Security analysis of ADRF analytics record lifecycle

This report in another language: 中文.

19.7.0 · ProVerif+ Agent report

Documents	5G System; Analytics Data Repository Services; Stage 3 (protocol definition)
Properties examined	11: 6 hold, 5 fail (8 attacks)
Findings	1

Contents

1 Protocol and scope
2 Findings
3 Properties examined
4 Limits of this analysis
Appendix A: Reading of each property
Appendix C: Claims and prover answers
Appendix D: How the analysis was run

1 Protocol and scope

The Nadrf_DataManagement API of the ADRF lets NF service consumers store analytics data, retrieve it and remove it. Three kinds of party take part: the NF that stores a record, the NF that reads it, and the ADRF that keeps it, with the NRF as the authorization server that issues every NF its OAuth2 access token for the API scope.

An exchange is protected end to end: every request is authenticated by the NF's own credential and carries its access token, and the ADRF answers on the requesting NF's own connection. The storing NF learns the storeTransId of the record it created; the reader names the record by its data set identifier or by that identifier; the removal names it by storeTransId or by a data specification.

The parties:

- **NF service consumer (storing)**: stores the analytics and later removes it. Holds: its TLS credential and its access token.
- **NF service consumer (reader)**: approved to read the record. Holds: its TLS credential and its access token.
- **ADRF**: keeps the records of the data store. Holds: record store with storeTransId and dataSetId.
- **NRF**: authorization server. Holds: NF profiles, client credentials.

Figure 1 shows the exchange as the documents specify it.

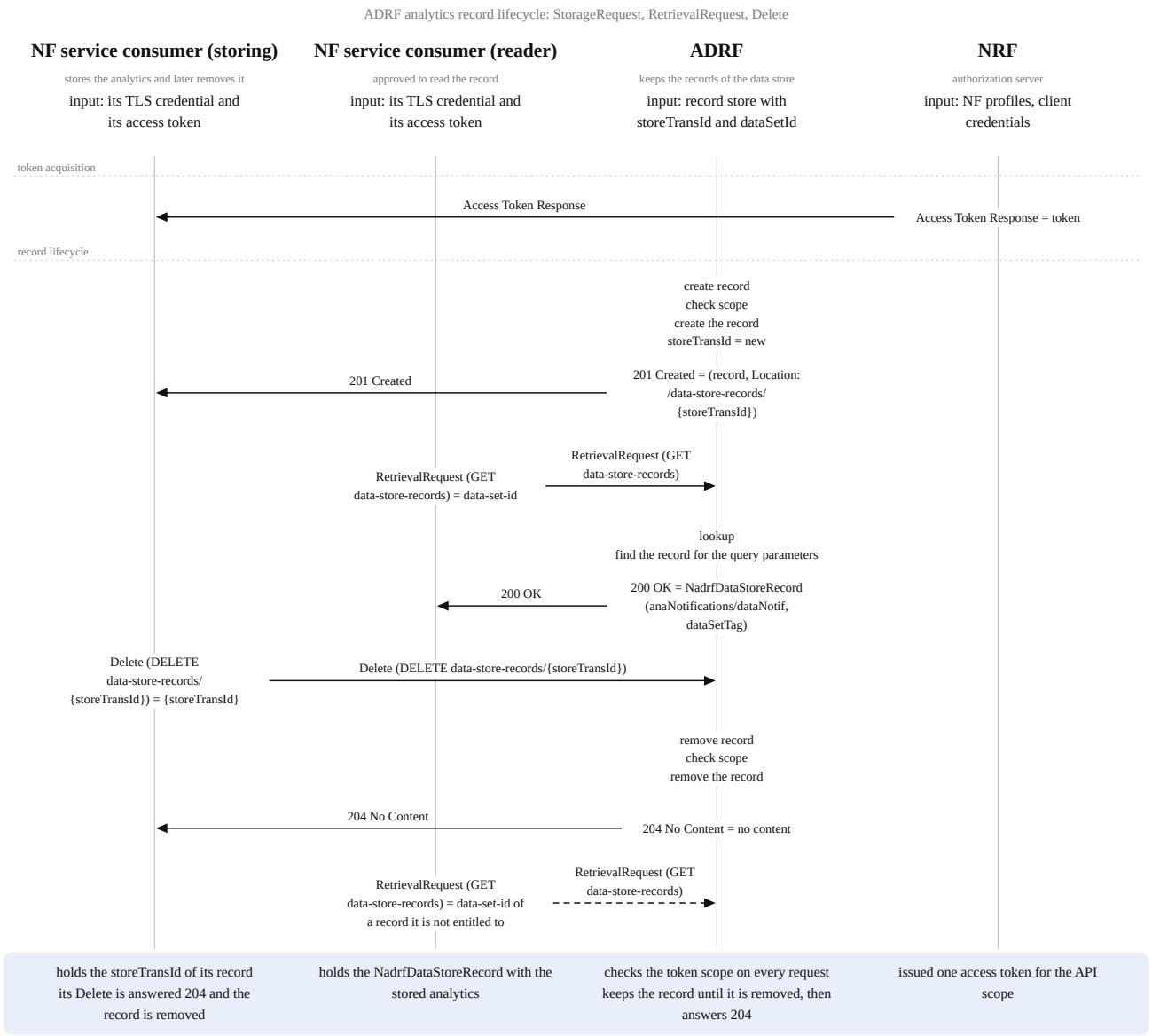


Figure 1. The protocol as specified.

The scope of the analysis is in Appendix D: what was analysed, what was left out, and the sections of the documents it is built from.

2 Findings

The analysis has 1 finding: what follows is not stated in the documents. For 1 of them it is unclear whether the documents state it.

The token gate and the wire protection hold: only an NF with its own credential and its own token is served or obeyed, and a record a consumer accepts is always one the ADRF stored. The protocol does not protect one NF's records from another legitimate NF, and that is the finding: with the single API scope and a data set identifier, a malicious NF reads another NF's analytics and removes its record. Eleven targets were answered, three of them on the model that unrolls the record state, and one claim of the authorization target (that the token is unexpired) cannot be asked of the prover. A checked change of the protocol - the ADRF acting on a record only for the NF that stored it

or one it approved - removes the three attacks of another NF while the normal run survives. The answers rest on TLS and OAuth2 being abstracted as per-endpoint keys and unforgeable tokens, and on the record lifecycle being unrolled over five steps.

The attack figures use the columns of Figure 1. Red is what the attacker does. Yellow marks a message that a party sends although the specified exchange does not have it at that point, and the outcome that the attack breaks.

2.1 A legitimate NF can read and erase another NF's analytics records

Kind	Weakness of the protocol as specified
New	Unclear: the documents state it in part, or could not be searched
Value	high
Properties	Z05, K02, Z03, U01 (section 3)
Source text	s0034, s0069, s0373
Shown by	ProVerif attack queries `z05_read`, `z05_rmspec`, `z05_del`; Figures 2, 3, 4 and 5

What happens. The scope of clause 5.1.9 covers storage, retrieval, delete and the subscription operations in one token, and the ADRF checks that scope only. A RetrievalRequest may name the record by its data set identifier and remove-stored-data-analytics removes every record matching a data specification, so a legitimate NF that names another NF's data set is served that record and can erase it. The model shows both: the malicious NF receives the storing NF's payload and removes the record, and the same holds through the Delete by storeTransId once it has learned the identifier.

What the attacker needs. One NF credential that the operator considers legitimate, the access token the NRF issues for the API scope nadrf-datamanagement:storage-read-delete-subs, and either the data set identifier of the record or its storeTransId.

What the attacker gains. The malicious NF reads other NFs' analytics and removes records another NF stored, which denies the storing NF its data and the deletion notification it asked for. No key of another NF is needed and no TLS session is broken.

