiki.org/dds/who-is-using-dds-2/

[2]https://issues.omg.org/issues/DDSSEC12-13

ing traceroute in which an attacker needs to query the network repeatedly to obtain information about the topology [3], that may trigger alarms to network administrators, the methods we present allow an attacker to construct the topology of the underlying data bus merely by passively sniffing the packets inside the network. As per the traditional case, administrators can employ techniques that obscure the network itself [4] to impede an attacker from reconstructing the true network topology, or that trigger intrusion detection countermeasures before an actual attack is executed. In a passive attack scenario, it becomes substantially harder to identify an attacker before any malicious operation is performed. Therefore, we investigate how revealing the data flow semantics for each node and its functional role in the network renders DDS networks more vulnerable when facing malicious adversaries.

The rest of the paper is divided as follows: Section II provides an overview and technical description of the secure DDS protocol and related components incorporated into our approach; Section III details our threat and attack model assumptions; Section IV details our approach in partially reconstructing data bus topology and inferencing reachability through the network at scale; Section V documents our experimental setup and testing infrastructure; Section VI demonstrates how an attacker may isolate information flow from a single node by identifying critical targets or verify reachablity from a selected source to target destination; Section VII discusses related work in relation to network reconnaissance, DDS networking and information flow control; Section VIII summarizes our main contributions and discusses potential mitigations and their caveats, as well as future work addressing remaining issues in remote access control attestation.

II. Background

A. Data Distribution Service

The Data Distribution Service (DDS) [1] is a standardized network middleware protocol that aims to provide reliable and scalable service based on a publish-subscribe model, i.e. a data centric model based on a conceptual *Global Data Space*. The decoupled nature of publisher-subscriber compared to an ordinary request-response model renders the protocol more suitable for real-time systems and IoT applications. Applications can choose to have publishers and/or subscribers, where the data model underlying the *Global Data Space*, or a DDS *Domain*, is a set of data objects. A *Publisher* is an object responsible for data distribution and may publish data of different data types. Similarly, a *Subscriber* is an object responsible for receiving published data and making it available for the receiving application. Topic objects conceptually fit between publications and subscriptions. These *DomainParticipant*s can respectively write or read in a *Domain*, which denotes the set of all applications that can communicate with each other. *Topic* objects conceptually fit between publications and subscriptions, and uniquely identify the name, data type and corresponding Quality of Service (QoS) associated with the data on both the publisher and the subscriber sides.

B. Authentication

Each DomainParticipant needs to go through an authentication before it can communicate inside a domain. On start, a DomainParticipant authenticates its local identity to others in the network using its own public certificate. This Identity Certificate is signed by the Identity Certificate Authority (CA) [2]. Each DomainParticipant will then verify the authentication of a discovered remote peer through a mutual handshake request and reply messages. Among other tokens inside the handshake request, the Identity Certificate and the Domain Participant Permissions (detailed in next section) of a remote peer will also be included; this becomes the major security vulnerability we exploit in this work.

C. Access Control Service

In order to ensure authorization of DDS publishers and subscribers, DDS defines an Access Control Plugin. The DomainParticipant must be provisioned access to given domains, publish access to topics for data it produces, and subscribe access to topics for data it consumes. In addition, there are more capability permissions that further segment data access, such as DDS *partitions*, *data tags*, and *domain tags*, that are omitted from our discussion for brevity but are accounted for in our approach. Three configuration documents are associated with the Access Control Service: a Permissions CA Certificate, a Domain Governance signed by Permission CA, and a Domain Participant Permission signed by the Permission CA. The Permissions CA Certificate contains a public key from the CA that signs the other two documents. The Domain Governance is a XML document specifying the protection policy inside this domain, including whether or not to enforce encryption, whether to set specific limitations on certain topics, etc. The Domain Participant Permission is a XML document containing the permissions of a DomainParticipant. Essentially, it is a set of *grants* that denotes the rules to either reject or allow the DomainParticipant to write or access certain *topics*, inside certain *partitions* of a domain, with certain *data tags* associated with the DomainParticipant. It also includes the domain the DomainParticipant allowed is to communicate in, and the time period that the DomainParticipant can communicate [2].

D. Imandra

Imandra[3] is a formal verification tool, originally purposed for model checking financial market software and exchange protocols [5]. It is highly adaptable and performs the nonlinear arithmetic, automated induction, etc. that we need to infer the proofs or counterexamples to resolve out SAT formulation of permission intersections. Formal verification techniques can reason about a large state space without exhaustive search. Still, when surveying DDS networks at scale, solving such SAT queries would remain a bottleneck, and thus should be optimized for by reducing the number of queries required when inferring about the network. With Imandra's public APIs, we can simplify the development process of our attack model and completely automate the vulnerability excavation pipeline.

[3]https://www.imandra.ai

III. THREAT

A. Threat Model

Considering the permission leakage in the network, the threat model proposed in this work includes the following:

- Attacker has access to victim's network traffic en-route.
- Attacker does not know network semantics completely.
- Attacker does not own a valid permission file from CA.
- Attacker could choose to actively control network traffic by selectively dropping network traffic, poisoning the routing, or physically disrupting a participant device.

