Audit and Fuzzing of Cyclone DDS for Open Source Technology and Investment Fund

Final Report and Management Summary

2024-05-29

PUBLIC

X41 D-Sec GmbH
Krefelder Str. 123
D-52070 Aachen
Amtsgericht Aachen: HRB19989

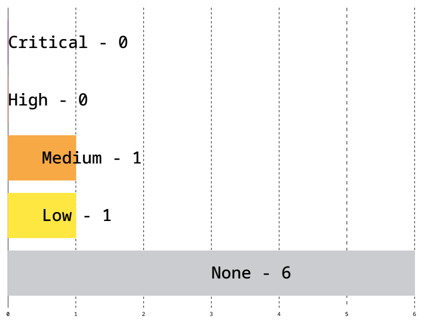
`https://x41-dsec.de/`
`info@x41-dsec.de`

Revision	*Date*	*Change*	*Author(s)*
1	2024-05-09	Final Draft Report	Robert Femmer and Markus Vervier
2	2024-05-29	Final Report and Management Summary	Robert Femmer and Markus Vervier

Contents

Dashboard

Target	
Customer	Open Source Technology and Investment Fund
Name	Cyclone DDS
Type	OMG DDS Implemetation
Version	As deployed between 2024-02-12 and 2024-03-31
Engagement	
Type	Security Audit
Consultants	2: Robert Femmer and Markus Vervier
Engagement Effort	23 person-days, 2024-02-12 to 2024-03-31
Total issues found	2



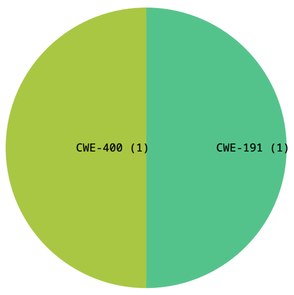


Figure 1: Issue Overview (l: Severity, r: CWE Distribution)

1 Executive Summary

In March 2024, X41 D-Sec GmbH performed a Security Audit against Cyclone DDS to identify vulnerabilities and weaknesses in the application, improve the security posture by adding fuzzers for critical attack surface and compare the implementation to the specification published by the OMG Standards Development Organization.

A total of two vulnerabilities were discovered during the test by X41. None were rated as critical, none were classified as high severity, one as medium, and one as low. Additionally, six issues without a direct security impact were identified.

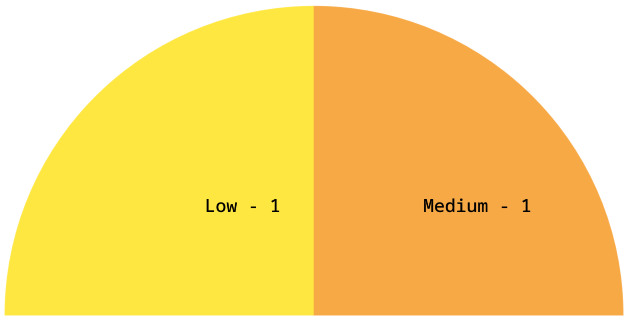


Figure 1.1: Issues and Severity

Cyclone DDS is an implementation of the Data Distribution Service specification published by the OMG Standards Development Organization. It is deployed in environments which require medium to high number of participants to communicate in an publisher-subscriber model. Applications can be found in robotics and defense. Cyclone DDS is in active development since decades and is rooted in a time, where the risks of exposing software to the Internet did not play a big role. The DDS specification does not include guidelines on how to secure an infrastructure from malicious participants and hence the recommendation is to not expose the infrastructure to possible malicious actors. However, in times of high inter-connectivity, network isolation is becoming increasingly impossible. OMG published the DDS Security Specification to address these issues by specifying security plugins, which provide (among other things) authentication, access control and cryptographic methods. The added functionality may be used to secure the publisher-subscriber communication in an open network.

This project focuses on the security plugins described by the DDS Security Specification. The efforts were divided into three approaches. Firstly the source code was audited for common security vulnerabilities. Secondly the implementation was checked for inconsistencies of the specification and the actual code. Lastly fuzzers for the main attack surfaces were contributed to the project to be run in the OSS-fuzz project.

The work was performed by two experienced security experts between 2024-02-12 and 2024-03-31.

