Security analysis of NWDAF model provision: Subscribe → Notify → Unsubscribe

This report in another language: 中文.

19.8.0 · ProVerif+ Agent report

Documents	5G System; Network Data Analytics Services; Stage 3 (protocol definition)
Properties examined	10: 3 hold, 7 fail (8 attacks)
Findings	4

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 Nnwdaf_MLModelProvision service of the 5G core lets an NF service consumer — an NWDAF containing an analytics function, another NWDAF containing a model training function, or the LMF — subscribe at an NWDAF containing the Model Training Logical Function and be told where trained ML model files are published. The consumer sends a subscription naming the events it wants and the URI at which it will receive notifications. The producer answers with the identifier of the created subscription and later posts a notification carrying that identifier, the event and the address of a model file.

When the exchange ends, the consumer holds a subscription identifier and a model address for each subscribed event, and the producer holds the notifUri and the events to notify. The consumer's only assurance is that the notification repeats the subscription identifier it was given. Nothing in the flow binds the notification to the producer, and the API's authorization is a single scope that covers every subscription and every operation.

The parties:

- **NF service consumer**: NWDAF containing AnLF, NWDAF containing MTLF, or LMF. Holds: identity, notifUri, OAuth2 token.
- **NWDAF containing MTLF**: the model producer. Holds: trained ML model files, NRF signing key.
- **NRF**: authorization server. Holds: client registrations, the nnwdaf-mlmodelprovision scope.

Figure 1 shows the exchange as the documents specify it.

