Security analysis of ADRF ML model update delete profile

19.7.0 · ProVerif+ Agent report

Documents	5G System; Analytics Data Repository Services; Stage 3 (protocol definition)
Properties examined	11: 4 hold, 7 fail (7 attacks)
Findings	2; known results reproduced in Appendix B: 1

Contents

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

1 Protocol and scope

The Nadrf_MLModelManagement service of the ADRF lets an NF service consumer store ML models as records, update a record, retrieve stored models and delete them. The consumer is usually the NWDAF containing MTLF, which holds the models and their file addresses; the ADRF keeps the records and the model files; the NRF issues the OAuth2 token for the single service scope nadrf-mlmodelmanagement.

A storage request creates a record with a new storeTransId and returns its URI; an update replaces the record; a retrieval returns the modelUniqueId and mlFileAddr of each model; a delete removes the record by storeTransId or a model by modelUniqueId. The documents name one check: for a retrieval the ADRF refuses a consumer that is neither in the model's allowConsumerList nor the NWDAF that stored it. They state no check for update or delete, and the whole API has one scope at service level, with none at resource or operation level.

The parties:

- **NF service consumer (NWDAF containing MTLF)**: stores, updates, retrieves and deletes its ML model records. Holds: its own ML model files and their addresses, its NRF credentials.
- **ADRF**: NF service producer that keeps the records and the model files. Holds: the ADRF ML Model Store records and the storeTransId values.
- **NRF**: OAuth2 authorization server for the service scope. Holds: NF registrations and client credentials.

Figure 1 shows the exchange as the documents specify it.

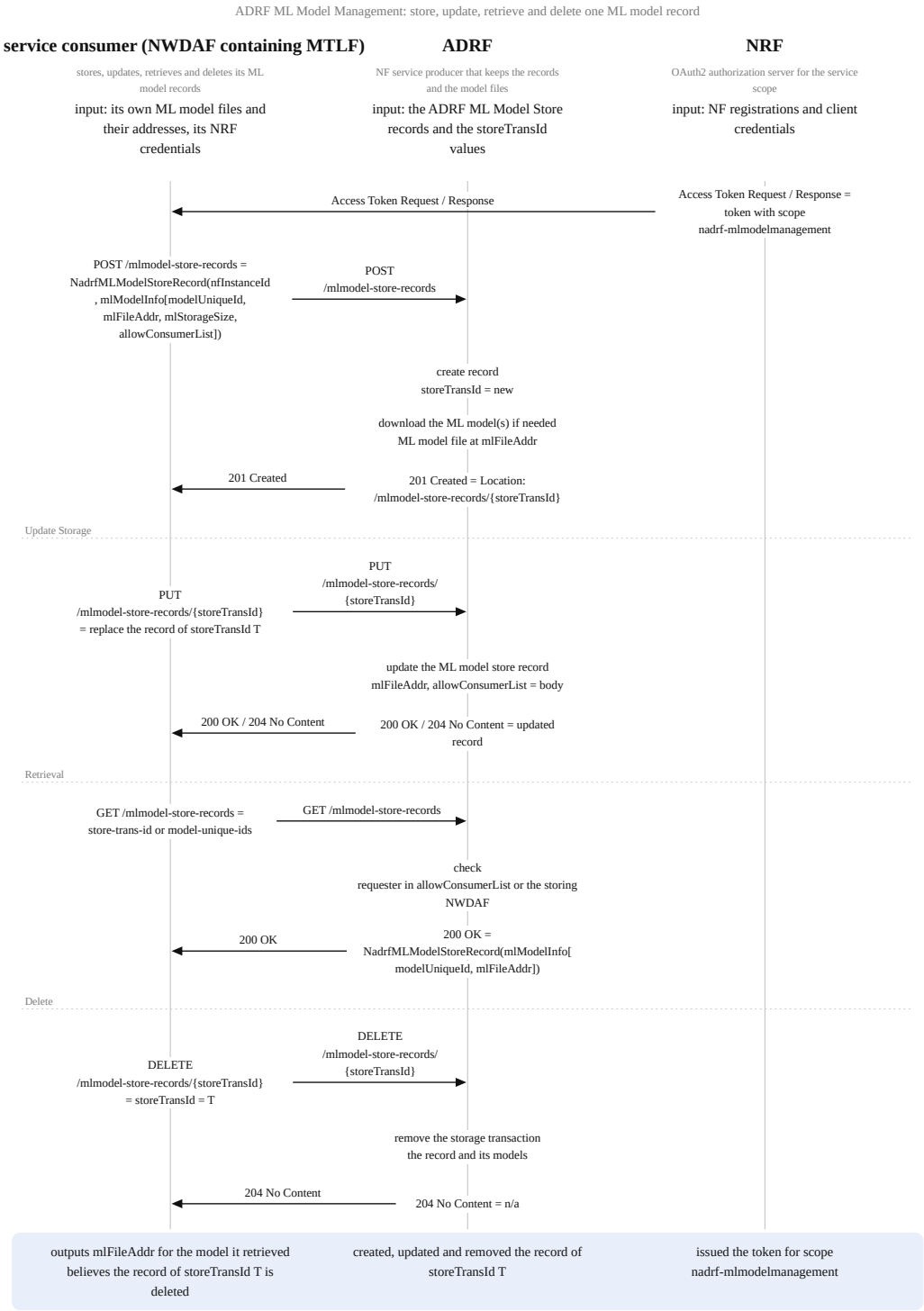


Figure 1. The protocol as specified.

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 2 findings: what follows is not stated in the documents. For 2 of them it is unclear whether the documents state them. 1 result that the documents already state is not counted and is in Appendix B.

The retrieval side of the profile behaves as written, but the write and delete side is unprotected: a legitimate NF that holds the single service-scope token deletes another NF's stored model, which the documents do not state as a weakness. A second weakness comes from the silence around modelUniqueId: another storer can register a record for an identifier already stored, and a retrieval then serves the attacker's mlFileAddr to the original storer's consumer. Within the six unrolled steps the update replaces the record as written and a delete leaves nothing retrievable. The results rest on idealised OAuth2 and TLS, on a monotonic record table outside the unrolled model, and on reading the rule of 4.5.3.2 as "in the allow list or the storer". One recommendation was checked on the model and removes the delete attack; the other is stated but not checked.

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 token holder that is not the storer deletes another NF's stored model

Kind	Weakness of the protocol as specified
New	Unclear: the documents state it in part, or could not be searched (ts-29-501:1801)
Value	high
Properties	Z03, Z05, D01 (section 3)
Source text	s0049, s0059, s0060, s0071, s0072, s0261
Shown by	ProVerif attack query `z03d`; Figures 2, 3 and 4

What happens. The retrieval procedure of 4.3.2.3.2 lets the ADRF answer only to a consumer in the allowConsumerList or to the storing NWDAF. The delete procedures of 4.3.2.4.2 and 4.3.2.4.3 and the update procedure of 4.3.2.2.3 state no such condition. A registered NF that is only a permitted reader of a model therefore removes that model from the ADRF with remove-stored-mlmodel, and a stored value is taken from any token holder's PUT.

What the attacker needs. The attacker is a registered NF service consumer with a valid access token for the single scope nadrf-mlmodelmanagement, and it knows a modelUniqueId, which its own permitted retrieval gives it or which it enumerates because the identifiers are Uintegers.

What the attacker gains. The attacker removes the model of another NF from the ADRF, or stores its own values into that NF's record, so consumers that the storer allowed lose the model or receive the attacker's file address.