Overall, Cyclone DDS is a mature software project and no high or critical issues were identified within the time limits of this effort. However, due to the history of the project, which was implemented under the assumption of a trusted network, Cyclone DDS does not show a high resistance against actors who merely seek to disrupt the communication by causing denial of service conditions.

2 Introduction

X41 reviewed Cyclone DDS, an implementation of the DDS specification published by OMG Standards Development Organization, and specifically the implementation of the plugin ecosystem described in the DDS Security specification. Further, fuzzers covering main attack surface to be run within the OSS-fuzz framework, were work contributed to the project.

The software is considered sensitive because of it's application in industry, robotics and defense.

Attackers could try to disrupt the communication established by the DDS network or intrude into infrastructure of the participants of the network.

2.1 Methodology

The review was based on review of the source code and the specifications published by the OMG Standards Development Organization.

X41 adheres to established standards for source code reviewing and penetration testing. These are in particular the *CERT Secure Coding*[1] standards and the *Study - A Penetration Testing Model*[2] of the German Federal Office for Information Security.

The workflow of source code reviews is shown in figure 2.1. In an initial, informal workshop regarding the design and architecture of the application a basic threat model is created. This is used to explore the source code for interesting attack surface and code paths. These are then audited manually and with the help of tools such as static analyzers and fuzzers. The identified issues are documented and can be used in a GAP analysis to highlight changes to previous audits.

[1] `https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards`
[2] `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Penetration/penetration_pdf.pdf?__blob=publicationFile&v=1`

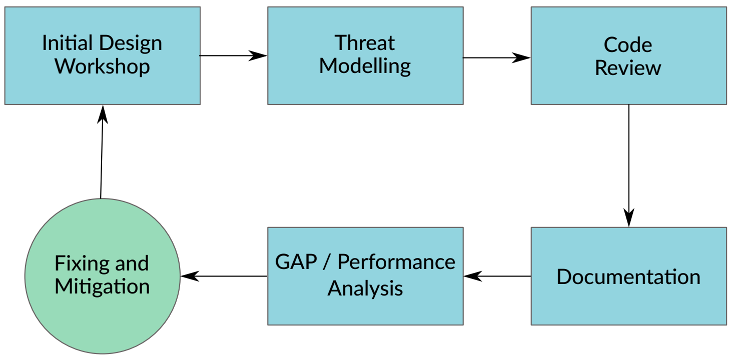


Figure 2.1: Code Review Methodology

2.2 Findings Overview

DESCRIPTION	SEVERITY	ID	REF
Integer Underflow in Security Deserializer	MEDIUM	CYCLN-CR-24-01	4.2.1
Possible Denial Of Service by Initiating Handshakes	LOW	CYCLN-CR-24-02	4.2.2
Return Values Unchecked in func validate_local_identity()	NONE	CYCLN-CR-24-100	4.3.1
Return Values Unchecked in func get_certificate_subject_name()	NONE	CYCLN-CR-24-101	4.3.2
Key Material not Erased Before Freeing	NONE	CYCLN-CR-24-102	4.3.3
Superfluous Code in func serdata_cdr_from_ser_common()	NONE	CYCLN-CR-24-103	4.3.4
Handles are not Unique over the Lifetime of the Program	NONE	CYCLN-CR-24-104	4.3.5
Note on the Squatter Attack	NONE	CYCLN-CR-24-105	4.3.6

Table 2.1: Security-Relevant Findings

2.3 Scope

The scope of this review was focused on the security plugins described in the DDS Security Specification. The plugins provide methods to handle authentication, access control and cryptography to facilitate secure and trusted communication between participants of DDS. Further, the specification describes threat models that describe security boundaries, which the correct usage of the security plugins establish.

2.4 Coverage

A security assessment attempts to find the most important or sometimes as many of the existing problems as possible, though it is practically never possible to rule out the possibility of additional weaknesses being found in the future.

The security plugins were audited for common security vulnerabilities, including memory corruption vulnerabilities, race conditions and logic errors.

The time allocated to X41 for this assessment was not sufficient to yield a good coverage of the given scope. While a best effort attempt was made to confirm that the implementation of the security plugins protect against the threats modeled in the DDS Security Specification, X41 must conclude that the complexity of the code base warrants deeper exploration of possible attack scenarios and specifically the interaction between the security plugin infrastructure and the networking layer of Cyclone DDS.