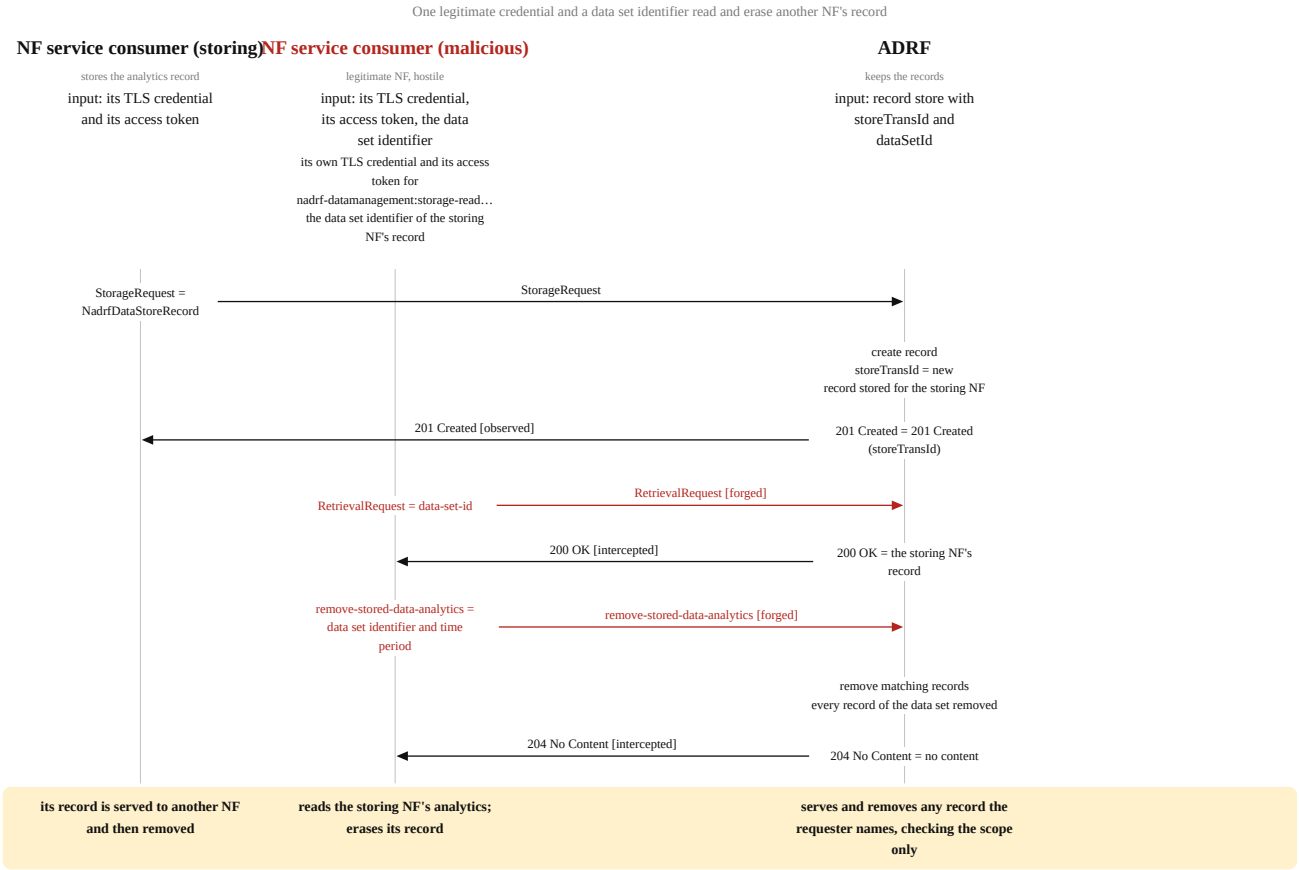


Figure 2. One legitimate credential and a data set identifier read and erase another NF's record. Property Z05; attack queries z05_read, z05_rmspec.

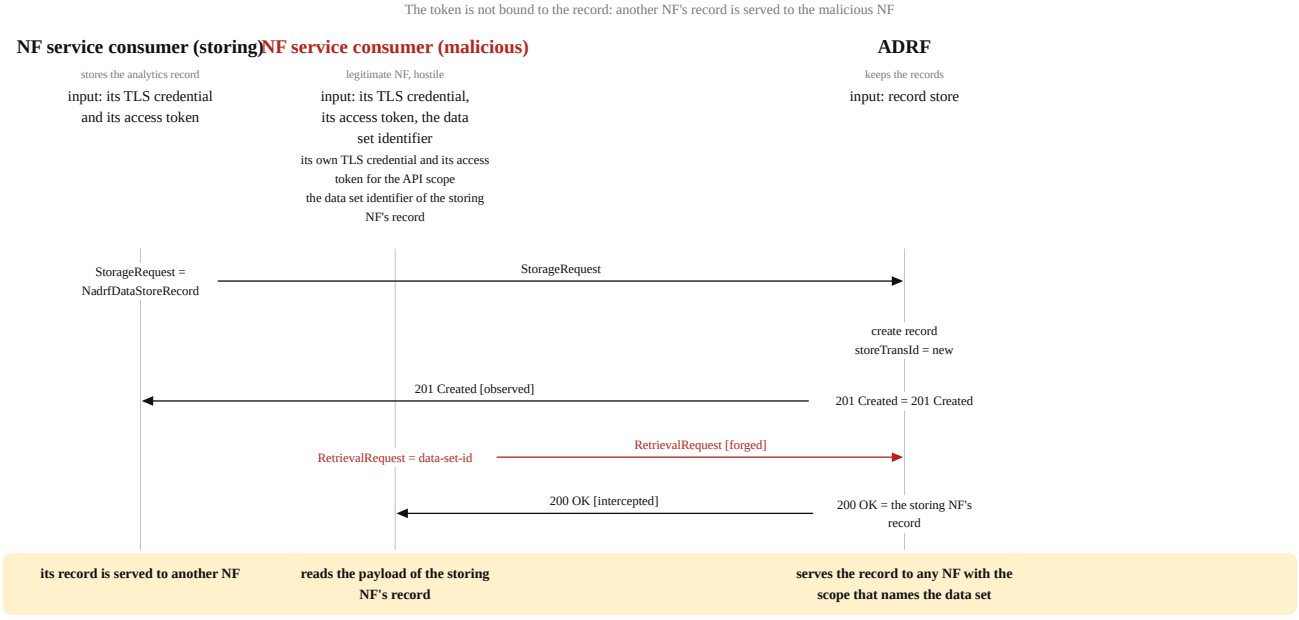


Figure 3. The token is not bound to the record: another NF's record is served to the malicious NF. Property Z03; attack query z05_read.

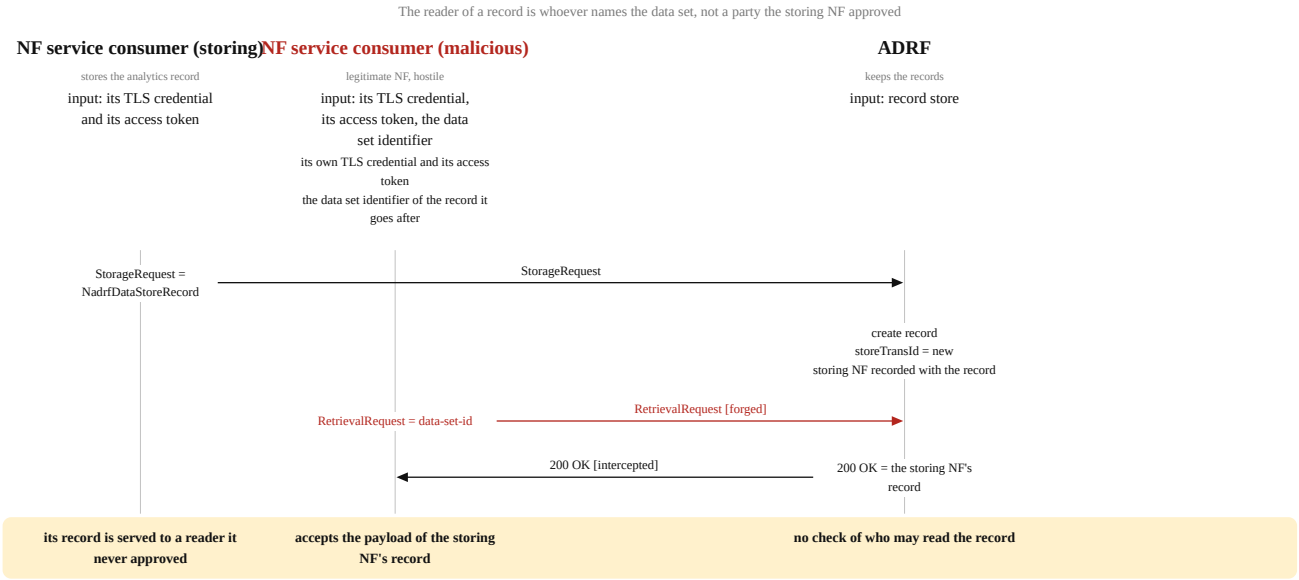


Figure 4. The reader of a record is whoever names the data set, not a party the storing NF approved. Property U01; attack query z05_read.