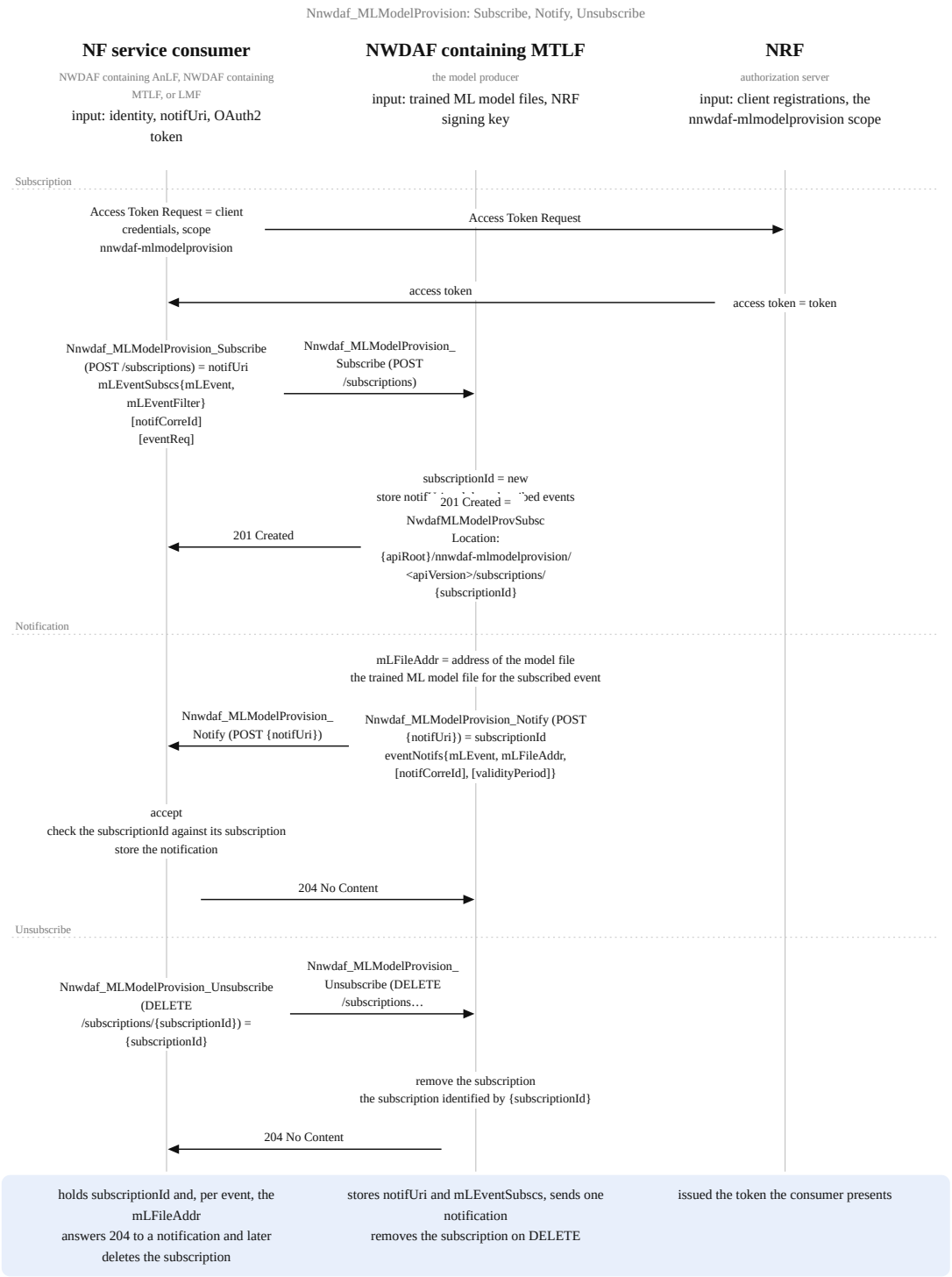


Figure 1. The protocol as specified.

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 4 findings: what follows is not stated in the documents. For 1 of them it is unclear whether the documents state it.

The API does not give what a reader of the flow would assume. The notification carries no origin check, no freshness value and no binding of its payload. An NF on the service based interface can therefore deliver a notification the consumer accepts with a model address of its own, replay a recorded one, or have one stored after the subscription was deleted. The authorization is a single service-wide scope, so any authorized NF can delete any other consumer's subscription. ProVerif shows both central weaknesses, and each is backed by a checked change that removes it. Of the ten properties, four fail for an attacker the network really gives, two are only reachable through the model's unprotected response channel, one restates what the documents already say, and the rest hold only in the reading in which the consumer checks the subscription identifier. The results rest on a model that gives the attacker the whole service based interface and does not model TLS, OAuth2 token validation or the model file itself.

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 Any NF on the SBI can deliver a notification the consumer accepts, with any model address

Kind	Weakness of the protocol as specified
New	Yes: the documents do not state it
Value	high
Properties	A03, A03.9, D01, A08 (section 3)
Shown by	ProVerif attack queries `a03`, `a08`; Figures 2, 3, 4 and 5

What happens. The Notify operation is an HTTP POST that the consumer accepts on its notifUri. The consumer compares the subscriptionId with the one it holds, but nothing identifies the sender and nothing binds the event identifier or the mLFileAddr. An NF that observed the subscriptionId can therefore post a notification with a model address, an event identifier and a validity period of its choice, and the consumer stores it and answers 204. The model shows the accepted notification with no send of the MTLF behind it.

What the attacker needs. A position on the service based interface; no token and no key. The subscriptionId is read from the 201 Created or from the notification itself, and the notifUri from the Subscribe request.

What the attacker gains. The consumer is directed to an ML model of the attacker's choice and will use it for analytics or positioning, although the NWDAF containing MTLF never published that model. The flow supplies an address only, so nothing downstream detects the substitution.