The time allocated to developing fuzzers for the main attack surface was sufficient. The three fuzzers contributed to the project cover the main attack surfaces, namely the CDR[3] deserializer and the implementation of the authentication handshake in the security plugin ecosystem.

A matrix chat group was created for direct communication between the developers and the testers.

2.5 Recommended Further Tests

X41 recommends to mitigate the issues described in this report. Afterwards, CVE[4] numbers should be requested and customers be informed (e.g. via a changelog or a special note for issues with higher severity) to ensure that they can make an informed decision about upgrading or other possible mitigations.

Further, it is recommended to audit and harden the DDS implementation further against Denial of Service and resource exhaustion attacks. X41 recommends to audit the networking stack implemented in Cyclone DDS separately.

[3] Common Data Representation
[4] Common Vulnerabilities and Exposures

3 Rating Methodology for Security Vulnerabilities

Security vulnerabilities are given a purely technical rating by the testers as they are discovered during the test. Business factors and financial risks for Open Source Technology and Investment Fund are beyond the scope of a penetration test which focuses entirely on technical factors. Yet technical results from a penetration test may be an integral part of a general risk assessment. A penetration test is based on a limited time frame and only covers vulnerabilities and security issues which have been found in the given time, there is no claim for full coverage.

In total, five different ratings exist, which are as follows:

Severity Rating
None
Low
Medium
High
Critical

A low rating indicates that the vulnerability is either very hard for an attacker to exploit due to special circumstances, or that the impact of exploitation is limited, whereas findings with a medium rating are more likely to be exploited or have a higher impact. High and critical ratings are assigned when the testers deem the prerequisites realistic or trivial and the impact significant or very significant.

Findings with the rating 'none' are called informational findings and are related to security hardening, affect functionality, or other topics that are not directly related to security. X41 recommends to mitigate these issues as well, because they often become exploitable in the future. Doing so will strengthen the security of the system and is recommended for defense in depth.

3.1 Common Weakness Enumeration

The CWE[1] is a set of software weaknesses that allows the categorization of vulnerabilities and weaknesses in software. If applicable, X41 provides the CWE-ID for each vulnerability that is discovered during a test.

CWE is a very powerful method to categorize a vulnerability and to give general descriptions and solution advice on recurring vulnerability types. CWE is developed by *MITRE*[2]. More information can be found on the CWE website at `https://cwe.mitre.org/`.

[1] Common Weakness Enumeration
[2] `https://www.mitre.org`

4 Results

This chapter describes the results of this test. The security-relevant findings are documented in Section 4.2. Additionally, findings without a direct security impact are documented in Section 4.3.

4.1 Contributions

About half of the time budget of this project was allotted to contributing fuzzers to be run in the OSS-fuzz framework. Towards this effort, three additional fuzzers were developed, targeting the topic deserialization methods, the security deserialization methods and the security handshake. During the audit these were identified as main attack surfaces.

4.1.1 Topic Deserialization

The communication between participants of the DDS is organized in so-called topics. The topic is defined by a certain format of data, which publishers use to publish data and subscribers parse to interpret the data. To exchange the data over the wire, DDS makes use of the CDR, which was also specified by the OMG Standards Development Organization. Types are defined in an domain specific IDL[1] and compiled to C source code, which provides instructions on how to serialize and deserialize an instance of the data type. To fuzz the deserializer, it is optimal to not only consider a fixed set of data types, but randomize the data type and then fuzz the deserializer routines for the randomized data type.

The contributed fuzzer does this as follows: It uses the current GIT HEAD commit hash as seed for generating a random data type. The fuzzer is then compiled using these random data types and fuzzed in OSS-fuzz [2].

[1] Interface Definition Language
[2] https://github.com/eclipse-cyclonedds/cyclonedds/pull/1940

4.1.2 Security Deserialization

A second deserializer exists for the purpose of parsing specific messages during the handshake. The most severe bug discovered in this audit was identified in the implementation of this deserializer and hence, a fuzzer targeting this code was developed. The fuzzer was able to trigger the bug as well[3].

4.1.3 Authentication Handshake

Before two participants can encrypt their communication, they perform a three-way handshake, which internally is driven by a state machine, which defines transitions in cases of errors, timeouts, etc. A structure aware fuzzer was designed to target this state machine including all the messages that are passed between two participants during the handshake[4].