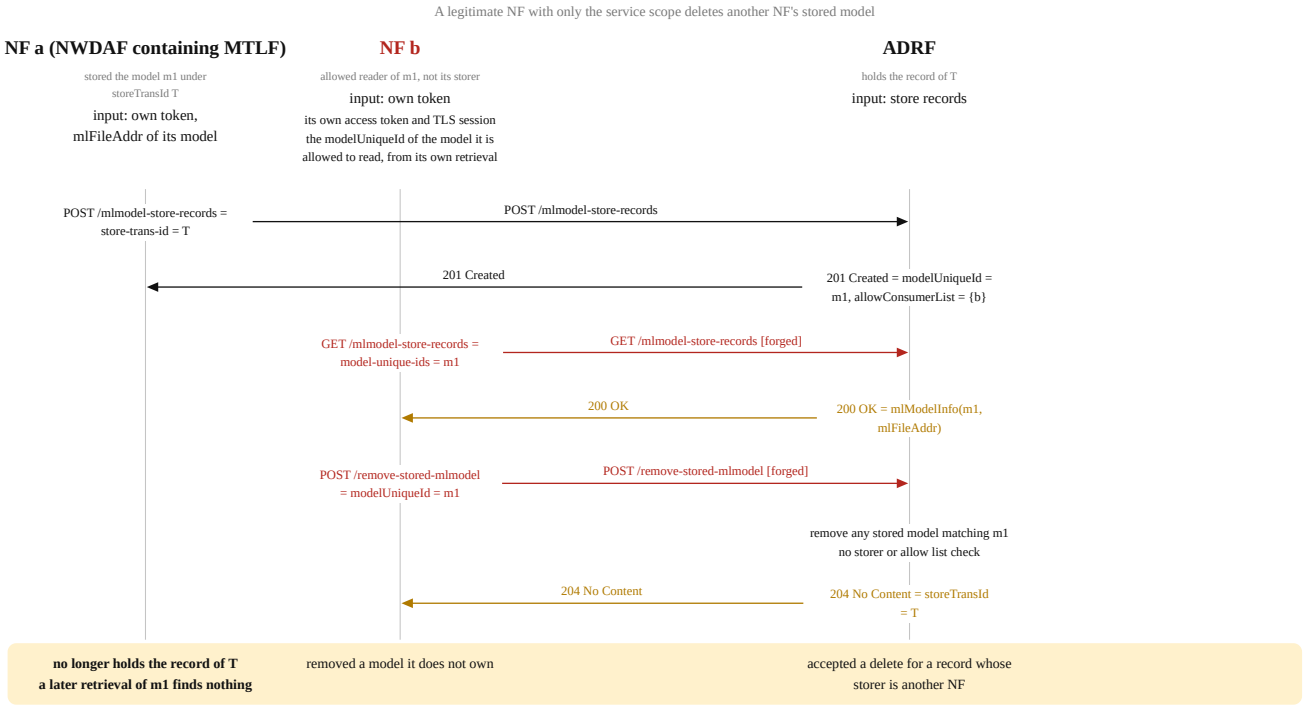


Figure 2. A legitimate NF with only the service scope deletes another NF's stored model. Property Z05; attack query z03d.

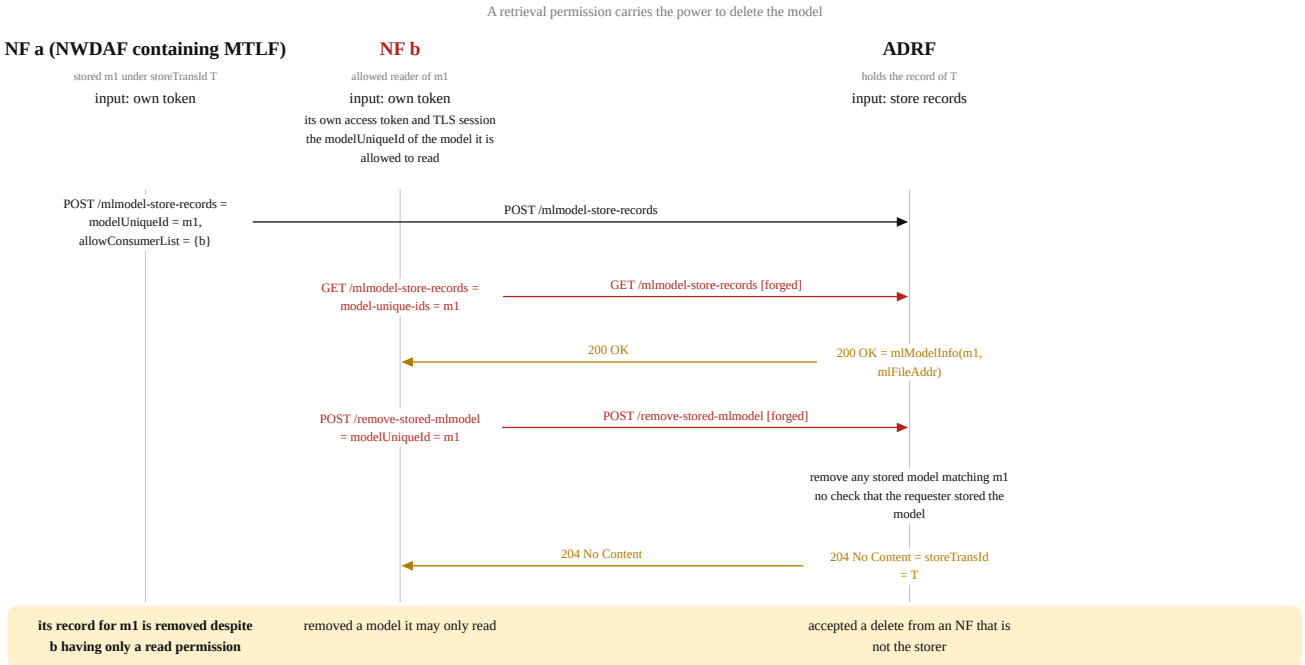


Figure 3. A retrieval permission carries the power to delete the model. Property Z03; attack query z03d.

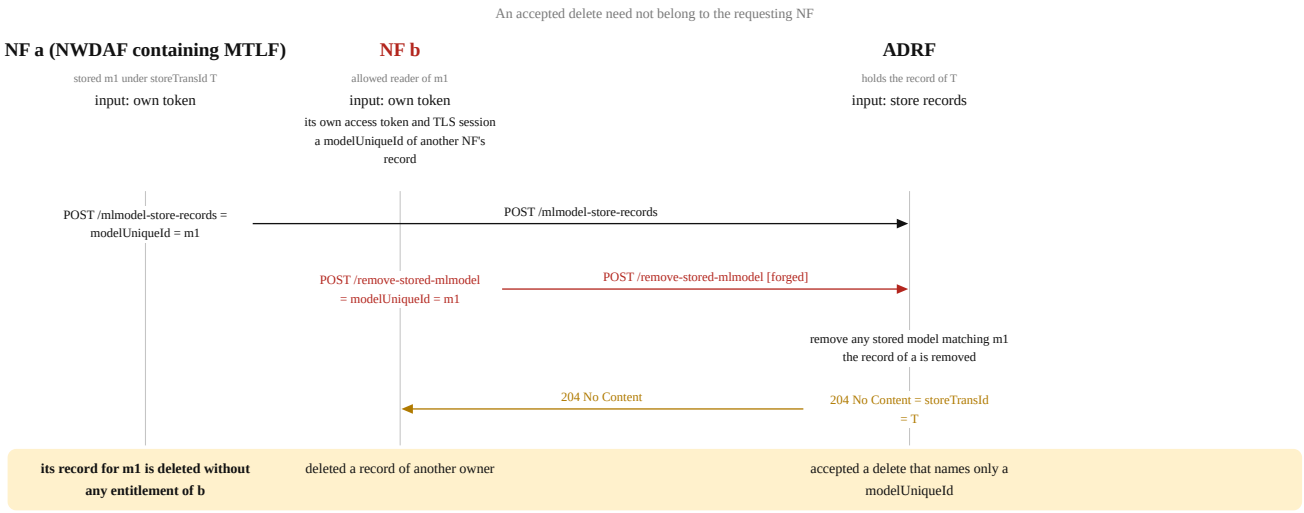


Figure 4. An accepted delete need not belong to the requesting NF. Property D01; attack query z03d.

Proposed change. Accept a PUT, a DELETE and a remove-stored-mlmodel only from the NF that stored the record: the ADRF compares the NF identified by the access token with the nfInstanceId or nfSetId of the NWDAF containing MTLF that stored it, as s0049 already does for retrieval, and never takes the owner of a record from the PUT body.

Check of the change with ProVerif. The change was put into a copy of the model: Require for the write operations the same ownership basis that s0049 states for retrieval: the ADRF accepts a PUT, a DELETE and a remove-stored-mlmodel only from the NF that stored the record (the nfInstanceId/nfSetId of the NWDAF containing MTLF that stored it), and the owner of the record is no longer taken from the PUT request body. On the changed model ProVerif proves `z03d`: the attack is no longer possible there. The normal run of the protocol is still found. The other 13 recorded answers are unchanged. 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`.

2.2 Another storer's file address is served for a stored modelUniqueId

Kind	Weakness of the protocol as specified
New	Unclear: the documents state it in part, or could not be searched
Value	high
Properties	A08, A08.9, U02 (section 3)
Source text	s0014, s0044, s0047, s0218, s0221
Shown by	ProVerif attack query `a08`; Figures 5 and 6

What happens. Any NF with the service scope may store a record for a modelUniqueId another NF already stored, and s0014 allows the ADRF to create a further record and a new storeTransId. A retrieval with model-unique-ids then returns one of the records for that identifier. In the trace the ADRF serves the record of a second storer to the first storer's consumer, which accepts the second storer's mlFileAddr.

What the attacker needs. The attacker is a registered NF service consumer with a valid access token, and it knows or guesses the modelUniqueId of a model another NF has stored; it lists a victim consumer in its own allowConsumerList.

What the attacker gains. A consumer that retrieves the model by its identifier receives a file address the attacker chose and downloads the attacker's model file instead of the one its storer published.