B. Attack Model

The minimum requirement in our model to execute an attack on DDS is to have access to the network. As discussed, having network access grants the ability to sniff the communications and acquire the packets exchanged by all participating nodes, including handshake packets, which contain the permission tokens. In detail, we can break the attack into two phases based on whether the attacker has the ability to control the network. If the attacker can only record the network then it can perform *passive* attacks; on the other hand, it has some degree of influence on the network, *active* realtime attacks become feasible.

Starting from the passive attack prospective, by just relying on the permission files captured by sniffing traffic, or received as a response by actively sending handshakes to the unknown node, it constructs a topology graph. In this way, the attacker is able to understand the semantics of the network, and learn how devices interact with each other on what specific topics. On the other hand, with some level of control of the network, an active attacker can do much more than learning sensitive information from it. For example, if some node of the network is well protected, an active attacker can leverage our approach to query a minimal set of nodes to take it over. In this way, it would be possible to cut the information flow from the network to the specific node, effectively killing the node by rendering it useless. Even if an attacker does not have the capability to control the traffic, in a wireless scenario, an attacker can trace the signal, locate the node's physical location and kill the node physically. In addition to the disruptive attacks, a more subtle technique is to tinker with the QoS in the network and introduce delays, thus degrade the performance and force realtime devices to fall back on backup solutions.

One example would be a highly connected peer-to-peer network such as the Cooperative Intelligent Transport Systems (C-ITS)[4] under development of the European Commission, whose goal is to build a smart-city scaled network to exchange realtime data among vehicles and other road infrastructural facilities to optimize traffic management and take full advantage of highly automated vehicles (level 4/5) [6]. Due to the safety oriented behaviours imposed to autonomous robotic systems by the regulator, those attacks can tamper the entire infrastructure and make it potentially unusable. In the discussed example, if an level 5 autonomous vehicle (fully automated) is abruptly excluded from the network, or it detect a delay on the communications, the only solution is to rely on the on-board sensors and securely pull over. However, a well coordinated attack would be unnoticed and the delays could potentially lead to accidents.

[4]https://ec.europa.eu/transport/themes/its_en

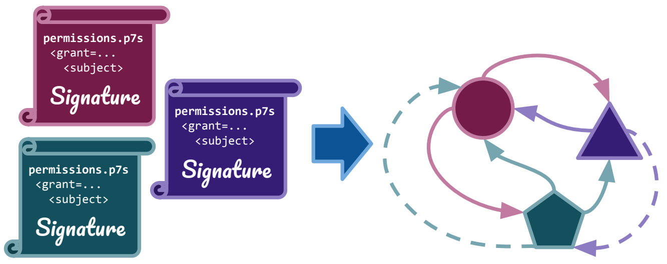


Fig. 2: Captured permission tokens are used to infer the data flow between participants as well as potential loopholes and administrator-defined policies that may violate the intended purpose of deployment.

IV. APPROACH

Under the assumptions discussed in the previous section, we know that once an attacker acquire the handshake packets it can construct the semantic network topology by interpreting the permission files. The sample snippets with Fig 9 in the appendix depict the example permission files that we will use to illustrate this process. After obtaining the network topology, we also explore how an attacker may formulate queries regarding the network's connectivity.

Example queries include: 1) Given the set of nodes A and B in the cyclic graph G, what minimal set nodes in G, exclusive of A and B, would need to be disrupted to discontinue information flow from A to B; 2) Given a source A, what are the nodes that we need to offline in order to isolate all information flow from set A; 3) Given a destination set B, what set of nodes would need to be compromised to prevent B from only receiving information from the rest of G. With this information, an attacker may then selectively partition any node from the rest of the network with minimal invested effort or detectable network disturbance.

A. Network Topology

We depict the network topology as a directed graph with vertices representing nodes in the network, and edges indicating that there exists at least one topic match between the two connected vertices. The primary reason for a directed graph is that we need to distinguish publish and subscribe actions, which can naturally be described using directional edges, with edges pointing from a publisher to a subscriber. Additionally, to account for a third 'relay' permission type, we decompose all relay actions to a combination of subscribe and publish capabilities on the topic. This reduction not only decreases the complexity of inferring information flow but also eases the graph visualization and introspection.