As a side effect, this fuzzer targets parts of the security deserializer as well. A corpus was provided, which includes valid handshakes in both directions. The fuzzer is therefore equipped to skip any part of the cryptographic routines which would otherwise prohibit exploring some of the attack surface.

[3] https://github.com/eclipse-cyclonedds/cyclonedds/pull/1967
[4] https://github.com/eclipse-cyclonedds/cyclonedds/pull/1968

4.2 Findings

The following subsections describe findings with a direct security impact that were discovered during the test.

4.2.1 CYCLN-CR-24-01: Integer Underflow in Security Deserializer

Severity:	MEDIUM
CWE:	191 – Integer Underflow (Wrap or Wraparound)
Affected Component:	src/security/core/src/dds_security_serialize.c

4.2.1.1 Description

An integer underflow was identified in ***DDS_Security_Deserialize_OctetSeq()*** and ***DDS_Security_Deserialize_String()***. In lst. 4.1, a sequence is parsed from serialized input stream by first reading the *_length* field from the stream and then the items of the sequences according to the given length. A bounds check ensures that the length does not exceed the remaining bytes in the buffer. However, the effective size for the sequence is then padded to *sizeof(uint32_t)*, which is 4 bytes. If the length of the sequence does not end on a 4-byte boundary, it may be aligned upwards and *a_asize* may be greater than *_length* and hence greater than the remaining bytes in the buffer. In that case, the subtraction of *a_size* from *remain* will underflow. As a result, parsing may continue, even though not enough bytes were submitted to parse the required data types. As a result, confidential information may be parsed into data structures that are not supposed to contain them or the process may simply crash due to reading unmapped memory.

```c
    if (dser->remain < seq->_length) {
        return 0;
    }

    if (seq->_length > 0) {
        /* Consider padding */
        size_t a_size = alignup_size(seq->_length, sizeof(uint32_t));
        seq->_buffer = ddsrt_malloc(seq->_length);
        memcpy(seq->_buffer, dser->cursor, seq->_length);
        dser->cursor += a_size;
        dser->remain -= a_size;
```

Listing 4.1: Integer Underflow due to Padding

4.2.1.2 Solution Advice

X41 recommends to perform alignment prior to the boundary check to ensure that no underflow occurs in the following arithmetic operations. The issue was reported via GitHub Security. A patch was issued[5]. The patch removes the alignment operation completely and as a consequence, the boundary check is sufficient.

[5] https://github.com/eclipse-cyclonedds/cyclonedds/commit/44ca08e0c200ee6188373c8bc5c17edcf06c0b54

4.2.2 CYCLN-CR-24-02: Possible Denial Of Service by Initiating Handshakes

Severity:	LOW
CWE:	400 – Uncontrolled Resource Consumption ('Resource Exhaustion')
Affected Component:	src/core/ddsi/src/ddsi_handshake.c

4.2.2.1 Description

When a secure participant is discovered, a handshake is initiated. The handshake process is implemented by a state machine and a global timeout of 100s is set in effect, which cancels the handshake, if the state machine has not managed to terminate. During the handshake, messages between the participants are exchanged, which include a key exchange and signatures. However, the type of messages are not limited to the specific parameters necessary for the handshake and arbitrary data may be sent, which is then parsed and stored in memory. Further, no limits aside from the aforementioned global timeout are placed on the handshake. This allows a malicious participant to initiate numerous handshakes, send a large amount of arbitrary data, which is then kept in memory by the peer and leave the handshake in limbo until the global timeout releases the resources associated with this handshake.

The state machine implementing the handshake supports this by implementing various loops, which are entered when one participant is expecting a message from the peer, but does not receive one. Assuming the previous message got lost, it is resend and the state for waiting for a reply is entered again. There is no limit on the amount of iterations the state machine may go through in case such a loop condition is met.

4.2.2.2 Solution Advice

X41 recommends to implement various limits on the resource consumption of handshakes:

1. Limit the amount of memory that can be allocated per handshake
2. Limit the rate by which handshakes can be initiated by a client
3. Limit the amount of iterations a loop in the state machine may go through

4.3 Informational Notes

The following observations do not have a direct security impact, but are related to security hardening, affect functionality, or other topics that are not directly related to security. X41 recommends to mitigate these issues as well, because they often become exploitable in the future. Doing so will strengthen the security of the system and is recommended for defense in depth.