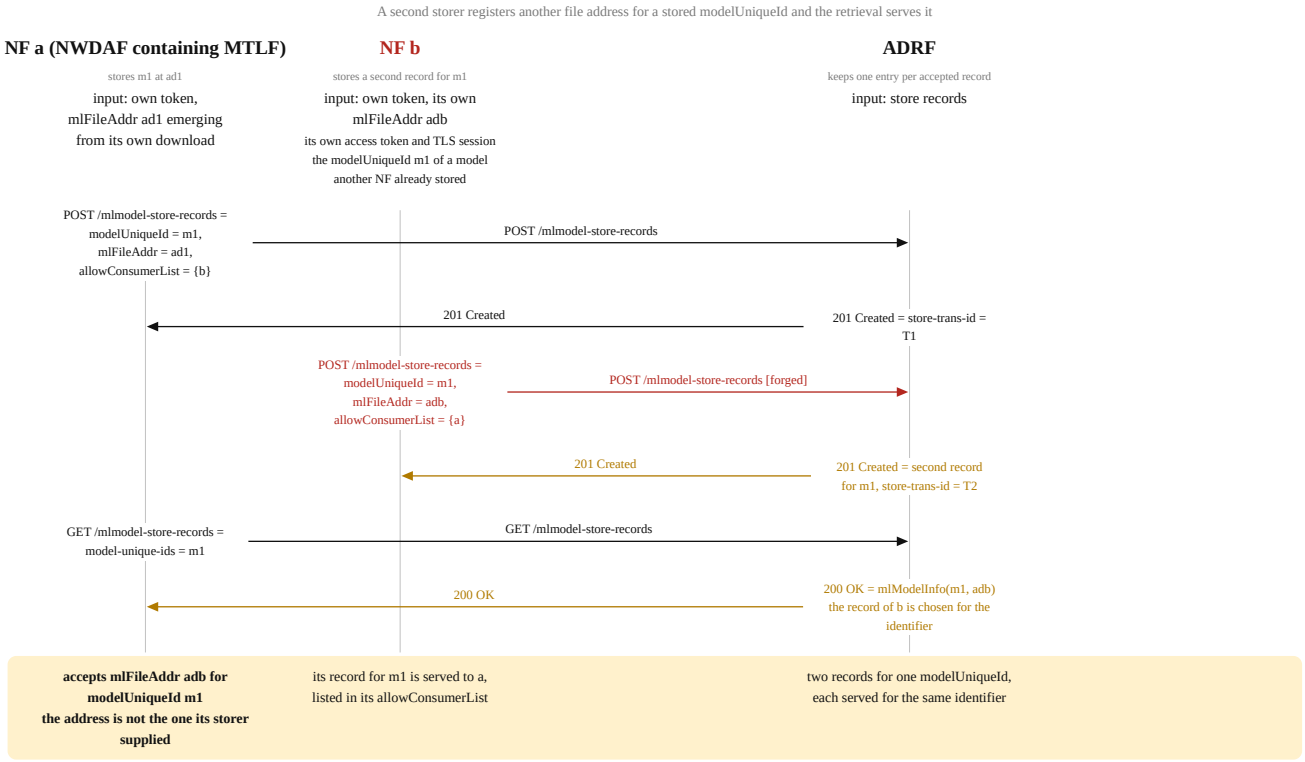


Figure 5. A second storer registers another file address for a stored modelUniqueId and the retrieval serves it. Property A08; attack query a08.

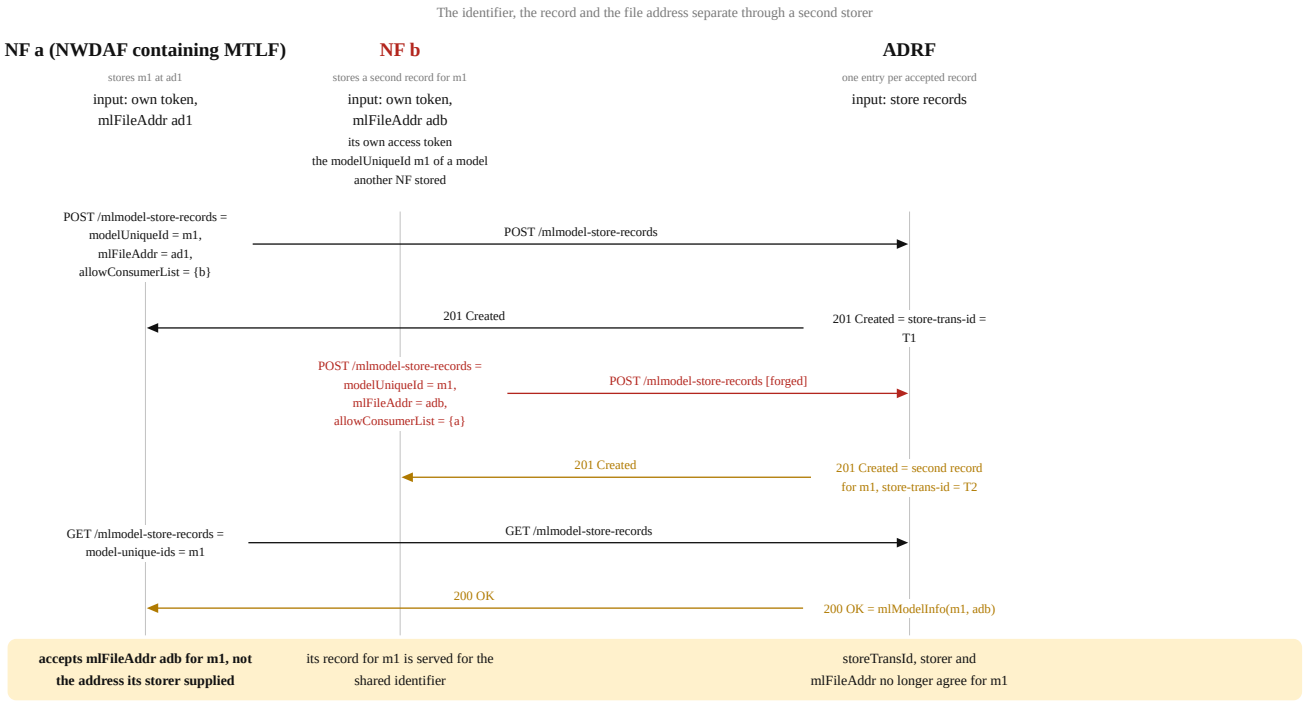


Figure 6. The identifier, the record and the file address separate through a second storer. Property A08.9; attack query a08.

Proposed change. Make modelUniqueId unique per storer in the ADRF: a POST that carries a modelUniqueId already stored by another nfInstanceId or nfSetId is refused, or the record is bound to the storer, and a GET with model-unique-ids returns only records of the storer whose identity the query names.

Check of the change with ProVerif. Not checked: The remedy needs the ADRF to look up existing records by modelUniqueId before it creates a record and to refuse a second record from another storer. A bounded model of the storage chain with that lookup was not built within the budget, so the change is stated but unchecked.

No figure was given for these attacks of a finding: U02: a08.

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	**Retrieval needs an authorization basis.** ADRFReturns model logs for each successful return (%2)200OKAnd? mlModelInfo/mlFileAddrThere must be a valid basis for authorization: the person requesting it in the modelallowConsumerList, or the requester stores the model.NWDAF(includes)MTLF); Nadref-mlmodelmanage service only does not constitute an authorization.	no compromise	Holds	medium	yes
Z03	**Retrieval permission item-bound.** Permission to retrieve must be tied individually: modelUniqueId recovered, identity of the person requesting it (nfInstanceId or nfSetId), identity of the person storing it and type of operation (retrieving); A licence on a model cannot be used for another model, another consumer or delete/update operations.	no compromise	Fails (1 attack)	high	unclear
Z05	**MaliciousNFkept from other owners' records.** In total control of a legitimate one by the assailantNF♪ Hold yourself ♪OAuth2If the evidence and legal service are available,NFCannot retrieve, update or delete model records that are not part of its permitted list and are not stored by it, nor read otherNFIt's...TLSSession.	no compromise	Fails (1 attack)	high	unclear
D01	**Update and delete of an authorized record.** ADRFEvery update accepted (PUT) or delete (DELETE/POSTRemove-stored-mlmode) must correspond to a record that is real and that the person requesting it has the right to operate; WrittenmodelUniqueId、mlFileAddr、allowConsumerListis consistent with the value submitted by the requesting party and has not been replaced by a third party.	no compromise	Fails (1 attack)	high	unclear
A08	**Record fields bound to the storer.** ADRFReturning or saving records,modelUniqueId、mlFileAddr、allowConsumerListWith StorernfInstanceId/nfSetIdThe correspondence must be consistent with the value submitted by the requesting party and authorized; Retrieving a return record by store-trans-id or model-unique-ids must be marked in accordance with the request.	no compromise	Fails (1 attack)	high	unclear
R03	**One storage or delete act per request.** The same logical storage or deletion operation (the same requester, the same model identifier, the same request content) produces at most one record creation or one deletion side effect on ADRF; Re-transmitting returns the cache result, but does not create a record or delete it again.	no compromise	Attack holds (stated in the documents)	low	yes