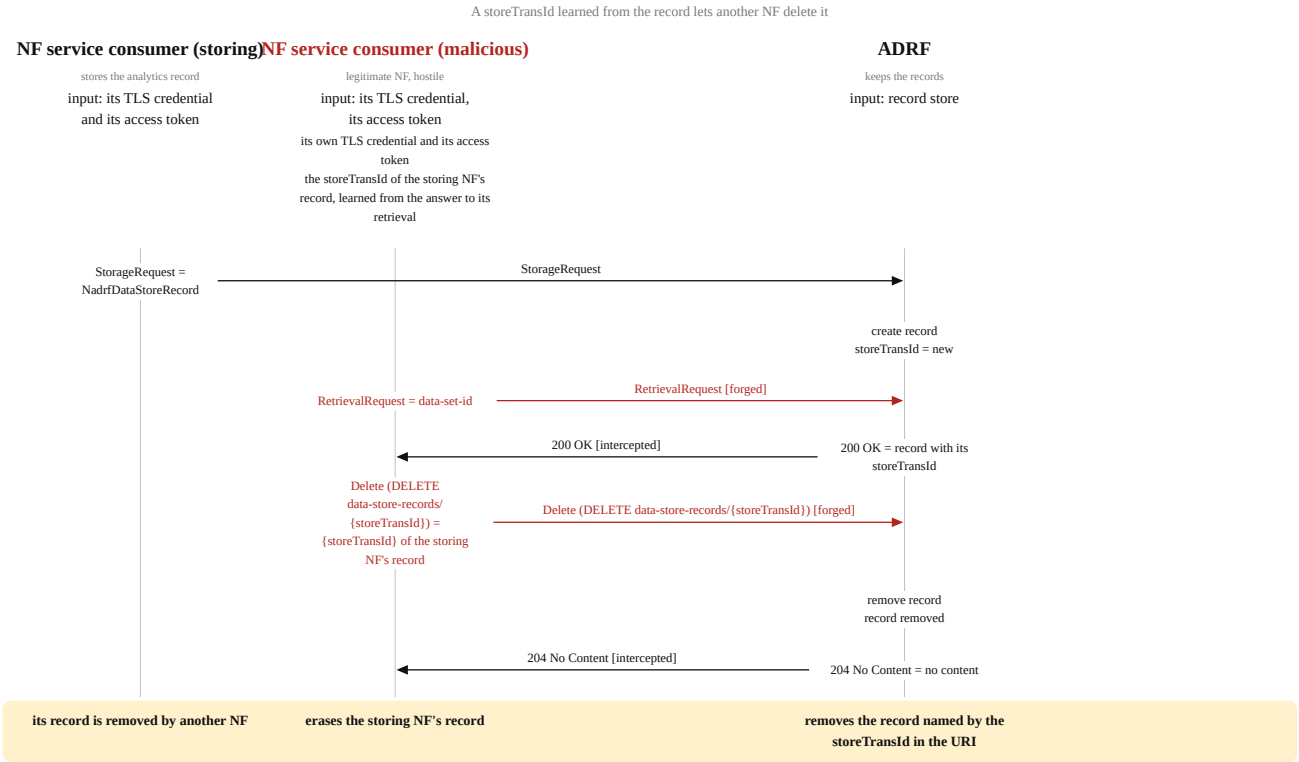


Figure 5. A storeTransId learned from the record lets another NF delete it. Property Z05; attack query z05_del.

Proposed change. Keep the identity of the storing NF with the record and act on the record only for that NF, or for an NF the storing NF named in the storage request: a retrieval is served only to them, and remove-stored-data-analytics removes only records the requesting NF stored, answering an error otherwise. The change was checked: on the modified model the three attacks of another NF are gone, the normal run is still found and the payload attack disappears with them. Splitting the single scope into storage, read and delete scopes is a second change that the model does not check.

Check of the change with ProVerif. The change was put into a copy of the model: The ADRF keeps the identity of the NF that stored the record (it already knows it, clause 4.2.2.2.2) and, for a retrieval, serves only the storing NF or an NF that storing NF named as allowed to read; for a Delete by storeTransId and for remove-stored-data-analytics it removes only records the requesting NF stored and answers an error otherwise. On the changed model ProVerif proves `z05_read`, `z05_rmspec`, `z05_del`: the attack is no longer possible there. The normal run of the protocol is still found. Of the other recorded answers, 13 still hold and 1 changed: `k02_nf` (K02, was attack) is now verified. This is a first check on the model: it shows that the change removes this attack there, not that it introduces no other problem. Changed model: `records/fixes/0047-fix-ownership.pv`.

No figure was given for these attacks of a finding: K02: z05_del, K02: z05_read, K02: z05_rmspec, U01: z05_del, U01: z05_rmspec, Z03: z05_del, Z03: z05_rmspec.

3 Properties examined

Each row is one property put to ProVerif. The result is ProVerif's answer on the analyst's model of the protocol; value and whether the documents already state the result are the analyst's judgement, with the reasons in Appendix A.

ID	Property	Compromise assumed	Result	Value	In the documents
Z01	**Token-gated retrieval and delete.** Whenever the ADRF accepts and executes a RetrievalRequest or a Delete (including deletion by specification), the requester must hold an unexpired permission covering that operation, issued to that requester by the authorization server.	no compromise	Holds	medium	yes
Z03	**Token bound to subject and record.** When the ADRF executes a read or a delete, the subject, audience, resource (storeTransId) and operation type of the permission used must match this request.	no compromise	Fails (2 attacks)	high	unclear
K02	**Payload hidden from other NFs.** The record payload in a read response is visible only to the NF that stored the record and to NFs approved to read it; a malicious legitimate NF and an external attacker cannot derive that payload.	no compromise	Fails (1 attack)	high	unclear
D01	**Served record matches stored record.** The data accepted by each read corresponds to the record the ADRF actually stores and that the request parameters locate, with payload and context consistent.	no compromise	Holds	medium	yes
S05	**No retrieval of a deleted record.** After the deletion takes effect, a read of a deleted record must not return successful content (it should return 204 or an error), and the old context cannot regain the original data.	no compromise	Holds within a bound (the changing value unrolled to 5 steps)	medium	yes
R02	**Delete accepted at most once.** The same logical delete message is accepted by the ADRF at most once, and a replay does not change the state of other records.	no compromise	Holds within a bound (the changing value unrolled to 5 steps)	medium	unclear
Z05	**Other NFs' records stay protected.** When a malicious legitimate NF controls its own domain, it cannot read or delete records stored by other NFs, and the confidentiality and authorization properties of the target domain still hold.	no compromise	Fails (3 attacks)	high	unclear
U01	**Record and reader both authentic.** The record returned by a read was genuinely stored by the party claiming to have stored it, and the requester is an approved reader, with storeTransId and content consistent.	no compromise	Fails (1 attack)	high	unclear

ID	Property	Compromise assumed	Result	Value	In the documents
U02	**Read and delete permissions separate.** Reading and deleting each require a permission covering that operation, and neither can substitute for the other.	no compromise	Attack holds (stated in the documents)	medium	yes
U03	**No stale success after deletion.** After a deletion takes effect or an error occurs, the old record must not be accepted as a fresh successful result.	no compromise	Holds within a bound (the changing value unrolled to 5 steps)	medium	yes
X01	**Token and credential confidentiality.** An NF's TLS credential and the access token the NRF issues to it stay known to that NF, the NRF and the ADRF only; neither the network attacker nor another legitimate NF can obtain or forge a token for another NF's identity, so it cannot act towards the ADRF as that NF.	no compromise	Holds	medium	yes

"Attack holds (stated in the documents)": the prover found the attack, and the documents themselves already state it. The property fails; this is a known result, not a finding (Appendix B).

Goals the documents state that the drafted list of properties did not have, added by the analyst (Appendix D): X01.

The questions asked, and the analyst's answers:

Question	Answer	Rests on
Q1: dataStoreId/storer/allowed consumers bound to the data	The storeTransId and the payload a reader accepts are always the ones the ADRF stored, and the NF that stored the record is the one the ADRF reports; but nothing binds the record to the NF that is allowed to read it, so the reader is whoever names the data set, not a party the storing NF approved.	D01, U01, K02
Q2: read permission and delete permission each rest on a specification condition	The API defines one scope for storage, retrieval and delete, so reading and removing are not separately authorized: a token that lets an NF read also lets it delete, and the model shows a single token accepted for both.	U02, Z01, Z03
Q3: after an error or deletion the data cannot be accepted as a fresh successful result	After the ADRF removes the record it answers 204 to the next retrieval and the payload is not returned again, and a repeated delete is refused rather than executed twice; what the model does not cover is an old answer of the ADRF being replayed later inside a new connection.	S05, R02, U03

4 Limits of this analysis

A result here is a result on a model of the protocol: it says what the specified messages and checks give or fail to give, with cryptography taken as ideal, and nothing about an implementation.

What this analysis does not cover, as the analyst states it:

- TLS and OAuth2 are abstracted: the transport is a per-endpoint symmetric key and the token an unforgeable value the NRF derives from the NF identity, so nothing is said about the strength of TLS, of the token format or of any algorithm.
- Token expiry is not modelled: ProVerif has no time, so the target's requirement that the permission be unexpired is not answered, and a token never goes stale in the model.
- The identity of the ADRF's own record state is modelled in bounded.pv only over five steps of one record; longer lifecycles, records expiring on their lifetime or deletion notifications to the storing NF are outside the analysis.
- model.pv keeps records in a table that only grows, so it cannot express removal; the answers about what follows a removal come from bounded.pv and hold for the five steps unrolled there.

- A storage consumer, a reader and one malicious NF are modelled; a compromised ADRF or NRF, a dishonest SCP or a roaming partner are outside the analysis.
- Replay of a stored response inside a TLS connection is not modelled: responses are bound to a per-request nonce under the endpoint key, and what a consumer does with a late answer is only covered by the unrolled model.
- The documents' clause 4.2.2.9.3 delete by data specification is modelled by matching on the data set identifier only; the time period and the data or analytics specification are not represented.
- The documents themselves state that a token does not bind to a record (RFC 6749 section 10.3) and give a single scope for the API (clause 5.1.9); the finding goes beyond those statements by showing what an NF with the scope can do.

What the host records:

- The answers of S05, R02, U03 rest on an operation the model simplifies (Appendix D).

What stays open for each property, and what would settle it, is in Appendix A.

Appendix A: Reading of each property

The analyst's own reading of the prover's answers, not an answer of the prover: what the answers mean for the protocol, what the reading rests on, whether it can be used as it stands, and what it is worth.

Z01: Token-gated retrieval and delete

Whenever the ADRF accepts and executes a RetrievalRequest or a Delete (including deletion by specification), the requester must hold an unexpired permission covering that operation, issued to that requester by the authorization server.