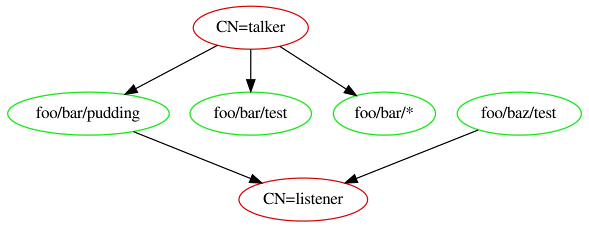


Fig. 3: Raw Graph Obtained by Scanning Permission Files

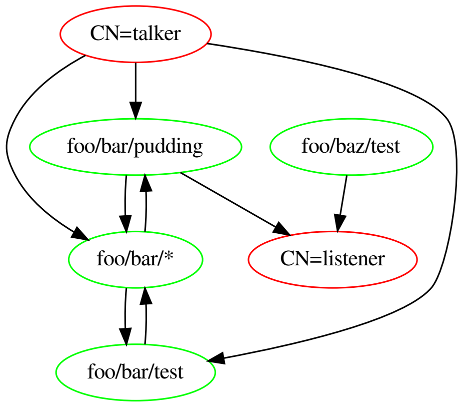
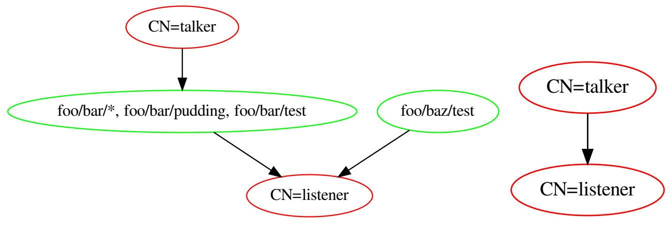


Fig. 4: Connected Graph Obtained by Connecting Topics

B. Heuristic Graph and Lazy Evaluation

In real world applications, a network may consist of hundreds or even thousands of nodes. Such tremendous scales inevitably make any graph construction or influencing a resource an intensive task. A naive approach to constructing a network topology requires the consideration of all permission files when computing for the potential intersect in respective permission grants. However, this is impractical given an exhaustive $O(n(n-1)/2)$ would be done using our formal verification of grant intersections; this is among the most computationally intensive steps in our attacker pipeline. Instead, our approach reduces query time latency via admissible heuristics and lazy evaluation. By first generating a heuristic graph to approximate the information flow, we substantially curtail the number of expensive inferences on grant intersections. Thus, while the initial model may exaggerate apparent connectivity, we can remain assured that resulting reachability queries via formal verification remain complete.

Generating a heuristic graph mostly relies on the fast and admissible approximation as to whether or not to connect two vertices. We decompose this approximation process into three phases: 1) First, a simple directed graph is created by indexing each grant in the permission files, adding respective vertices for both nodes as well as topics to the graph without duplicates, and then connecting nodes and topics according to the direction of information flow. This results in a directed bipartite graph such that vertex set U consists of all nodes in the network and vertex set V includes all topics involved in the network. Figure 3 shows a sample graph on a simple network with a talker and a listener. In this simple network, vertex set U consists of nodes talker and listener, whereas vertex set V is comprised of four topics.



(a) Contracted G (b) Heuristic G

Fig. 5: Contracted Graph is obtained by collapsing related topics into single node, while the Heuristic Graph then is obtained by collapsing topic vertices.

This graph is quick to generate as nodes as well as topics can be iteratively appended on the fly, rather than holistically batching the entire graph all at once and performing intersection checks between any two nodes' publish and subscribe topic expression.

2) The second phase focuses on combining related topics to form connected components of topics and then collapses the topics into a single vertex. By combining related topics, we mean drawing bidirectional edges between any two topics that match at least once using two way 'fnmatch': the POSIX string matching function chosen in the Secure DDS standard. An example of this is in Figure 4, where we have one such connected component formed by three topics including *foo/bar/pudding*, *foo/bar/**, and *foo/bar/test*. The transition from Figure 4 to Figure 5a illustrates the process of collapsing the connected components into a single vertex. Although this process is simple, it may potentially increase the total number of paths between different nodes in the network. The extra paths we get do not exist in the real network topology, hence a heuristic graph instead of an exact model.

3) During the last phase, we further reduce the bipartite graph to a regular network topology by eliminating topics vertex set and connect nodes that might have the capabilities to communicate on some topic. In our simple example, we get Figure 5b as a heuristic graph after completing this step, which serves as a foundation to answer the connectivity query.

Given the retrieving of a heuristic graph, naive queries on reachability using the simple edge traversal would be inaccurate; our approach resolves this via lazy evaluation. First, using the naive path computed on the heuristic graph, i.e. using Dijkstra Algorithm or A*, the resulting edge sequence or node pairs are iteratively verified for directional connectivity using a satisfaction constraint solver. We describe the reachability verification process in detail under section IV-C. By pruning paths and edges sequences at query time, we avoid unnecessarily checking unfeasible flows derived from topic permission mismatches.

C. Reachability Verification

During the handshake phase, two DDS DomainParticipants will each verify that the other has the permission to access the resource in question. For the subject node that is advertising its access, we will abstract this into a *subject* representation; containing the information about the name of the subject, the action it is requesting, the topics that it advertises to publish or subscribe, and other subject criteria regulated by access control. Algorithm 1 in the index details how each node will validate the provided subject representation with the subject's respective permission file, and return a qualifier: ALLOW or DENY of the request.

The access control algorithm checks the grant in the permissions file that matches the supplied subject and is valid at the time it is evaluated. For this grant, it sequentially enumerates through all the rules *in order*, and returns immediately if there is a match between the rule and the subject. The matching is conditioned upon many *criteria*s including topics, partitions and data tags. If no rule is matched, the returned qualifier falls through to the grant's default behavior.