ID	Property	Compromise assumed	Result	Value	In the documents
S03	**Updated record not overwritten by an old version.** For the same storeTransId, the updated state of record should not be covered by the old version; After the deletion has entered into force, the old record may not be re-trievable through the re-establishment of the old request or the old marking.	no compromise	Holds within a bound (the changing value unrolled to 6 steps)	medium	no
S05	**Deleted record not reachable by old identifiers.** When the deletion enters into force, search, update, delete requests from old storeTransId or old modelUniqueId may no longer return or modify the original model records; Old markings may not be re-entitled.	no compromise	Holds within a bound (the changing value unrolled to 6 steps)	medium	no
A08.9	**Unique identifier bound to record and file.** ADRFIn the records kept and returned,modelUniqueId, StorernfInstanceId/nfSetId、storeTransIdandmlFileAddrThe correspondence must be consistent with the value submitted by the person requesting it and authorized.	no compromise	Fails (1 attack)	high	unclear
U01	**Old file address and allow list not accepted.** Up-to-date entry point, oldmlFileAddrWith the oldallowConsumerListThey must no longer be accepted as valid data.	no compromise	Holds within a bound (the changing value unrolled to 6 steps)	medium	no
U02	**Two storers, one model identifier.** Two different legal storagesNFWith the samemodelUniqueIdWhen you store it,ADRFThe conduct must fall within a branch permitted by the norm or be recorded as a branch of the explicit hypothesis and not be covered silently.	no compromise	Fails (1 attack)	medium	unclear

"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).

The questions asked, and the analyst's answers:

Question	Answer	Rests on
Q1: Model uniqueIDWithstor, recordIDThe binding of the file address.	The binding is not protected: any NF with the service scope can store a second record for a modelUniqueId another NF already holds, and a retrieval by that identifier can then return the other record's mlFileAddr, so storeTransId, storer and file address do not stay tied to one another.	A08, A08.9, U02
Q2:URL/ Allow consumers to update old data to accept borders	The update itself is clean: once the PUT takes effect the ADRF serves only the new mlFileAddr and refuses the consumer the update removed from the allow list. What is not clean is who may update: no ownership check is prescribed, and two writers can leave the older values in place because nothing orders them.	U01, S03
Q3: Two legal storagesNFThe same as in uniqueIDWhether or not the conduct is permitted by the norm or should be judged as a hypothetical branch	The specification does not settle it. It allows the ADRF to create a further record for a model already stored and says nothing about a different storer using the same modelUniqueId, so the behaviour has to be fixed as an explicit assumption; in the model both records coexist and either can be served for the identifier.	U02, A08

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:

- The download of the ML model file from mlFileAddr and the partial storage results of the EnModelMgmt feature are not modelled, so a record whose model file was never stored, and the effect of a failed download, are outside the results.
- The OAuth2 token, the NRF and the scope check are idealised: the token is an unforgeable term that binds the consumer identity and the single service scope; subject, audience and expiry, and the case of several NRFs, are not modelled.
- TLS is modelled as one session per pair of NF endpoints: the network can neither read nor modify another NF's traffic, and no NF can read another's session. Replay of an application request is possible in the model even though TLS would stop it.
- The records are kept in a monotonic table in model.pv, so replacement and removal are only modelled in bounded.pv, and there the ADRF is unrolled to six steps with one record and two models; other numbers of records, models and concurrent writers are outside the bounds.
- The retrieval rule of s0049 is read as "in the allowConsumerList or the storing NWDAF"; the text says "and/or", and the stricter reading (only the storer, or the storer and a listed consumer) was not analysed.
- The behaviour of a retrieval when allowConsumerList is absent is not modelled: the model always has a list, and the source does not say what the ADRF does without one.
- Whether the ADRF is honest, whether the NRF is honest, redirects through an SCP (307, 308), and availability are outside the analysis.
- A race of two concurrent PUTs on one record, and the distinction between 403 and 204 answers that reveals whether a modelUniqueId exists, were worked through but not confirmed by a query.

What the host records:

- The answers of D01, A08, R03, S03, S05, A08.9, U01, U02 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: Retrieval needs an authorization basis

ADRFReturns model logs for each successful return (%2)200OKAnd?mlModelInfo/mlFileAddrThere must be a valid basis for authorization: the person requesting it in the modelallowConsumerList, or the requester stores the model.NWDAF(includes)MTLF)； Nadref-mlmodelmanage service only does not constitute an authorization.

Retrieval is the one operation the documents fence: the ADRF gives a record only to a consumer in the model's allowConsumerList or to the NF that stored it. A holder of the service-scope token that is neither is refused, so the scope alone yields no model.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: The documents require this check in so many words, so the result confirms the rule for the model rather than finding something new; it is worth having because retrieval is the only place where the documents state a basis.
- Stated in the documents: yes. Where the documents say it: s0049, s0047
- Rests on: s0049 is read as "in the allowConsumerList or the storer" although the text says "and/or".; The allow list is always present in the model.; TLS is modelled per endpoint pair, so no other NF reads the response.
- Open: What the ADRF does when allowConsumerList is absent is not modelled.

Z03: Retrieval permission item-bound

Permission to recover must be tied to each item:modelUniqueId, the identity of the person requesting the request (nfInstanceIdornfSetId), storage person identity and type of operation (retrieving); A licence on a model cannot be used for another model, another consumer or delete/update operations.

A permission is bound to its model: an NF allowed for one model gets nothing of another model it did not store. The same permission is not separated from the other operations, and the NF uses its token to delete the stored model of the NF that owns it.

- Result: Fails (1 attack); answers what is asked: partly; usable
- Value high: The analysis shows that a permission the documents grant for reading carries the power to delete, which the documents do not state and which contradicts the purpose of the allow list.
- Stated in the documents: unclear. Source text it rests on: s0049, s0071, s0072
- Rests on: The retrieval permission is checked per model, as the model stores one entry per model with its own allowConsumerList.; Delete by storeTransId was not used to show the attack; remove-stored-mlmodel by modelUniqueId was, because that identifier is known to the permitted reader.
- Open: The update operation, and whether a retrieval permission is usable for it, is not shown by an attack because another NF cannot address the record URI in the model.
- Next step: Model a record URI that another NF can learn, for instance through a redirect or a shared NF set, to see whether the update path is as open as the delete path.

Z05: Malicious NF kept from other owners' records

In the attacker's complete control of a legitimate one.NF♪ Hold yourself ♪OAuth2If the evidence and legal service are available,NFCannot retrieve, update or delete model records that are not part of its permitted list and are not stored by it, nor read otherNFIt's...TLSSession.

A legitimate NF with its own token reads only the models of others it is allowed to read, and its own models without limit. It deletes the stored model of another NF by naming that model's identifier, because the delete operation asks for no ownership.

- Result: Fails (1 attack); answers what is asked: partly; usable
- Value high: A malicious but legitimate NF is the attacker 3GPP names for cross-tenant problems, and the analysis shows that the single service scope lets it delete another tenant's model, which is exactly what TS 29.501 warns a service-level scope must not do.
- Stated in the documents: unclear. Source text it rests on: s0049, s0071, s0072, s0261
- Rests on: The attacker holds its own token and cannot read another NF's TLS session, as the scope states.; The delete attack uses remove-stored-mlmodel, whose only input is the modelUniqueId.
- Open: The update operation is not exercised against another NF's record.; The claim about TLS sessions is an assumption of the model, not a query.
- Next step: Add a way for another NF to obtain a record URI, as a shared NF set would, and check the update path with the same property.

D01: Update and delete of an authorized record

ADRFEvery update accepted (PUT) or delete (DELETE/POSTRemove-stored-mlmode) must correspond to a record that is real and that the person requesting it has the right to operate; WrittenmodelUniqueId、 mlFileAddr、 allowConsumerListis consistent with the value submitted by the requesting party and has not been replaced by a third party.

An accepted update rests on a request that carried the requester's own token over its own TLS session, so no third party substitutes the values that are stored. The entitlement of the requester is not checked: a delete by modelUniqueId is accepted from an NF that does not own the record.

- Result: Fails (1 attack); answers what is asked: partly; usable with caveats
- Value high: The delete attack shows that the procedures which change a record have no entitlement check, which is the property this target asks for.
- Stated in the documents: unclear. Source text it rests on: s0025, s0027, s0059, s0071, s0176
- Rests on: The storeTransId is not reachable by another NF in the model, so the same missing check is only shown on the delete path.; Requests and responses are authenticated as one TLS session per pair of endpoints.
- Open: Whether an accepted PUT can concern a record the requester does not own is not shown by a trace.
- Next step: A query that lets another NF learn a record URI would settle the update claim.