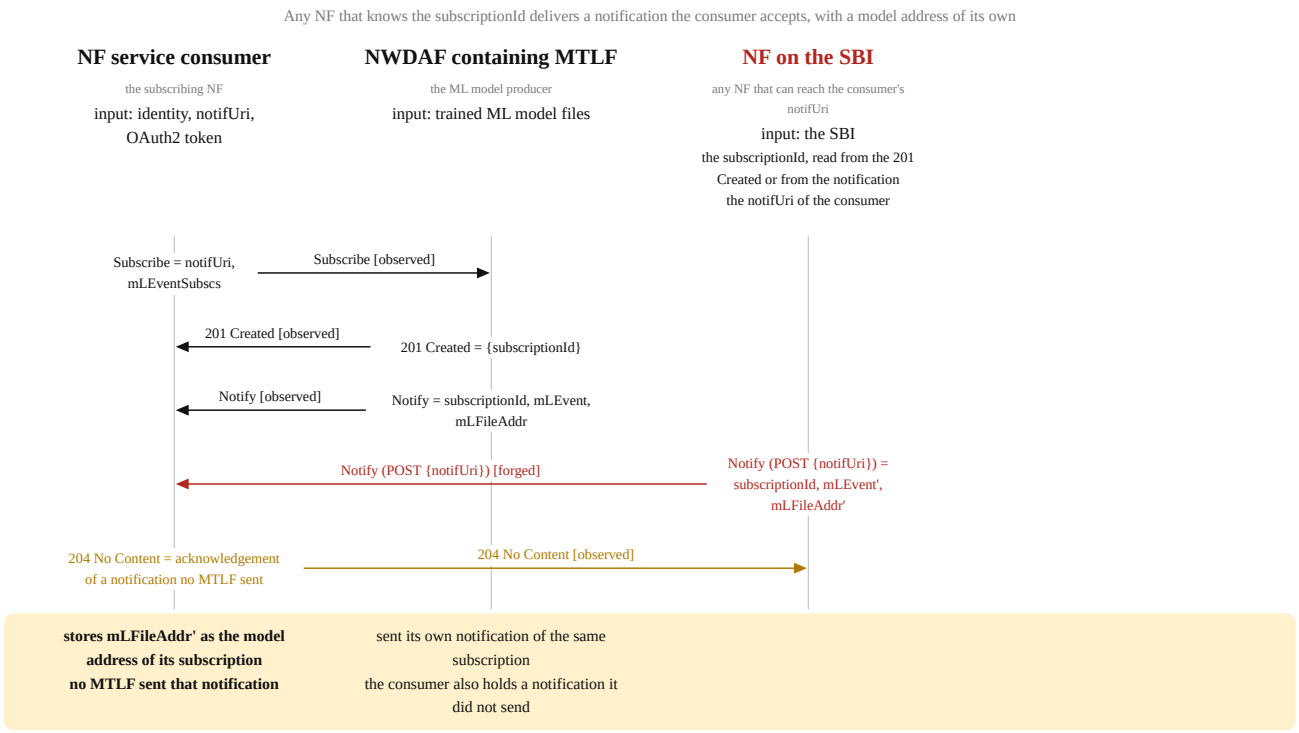


Figure 2. Any NF that knows the subscriptionId delivers a notification the consumer accepts, with a model address of its own. Property A03; attack query a03.