Only an NF that authenticates with its own credential and presents the token the NRF issued to it is served or obeyed, for retrieval and for both forms of deletion. A malicious NF is served, but only under its own identity and its own token; nothing it can do makes the ADRF act for another NF. Whether the token is still valid is outside what the model can say.

- Result: Holds; answers what is asked: partly; usable with caveats
- Value medium: It confirms the OAuth2 gate the specification requires, with the exact condition under which the ADRF acts, but says nothing about expiry, which the target also asks.
- Stated in the documents: yes. Where the documents say it: s0368, s0369, s0373
- Rests on: The token is an unforgeable value the NRF derives from the NF identity, and the request is authenticated by the NF's own endpoint key; token format, audience checking and expiry are not modelled.
- Open: Whether the token is unexpired at the time of the request: the model has no time, so the expiry claim is not answered.

Z03: Token bound to subject and record

When the ADRF executes a read or a delete, the subject, audience, resource (storeTransId) and operation type of the permission used must match this request.

The token is bound to the NF that presents it: an NF can only act under its own identity, and the NRF issues each token to one NF. It is bound neither to the record the request names nor to the operation: any NF with the scope can name another NF's record and act on it, and one token is taken for reading and for deleting alike.

- Result: Fails (2 attacks); answers what is asked: yes; usable with caveats
- Value high: The subject binding is proved, and the failure of binding to record and operation is the mechanism of the finding; it is the part of the target the specification never states.
- Stated in the documents: unclear. Source text it rests on: rfc6749:2879, s0034, s0373
- Rests on: The model gives the ADRF no per-record entitlement because the specification defines none; if a deployment adds one outside the API, the answer changes.; The malicious NF knows the data set identifier of the record; the specification treats it as a descriptive identifier, not a secret.
- Open: Whether a deployment expects the ADRF to bind a token to a record without the specification saying so.

K02: Payload hidden from other NFs

The record payload in a read response is visible only to the NF that stored the record and to NFs approved to read it; a malicious legitimate NF and an external attacker cannot derive that payload.

The stored analytics are kept from the network attacker, which holds no NF credential and cannot read the exchange. They are not kept from another legitimate NF: an NF that names the data set identifier is served the payload and reads it.

- Result: Fails (1 attack); answers what is asked: yes; usable
- Value high: It separates what the protocol does protect (the wire) from what it does not (one legitimate NF against another), which is the finding.
- Stated in the documents: unclear. Source text it rests on: s0037, s0373
- Rests on: The payload is a private value in the model and the exchange is protected by the endpoint key, so the network attacker can only get it through a legitimate credential.; The successful retrieval path is modelled; a reader's handling of a 204 answer is only covered in the unrolled model.
- Open: Whether a data set identifier is secret in a deployment; the model assumes the malicious NF knows it.

D01: Served record matches stored record

Every record a read accepts corresponds to a record the ADRF really stored, located by the request parameters, with payload and context consistent.

What a consumer accepts is always a record the ADRF really stored, with the same storeTransId, the same storing NF and the same payload. A malicious NF can store a record under a data set identifier another NF uses, so a reader asking for that data set may be given the malicious NF's record - the correspondence holds, but not with the NF the reader expected.

- Result: Holds; answers what is asked: yes; usable
- Value medium: A clean correspondence between what is stored and what is served; its control shows the limit, that the storer need not be the NF the reader expected.
- Stated in the documents: yes. Where the documents say it: s0019, s0036, s0037
- Rests on: The record is returned whole to the reader, including the storing NF identity, which the model puts in the record.; The model's table only grows, so nothing is said here about a record after removal.

S05: No retrieval of a deleted record

After the point at which removal takes effect, a read of a deleted record must not return successful content (it should return 204 or an error), and an old context must not regain the original data.

Once the ADRF has removed the record, a later retrieval for it is answered 204 and no payload is returned, to the reader or to anyone else. The answer depends on the ADRF replacing the state it keeps when it removes the record and testing it before it answers.

- Result: Holds within a bound (the changing value unrolled to 5 steps); answers what is asked: yes; usable with caveats
- Value medium: It shows the removal really takes effect in the API's answers, with a control that the after-removal retrieval is reached; the caveat is the bound of five steps.
- Stated in the documents: yes. Where the documents say it: s0045, s0054, s0055
- Rests on: The lifecycle is unrolled over five steps of one record; longer lifecycles and removal by expiry are outside it.

R02: Delete accepted at most once

The same logical delete message is accepted by the ADRF at most once, and a replay does not change the state of other records.

A delete message that arrives twice removes the record once; the second one is refused because the ADRF's state has moved to removed. A rejected delete leaves the state as it was, so a failure does not make the record look deleted.

- Result: Holds within a bound (the changing value unrolled to 5 steps); answers what is asked: yes; usable with caveats
- Value medium: The specification does not say what a repeated delete does; the model shows the reading in which it is harmless, with a control that the refusal is reached.
- Stated in the documents: unclear. Source text it rests on: ts-33-501:2875, s0053, s0054, s0055, s0057
- Rests on: Five unrolled steps of one record; the error answer is modelled as a distinct reply rather than with the HTTP status codes of clause 5.1.7.
- Open: What the specification intends for a repeated delete of a record that no longer exists; nothing in the selected text says whether an error, 204 or a re-creation is expected.

Z05: Other NFs' records stay protected

When a malicious legitimate NF controls its own domain, it cannot read or delete records stored by other NFs, and the confidentiality and authorization properties of the target domain still hold.

Against the network alone the property holds: without another NF's credential no record is ever served or removed for it. Against a legitimate NF it fails: an NF with the API scope reads another NF's record by naming its data set, and removes it either by data specification or, once it has the identifier, by storeTransId.

- Result: Fails (3 attacks); answers what is asked: yes; usable
- Value high: This is the finding: a weakness of the protocol as specified that the documents do not state, shown with the exact attacker and the exact requests.
- Stated in the documents: unclear. Source text it rests on: s0034, s0063, s0069, s0373
- Rests on: The malicious NF's knowledge includes the data set identifier and, after a first removal attempt, the storeTransId it learns from the answer.; The ADRF is assumed to enforce no per-record entitlement, because the API defines none.
- Open: Whether the specification expects the ADRF to authorize a record at all; nothing in the selected text says so.

U01: Record and reader both authentic

The record returned by a read was really stored by the claiming storer and the requester is an approved reader, with storeTransId and content consistent.

The record a reader accepts is authentic and complete, but the reader need not be a party the storing NF approved: the ADRF serves any NF that names the data set. Integrity of the answer holds; the approval of the reader does not exist in the API.

- Result: Fails (1 attack); answers what is asked: yes; usable
- Value high: It states exactly which half of the question holds and which does not, which is what an editor needs.
- Stated in the documents: unclear. Source text it rests on: s0008, s0034, s0373

- Rests on: The storage request carries no field naming the NFs that may read the record, so the model has nothing to check.; The successful retrieval path only; a 204 answer to a reader is covered in the unrolled model.
- Open: Whether a deployment adds its own approval of readers outside the API.

U02: Read and delete permissions separate

Reading and deleting each require a permission covering that operation, and neither can substitute for the other.

Reading and deleting are not separately authorized: the table of clause 5.1.9 gives one scope, nadrf-datamanagement:storage-read-delete-subs, for retrieval, delete and the subscription operations, so a token obtained to read is accepted to delete and the other way round. The tool shows one token used for both operations; the network attacker cannot use the difference because it holds no token at all.

- Result: Attack holds (stated in the documents); answers what is asked: yes; usable
- Value medium: The specification states the single scope in its scope table, so the loss of separation is a documented property of the design rather than a new finding; the model makes its consequence explicit.
- Stated in the documents: yes. Where the documents say it: ts-33-501:2875, s0373, s0028, s0048
- Rests on: The scope is modelled as the single value the NRF derives per identity; since the API defines no per-operation scope there is nothing finer for the ADRF to check.

U03: No stale success after deletion

After a removal takes effect or fails, the old record must not be accepted as a fresh successful result.

After the removal the reader's next retrieval is answered 204 and the old payload does not come back as a fresh result; the answer is bound to the reader's own request by a fresh value the reader chose. The model does not say what a consumer does with an old answer that is delivered again later.

- Result: Holds within a bound (the changing value unrolled to 5 steps); answers what is asked: yes; usable with caveats
- Value medium: It shows the API's answer after removal is not a stale success, with a control that the refusal is reached; the caveat is the bound and the replay abstraction.
- Stated in the documents: yes. Where the documents say it: s0045, s0054, s0056
- Rests on: Each answer is bound to a per-request value under the reader's key; a replay inside a TLS connection is outside the model.; Five unrolled steps of one record.
- Open: Replay of an old answer to a later request inside a fresh connection is not covered.

X01: Token and credential confidentiality

An NF's TLS credential and the access token the NRF issues to it stay known to that NF, the NRF and the ADRF only; neither the network attacker nor another legitimate NF can obtain or forge a token for another NF's identity, so it cannot act towards the ADRF as that NF.

An NF's credential and the token the NRF issues to it stay with that NF: neither the network attacker nor another legitimate NF obtains them, so neither can act towards the ADRF under that identity. The control shows the attacker does obtain its own token and credential, so the query is not empty.