A08: Record fields bound to the storer

ADRFReturning or saving records,modelUniqueId 、 mlFileAddr 、 allowConsumerListWith StorernfInstanceId/nfSetIdThe correspondence must be consistent with the value submitted by the requesting party and authorized; Retrieving a return record by store-trans-id or model-unique-ids must be marked in accordance with the request.

The network cannot substitute a field of a served record, so the binding holds against the network. A second NF that stores the same modelUniqueId can be served instead of the model's own storer, and a consumer then accepts a file address its storer never gave.

- Result: Fails (1 attack); answers what is asked: partly; usable with caveats
- Value high: It shows a consumer of a model being handed the file address of another storer, which the documents neither state nor forbid, and the cause is a rule they give only for the same storer storing again.
- Stated in the documents: unclear. Source text it rests on: s0014, s0044, s0047, s0221
- Rests on: Two records for one modelUniqueId coexist in the table, and a retrieval by identifier may return either; the source does not say which.; The consumer a lists the second storer's record as allowed for itself only because that record's allowConsumerList names a, which the attacker can do freely.
- Open: Which record a retrieval returns when several share a modelUniqueId is left open by the source and by the model.
- Next step: A model in which retrieval is bound to the storer of the identifier would settle whether the ambiguity can be closed by the ADRF alone.

R03: One storage or delete act per request

The same logical storage or deletion operation (the same requester, the same model identifier, the same request content) produces at most one record creation or one deletion side effect on ADRF; Re-transmitting returns the cache result, but does not create a record or delete it again.

The ADRF creates a record for every accepted storage request, and a replayed request is accepted again, so one storage act yields two records and two storeTransIds.

- Result: Attack holds (stated in the documents); answers what is asked: yes; usable with caveats
- Value low: The documents state this behaviour themselves in the NOTE of s0014, so the result only reproduces it with the exact conditions.
- Stated in the documents: yes. Where the documents say it: s0010, s0011, s0014
- Rests on: Replay is possible in the model; TLS in a real deployment would stop a network attacker from replaying the request, so this is the case of a repeated or retried request rather than of an attacker on the wire.

S03: Updated record not overwritten by an old version

For the same storeTransId, the updated state of record should not be covered by the old version; After the deletion has entered into force, the old record may not be re-trievable through the re-establishment of the old request or the old marking.

An update replaces the record: after it takes effect the ADRF honours the new allowConsumerList only, so the consumer the update removed is refused and the old list is not a second way in.

- Result: Holds within a bound (the changing value unrolled to 6 steps); answers what is asked: yes; usable with caveats
- Value medium: It confirms that the update replaces the record as the procedure says, within the six steps modelled; the interesting race is left open.
- Stated in the documents: no. Source text it rests on: s0025, s0027, s0049
- Rests on: The record is unrolled to six steps with one writer and one update.; The ADRF takes the values from the PUT body, as s0176 says, and keeps them as the record.
- Open: A race of two writers on one record is not modelled.

- Next step: Unroll two updates in opposite orders to show whether a race leaves an older version in place.

S05: Deleted record not reachable by old identifiers

When the deletion enters into force, the search, update, delete requests from old storeTransId or old modelUniqueId may no longer return or modify the original model records; Old markings may not be re-entitled.

After a successful delete the record is gone: a later retrieval with the same identifier is refused, both for the network and for a token holder.

- Result: Holds within a bound (the changing value unrolled to 6 steps); answers what is asked: yes; usable with caveats
- Value medium: It shows the delete leaves nothing retrievable in the unrolled model, which the documents require but never state as a check.
- Stated in the documents: no. Source text it rests on: s0060, s0072
- Rests on: The ADRF chain is unrolled to six steps and the retrieval after the delete is the sixth.; The delete is modelled as the record's absence, not as a change to a stored row.
- Open: Retrieval after a delete when the same modelUniqueId was stored again by another NF.
- Next step: A GET by storeTransId of a deleted record, and a GET by a modelUniqueId that a second storer stored again, would cover the remaining identifiers.

U01: Old file address and allow list not accepted

Update entry point, oldmlFileAddrWith the oldallowConsumerListThey must no longer be accepted as valid data.

Once the update takes effect the ADRF serves the new mlFileAddr only; the old address is no longer returned for the identifier.

- Result: Holds within a bound (the changing value unrolled to 6 steps); answers what is asked: yes; usable with caveats
- Value medium: It confirms that the replacement the PUT prescribes takes effect at the served value, in the unrolled bound.
- Stated in the documents: no. Source text it rests on: s0027, s0047, s0176
- Rests on: One record, one update, six steps; the new values are handed to the next step only after the PUT is accepted.
- Open: A retrieval that races the update is not modelled.

U02: Two storers, one model identifier

Two different legal storagesNFWith the samemodelUniqueIdWhen you store it,ADRFThe conduct must fall within a branch permitted by the norm or be recorded as a branch of the explicit hypothesis and not be covered silently.

Two records for the same modelUniqueId, owned by two different NFs, coexist in the ADRF and either can be served for the identifier. The specification allows the further record for the same storer and says nothing of a different storer.

- Result: Fails (1 attack); answers what is asked: partly; usable with caveats
- Value medium: It shows mechanically that the text leaves the case open, which is what an editor needs, but it is not a statement about a specified behaviour.
- Stated in the documents: unclear. Source text it rests on: s0010, s0011, s0014, s0218

- Rests on: The note of s0014 is read as permission to create a further record whenever the ADRF is asked again.; The retrieval takes any record matching the identifier.
- Open: The source does not settle it, so no model can: the answer is a question for the specification.
- Next step: A clarification from the editors, or a rule that modelUniqueId is unique per storer, would settle it.

A08.9: Unique identifier bound to record and file

ADRFIn the records kept and returned,modelUniqueId, StorernfInstanceId/nfSetId、storeTransIdandmlFileAddrThe correspondence must be consistent with the value submitted by the person requesting it and authorized.

The correspondence survives the network: what a consumer accepts for a model identifier was served to it by the ADRF. It does not survive a second storer: the identifier, the record and the file address separate, and the consumer receives the other storer's address.

- Result: Fails (1 attack); answers what is asked: partly; usable with caveats
- Value high: The three identifiers the target names stop agreeing with each other through a rule the documents give only for a storer storing its own model again.
- Stated in the documents: unclear. Source text it rests on: s0011, s0014, s0015, s0044
- Rests on: Responses are authenticated as one TLS session per pair of endpoints.; Two records for one modelUniqueId may coexist and the retrieval picks one.
- Open: Whether the same substitution happens through a retrieval by storeTransId depends on a URI another NF cannot learn in the model.

Appendix B: Known results reproduced

A property that fails as the documents already state, or that they set aside, is not a finding. What follows is of that kind: the analysis reproduces it on the model, which gives the exact conditions.

B.1 A replayed storage request creates a second record

Kind	Weakness of the protocol as specified
Stated in the documents	place not given
Value	low
Properties	R03 (section 3)
Source text	s0010, s0011, s0014
Shown by	ProVerif attack query `r03`

What happens. A replayed POST to /mlmodel-store-records is accepted again and the ADRF creates a second Individual ADRF ML Model Store Record with a new storeTransId. The NOTE of s0014 states that a new record and a new storeTransId are assigned when the models are already stored.

What the attacker needs. The attacker has the network and replays a storage request it observed; the request is a plain HTTPS request body, so no token of its own is needed.

What the attacker gains. The same models are stored twice and two records exist for one storage act; a delete of one record leaves the other, so the storer believes the model is gone while it is still stored.