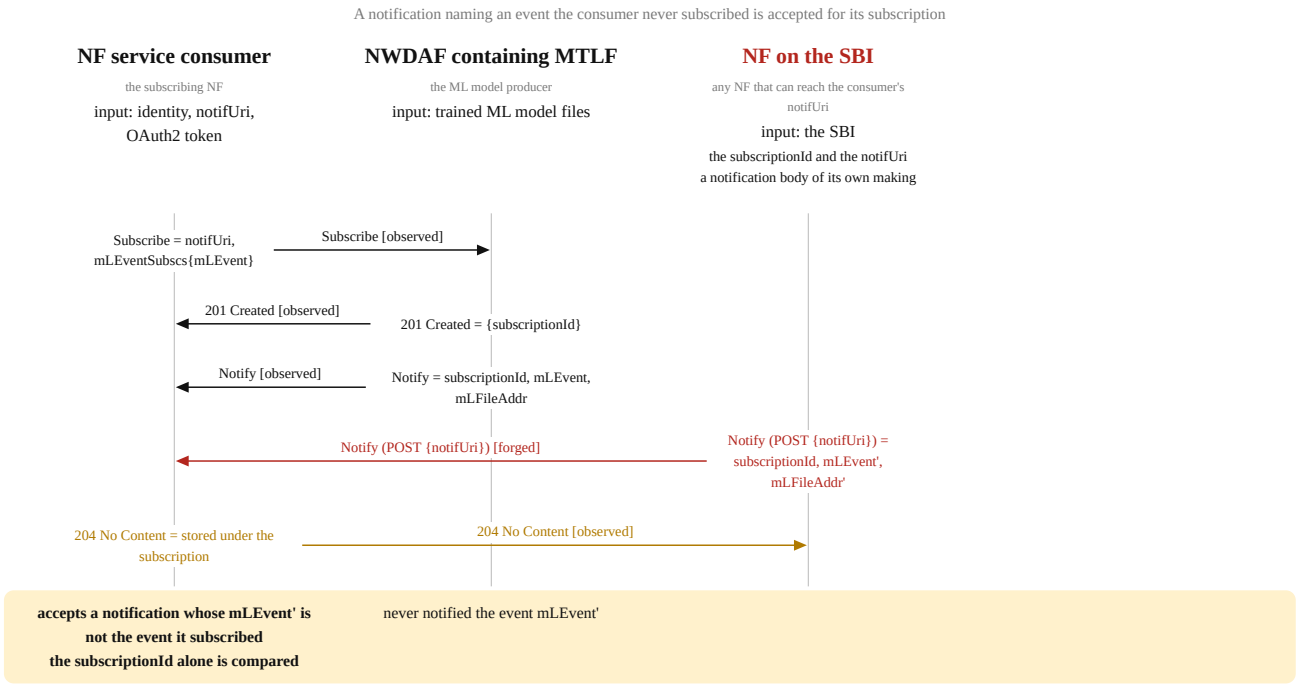


Figure 3. A notification naming an event the consumer never subscribed is accepted for its subscription. Property A08; attack query a08.

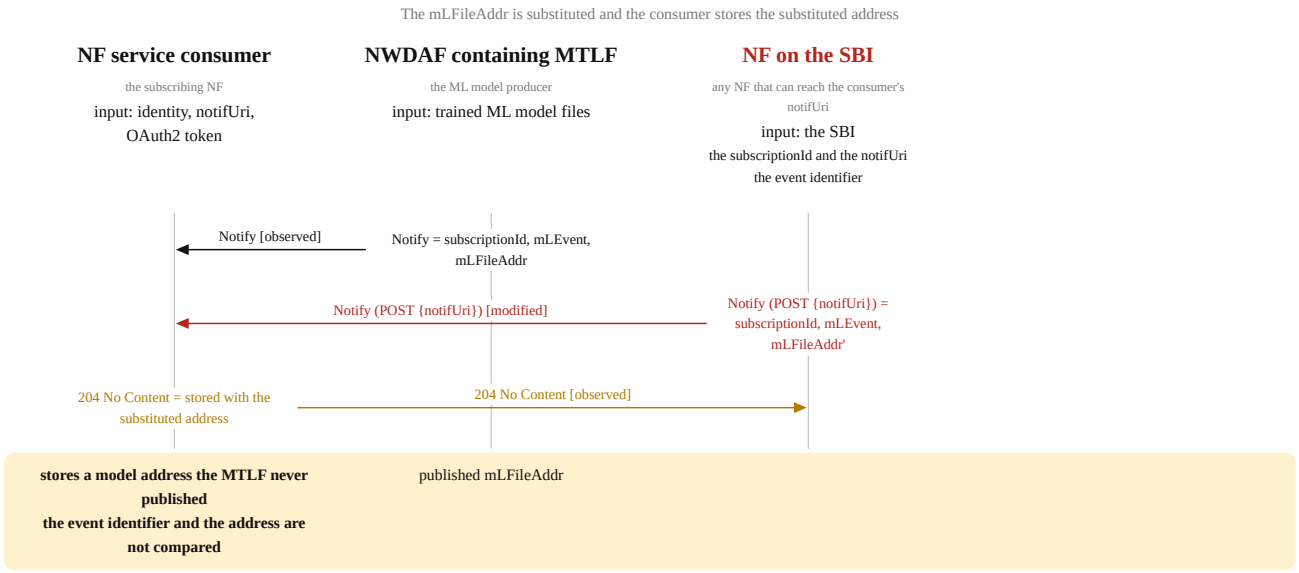


Figure 4. The mLFileAddr is substituted and the consumer stores the substituted address. Property D01; attack query a03.