- Result: Holds; answers what is asked: yes; usable
- Value medium: It confirms the confidentiality RFC 6749 requires for tokens, in the exact setting where one NF is hostile.

- Stated in the documents: yes. Where the documents say it: rfc6749:2859, s0369, s0373
- Rests on: Token and credential are modelled as private derivations of the NF identity and are only ever sent under that NF's key.

Appendix C: Claims and prover answers

The prover's record for each target: the level reached, the query that reached it, whether a control was checked, and the attacks with the analyst's verdict on each.

target	level	bound	lemma	control	attacks	claims
Z01	proved		z01_get	checked		4 proved
Z03	proved		z01_get	checked	z05_read (holds_in_source); u02 (holds_in_source)	2 proved, 2 attack
K02	proved		k02_net	checked	k02_nf (holds_in_source)	1 proved, 1 attack
D01	proved		d01	checked		2 proved
S05	bounded	the changing value unrolled to 5 steps	b_s05	checked		2 bounded, 1 reachable
R02	bounded	the changing value unrolled to 5 steps	b_r02	checked		2 bounded, 1 reachable
Z05	proved		z05_net_get	checked	z05_read (holds_in_source); z05_rmspec (holds_in_source); z05_del (holds_in_source)	1 proved, 3 attack
U01	proved		d01	checked	z05_read (holds_in_source)	1 proved, 1 attack
U02	proved		x01_tok	checked	u02 (holds_in_source)	1 proved, 1 attack
U03	bounded	the changing value unrolled to 5 steps	b_u03	checked		2 bounded, 1 reachable
X01	proved		x01_tok	checked		2 proved

Claim by claim, in the analyst's wording of each claim; the answer to each is the prover's:

target	claim	attacker	answer	query	control
Z01	The ADRF answers a RetrievalRequest only for a requester that presented an access token issued to that same NF by the NRF; the network attacker holds no NF credential and is never served.	`network`	proved	z01_get	checked
Z01	The ADRF answers a RetrievalRequest only for a requester that presented a token issued to it by the NRF; the malicious NF is served, but only with the token the NRF issued to itself.	`malicious_nf`	proved	z01_get	checked
Z01	Delete by storeTransId is gated by the caller's own token; the malicious NF has its own token only.	`malicious_nf`	proved	z01_del	checked
Z01	Delete by data specification is gated by the caller's own token; the malicious NF has its own token only.	`malicious_nf`	proved	z01_rmspec	checked
Z03	The token presented is the caller's own: no request is served unless it carries the token the NRF issued to the identity it claims.	`network`	proved	z01_get	checked
Z03	Subject binding holds for the malicious NF too: it can only act under its own identity and its own token.	`malicious_nf`	proved	z01_get	checked

target	claim	attacker	answer	query	control
Z03	The token is bound to the record it names: a caller may act only on a record it is entitled to. The malicious NF has its own token and the data set identifier.	`malicious_nf`	attack (holds_in_source)	z05_read (the control of z05_net_get)	
Z03	The token is bound to the operation it was issued for. The malicious NF has a single token for the API scope.	`malicious_nf`	attack (holds_in_source)	u02	
K02	The analytics payload stored by the storing NF reaches the network attacker only if a legitimate NF's credential is in its hands.	`network`	proved	k02_net	checked
K02	The payload of another NF's record is not kept from a legitimate NF: a malicious NF that names the data set identifier is served it and reads it.	`malicious_nf`	attack (holds_in_source)	k02_nf (the control of k02_net)	
D01	Every record a consumer accepts was really stored at the ADRF, with the same storeTransId, the same storing NF and the same payload.	`network`	proved	d01	checked
D01	Same correspondence holds while a malicious NF is active: what a consumer accepts is a record the ADRF stored, though not necessarily one the storing NF it expected created (the control shows that).	`malicious_nf`	proved	d01	checked
S05	A retrieval that follows the removal is answered 204 - the situation the target speaks of is reached.	`network`	reachable	b_s05_ctl	
S05	After the ADRF has removed the record, no retrieval returns its content; the state it keeps is replaced on removal and checked before it answers.	`network`	bounded (the changing value unrolled to 5 steps)	b_s05	checked
S05	After the removal no retrieval returns its content, whoever asks; the malicious NF has its own credential and the data set identifier.	`malicious_nf`	bounded (the changing value unrolled to 5 steps)	b_s05	checked
R02	A repeated delete message reaches the ADRF and is refused - the situation the target speaks of is reached.	`network`	reachable	b_r02_ctl	
R02	A second delete of the same record is refused: the ADRF replaces the state it keeps, so the same message does not make it act again.	`network`	bounded (the changing value unrolled to 5 steps)	b_r02	checked
R02	A replayed delete, from another NF, does not remove the record a second time and does not change any other record.	`malicious_nf`	bounded (the changing value unrolled to 5 steps)	b_r02	checked
Z05	A malicious legitimate NF cannot read another NF's record: with no other NF's credential, no serving of that record ever happens.	`network`	proved	z05_net_get	checked
Z05	A malicious legitimate NF cannot read another NF's record; it has its own credential and token, the API scope, and the data set identifier of the record.	`malicious_nf`	attack (holds_in_source)	z05_read (the control of z05_net_get)	
Z05	A malicious legitimate NF cannot remove another NF's record; it has its own credential and token and the data specification of the record.	`malicious_nf`	attack (holds_in_source)	z05_rmspec	
Z05	A malicious legitimate NF cannot remove another NF's record by storeTransId; it has its own credential and token and learns the storeTransId.	`malicious_nf`	attack (holds_in_source)	z05_del	

target	claim	attacker	answer	query	control
U01	The content a reader accepts was really stored by the storing NF with the same storeTransId and content; nothing on the network can make it accept anything else.	`network`	proved	d01	checked
U01	The reader is an NF the storing NF approved; the malicious NF has its own credential and token and the data set identifier.	`malicious_nf`	attack (holds_in_source)	z05_read (the control of z05_net_get)	
U02	Reading and removing each require a permission covering that operation and one cannot stand for the other; the malicious NF has one token for the API scope.	`malicious_nf`	attack (holds_in_source)	u02	
U02	Reading and removing require a permission; the network attacker holds no access token at all, so it can invoke neither operation.	`network`	proved	x01_tok	checked
U03	A consumer accepts a retrieval result only as the answer to its own request; the situation this target speaks of is reached (an ADRF answer to a later request).	`network`	reachable	b_u03_ctl	
U03	After the removal the reader's next retrieval is answered 204 and no old payload comes back as a fresh result.	`network`	bounded (the changing value unrolled to 5 steps)	b_u03	checked
U03	After the removal no consumer, the malicious NF included, obtains the old record as a fresh retrieval result.	`malicious_nf`	bounded (the changing value unrolled to 5 steps)	b_u03	checked
X01	The access token issued to an NF is known only to that NF and the NRF: the network attacker never obtains another NF's token.	`network`	proved	x01_tok	checked
X01	The TLS credential of an NF stays with that NF: another legitimate NF cannot obtain it and so cannot act towards the ADRF as that NF.	`malicious_nf`	proved	x01_key	checked

Appendix D: How the analysis was run

An LLM analyst builds the model of the protocol from the documents and puts each property to ProVerif; every result in section 3 is an answer of the prover.

ProVerif+ Agent session session-20261114T230426, 2026-11-14. Status: finished. Model: deepseek-v4-1-flash-261017. Steps: 51, prover 6 s, wall 16 min, cost 0.150 USD.

ProVerif: 127 proofs run, 126 decided, 1 without an answer, 4 s, 36 answers recorded.

What was analysed

Protocol	ADRF analytics record lifecycle
Version	19.7.0
Analysed	A single non-UE aggregated analytics record, StorageRequest→RetrievalRequest→Delete, one storing NF, one NF permitted to read, one malicious legitimate NF; bounded delete with a fixed two states; upstream subscription, UE consent and record expiry scheduling are excluded.
Not analysed	The symbolic model does not prove the computational security or the post-quantum resistance of the algorithms; the attack traces hold only under explicit assumptions and are not directly equivalent to a specification flaw. Every property must have a reachability/positive control. Fixed to Rel-19. The external network attacker and the malicious legitimate NF holding its own credentials are distinguished separately; TLS authenticated encryption is modelled per endpoint, and a legitimate NF cannot read another NF's TLS session. OAuth2 subject/audience/scope/expiry and TLS identity binding follow the attached baseline; a service scope does not automatically equal resource ownership. Monotonic tables cannot model real deletion, so the finite-state lifecycle must be explicitly delimited. The target is the specified API sub-flow, not complete 5G security.
Source text	443 numbered passages (331162 bytes) from the sections listed below. A reference such as s0001 in this report is one of these passages

The sections of the documents the analysis is built from:

Document	Section	Title
main	4.2.2.2	Nadrf_DataManagement_StorageRequest service operation
main	4.2.2.2.1	General
main	4.2.2.2.2	Request storage of data or analytics
main	4.2.2.5	Nadrf_DataManagement_RetrievalRequest service operation
main	4.2.2.5.1	General
main	4.2.2.5.2	Request and retrieve stored data or analytics from the ADRF Data Store
main	4.2.2.9	Nadrf_DataManagement_Delete service operation
main	4.2.2.9.1	General
main	4.2.2.9.2	Requesting removal of stored data or analytics
main	4.2.2.9.3	Requesting removal of stored data or analytics using a data or analytics specification
main	5.1	Nadrf_DataManagement Service API
main	5.1.1	Introduction
main	5.1.2	Usage of HTTP