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	checked		2 proved
Z03	proved		z03m	checked	z03d (holds_in_source)	1 proved, 1 attack
Z05	proved		z03m	checked	z03d (holds_in_source)	1 proved, 1 attack
D01	proved		d01v	checked	z03d (holds_in_source)	1 proved, 1 attack
A08	proved		a08r	checked	a08 (depends_on_ambiguity)	1 proved, 1 attack
R03	none				r03 (holds_in_source)	1 attack
S03	bounded	the changing value unrolled to 6 steps	s03b	checked		1 bounded
S05	bounded	the changing value unrolled to 6 steps	s05b	checked		2 bounded
A08.9	proved		a08r	checked	a08 (depends_on_ambiguity)	1 proved, 1 attack
U01	bounded	the changing value unrolled to 6 steps	u01b	checked		1 bounded
U02	none				u02 (depends_on_ambiguity)	1 attack

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	Every 200 OK that carries an ML model record is sent only after the ADRF checked that the requester is in that model's allowConsumerList or is the NF of the NWDAF containing MTLF that stored it; the single service scope alone never yields a record.	`network`	proved	z01	checked
Z01	Same claim with the attacker holding a valid OAuth2 token for the single service scope nadrf-mlmodelmanagement: a token holder that is neither in the allow list nor the storer obtains no record.	`malicious_nf`	proved	z01	checked
Z03	A retrieval permission is bound to the model: an NF that holds a token and is allowed for m1 obtains no record of m2 unless it stored m2 itself, while its permitted retrieval of m1 succeeds.	`malicious_nf`	proved	z03m	checked
Z03	A permission for one model does not extend to the delete operation: an NF that is only in the allowConsumerList of m1 cannot delete the stored model.	`malicious_nf`	attack (holds_in_sourc e)	z03d	
Z05	A malicious legitimate NF cannot retrieve a model that is neither in its allowConsumerList nor stored by it; its own permitted retrieval of m1 succeeds.	`malicious_nf`	proved	z03m	checked
Z05	A malicious legitimate NF does not delete ML model records that are neither in its allowConsumerList nor stored by it.	`malicious_nf`	attack (holds_in_sourc e)	z03d	
D01	Every accepted update follows a request that carried a valid token for the service scope and was protected by the TLS session towards the ADRF, so the stored modelUniqueId, mlFileAddr and allowConsumerList are the values that requesting NF submitted and no third party substituted them.	`malicious_nf`	proved	d01v	checked

target	claim	attacker	answer	query	control
D01	An accepted delete request corresponds to a record that the requesting NF is entitled to change: it is not enough to hold a token and name a modelUniqueId.	malicious_nf	attack (holds_in_source)	z03d	
A08	The record representation a consumer accepts for a modelUniqueId can only be one the ADRF served to that consumer; the network cannot substitute modelUniqueId, mlFileAddr or allowConsumerList in transit.	network	proved	a08r	checked
A08	The mlFileAddr a consumer receives for a modelUniqueId is the address supplied by that model's storer.	malicious_nf	attack (depends_on_ambiguity)	a08	
R03	One logical storage request creates one record: a replayed POST does not create a second record with a new storeTransId.	network	attack (holds_in_source)	r03	
S03	After an update the record holds the new version only: an NF that the update removed from the allowConsumerList is refused, so the old allow list does not survive as a second basis for access.	malicious_nf	bounded (the changing value unrolled to 6 steps)	s03b	checked
S05	After a successful delete the removed ML model record is no longer returned for the old storeTransId or modelUniqueId: a later retrieval with the same identifier is refused.	network	bounded (the changing value unrolled to 6 steps)	s05b	checked
S05	Same claim when the attacker holds a valid service-scope token and asks again with the identifier of the deleted record.	malicious_nf	bounded (the changing value unrolled to 6 steps)	s05b	checked
A08.9	Neither modelUniqueId, storeTransId nor mlFileAddr can be substituted by the network in a record served to a consumer; the correspondence a consumer sees is the one the ADRF stored.	network	proved	a08r	checked
A08.9	modelUniqueId and mlFileAddr keep the correspondence the authorized storer gave them, so the address retrieved for a model is the one its storer supplied.	malicious_nf	attack (depends_on_ambiguity)	a08	
U01	After the update takes effect, a retrieval of the model returns the new mlFileAddr only; the old address is no longer served.	malicious_nf	bounded (the changing value unrolled to 6 steps)	u01b	checked
U02	When two legitimate storing NFs use the same modelUniqueId, the ADRF's behaviour falls in a branch the specification allows and neither storer's record is silently replaced.	malicious_nf	attack (depends_on_ambiguity)	u02	

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-20261115T092542, 2026-11-15. Status: finished. Model: deepseek-v4-1-flash-261017. Steps: 57, prover 20 s, wall 14 min, cost 0.114 USD.

ProVerif: 68 proofs run, 67 decided, 1 without an answer, 8 s, 22 answers recorded.

What was analysed

Protocol	ADRF ML model update delete profile
Version	19.7.0
Analysed	Expand the scope of the first round only store/retrieve: single model record creation, updateURLand/or allow consumers, access and deletion; The only identification and storage of a fixed modelNF, limited status; DistinctionMLModel metadata transfers with the actual model file. Could not automatically grant cross-owner writing privileges with service scope.
Not analysed	Symbolic models do not support the calculation of algorithms as safe or as resistant to quantum security; The trajectory of the attack is established only under clear assumptions and is not directly equivalent to a normative loophole. Each nature must be subject to accessibility/proportional controls. FixedRel-19。 Distinguishing between an outside cyber attacker and an ill-intentioned and legitimate person holding his own evidence.NF； TLSAuthenticate encryption modeling by endpoint, validNFCan not open messageNFIt's...TLSSession.OAuth2Subject/udience/scope/expiry andTLS(a) The identity is bound in accordance with the attached baseline; Service scope does not automatically equal ownership of resources. Single-modules cannot simulate real deletions, and the life cycle of a limited state must be clearly defined. Targets are specified.APISubprocess, not complete 5G secure.
Source text	298 numbered passages (163580 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	4.3.2.2	Nadrf_MLModelManagement_StorageRequest service operation
main	4.3.2.2.1	General
main	4.3.2.2.2	Request Storage of ML model(s)
main	4.3.2.2.3	Update Storage of ML model(s)
main	4.3.2.3	Nadrf_MLModelManagement_RetrievalRequest service operation
main	4.3.2.3.1	General
main	4.3.2.3.2	Request and get stored ML model(s) from ADRF ML Model Store
main	4.3.2.4	Nadrf_MLModelManagement_Delete service operation
main	4.3.2.4.1	General
main	4.3.2.4.2	Requesting removal of stored ML model(s)
main	4.3.2.4.3	Requesting removal of stored ML model(s) using unique ML model identifier
main	5.2	Nadrf_MLModelManagement Service API
main	5.2.1	Introduction

Document	Section	Title
main	5.2.2	Usage of HTTP
main	5.2.2.1	General
main	5.2.2.2	HTTP standard headers
main	5.2.2.2.1	General
main	5.2.2.2.2	Content type
main	5.2.2.3	HTTP custom headers
main	5.2.3	Resources
main	5.2.3.1	Overview
main	5.2.3.2	Resource: ADRF ML Model Store Records
main	5.2.3.2.1	Description
main	5.2.3.2.2	Resource Definition
main	5.2.3.2.3	Resource Standard Methods
main	5.2.3.2.3.1	POST
main	5.2.3.2.3.2	GET
main	5.2.3.2.4	Resource Custom Operations
main	5.2.3.3	Resource: Individual ADRF ML Model Store Record
main	5.2.3.3.1	Description
main	5.2.3.3.2	Resource Definition
main	5.2.3.3.3	Resource Standard Methods
main	5.2.3.3.3.1	DELETE
main	5.2.3.3.3.2	PUT
main	5.2.3.3.4	Resource Custom Operations
main	5.2.4	Custom Operations without associated resources
main	5.2.4.1	Overview
main	5.2.4.4	Operation: remove-stored-mlmodel
main	5.2.4.4.1	Description
main	5.2.4.4.2	Operation Definition
main	5.2.5	Notifications
main	5.2.6	Data Model
main	5.2.6.1	General
main	5.2.6.2	Structured data types
main	5.2.6.2.1	Introduction
main	5.2.6.2.2	Type: NadrfMLModelStoreRecord
main	5.2.6.2.3	Type: MLModelInfo
main	5.2.6.2.4	Type: MLModel
main	5.2.6.2.5	Type: MLModelDelResult

Document	Section	Title
main	5.2.6.2.6	Type: AllowedConsumer
main	5.2.6.2.7	Type: ModelStoreResult
main	5.2.6.3	Simple data types and enumerations
main	5.2.6.3.2	Enumeration: StoreResult
main	5.2.6.4	Data types describing alternative data types or combinations of data types
main	5.2.7	Error Handling
main	5.2.7.1	General
main	5.2.7.2	Protocol Errors
main	5.2.7.3	Application Errors
main	5.2.8	Feature negotiation
main	5.2.9	Security
main	5.2.6.3.1	

The scope of the protocol as the source step listed it in the documents: expand the scope of the first round only store/retrieve: create and update single model recordsURLand/or allow consumers, access and deletion; The only identification and storage of a fixed modelNF, limited status; DistinctionMLModel metadata transfers with the actual model file. Could not automatically grant cross-owner writing privileges with service scope.