To check for permissive exchanges between grants and determine whether data flow between given nodes is possible, we must formally verify the intersection of the two permissions files; i.e. either assert or refute the existence of a pair of matching subjects that satisfy all pairwise constraints. More precisely, given two nodes A and B, and their corresponding permission files PermA and PermB, find two subjects SubA and SubB such that all the following hold:

$$Evaluate(PermA, SubA) = ALLOW \tag{1}$$
$$Evaluate(PermB, SubB) = ALLOW \tag{2}$$
$$Match(SubA, SubB) \text{ or } Match(SubB, SubA) \tag{3}$$

The constraints above dictate that both subject instances must conform to the respective permissions, while the QoS attributes of both subjects such as topic, partition, and data tags must also correspond. The following section details the construction and consumption of such constraints.

V. Implementation

To validate our approach, we construct an experimental setup with a reproducible test harness as a pipeline for the entire attacker model. Docker is used to containerize three main processes, as well as vitalize a target Secure DDS deployment, as shown in Fig 2.

We first programmatically synthesize a DDS application with minimal spanning permissions, valid PKI and CA trust anchors, where the digitally signed governance enforces authenticated encryption for all transport. This experimental configuration is then provided to an isolated simulation control that launches each participant in separate containers within a controlled software defined network. The first few seconds of network traffic is consecutively recorded to capture initial Real Time Publish Subscribe (RTPS) protocol discovery data, see Fig 7, and then given to the attacker.

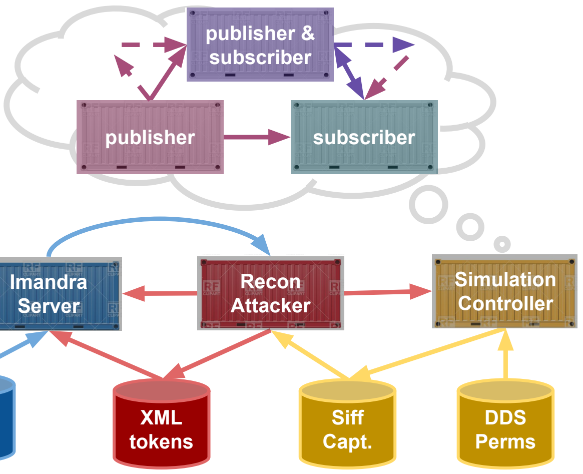
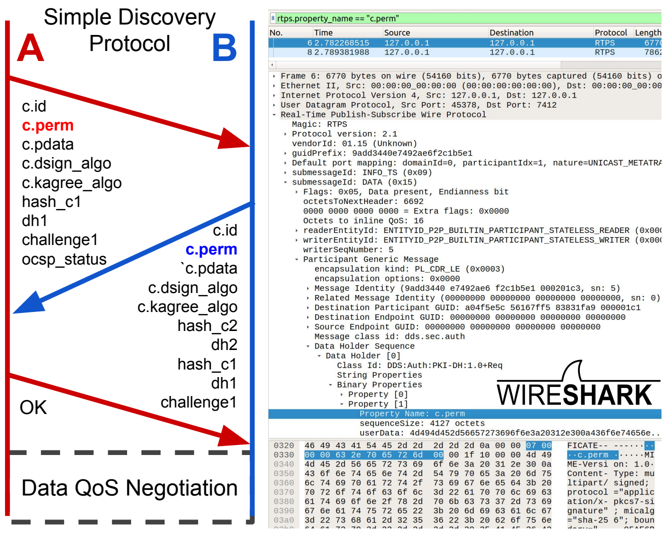


Fig. 6: Visual of experimental setup and test harness. Network discovery traffic between Secure DDS participants is captured and used in concert with the SAT solver to infer application typology from intersecting permissions. The attacker then uses this feedback to precisely influence the information flow.