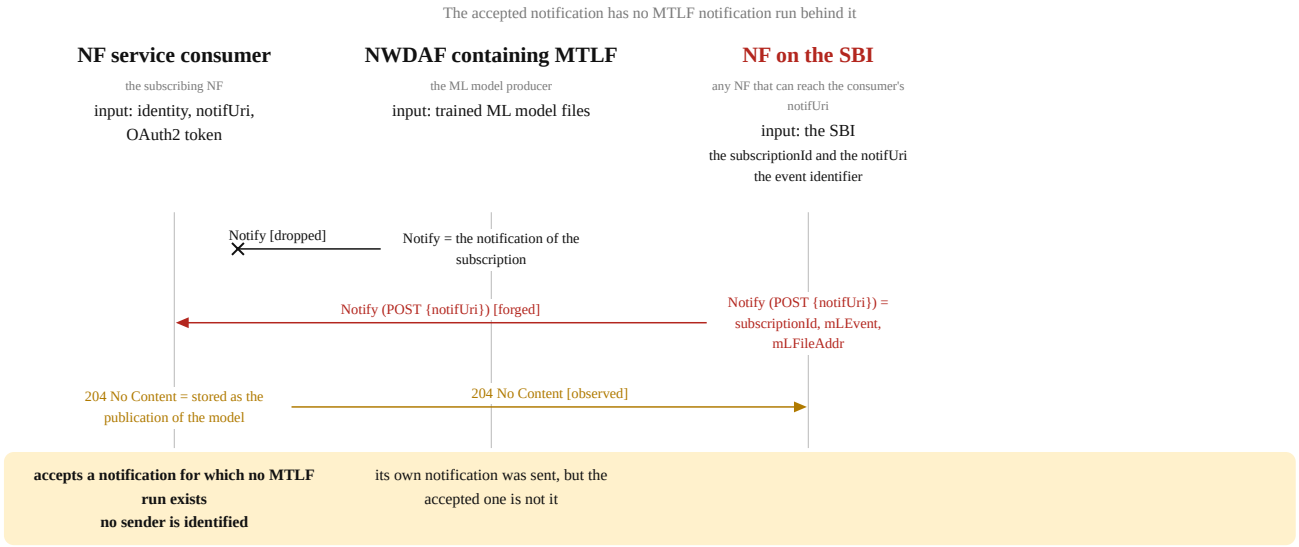


Figure 5. The accepted notification has no MTLF notification run behind it. Property A03.9; attack query a03.

Proposed change. Have the NWDAF containing MTLF sign the notification body (subscriptionId, notifUri, event identifier, mLFileAddr) and have the consumer verify that signature against the MTLF it subscribed at before storing the notification and answering 204. Checked on a model of this change: the query for the correspondence between an accepted notification and a MTLF send is verified, the normal run is still found and the other recorded answers are unchanged. Signing the body without the notifUri is not enough: the check then still accepts a notification meant for another recipient.

Check of the change with ProVerif. The change was put into a copy of the model: The Nnwdaf_MLModelProvision_Notify request carries a signature by the NWDAF containing MTLF over (subscriptionId, notifUri, event identifier, mLFileAddr), and the NF service consumer verifies that signature against the MTLF it subscribed at before it stores the notification and answers 204, rejecting it otherwise. The notifUri is inside the signed body so a notification cannot be redirected to another consumer. On the changed model ProVerif proves `a03`: the attack is no longer possible there. The normal run of the protocol is still found. The other 9 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/0044-fix-signed.pv`.

2.2 A token for the service deletes any consumer's subscription

Kind	Weakness of the protocol as specified
New	Unclear: the documents state it in part, or could not be searched
Value	high
Properties	Z03 (section 3)
Shown by	ProVerif attack query `z03`; Figure 6

What happens. The API defines a single OAuth2 scope for the whole service and no scopes at resource or operation level, and the DELETE addresses the subscription only by {subscriptionId} in the path. The NWDAF therefore removes a subscription for any NF that presents a token of the service, whether or not that NF created the subscription. The model shows a subscription created with one consumer's token and removed with a token issued for another.

What the attacker needs. An OAuth2 access token for the nnwdaf-mlmodelprovision scope, which any authorized NF service consumer holds, and the subscriptionId of the victim, which travels in the clear in the Location header and in every notification.

What the attacker gains. Any authorized NF can terminate another consumer's ML model subscription, and with it the flow of model updates that consumer relies on. The victim learns only that no further notifications arrive.