Document	Section	Title
main	5.1.2.1	General
main	5.1.2.2	HTTP standard headers
main	5.1.2.2.1	General
main	5.1.2.2.2	Content type
main	5.1.2.3	HTTP custom headers
main	5.1.3	Resources
main	5.1.3.1	Overview
main	5.1.3.2	Resource: ADRF Data Store Records
main	5.1.3.2.1	Description
main	5.1.3.2.2	Resource Definition
main	5.1.3.2.3	Resource Standard Methods
main	5.1.3.2.3.1	POST
main	5.1.3.2.3.2	GET
main	5.1.3.2.4	Resource Custom Operations
main	5.1.3.3	Resource: Individual ADRF Data Store Record
main	5.1.3.3.1	Description
main	5.1.3.3.2	Resource Definition
main	5.1.3.3.3	Resource Standard Methods
main	5.1.3.3.3.1	DELETE
main	5.1.3.3.4	Resource Custom Operations
main	5.1.3.4	Resource: ADRF Data Retrieval Subscriptions
main	5.1.3.4.1	Description
main	5.1.3.4.2	Resource Definition
main	5.1.3.4.3	Resource Standard Methods
main	5.1.3.4.3.1	POST
main	5.1.3.4.4	Resource Custom Operations
main	5.1.3.5	Resource: Individual ADRF Data Retrieval Subscription
main	5.1.3.5.1	Description
main	5.1.3.5.2	Resource Definition
main	5.1.3.5.3	Resource Standard Methods
main	5.1.3.5.3.1	DELETE
main	5.1.3.5.4	Resource Custom Operations
main	5.1.4	Custom Operations without associated resources
main	5.1.4.1	Overview
main	5.1.4.2	Operation: request-storage-sub
main	5.1.4.2.1	Description

Document	Section	Title
main	5.1.4.2.2	Operation Definition
main	5.1.4.3	Operation: request-storage-sub-removal
main	5.1.4.3.1	Description
main	5.1.4.3.2	Operation Definition
main	5.1.4.4	Operation: remove-stored-data-analytics
main	5.1.4.4.1	Description
main	5.1.4.4.2	Operation Definition
main	5.1.5	Notifications
main	5.1.5.1	General
main	5.1.5.2	Retrieval Notification
main	5.1.5.2.1	Description
main	5.1.5.2.2	Target URI
main	5.1.5.2.3	Standard Methods
main	5.1.5.2.3.1	POST
main	5.1.5.3	ADRF Alert Notification
main	5.1.5.3.1	Description
main	5.1.5.3.2	Target URI
main	5.1.5.3.3	Standard Methods
main	5.1.5.3.3.1	POST
main	5.1.6	Data Model
main	5.1.6.1	General
main	5.1.6.2	Structured data types
main	5.1.6.2.1	Introduction
main	5.1.6.2.2	Type: NadrfDataStoreRecord
main	5.1.6.2.3	Type: NadrfDataStoreSubscription
main	5.1.6.2.4	Type: NadrfDataRetrievalSubscription
main	5.1.6.2.5	Type: NadrfDataRetrievalNotification
main	5.1.6.2.6	Type: NadrfDataStoreSubscriptionRef
main	5.1.6.2.7	Type: NadrfStoredDataSpec
main	5.1.6.2.8	Type: DataSubscription
main	5.1.6.2.10	Type: StorageHandlingInfo
main	5.1.6.2.11	Type: NadrfAlertNotification
main	5.1.6.2.12	Type: NadrfAlertNotificationResponse
main	5.1.6.2.13	Type: DataSetTag
main	5.1.6.3	Simple data types and enumerations
main	5.1.6.4	Data types describing alternative data types or combinations of data types

Document	Section	Title
main	5.1.7	Error Handling
main	5.1.7.1	General
main	5.1.7.2	Protocol Errors
main	5.1.7.3	Application Errors
main	5.1.8	Feature negotiation
main	5.1.9	Security
main	5.1.6.2.9	

The purpose of the protocol, as the source step located it in the documents: a single non-UE aggregated analytics record; StorageRequest → RetrievalRequest → Delete; one storing NF, one NF allowed to read, and one malicious legitimate NF; fixed two-state bounded deletion; excluding upstream subscriptions, UE consent, and record expiry scheduling.

The properties against the documents

Before writing a query, the analyst set the drafted list of properties against what the documents themselves say the protocol is for. The name, what a property is about, and what the documents say are the analyst's reading; the places named exist in the documents.

ID	Name	About	What the documents say	Where
Z01	Token-gated retrieval and delete	The ADRF acts only for an NF service consumer that presents an access token the NRF issued to that same NF for the Nadrf_DataManagement scope; the token covers the operation requested.	Clause 5.1.9 says access to the API may be authorized by OAuth2 with the NRF as authorization server, that a consumer obtains a token from it, and lists the scope nadrf-datamanagement:storage-read-delete-subs for the RetrievalRequest and Delete operations.	s0368, s0369, s0371, s0373
Z03	Token bound to subject and record	The ADRF checks that the token is the caller's own, that its audience is the ADRF, and that the record named by the request is one the caller may act on.	The API defines one scope for retrieval, delete and subscription operations (5.1.9) and says nothing about binding a token to a storeTransId or to a resource owner; RFC 6749 section 10.3 states that OAuth offers the resource server no method to check that a token was issued to the client presenting it, and section 10.16 points at audience restriction.	rfc6749:2879, s0373, s0034, s0053
K02	Payload hidden from other NFs	The analytics payload carried in a retrieved NadrfDataStoreRecord is known only to the storing NF and to the approved reader, not to another legitimate NF nor to an attacker on the network.	The retrieval response carries the stored analytics in anaNotifications/dataNotif (4.2.2.5.2); clause 5.1.9 and TS 33.501 5.9.2.1 require the SBI exchange to be mutually authenticated and protected.	s0037, s0039, s0040, s0368
D01	Served record matches stored record	The reader is assured that what it accepts is the record the ADRF really stores under the identifiers it asked for, with the same storeTransId and the same payload.	Retrieval finds the data according to the requested parameters and returns the NadrfDataStoreRecord (4.2.2.5.2); TS 33.501 5.9.2.1 requires mutual authentication between consumer and producer and that each NF validate incoming messages.	ts-33-501:2871, s0034, s0036, s0037
S05	No retrieval of a deleted record	After the ADRF has removed an Individual ADRF Data Store Record, no consumer, legitimate or malicious, obtains its content again; the ADRF answers 204 or an error.	4.2.2.9.2 says the ADRF shall remove the corresponding stored record and answer 204, and 4.2.2.5.2 says a retrieval of data that does not exist is answered 204.	s0054, s0055, s0056, s0045

ID	Name	About	What the documents say	Where
R02	Delete accepted at most once	A recorded delete request that the network or another NF sends again does not make the ADRF act a second time and does not change any other record.	The delete request names the record by storeTransId or by a data specification and the ADRF removes matching records and answers 204 (4.2.2.9.2, 4.2.2.9.3); TS 33.501 5.9.2.1 requires that each NF validate incoming messages against the protocol specification and the network state.	ts-33-501:2875, s0053, s0054, s0055, s0069
Z05	Other NFs' records stay protected	A malicious but legitimate NF service consumer, holding its own credentials and a token for the API scope, cannot read or remove a record another NF stored.	The single scope of 5.1.9 covers storage, retrieval and delete for the whole API; the document states no per-record ownership check, so what protects one NF's records from another legitimate NF is only the scope and the record identifier.	s0373, s0034, s0053, s0069
U01	Record and reader both authentic	The content a reader accepts was really stored by the storing NF, and the reader is an NF the storing NF approved; storeTransId and content agree with what was stored.	The ADRF assigns the storeTransId and returns the record (4.2.2.2.2 step 2), retrieval returns the record found for the request parameters (4.2.2.5.2); the documents say nothing about binding a stored record to the NF that may read it.	s0019, s0022, s0034, s0037
U02	Read and delete permissions separate	Reading and removing a record each require a permission covering that operation, and one cannot stand for the other.	Clause 5.1.9 defines one scope, nadrf-datamanagement:storage-read-delete-subs, for both the RetrievalRequest and the Delete operation, so the documents do not separate the two permissions.	s0028, s0048, s0373
U03	No stale success after deletion	After a deletion or an error, an old success response cannot be presented to a consumer as a fresh retrieval result.	The ADRF answers 204 when the data does not exist and 204 after a successful delete (4.2.2.5.2, 4.2.2.9.2); the documents say nothing about replay of responses or about caching.	s0045, s0054, s0055, s0056
X01	Token and credential confidentiality	The access token and client credential of an NF service consumer, against another NF and against the network.	RFC 6749 section 10.3 requires access token credentials to be kept confidential and forbids unauthorized parties to generate, modify or guess valid tokens, and section 10.10 bounds the guessing probability; TS 33.501 5.9.2.2 makes the NRF the authorization server and requires it and the NFs to be mutually authenticated.	rfc6749:2859, rfc6749:2870, rfc6749:3032, ts-33-501:2883

Added from the documents by the analyst: X01.