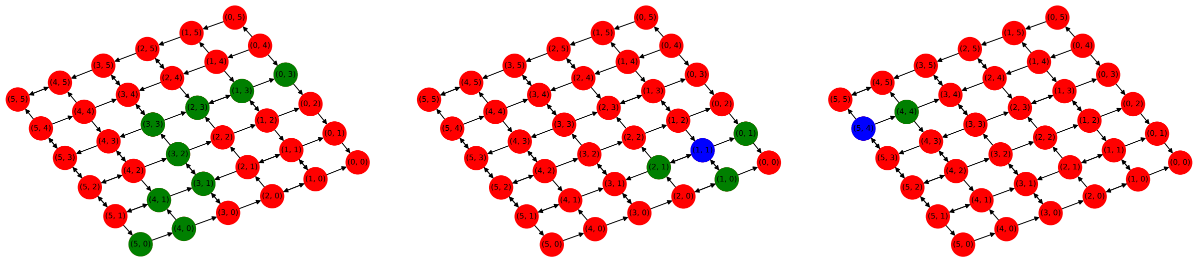
```
rtps.property_name == "c.perm"
No.   Time          Source      Destination   Protocol  Length
  6   2.782268515   127.0.0.1   127.0.0.1     RTPS      6770
  8   2.789381988   127.0.0.1   127.0.0.1     RTPS      7862

› Frame 6: 6770 bytes on wire (54160 bits), 6770 bytes captured (54160 bits) o
› Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00
› Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
› User Datagram Protocol, Src Port: 45378, Dst Port: 7412
˅ Real-Time Publish-Subscribe Wire Protocol
    Magic: RTPS
  › Protocol version: 2.1
    vendorId: 01.15 (Unknown)
  › guidPrefix: 9add3440e7492ae6f2c1b5e1
  › Default port mapping: domainId=0, participantIdx=1, nature=UNICAST_METATRA
  › submessageId: INFO_TS (0x09)
  ˅ submessageId: DATA (0x15)
    › Flags: 0x05, Data present, Endianness bit
      octetsToNextHeader: 6692
      0000 0000 0000 0000 = Extra flags: 0x0000
      Octets to inline QoS: 16
    › readerEntityId: ENTITYID_P2P_BUILTIN_PARTICIPANT_STATELESS_READER (0x000
    › writerEntityId: ENTITYID_P2P_BUILTIN_PARTICIPANT_STATELESS_WRITER (0x000
      writerSeqNumber: 5
    ˅ Participant Generic Message
        encapsulation kind: PL_CDR_LE (0x0003)
        encapsulation options: 0x0000
      › Message Identity (9add3440 e7492ae6 f2c1b5e1 000201c3, sn: 5)
      › Related Message Identity (00000000 00000000 00000000 00000000, sn: 0)
      › Destination Participant GUID: a04f5e5c 56167ff5 83831fa9 000001c1
      › Destination Endpoint GUID: 00000000 00000000 00000000 00000000
      › Source Endpoint GUID: 00000000 00000000 00000000 00000000
        Message class id: dds.sec.auth
      ˅ Data Holder Sequence
        ˅ Data Holder [0]
            Class Id: DDS:Auth:PKI-DH:1.0+Req
            String Properties
          ˅ Binary Properties
            › Property [0]
            ˅ Property [1]
                Property Name: c.perm
                sequenceSize: 4127 octets
                userData: 4d494d452d56657273696f6e3a20312e300a436f6e74656e..
0320  46 49 43 41 54 45 2d 2d  2d 2d 2d 0a 00 00 07 00   FICATE-- ------··
0330  00 00 63 2e 70 65 72 6d  00 00 1f 10 00 00 4d 49   ··c.perm ······MI
0340  4d 45 2d 56 65 72 73 69  6f 6e 3a 20 31 2e 30 0a   ME-Versi on: 1.0·
0350  43 6f 6e 74 65 6e 74 2d  54 79 70 65 3a 20 6d 75   Content- Type: mu
0360  6c 74 69 70 61 72 74 2f  73 69 67 6e 65 64 3b 20   ltipart/ signed;
0370  70 72 6f 74 6f 63 6f 6c  3d 22 61 70 70 6c 69 63   protocol ="applic
0380  61 74 69 6f 6e 2f 78 2d  70 6b 63 73 37 2d 73 69   ation/x- pkcs7-si
0390  67 6e 61 74 75 72 65 22  3b 20 6d 69 63 61 6c 67   gnature" ; micalg
03a0  3d 22 73 68 61 2d 32 35  36 22 3b 20 62 6f 75 6e   ="sha-25 6"; boun
```

Fig. 7: Simple Discovery Protocol, shown in the network ladder diagram, takes place when participants exchange identities and permission tokens. An RTPS packet containing clear text permission tokens bytes is highlighted in Wireshark capture.

The attacker process strips all permission tokens for the raw packet capture and constructs a graph based database of permission tokens and respective origin/destination IP address. This database is then shared with the SAT solver to call queries' agents.

For formal verification, we utilize Imandra as our selected SAT solver by replicating the access control evaluation logic as defined by the DDS specification in OCaml, a strongly typed functional programming language supported with Imandra. This allows us to quickly prototype and experiment with alternate security plugin designers with minimal modification.



(a) Prove Data Dependency (b) Isolate Publisher (c) Isolate Subscriber

Fig. 8: Left to right. Query 1: given source and destination, prove data dependency. Query 2: given source, determine minimum set of nodes to isolate incoming data. Query 3: given target, determine the minimum set of nodes to cut off outgoing data.

As recounted in section VIII, this becomes invaluable in discovering vendor implementation bugs and deviation from the default security plugin specification.

The model of the access control logic and accompanying token database is used by the Imandra service to solve for incremental reachability inquiries from the inquisitive attacker. The attacker uses the proved or refuted subject instances as feedback to prune the heuristic graph until the overall reachability inquiry is determined.

Armed with the associative model of DDS objects to physical network address, the attacker may finally sabotage the target application by selectively deteriorating DDS connections by commanding the simulation controller to drop specified containers from the software defined network.

To observe this disturbance, our simulated DDS network is simply composed of broadcast nodes that periodically publish a KeepAllive diagnostic message to all topics they can publish. Each node also serves as repeater, relaying any subscribed KeepAllive messages to all topics it can publish after appending its own id to avoid cyclic packets. By sampling the packet lineage from different points in the distributed application, unit tests for bisecting information flow can be verified. In the following section an illustrative example of this is presented.