The properties against the documents

Before a query was written, 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	Retrieval needs an authorizatio n basis	The ADRF (NF service producer) is assured to hand a stored ML model record only to a consumer that is in that model's allowConsumerList or is the NF of the NWDAF containing MTLF that stored it; the model record is kept from every other NF.	s0049 states that if the consumer is not in the allowed NF consumer list and/or is not the NF that stored the model, the ADRF shall answer 403 RETRIEVAL_ML_MODEL_NOT_ALLOW ED; s0047 defines the 200 OK body with mlFileAddr.	main:1364, s0044, s0047, s0049, s0221, s0230
Z03	Retrieval permission item-bound	The ADRF is assured that a retrieval permission is bound to one modelUniqueId, one consumer identity and the retrieval operation: a permission for one model is not usable for another model, for another consumer, or for update and delete of that record.	s0049 binds the retrieval permission to the model and the consumer; s0221 and s0230 put allowConsumerList per MLModelInfo and give it as nfInstanceId/nfSetId. The documents say nothing of such a binding for the update and delete operations.	s0049, s0218, s0221, s0230, s0261
Z05	Malicious NF kept from other owners' records	A malicious but legitimate NF service consumer holding its own OAuth2 credentials for the service scope must not retrieve, update or delete ML model records that are neither in its allowConsumerList nor stored by it, and must not read another NF's TLS session.	s0049 requires the allow-list / storing-NF condition for retrieval; s0261 gives one scope for the whole service and no resource-level scope; TS 29.501 requires that such a design not grant inappropriate write or delete access to undue NF service consumers.	ts-29-501:1801, s0049, s0221, s0224, s0258, s0261

ID	Name	About	What the documents say	Where
D01	Update and delete of an authorized record	The ADRF and the storing NWDAF are assured that every accepted PUT, DELETE or remove-stored-mlmodel request concerns a record that exists and that the requesting NF is entitled to change, and that the stored modelUniqueId, mlFileAddr and allowConsumerList are the values the entitled requester submitted.	s0024, s0058, s0071 and s0072 describe the update and delete procedures without any authorization check; s0176 says the PUT body replaces the record. The documents state no ownership rule for these operations.	s0024, s0025, s0058, s0071, s0072, s0176
A08	Record fields bound to the storer	The consumer is assured that the modelUniqueId, mlFileAddr, allowConsumerList and storing nfInstanceId/nfSetId it receives for a model are the values written by the storer it asked about, and that a record returned for a requested identifier belongs to that identifier.	s0047 says the response carries MLModelInfo with modelUniqueId and mlFileAddr; s0218/s0221 bind the fields into one record; s0014 allows a second record for a model already stored without saying what a retrieval then returns.	s0044, s0047, s0135, s0176, s0014, s0218, s0221
R03	One storage or delete act per request	The ADRF is assured that one logical storage or delete request creates one record or one deletion, also when the request is replayed on the SBI.	s0010 and s0011 say the ADRF creates a record and assigns a storeTransId per received POST; s0014 states that a new record and a new storeTransId are assigned even when the model is already stored.	s0010, s0011, s0014, s0058, s0072
S03	Updated record not overwritten by an old version	The ADRF and the storing NWDAF are assured that after an update the record holds the new version only, so an old version cannot become the current one again.	s0025 to s0030 make the PUT update the record; the documents do not state a version or ordering rule for concurrent updates.	s0025, s0027, s0176
S05	Deleted record not reachable by old identifiers	The ADRF and the storing NWDAF are assured that after a successful delete the old storeTransId and the old modelUniqueId no longer return or modify the removed ML model record.	s0060 and s0072 make a successful delete remove the record and the models; the documents do not state what a later request with the old identifier returns.	s0060, s0072, s0044, s0063
A08.9	Unique identifier bound to record and file	The consumer is assured that modelUniqueId, storeTransId and mlFileAddr keep the correspondence the authorized storer gave them, so the file address retrieved for a model is the one the storer of that model supplied.	s0021 makes modelUniqueId and mlFileAddr mandatory fields of MLModelInfo; s0015 says the Location header carries the record URI with storeTransId; s0014 allows duplicate records for one model.	s0011, s0014, s0015, s0044, s0218, s0221
U01	Old file address and allow list not accepted	The consumer and the ADRF are assured that after an update the old mlFileAddr and the old allowConsumerList are no longer accepted as the record's data.	s0027 makes the PUT update the record and s0047 says a retrieval returns the mlFileAddr stored in the ADRF; the documents state no rule for a retrieval that races an update.	s0024, s0027, s0047, s0176, s0221
U02	Two storers, one model identifier	The ADRF's behaviour when two legitimate storing NFs use the same modelUniqueId must fall in a branch the specification allows, or be recorded as an explicit assumption, and must not silently replace one storer's record.	s0014 covers only a model already stored or being stored, without distinguishing storers; s0218 gives nfInstanceId/nfSetId of the storing NWDAF per record and says nothing of an identifier already used by another storer.	s0010, s0011, s0014, s0218, s0221

The analyst added no property: the drafted list covers what the documents state.

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: 54. Not taken: 344. 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 P010	rfc6749:2778-2850	Z05, `malicious_nf`	
P011 to P015	rfc6749:2859-2881	Z01, Z05, `malicious_nf`	
P016 to P033	rfc6749:2885-3003	Outside the scope	Refresh tokens, authorization codes, redirection URIs and the resource owner password grant belong to OAuth2 grants this service API does not use: it uses the client credentials grant only (main 5.2.9).
P034 to P035	rfc6749:3013-3019	No goal stated	Advice on keeping credentials out of request URIs; the API carries its credentials in headers and this states no property of the message flow.
P036 to P062	rfc6749:3023-3210	Outside the scope	Endpoint authenticity is given by TLS to the NRF and the ADRF (baseline TS 33.501/29.500), phishing, CSRF, clickjacking, code injection, open redirectors and the implicit grant concern a browser-based client this API has not.
P063	ts-33-501:2491-2491	Outside the scope	Bidding down of cipher suites and algorithms is a property of the primitives and of the TLS profile, outside the modelled API sub-flow.
P064 to P072	ts-33-501:2495-2511	Z01, Z05, `malicious_nf`	
P073 to P078	ts-33-501:2515-2529	No goal stated	Requirements on key hierarchies and on the general 5GC security architecture; they state nothing that must hold or can fail in this API flow.
P079 to P213	ts-33-501:2533-2853	Outside the scope	Over-the-air, gNB, ng-eNB, subscription credential, subscriber privacy, AUSF and UDR requirements concern access-network and subscriber security, a part of the system the task leaves out.
P214	ts-33-501:2859-2859	Outside the scope	Trust boundaries between SBA security domains (SEPP, roaming) are not part of the intra-PLMN ML model management flow modelled here.
P215 to P227	ts-33-501:2865-2891	Z01, Z05, `malicious_nf`	
P228 to P241	ts-33-501:2895-2923	Outside the scope	NEF, SCP and message-forwarding requirements concern functions and intermediaries that are not parties of this API's message flow; the task models the endpoints only.
P242 to P254	ts-33-501:2929-2953	Z01, Z05, `network`	
P255 to P303	ts-33-501:2957-3055	Outside the scope	SEPP and roaming intermediary protection concern inter-PLMN signalling, outside the intra-PLMN profile modelled here.
P304 to P314	ts-33-501:3059-3081	Outside the scope	Protection of attributes at the SEPP and IPUPS filtering concern functions the task leaves out.
P315 to P318	ts-33-501:3085-3091	Z05, `malicious_nf`	

Paragraphs	Where	Taken up as	Why not
P319 to P327	ts-33-501:3097-3115	Outside the scope	Signalling monitoring and its configuration concern monitoring functions the task leaves out.
P328 to P337	ts-33-501:3121-3141	No goal stated	Security visibility and configurability are operator requirements; they state no property of this protocol that could hold or fail.
P338 to P369	ts-33-501:3149-3215	The prover cannot be asked	The strength and identifiers of ciphering and integrity algorithms are properties of the primitives; the symbolic model proves neither computational nor quantum resistance.
P370 to P398	ts-33-501:3219-11901	Outside the scope	5G-RG, NSSAAF, N3, EAP-TLS annexes, IAB, MBSF and NEF requirements concern other functions and interfaces.