The security statements of the documents, paragraph by paragraph

The host listed the 398 paragraphs of the sections in which the documents say what the protocol protects and against what, and the analyst said of each what it made of it: a property, an attacker, or why neither. Taken up: 30. Not taken: 368. Which paragraphs are listed follows from the layout of the standard; where each went is the analyst's judgement.

Paragraphs	Where	Taken up as	Why not
P001 to P002	rfc6749:2778-2786	No goal stated	General remark that OAuth security depends on the deployment; states nothing this protocol must give.
P003 to P006	rfc6749:2790-2823	X01, `malicious_nf`	
P007 to P010	rfc6749:2827-2850	Z03, `malicious_nf`	
P011 to P015	rfc6749:2859-2881	Z01, Z03, X01, `malicious_nf`	

Paragraphs	Where	Taken up as	Why not
P016 to P021	rfc6749:2885-2917	Outside the scope	Refresh tokens; this API uses the client credentials grant only (5.1.9).
P022 to P025	rfc6749:2921-2943	Outside the scope	Authorization codes; not used by the client credentials grant.
P026 to P029	rfc6749:2947-2980	Outside the scope	Authorization code redirection; no user agent or redirection in this flow.
P030 to P033	rfc6749:2984-3003	Outside the scope	Resource owner password credentials grant; not used.
P034 to P035	rfc6749:3013-3019	X01	
P036	rfc6749:3023-3028	Z01, D01	
P037 to P039	rfc6749:3032-3041	X01	
P040 to P042	rfc6749:3045-3066	Outside the scope	Phishing of end users; no end user takes part in this API sub-flow.
P043 to P048	rfc6749:3070-3110	Outside the scope	CSRF against a user agent redirection endpoint; no user agent here.
P049 to P050	rfc6749:3119-3138	Outside the scope	Clickjacking of an authorization page; no browser interaction.
P051 to P052	rfc6749:3142-3150	No goal stated	Advice to implementers to sanitize inputs; it states no property of the exchange.
P053 to P054	rfc6749:3154-3167	Outside the scope	Open redirectors; no redirection in this flow.
P055 to P062	rfc6749:3170-3210	Outside the scope	Implicit flow and impersonation of a resource owner; this API uses the client credentials grant with no resource owner.
P063	ts-33-501:2491-2491	The prover cannot be asked	Bidding down of negotiated security features; the model has no negotiation and ProVerif is not asked about algorithm choice.
P064 to P072	ts-33-501:2495-2511	Outside the scope	Authentication and authorization of the UE and the serving network; this API sub-flow has no UE.
P073 to P074	ts-33-501:2515-2517	Outside the scope	Key lengths and key derivation for AS/NAS protection; not the ADRF API.
P075 to P078	ts-33-501:2523-2529	Outside the scope	Requirements on the UE.
P079 to P089	ts-33-501:2533-2553	Outside the scope	UE user and signalling data confidentiality over the radio interface.
P090 to P103	ts-33-501:2557-2583	Outside the scope	UE user and signalling data integrity over the radio interface.
P104 to P110	ts-33-501:2587-2599	Outside the scope	Storage of subscription credentials; no subscription data in this bounded sub-flow.
P111 to P125	ts-33-501:2603-2631	Outside the scope	Subscriber privacy; the record analysed is a non-UE aggregated analytics record and the task excludes UE consent and subscription.
P126	ts-33-501:2637-2637	Outside the scope	General heading for gNB requirements.
P127 to P136	ts-33-501:2641-2659	Outside the scope	User data and signalling data confidentiality between UE and gNB.

Paragraphs	Where	Taken up as	Why not
P137 to P147	ts-33-501:2663-2683	Outside the scope	User data and signalling data integrity between UE and gNB.
P148 to P178	ts-33-501:2687-2761	Outside the scope	Requirements on gNB setup, key management, F1/E1 interfaces and on the ng-eNB; none of it is the ADRF API.
P179 to P192	ts-33-501:2767-2795	Outside the scope	Signalling confidentiality and integrity on the access network side, not over the service based interface of this API.
P193 to P198	ts-33-501:2799-2811	Outside the scope	Subscriber privacy and SEAF requirements.
P199 to P213	ts-33-501:2819-2853	Outside the scope	Requirements on UDM, SIDF, AUSF and UDR; the analysed sub-flow excludes subscription data and user consent.
P214	ts-33-501:2859-2859	Z01, K02, D01, `malicious_nf`	
P215 to P217	ts-33-501:2865-2869	Outside the scope	Service registration and discovery and topology hiding; the task excludes discovery and the model assumes the consumer knows the ADRF.
P218	ts-33-501:2871-2871	Z01, Z03, D01, `malicious_nf`	
P219 to P220	ts-33-501:2873-2875	D01, S05, R02	
P221 to P227	ts-33-501:2879-2891	Z01, X01, `malicious_nf`	
P228 to P234	ts-33-501:2895-2907	Outside the scope	NEF security requirements; the NEF is not a party of this sub-flow.
P235 to P241	ts-33-501:2911-2923	Outside the scope	Service Communication Proxy; the model assumes direct communication between consumer and ADRF.
P242 to P254	ts-33-501:2929-2953	Outside the scope	End-to-end core network interconnection and roaming; the scenario is inside one operator domain.
P255 to P292	ts-33-501:2957-3031	Outside the scope	SEPP requirements on N32; no roaming interface in this sub-flow.
P293 to P303	ts-33-501:3035-3055	Outside the scope	Roaming intermediaries on N32.
P304 to P312	ts-33-501:3059-3075	Outside the scope	Protection of attributes over N32-f; no SEPP here.
P313 to P314	ts-33-501:3079-3081	Outside the scope	IPUPS packet filtering; not part of the ADRF API.
P315 to P318	ts-33-501:3085-3091	No goal stated	Deployment rule for the 3gpp-Sbi-Originating-Network-Id header, which the documents describe without any property of the record lifecycle.
P319 to P327	ts-33-501:3097-3115	Outside the scope	Signalling traffic monitoring; a separate feature.
P328 to P337	ts-33-501:3121-3141	Outside the scope	Visibility and configurability of security features towards the UE and the user.
P338 to P369	ts-33-501:3149-3215	The prover cannot be asked	Algorithm identifiers, algorithm strength and selection; ProVerif is not asked about cryptographic strength.
P370 to P375	ts-33-501:3219-3231	Outside the scope	Requirements on 5G-RG and NSSAAF.
P376 to P380	ts-33-501:6540-6548	Outside the scope	Security on N3.

Paragraphs	Where	Taken up as	Why not
P381 to P394	ts-33-501:9656-10917	Outside the scope	EAP-TLS annex examples and IAB requirements.
P395 to P398	ts-33-501:11439-11901	Outside the scope	MBSF/MBSTF and NEF requirements in the annexes.

The parties and the protected values, one by one

The host listed the 23 parties and protected values that the drafted properties name, and the analyst said of each which attacker controls the party or holds the value, or why no such attacker was set. With an attacker: 11. Without: 12. The list follows from the properties; what was made of each entry is the analyst's judgement.

ID	Party or value	Kind	Attacker	Why none
S01	NF service consumer	party	`malicious_nf`	
S02	Malicious legitimate NF	party	`malicious_nf`	
S03	ADRF	party	Everything lost by definition	The ADRF is the party that stores and serves the record. With it under the attacker, every property asked here is lost by definition, and the documents ask none of them under a dishonest ADRF.
S04	NRF	party	Everything lost by definition	A dishonest authorization server issues a token for any identity, so every authorization property asked here is lost by definition. The documents ask none of them under a compromised NRF.
S05	NF allowed to read	party	`malicious_nf`	
S06	Storing NF	party	`malicious_nf`	
S07	OAuth2 access token and its scope	value	`malicious_nf`	
S08	Stored record storeTransId	value	`malicious_nf`	
S09	Data/analytics in a record	value	Everything lost by definition	The analytics content itself. An attacker holding it has the very secret the confidentiality target protects, and that is the breach, not a further attacker.
S10	Access token	value	`malicious_nf`	
S11	storeTransId	value	`malicious_nf`	
S12	Record content	value	Everything lost by definition	Record content held by the attacker is the breach of the confidentiality target itself.
S13	Contents of dataNotif/anaNotifications in NadrfDataStoreRecord	value	Everything lost by definition	The retrieved payload held by the attacker is the breach of the confidentiality target itself.
S14	Record payload	value	Everything lost by definition	The record payload held by the attacker is the breach of the confidentiality target itself.
S15	Deleted record	value	Nothing gained	A copy of a deleted record tells the attacker nothing about whether the ADRF still serves it. The deletion targets speak about what the ADRF returns.
S16	Delete request	value	`malicious_nf`	
S17	Record state	value	Everything lost by definition	The record state is kept inside the ADRF. An attacker that controls it has the ADRF, under which no property asked here survives.