VI. Results

In this section, we showcase some example queries to elucidate our work and demonstrate the correctness of our implementation on a more complex network; a 2D grid of consisting of 36 nodes is used to maintain readability.

A. Source and Target

If both source node and target node are specified, our model outputs a list of nodes as the path from source to target. For example, in the first subfigure of Figure 8a, if the source node is *(5, 0)*, and the target node is *(0, 3)*, then the model outputs a list of nodes containing all the green nodes.

B. Source Only

Given only a source node, our model displays a minimal set of nodes that an attacker needs to take down to prevent the source from passing data to all its subscribers. As shown in the second subfigure of Figure 8b, the source node is colored in blue, and the possible target nodes are colored in green. If the input is the blue node, then the model outputs a set including the three green nodes.

C. Target Only

Similarly, if only a target node is given, we will obtain a minimal set of nodes an attacker needs to attack to prevent the target from acquiring any new information from the network. This is illustrated in the third subfigure of Figure 8c, where the target node is colored in green and its source node is colored in blue.

VII. Related Work

A. Network Reconnaissance

We have demonstrated that the permission files in clear text leak application layer topology to anyone in the same network. In fact, the encrypted packets can still leak topology information to an attacker. Other techniques are needed for us to fully defend against network reconnaissance. McClure *et al.* [3] presented how tools like traceroute can be used to construct the internet layer topology. An attacker may use the topology information to find the weak links in the network and DDoS attack the weakest link. To thwart reconnaissance via traceroute, Meier *et al.* [4] proposed to limit the ICMP traffic in the network or obfuscate the traceroute result. The other problem is that IoT devices usually connect wirelessly, which allows an insider attacker to eavesdrop on a large chunk of the network. If the chunk is too large, an insider attacker would be able to rebuild the application layer topology by merely examining the flow of traffic. Hakiri *et al.* [7] proposed to connect the IoT devices with wired software defined networks using OpenFlow. Wired connection may limit the scope of possible eavesdropping and also the dynamic flexible internet layer topology nullifies the reconnaissance attempts via traceroute.

B. Flow Control

Secure DDS uses topic and partition match to enforce the flow control policy. The topic and partition expressions support fnmatch, allowing developers to build a hierarchical trust model. Secure DDS's label scheme is similar to the DStar labels [8] for single topic and partition but secure DDS assumes every node has the privileges to downgrade data it owns. The flow is possible as long as the subscribe set and publish set have an intersection, instead of a publish set needing to be a subset of the subscribe set. The policy opens probability not only for bad configuration but also for covert channels and allows an inside attacker to leak sensitive data. Therefore, the current flow control model works only if nodes that are granted a certificate by CA can be entirely trusted.

C. DDS

White *et al.* [9] present a framework that procedurally provisions access control policies for distributed middleware. Our work extends this by adding more reachability verification on fnmatch expression to ensure that no covert channels exist in candidate policies. Khaefi *et al.* [10] presented how using a bloom filter in DDS node discovery phase could significantly reduce the payload of handshake traffic at the expense of a tiny chance of collision. Encoding topic discovery data into a bloom filter indeed obscures the topic expressions while providing some probabilistic integrity of the topic permissions. However, given probabilistic data structures are subject to collisions, e.g. false-positive set matches, it remains unsuitable for access control policy enforcement.

D. Formal Verification for XACML

eXtensible Access Control Markup Language (XACML) has become an attractive standard for the specification of Access Control policies given the prevalence of existing XACML tools, human and machine readable syntax, and rich set of constructs. However these same features can also make authoring XAMCL policies prone to human error. Turkmen *et al.* [11] present a formal analysis of XACML policies by encoding them into Satisfiability Modulo Theories (SMT) formulas, facilitating formal policy analysis while relieving authors of the burden of manually proving soundness gradually. While this work remains more general in comparison, our work additionally affords soundness checks for the Policy Decision Point (PDP) business logic implemented by DDS vendors, and thus not limited to only policy definition files themselves.

VIII. Conclusion

In this work we introduced an approach for conducting passive network reconnaissance on systems relying upon Secure DDS, ascertaining a partial topological model of the underlying data bus, and associative mapping between data objects to network addressable participants. Using formal verification and model checking, we can then inquire about directed reachability through the distributed computation graph to efficiently perform vulnerability excavation offline without ever actively engaging with the targeted system. We then demonstrate how such acquired system models may then be used by an active attacker to prioritize targeted participants based on the data objects they represent or the connectivity they facilitate in the larger picture of the system, either by selectively isolating data flow to or from a given data producer/consumer without directly disturbing other participants.

Furthermore, our methods for formal verification have been used to prove two notable exploits in existing Secure DDS vendor implementations. Firstly, a policy decision point in the handshake protocol statemachine omits checking the partition criteria of remote participant connections, allowing data to be declassified to erroneous DDS partitions for which the remote participant was granted no privilege.

Secondly, the same implementation was also found to interpret topic expression match incorrectly. It naively swaps the fnmatch expression with the query arguments both ways to find the matching intersection. This allows any two malicious nodes to establish a connection using self-made-up expressions which are supersets of their permissions. Again this was verified by replicating our OCaml model to reflect the implementation and solve for the counterexample intersection of two permissions understood to be correctly non overlapping.