4.3.1 CYCLN-CR-24-100: Return Values Unchecked in *validate_local_identity()*

Affected Component:	Authentication plugin

4.3.1.1 Description

Lst. 4.2 shows code in `src/security/builtin_plugins/authentication/src/authentication.c` which is lacking checks for return values of ***X509_digest()***, which may fail. In that case, the following call to ***memcmp()*** may compare uninitialized stack values.

```
X509_digest(identityCA, digest, hash_buffer, &size);
for (unsigned i = 0; i < implementation->trustedCAList.length; ++i)
{
  X509_digest(implementation->trustedCAList.buffer[i], digest, hash_buffer_trusted, &size);
  if (memcmp(hash_buffer_trusted, hash_buffer, 20) == 0)
  {
    result = DDS_SECURITY_VALIDATION_OK;
    break;
  }
}
```

Listing 4.2: Missing Return Value Check

4.3.1.2 Solution Advice

X41 recommends to check return values for success.

4.3.2 CYCLN-CR-24-101: Return Values Unchecked in *get_certificate_subject_name()*

Affected Component:	Authentication plugin

4.3.2.1 Description

Lst. 4.3 shows code in `src/security/builtin_plugins/authentication/src/auth_utils.c` which is lacking checks for return values of ***X509_NAME_oneline()***, which may fail. In that case, the following call to ***ddsrt_strdup()*** may trigger an assertion in debug builds or crash due to a NULL pointer dereference in release builds.

```
char *subject_openssl = X509_NAME_oneline(name, NULL, 0);
char *subject = ddsrt_strdup(subject_openssl);
OPENSSL_free(subject_openssl);
return subject;
```

Listing 4.3: Missing Return Value Check

4.3.2.2 Solution Advice

X41 recommends to check return values for success.

4.3.3 CYCLN-CR-24-102: Key Material not Erased Before Freeing

Affected Component:	Cryptography plugin

4.3.3.1 Description

Lst. 4.4 shows code in `src/security/builtin_plugins/cryptographic/src/crypto\_objects.c`, which does not zero the key material before returning the memory to the allocator. While the code does zero the structure holding pointers to the key material, it does not zero the key material itself.

```
static void master_key_material__free(CryptoObject *obj)
{
  master_key_material *keymat = (master_key_material *)obj;
  if (obj)
  {
    CHECK_CRYPTO_OBJECT_KIND(obj, CRYPTO_OBJECT_KIND_KEY_MATERIAL);
    if (CRYPTO_TRANSFORM_HAS_KEYS(keymat->transformation_kind))
    {
      ddsrt_free (keymat->master_salt);
      ddsrt_free (keymat->master_sender_key);
      ddsrt_free (keymat->master_receiver_specific_key);
    }
    crypto_object_deinit ((CryptoObject *)keymat);
    memset (keymat, 0, sizeof (*keymat));
    ddsrt_free (keymat);
  }
}
```

Listing 4.4: Key Material is freed, but not zeroed

4.3.3.2 Solution Advice

X41 recommends to zero the key material before returning memory to the allocator.

4.3.4 CYCLN-CR-24-103: Superfluous Code in *serdata_cdr_from_ser_common()*

Affected Component:	DDS

4.3.4.1 Description

Lst. 4.5 shows code in `src/core/ddsi/src/ddsi_serdata_cdr.c`, which is executed before returning from the function. The code initializes a structure on the stack, which is unused and goes out of scope immediately.

```
dds_istream_t is;
dds_istream_init (&is, actual_size, d->data, DDSI_RTPS_CDR_ENC_VERSION_2);
return d;
```

Listing 4.5: Superfluous Code

4.3.4.2 Solution Advice

X41 recommends to remove the superfluous code.

4.3.5 CYCLN-CR-24-104: Handles are not Unique over the Lifetime of the Program

Affected Component:	Security plugins

4.3.5.1 Description

The DDS Security Specification requires the security plugin to be portable between different DDS implementations. Therefore the plugins are compiled to dynamically loadable shared objects with a specific API. The API uses opaque handles to identify resources, which are managed by the plugin. For example, these handles may refer to a specific identity belonging to a participant or key material. The objects that these handles refer to have a certain lifetime during which is tracked by reference counting. When all handles to an object are returned via API calls, the associated object is released.