ID	Party or value	Kind	Attacker	Why none
S18	Another NF's record	value	Everything lost by definition	Holding another NF's record is the breach the confidentiality and authorization targets describe.
S19	Read permission	value	`malicious_nf`	
S20	Delete permission	value	`malicious_nf`	
S21	Record	value	Everything lost by definition	Holding the record is the breach itself.
S22	Error response	value	Nothing gained	An error response is produced by the ADRF and protected to the requiring NF. Producing or replaying one needs nothing beyond the network, which is the base attacker these targets are answered under.
S23	Cached response	value	Nothing gained	A replayed cached response needs nothing beyond the network, which is the base attacker. A stored copy held by another NF is covered by the malicious NF.

The properties under each attacker

The attackers are the analyst's, set before a query was written; each answer was recorded under one of them. A cell is the prover's answer for the property under that attacker; "No answer" is an attacker set for the property under which the prover left a claim undecided or nothing was recorded; an empty cell is an attacker the analyst did not set for the property.

Attacker	Has	Basis	Where
`network`	the network only: it reads, drops, changes and sends messages, and holds no secret of a party		
`malicious_nf`	The credentials of one NF service consumer that the operator considers legitimate: its own TLS endpoint credential, an access token the NRF issued to it for the scope nadrf-datamanagement:storage-read-delete-subs, and the identifier of the record it goes after (the storeTransId it received or the dataSetId describing the analytics). It has no key of any other NF and does not read other NFs' TLS sessions.	The scope table of 5.1.9 gives one token for storage, retrieval and delete to every NF that is authorized for the API. The retrieval request may name the record by store-trans-id or data-set-id (4.2.2.5.2) and the delete by a data specification (4.2.2.9.3), so a consumer addresses records it did not store. TS 33.501 5.9.1 puts every NF of a trust zone inside the boundary, so a hostile NF is a legitimate insider with its own credentials.	s0373, s0034, s0069, s0368

ID	`network`	`malicious_nf`
Z01	Holds	Holds
Z03	Holds	Fails
K02	Holds	Fails
D01	Holds	Holds
S05	Holds	Holds
R02	Holds	Holds
Z05	Holds	Fails
U01	Holds	Fails
U02	Holds	Fails
U03	Holds	Holds

ID	network	malicious_nf
X01	Holds	Holds

Source and targets

Source: 443 units, 331162 bytes, frozen in source-delivery-01.json. Targets: 10 approved (Z01, Z03, K02, D01, S05, R02, Z05, U01, U02, U03); the questions asked: 3. The list of targets was approved as drafted, beforehand, with --approve-targets.

Decisions the host took by its documented default (task `decisions: auto`), for you to check:

- approve_package: approved source package protocol-f1f52cf258d8c7809ece111ccf09

This step as it reports itself: 1 requests (1 failed), 0.000 USD, 152430 tokens in and 4129 out, 202 s.

Notes on the prover and the model

- S05: answered in ProVerif, and the analyst names changing_state for this target
- R02: answered in ProVerif, and the analyst names changing_state for this target
- U03: answered in ProVerif, and the analyst names changing_state for this target

A ProVerif model that simplifies algebra (xor, dh_algebra, other_algebra) gives the attacker less than the real operation: a proof there is a proof against a weaker attacker, an attack there is an attack. A ProVerif model that leaves out a check on a value that changes (changing_state) lets parties accept more: a proof there holds, an attack there may be nothing but the missing check. Tamarin+ Agent models these as they are.

A bound in steps is an answer on bounded.pv: the analyst's second ProVerif model, in which the party that keeps a value that changes is unrolled to that many steps with its check in place. That model does less than the protocol does, so the property holds for those steps and no further, and an attack found there is an attack. The number of steps is the analyst's statement.

State review by the analyst

This part is the analyst's own reading of the source, not an answer of the prover. "yes" under shown by ProVerif means that ProVerif found a trace with these steps in the unrolled model, so the steps are possible; that they break a target is said only by an attack in the table of answers, and what else they cost is the analyst's reading. "nothing found" says which patterns were looked at and proves nothing.

value	kept by	changes when	checked when	refs
existence of one Individual ADRF Data Store Record together with its payload and its storeTransId	the ADRF	the StorageRequest creates it and the ADRF assigns a storeTransId (s0018, s0019); a Delete it processes and accepts removes it, by storeTransId (s0054, s0055) or by data or analytics specification (s0069).	on every RetrievalRequest, when the ADRF looks for records matching the request parameters (s0036) and on every Delete before it removes anything (s0054).	s0018, s0019, s0036, s0045, s0054, s0055, s0057, s0069
storeTransId of a stored record	the ADRF and the NF service consumer that stored the record	assigned afresh for every StorageRequest, so every stored record gets its own (s0019) and the storing NF learns it from the Location header (s0022).	the ADRF looks the record up by it in a Delete (s0053) and a consumer may name it in a RetrievalRequest (s0034).	s0019, s0022, s0034, s0053

pattern	result	target	shown by ProVerif	steps
replay	nothing found			bounded.pv St3 tests the kept state and only then fires the removal, passing gone0 to the next step; the second delete reaching the ADRF is refused, so no second acceptance happens (b_r02 verified, b_r02_ctl shows the refusal reached).
repeat_one_time_action	nothing found			The removal is a step the ADRF performs once per record; after it the state is replaced and a repeated or replayed delete is answered with an error rather than removing anything again.
desynchronisation	nothing found			The record state is held only by the ADRF; the storing NF keeps nothing but the storeTransId, so there is no second copy that could drift, and the search over five steps finds no mismatch to use.
update_before_check	nothing found			In bounded.pv St3 the state is tested before the removal is performed, and the new state is handed to the next step only after the check; no trace removes the record on a check that failed.
no_rollback_on_failure	nothing found			A delete that finds no record leaves the kept state as it was: the refusing branch hands the unchanged state on (bounded.pv St3 and St5), so a failed or abandoned run does not pretend the record is gone.
race_between_sessions	nothing found			model.pv keeps records in a table that only grows, so two deletes of one record are both accepted there; that is the monotone table, not the protocol. In bounded.pv, where the state is replaced, the second delete is refused, and the specification gives the ADRF one record state to read and change in each step, so no order of two sessions removes a record twice in a way a target forbids.
distinguishable_failure	candidate		yes: ProVerif falsifies `b_s05_ctl` on bounded.pv	The malicious NF holds its own credential and token for the scope nadrf-datamanagement:storage-read-delete-subs (s0373). / It sends a RetrievalRequest naming a data set identifier (s0034). / The ADRF finds a record and answers 200 with the record, or finds none and answers 204 (s0037, s0045). / The two answers differ, so the NF learns whether another NF's record exists.
wrap_or_window	nothing found			No counter, sequence number or acceptance window is used for the record state; the storeTransId is freshly assigned per StorageRequest and never compared against an earlier one, so there is nothing to wrap.

What the analyst named for targets:

step	targets	features	reason
38	S05, R02, U03	changing_state	The ADRF keeps the state of one Individual ADRF Data Store Record - it exists after the StorageRequest and is removed by the Delete - and these three targets are about what follows from that change: no retrieval returns the payload afterwards, a second delete is not accepted, and no old payload is accepted as a fresh result. model.pv cannot express the removal (a table only grows), so the answers come from bounded.pv, where the state is unrolled over five steps and the check on the kept value is written as clause 4.2.2.9.2 and 4.2.2.5.2 state it.
38	Z01, Z03, K02, D01, Z05, U01, U02, X01	plain_primitives, many_sessions	TLS between an NF and the ADRF or NRF is modelled as a per-endpoint symmetric key and OAuth2 tokens as unforgeable private functions of the NF identity, so the answers rest on hashing, MAC and encryption only, without any algebraic law a prover could get wrong. The roles are replicated, so each property is answered for any number of sessions of the storing NF, the reader and the malicious NF.

The analyst's closing note

Eleven targets (Z01, Z03, K02, D01, S05, R02, Z05, U01, U02, U03 and the added goal X01) were answered for the network attacker and for a malicious but legitimate NF, and the reading and figures are recorded.

What holds: only an NF that authenticates with its own credential and presents the token the NRF issued to it is served or obeyed, for retrieval and for both forms of deletion (Z01 proved); an NF's credential and token stay with it (X01 proved); what a consumer accepts is a record the ADRF really stored, with the same storeTransId, storing NF and payload (D01 proved, with a falsified control showing the storer need not be the NF the reader expected); after the removal the Adrf answers 204 and a repeated delete is refused (S05, R02, U03 on the unrolled model, controls falsified); the payload never reaches the network attacker (K02 proved).

The finding: because clause 5.1.9 defines one scope for storage, retrieval and delete and the ADRF checks that scope only, a legitimate NF that knows a data set identifier reads another NF's analytics and erases its record, by data specification or by storeTransId (Z05, K02, Z03, U01 attacks; also the loss of separation of read and delete permission, U02, and the existence oracle from the 200/204 answers). A checked change - the ADRF acting on a record only for the NF that stored it or one it approved - removes the three attacks of another NF and keeps the normal run.

What blocks more: token expiry cannot be asked of ProVerif (no time), so that claim of Z01 stays partly answered; the record lifecycle is unrolled over five steps only, and it is a model that does the removal, not the monotonic table of the unbounded model. TLS and OAuth2 are abstracted as per-endpoint keys and unforgeable tokens, so nothing is said about the primitives themselves; a compromised ADRF or NRF, an SCP and roaming are outside the analysis.