Although the reconnaissance methods and vulnerability excavation tooling developed over the course of our approach may inevitably prove to be of use to malicious actors, they are also immediately beneficial for general system validation and penetration testing, as when auditing mission critical systems for flaws in access control design or implementation. For example, when certifying the interfaces between the multimedia and drive-by-wire subsystem in an autonomous automotive, manufacturers may be required to formally prove or refute the set of all satisfiable data channels between the two that would be admissible by the factory permission policy, and assure that no satisfiable covert channels exist outside of the anticipated set.

The approach presented predominantly makes use of the current Secure DDS default plugin implementation, thus resolving this issue would largely serve to mitigate the practicality of the attacks demonstrated. Specifically, exchanging permission tokens in the clear during the initial crypto handshake, thus breaching confidentiality of the context of the connection is perhaps the focal issue at present. Revising the integration between the crypto and access control plugins to alternatively postponing permission token exchange through a secure channel after the crypto handshake has concluded is perhaps the most straightforward improvement. This could subsequently add another round trip delay to the overhead introduced in securing connections; however, granted the crypto handshake does not include the action request or response to begin with, it stands to reason that the permission token could be appended to the payload of the subsequent secured requests or responses.

Alternatively, one could seek to obscure the permissions embedded in the token by using an HMAC with a known key, either embedded in the token or distributing it out of band. Each topic/partition/data-tag element in the XML permission document could be replaced with say the base64 encoded digest of the expression string it replaces. Thus, upon receiving a action request from a remote participant, the local participant merely applies the same HMAC to the action searches for the matching digests in the remote permission criteria. This has the benefit of obscuring permissions from those sniffing handshake network traffic while making minimal changes to existing vendor libraries and is implementable in less than 60 additional lines of OpenSSL in for Fast RTPS. However, aside from simple string matching, additional changes would be required to fully support fnmatch expression, perhaps by having the remote participant provide the exact expression it wishes to invoke in its own permission list.

However, both of these mitigations thus far, either postponing permission exchange or obfuscating the fields in the permission token have their potential drawbacks or weaknesses. The mitigation using HMAC is particularly vulnerable as message authentication codes only really afford integrity and not confidentiality, i.e. once an attacker knows what they are looking for, it can easily ascertain whether the permission it seeks is present in the token. Given that systems that build upon DDS, like ROS2, commonly use predictably or standardized topic names, it could become trivial to brute force obscured permissions from a limited corpus of options, or correlate matching digests across tokens to infer connectivity.

In postponing permission exchange, we merely delay the invocation of the Policy Decision Point, affording a secure channel to remote participants whose privileges we have not yet attested to. Only a single trusted identity need be compromised to begin scraping the permission tokens of others in the same secure distributed network. While DDS discovery information could also be decrypted with the same compromised participant identity, the permission tokens that divulge what data a participant can access versus what they currently advertise can still be advantageous to an attacker as described previously.

IX. Future Work

A wider issue facing traditional attestation of remote privileges using digitally signed tokens is that the entire token must first be revealed in order to verify the trusted signature locally, effectively divulging all of the remote agents' capabilities, be they applicable to the current session or not. Ideally, an attestation method would allow a remote participant to prove to the local recipient only that part of it adequate to privilege; nothing more, nothing less.

An alternative approach could be to fracture the token into multiple sub-tokens that are individually verifiable and only encompass a single permission. As discussed by Caiazza *el al.* [12], the remote agent could then pick and choose the minimal required set of sub-tokens to be shared to gain access. Potentially, this adds to the complexity of the CA provisioning and expiration of permissions, as well as the coordination of sharing sub-tokens during runtime.

This sub-token scheme would not however ultimately prevent divulging the scope of privilege for a single permission, as in the case when the permission is not just a string, but also an expression, such as a matching prefix rule for all topics starting with $/foo/$ revealing that the remote participant also has access to $/foo/bar$.

To address this, future work could investigate the application of non-interactive zero-knowledge proofs to provide a mechanism for remote attestation of privilege in an access controlled protocol without divulging anything more than necessary. Aside from the provisioning of proving and verification key materials for PKI identities with periods of validity, particular challenges in using frameworks such as zk-SNARK [13] (zero-knowledge succinct non-interactive argument of knowledge) with applications using DDS networks is maintaining real time performance in terms of security overhead and scalability; that is, limiting the upper bound of computation time for verification, conserving bandwidth for sending larger proofs over the wire, and limited input sizes when transforming permission sets into a boolean circuit.

References

[1] *About the Data Distribution Service Specification Version 1.4*, Object Management Group, 04 2015.

[2] *About the Data Distribution Service Security Specification Version 1.1*, Object Management Group, 07 2018.

[3] S. McClure, J. Scambray, G. Kurtz, and Kurtz, "Hacking exposed: network security secrets and solutions," 2009.

[4] R. Meier, P. Tsankov, V. Lenders, L. Vanbever, and M. Vechev, "Nethide: secure and practical network topology obfuscation," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 693–709.