The parties and the protected values, one by one

The host listed the 24 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: 13. Without: 11. 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 Consumers of services	party	`malicious_nf`	
S02	ADRF	party	Everything lost by definition	The ADRF holds every record and performs every authorization check; with it controlled, each property asked here is lost by definition and the documents ask none of them under a compromised ADRF.
S03	NWDAFHamMTLF	party	`malicious_nf`	
S04	NRF	party	Everything lost by definition	The NRF issues the access tokens; with it controlled every token-based property is lost by definition.
S05	Maliciously legal NF service consumers	party	`malicious_nf`	
S06	Integrity NF service consumers	party	`malicious_nf`	
S07	NWDAFHamMLProvider of model documents	party	Outside the scope	The download of the ML model file from its address (s0012, s0026) is outside the modelled sub-flow.
S08	Storage NF A	party	`malicious_nf`	
S09	Storage NFB	party	`malicious_nf`	
S10	ML Model records	value	`malicious_nf`	
S11	allowConsumerList	value	`malicious_nf`	
S12	OAuth2 Access tokens and services	value	`malicious_nf`	
S13	modelUniqueId	value	Nothing gained	modelUniqueId is a Uinteger carried by every record and returned to allowed consumers (s0047); it is an identifier, not a protected value.
S14	allowConsumerList Entry	value	Nothing gained	AllowConsumer list entries are NF instance or set identifiers, public identities of NFs.
S15	Storer nfInstanceId/nfSetId	value	Nothing gained	The storing NF's nfInstanceId/nfSetId is a public NF identity.
S16	Services	value	`malicious_nf`	

ID	Party or value	Kind	Attacker	Why none
S17	ML model logs for other owners	value	`malicious_nf`	
S18	TLS Sessions	value	Everything lost by definition	If the attacker holds another NF's TLS session, every confidentiality and authentication property of that exchange is lost by definition; the scope states a legitimate NF cannot read another NF's TLS session, and no document asks a property under that.
S19	storeTransId	value	Nothing gained	storeTransId is the resource identifier of the record URI (s0015); the documents make it no secret, so knowing it gives nothing the network attacker lacks.
S20	mlFileAddr	value	`malicious_nf`	
S21	ML Model File	value	Outside the scope	The ML model file itself is downloaded from its address outside the modelled messages.
S22	nfInstanceId/nfSetId	value	Nothing gained	nfInstanceId/nfSetId are public NF identities used in the record and in the allow list.
S23	Delete Result	value	Nothing gained	The delete result only reports whether a model was found and deleted; it protects nothing.
S24	Record Status	value	`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`	a valid OAuth2 access token for the single service scope nadrf-mlmodelmanagement, obtained as a registered NF service consumer; it cannot read other NFs' TLS sessions	The API defines one service-level scope only (s0261) and TS 29.501 declares that such a scope grants access to all service operations to every consumer allowed to access the service (ts-29-501:1801); a registered consumer is therefore an attacker with a token.	ts-29-501:1801, s0258, s0259, s0261

ID	`network`	`malicious_nf`
Z01	Holds	Holds
Z03		Fails
Z05		Fails
D01		Fails
A08	Holds	Fails
R03	Fails	
S03		Holds
S05	Holds	Holds
A08.9	Holds	Fails

ID	network	malicious_nf
U01		Holds
U02		Fails

Notes on the prover and the model

- D01: answered in ProVerif, and the analyst names changing_state for this target
- A08: answered in ProVerif, and the analyst names changing_state for this target
- R03: answered in ProVerif, and the analyst names changing_state for this target
- S03: answered in ProVerif, and the analyst names changing_state for this target
- S05: answered in ProVerif, and the analyst names changing_state for this target
- A08.9: answered in ProVerif, and the analyst names changing_state for this target
- U01: answered in ProVerif, and the analyst names changing_state for this target
- U02: 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
the stored ML model record of one storeTransId: modelUniqueId, mlFileAddr, mlStorageSize and allowConsumerList per model, and the storing nfInstanceId/nfSetId	ADRF (NF service producer)	created on the POST (s0010, s0011); replaced by the values of the PUT body on the PUT (s0027, s0176); removed on the DELETE by storeTransId (s0060) or on remove-stored-mlmodel by modelUniqueId (s0072)	on every GET against the allowConsumerList and the storing NF (s0049); on PUT and DELETE only that a record with that storeTransId exists (s0025, s0059), with no owner or allow-list check prescribed	s0010, s0011, s0027, s0049, s0059, s0060, s0072
storeTransId, the identifier that the storing NF keeps and addresses later requests with	ADRF, which assigns it, and the NF service consumer that stored the record, which keeps it from the Location header and the 201 body	a new value on every accepted POST (s0011, s0015)	on PUT, DELETE and GET by store-trans-id as the resource path or query parameter (s0024, s0058, s0044)	s0011, s0015, s0044, s0058

pattern	result	target	shown by ProVerif	steps
replay	candidate	R03	no: not tried with ProVerif	The network attacker records the POST to /mlmodel-store-records sent by the storing NWDAF a. / It delivers the same request again to the ADRF. / The ADRF creates a second Individual ADRF ML Model Store Record and assigns a new storeTransId, so the same models are stored twice and two records exist for one storage act.
repeat_one_time_action	candidate	R03	no: not tried with ProVerif	a stores the model and receives storeTransId T. / The attacker replays a's POST, the ADRF assigns T2 and stores the model a second time. / The one-time act of installing the model under a transaction identifier happens twice, and the two records are independent: a later DELETE of T leaves the model stored under T2.
desynchronisation	nothing found			The record state lives in the ADRF only; the storing NF keeps no copy of mlFileAddr or allowConsumerList that it compares with the ADRF, so no exchange can leave the two with different values.
update_before_check	nothing found			The PUT procedure downloads the models and then updates the record (s0026, s0027); the identity and scope of the requester are taken from the token and the TLS session before the store is touched. bounded.pv hands the new values to the next step only after the PUT was received, so no value changes before the message is verified.
no_rollback_on_failure	candidate		no: not tried with ProVerif	a POSTs a record whose mlFileAddr does not resolve; the ADRF creates the record and assigns storeTransId first (s0010, s0011) and downloads the models afterwards (s0012). / The download fails, so the ADRF answers 500 with ML_MODEL_FILE_DOWNLOAD_FAILED (s0018). / The source does not say that the already created record is removed, so a record can remain whose ML model file was never stored, and the Location header already handed its URI to a.
race_between_sessions	candidate	S03	no: not tried with ProVerif	a PUTs the record T with the new mlFileAddr and allowConsumerList {a}. / A second NF that holds the service-scope token PUTs the same T with the values it read before, or with its own values; the source prescribes no ordering, version or owner check for the PUT (s0173 lists no query parameter, s0025 to s0027 no comparison). / The later PUT replaces the record, so the earlier update is lost and the record can show an older version than the last write.
distinguishable_failure	candidate		no: not tried with ProVerif	An NF holding the service-scope token but in no allow list guesses modelUniqueId values in a GET. / For an identifier that exists it receives 403 RETRIEVAL_ML_MODEL_NOT_ALLOWED (s0049); for one that does not it receives 204 No Content (s0050). / The difference tells the NF which modelUniqueIds are stored in the ADRF, although it is allowed to see none of them, and the identifiers are Uintegers and therefore enumerable (s0221).
wrap_or_window	nothing found			storeTransId is an opaque string assigned by the ADRF and modelUniqueId is a Uinteger with no acceptance window; no comparison of an ordered value is prescribed, so nothing can wrap.

What the analyst named for targets:

step	targets	features	reason
39	A08, A08.9, D01, R03, S03, S05, U01, U02	changing_state	These targets speak about the state of a stored record, which changes: a POST creates it, a PUT replaces mlFileAddr and allowConsumerList, a DELETE removes it, and a second storer can create a further record for the same modelUniqueId. The monotonic table of model.pv cannot express replacement or removal, so the answers on that file only reach the situation; the answers about the old value surviving an update, about access after a removal from the allow list and about retrieval after a delete come from bounded.pv, where the record is unrolled to six steps with the new value handed to the next step.

The analyst's closing note

Model of the Nadrf_MLModelManagement profile: a consumer (NWDAF containing MTLF) stores a record with two models, updates, retrieves and deletes it; a second legitimate NF with its own token for the single service scope is in the allow list of one model only. Requests and responses are protected as one TLS session per pair of endpoints; the ADRF is one process per operation over a record table, and bounded.pv unrolls the record to six steps.

All 11 targets are answered under every attacker set for them. Proved: retrieval always follows the allow-list/storer check (Z01, denial control), a permission is bound to its model (Z03), a malicious NF gets no model it may not read (Z05), an accepted update follows the requester's own token (D01), and the network cannot substitute the fields of a served record (A08, A08.9). Attacks: a permitted reader deletes the storer's model with remove-stored-mlmodel (Z03, Z05, D01; lemma z03d; holds in the source, no entitlement check is prescribed); a second storer registers another mlFileAddr for an already stored modelUniqueId and the retrieval serves it to the original storer's consumer (A08, A08.9; lemma a08; depends on the ambiguity of s0014); two records for one modelUniqueId coexist (U02); a replayed storage request creates a second record (R03, stated in s0014). Bounded, each with a falsified control: only the new mlFileAddr after the update (U01), the consumer the update removed is refused (S03), nothing is served after the delete (S05). The recommendation to require the storing NF for PUT, DELETE and remove-stored-mlmodel was checked: z03d verified, normal run kept, all 13 other answers unchanged; the recommendation on modelUniqueId uniqueness is stated but unchecked.

Blocking the rest: another NF cannot reach a storeTransId in the model, so the update path is not shown by an attack; two concurrent writers, the 403-versus-204 probe that reveals existing identifiers, and the record left behind when a model download fails were worked through but not confirmed by a query; the ML model file download is outside the model.