It was found that the opaque handles are in fact the memory addresses of the objects where they are allocated on the heap. This means that they are not unique over the lifetime of the program, as an object may be released and a new object of the same type may be allocated at the same memory location. Consequently, a handle that was not forgotten by the API client becomes a valid handle again, but does not refer to the same object anymore that it used to. If this handle was (due to some bug) to be returned to the API again, use-after-free conditions may arise.

While the audit did not turn up instances of this bug class, it must be noted that using the memory address as opaque handle shifts the burden of lifetime management of objects owned by the security plugins to the user of that plugin.

4.3.5.2 Solution Advice

X41 recommends to use handles which can be expected to be unique over the lifetime of the program, e.g. an atomic counter which is increased on every allocation. While this counter may overflow at some point, the risks of a critical bug resulting in an exploitable condition are reduced.

4.3.6 CYCLN-CR-24-105: Note on the Squatter Attack

Affected Component:	DDS Security Specification

4.3.6.1 Description

The DDS Security Specification describes the Squatter attack, in which a malicious participant forges their own participant GUID[6], "whereby a participant with a valid certificate could announce itself into the DDS Domain with the GUID of some other participant. Once authenticated the squatter participant would preclude the real participant from being discovered, because its GUID would be detected as a duplicate of the already existing one."

In order to prevent this attack, the GUID is to be generated from the subject name of the certificate signed by the certificate authority. The specification describes that 47 bits taken from the SHA256 hash of the subject name (plus some transformations) are to be used as parts of the GUID. It must be noted that in the case where the participant may choose their subject name for the certificate to be signed by the certificate authority, they may be able to choose a subject name that results in a colliding GUID to still perform the squatter attack.

4.3.6.2 Solution Advice

Since the issue arises solely from the specification, there can be no recommendation to change the implementation. X41 recommends to document this issue, so that users are deterred from offering possibly malicious participants to choose their own subject names for certificates.

[6] Globally Unique Identifier

5 About X41 D-Sec GmbH

X41 D-Sec GmbH is an expert provider for application security and penetration testing services. Having extensive industry experience and expertise in the area of information security, a strong core security team of world-class security experts enables X41 D-Sec GmbH to perform premium security services.

X41 has the following references that show their experience in the field:

- Source code audit of the Git source code version control system[1]
- Review of the Mozilla Firefox updater[2]
- X41 Browser Security White Paper[3]
- Review of Cryptographic Protocols (Wire)[4]
- Identification of flaws in Fax Machines[5,6]
- Smartcard Stack Fuzzing[7]

The testers at X41 have extensive experience with penetration testing and red teaming exercises in complex environments. This includes enterprise environments with thousands of users and vendor infrastructures such as the Mozilla Firefox Updater (Balrog).

Fields of expertise in the area of application security encompass security-centered code reviews, binary reverse-engineering and vulnerability-discovery. Custom research and IT security consulting, as well as support services, are the core competencies of X41. The team has a strong technical background and performs security reviews of complex and high-profile applications such as Google Chrome and Microsoft Edge web browsers.

X41 D-Sec GmbH can be reached via `https://x41-dsec.de` or `mailto:info@x41-dsec.de`.

[1] `https://x41-dsec.de/security/research/news/2023/01/17/git-security-audit-ostif/`
[2] `https://blog.mozilla.org/security/2018/10/09/trusting-the-delivery-of-firefox-updates/`
[3] `https://browser-security.x41-dsec.de/X41-Browser-Security-White-Paper.pdf`
[4] `https://www.x41-dsec.de/reports/Kudelski-X41-Wire-Report-phase1-20170208.pdf`
[5] `https://www.x41-dsec.de/lab/blog/fax/`
[6] `https://2018.zeronights.ru/en/reports/zero-fax-given/`
[7] `https://www.x41-dsec.de/lab/blog/smartcards/`

Acronyms

CDR	Common Data Representation	8
IDL	Interface Definition Language	11
CVE	Common Vulnerabilities and Exposures	8
CWE	Common Weakness Enumeration	10
GUID	Globally Unique Identifier	21