[5] D. A. Ignatovich and G. O. Passmore, *Creating Safe and Fair Markets*, AESTHETIC INTEGRATION, LTD., 02 2015.

[6] D. GROW, "Gear 2030 - high level group - final report," 2017.

[7] A. Hakiri, P. Berthou, A. Gokhale, and S. Abdellatif, "Publish/subscribe-enabled software defined networking for efficient and scalable iot communications," *IEEE communications magazine*, vol. 53, no. 9, pp. 48–54, 2015.

[8] W. G. Halfond, A. Orso, and P. Manolios, "Using positive tainting and syntax-aware evaluation to counter sql injection attacks," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2006, pp. 175–185.

[9] R. White, G. Caiazza, H. I. Christensen, and A. Cortesi, "Procedurally provisioned access control for robotic systems," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018, pp. 1–9.

[10] M. R. Khaefi and D. Kim, "Node discovery scheme of dds using dynamic bloom filters," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, Sept 2014, pp. 1–4.

[11] F. Turkmen, J. den Hartog, S. Ranise, and N. Zannone, "Formal analysis of xacml policies using smt," *Computers and Security*, vol. 66, pp. 185–203, 5 2017.

[12] G. Caiazza, R. White, and A. Cortesi, *Enhancing Security in ROS: Volume Eight*, 01 2019, pp. 3–15.

[13] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 90–108.

X. Appendix

Algorithm 1 DDS Security v1.0 Default Access Control Logic

```
procedure EVALUATE($permissions, subject$)
    for $grant$ in $permissions$ do
        $match \leftarrow grant.subject\_name.match(subject)$
        $valid \leftarrow grant.validity(current\_date\_time)$
        if $match$ and $valid$ then
            $qualifier \leftarrow$ CHECKRULES($rules$, $subject$)
            if $qualifier$ is $None$ then
                return $grant.default$
            else
                return $qualifier$
            end if
        end if
    end for
    return $ERROR$
end procedure
function CHECKRULES($rules$, $subject$)
    for $rule$ in $rules$ do
        $domain \leftarrow subject.domain$ in $rule.domainSet$
        $criteria \leftarrow rule.get(subject.action.type)$
                ▷ Action types: $publish, subscribe, relay$
        $match \leftarrow$ CHECKCRITERIA($criteria$, $subject$)
        if $domain$ and $match$ then
            return $rules.qualifier$
                    ▷ Qualifier types: $ALLOW, DENY$
        end if
    end for
    return $None$
end function
function CHECKCRITERIA($criteria$, $subject$)
    for $criterion, i$ in $criteria.criterions$ do
        $matches[i] \leftarrow$ any $(criterion.match(subject))$
                ▷ Criterion types: $topics, partitions, tags$
    end for
    return all $(matches)$
end function
function MATCH($publisher$, $subscriber$)
    $isMatched \leftarrow publisher.action = PUBLISH$ and
    $subscriber.action = SUBSCRIBE$ and
    $publisher.topic = subscriber.topic$ and
    $publisher.partition = subscriber.partition$ and
    $publisher.data_tag = subscriber.data_tag$
    return $isMatched$
end function
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dds>
  <permissions>
    <grant name="/talker">
      <subject_name>CN=/talker</subject_name>
      <validity>
        <not_before>2013-10-26T00:00:00</not_before>
        <not_after>2018-10-26T22:45:30</not_after>
      </validity>
      <allow_rule> <!-- multi and/or <deny_rule> -->
        <domains>
          <id_range> <!-- multi and/or <id> -->
            <min>10</min>
            <max>42</max>
          </id_range>
        </domains>
        <publish> <!-- multi and/or pub/sub -->
          <partitions> <!-- multi and/or <tags> -->
            <partition>food</partition>

          </partitions>
          <topics>
            <topic>foo/bar/pudding</topic>
            <topic>foo/bar/test</topic>
            <topic>foo/bar/*</topic>
          </topics>
        </publish>
      </allow_rule>
      <default>DENY</default> <!-- or >ALLOW< -->
    </grant>
```

(a) Talker Permissions

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dds>
  <permissions>
    <grant name="/listener">
      <subject_name>CN=/listener</subject_name>
      <validity>
        <not_before>2014-10-26T00:00:00</not_before>
        <not_after>2019-10-26T22:45:30</not_after>
      </validity>
      <allow_rule> <!-- multi and/or <deny_rule> -->
        <domains>
          <id_range> <!-- multi and/or <id> -->
            <min>20</min>
            <max>50</max>
          </id_range>
        </domains>
        <subscribe> <!-- multi and/or pub/sub -->
          <partitions> <!-- multi and/or <tags> -->
            <partition>food</partition>
            <partition>spam/*</partition>
          </partitions>
          <topics>
            <topic>foo/bar/pudding</topic>
            <topic>foo/baz/test</topic>

          </topics>
        </subscribe>
      </allow_rule>
      <default>DENY</default> <!-- or >ALLOW< -->
    </grant>
```

(b) Listener Permissions

Fig. 9: Highlighted diff between two Secure DDS permission.xml files depicting degrees of overlapping capabilities.