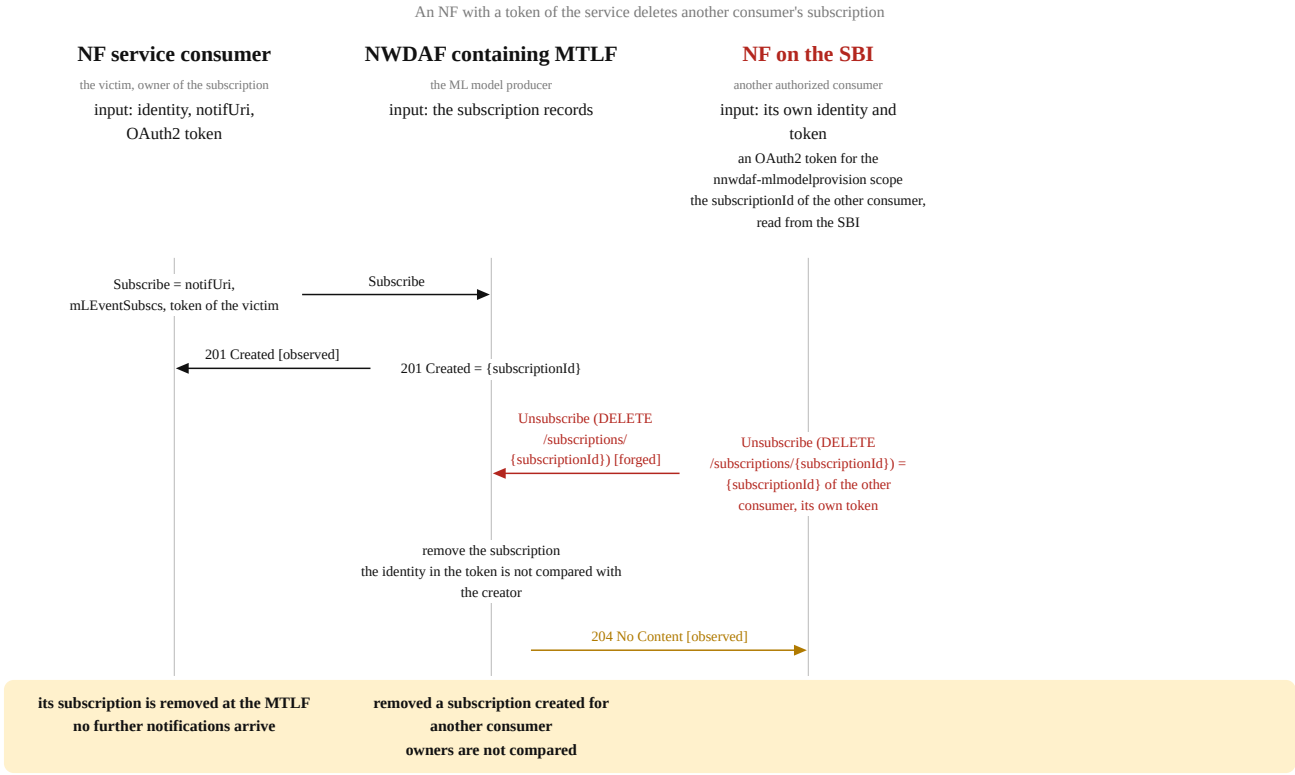


Figure 6. An NF with a token of the service deletes another consumer's subscription. Property Z03; attack query z03.

Proposed change. Have the NWDAF containing MTLF authorize the DELETE against the owner of the subscription: compare the NF identity in the presented token with the identity that created that subscription, and answer 403 when they differ. Checked on a model of this change: the query that an unsubscribe is accepted only from the subscription's owner is verified, the normal run is still found and the other recorded answers are unchanged.

Check of the change with ProVerif. The change was put into a copy of the model: The NWDAF containing MTLF authorizes the DELETE against the owner of the subscription: it compares the NF identity in the presented token with the identity that created that subscription (the one that sent the creating POST), instead of accepting any token that carries the nnwdaf-mlmodelprovision scope. On the changed model ProVerif proves `z03`: the attack is no longer possible there. The normal run of the protocol is still found. The other 11 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/0035-fix-owner.pv`.

2.3 A recorded notification can be replayed and accepted as a new one

Kind	Weakness of the protocol as specified
New	Yes: the documents do not state it
Value	medium
Properties	A04 (section 3)
Shown by	ProVerif attack query `a04`; Figure 7

What happens. NwdafMLModelProvNotif carries no sequence number, nonce or time stamp that the consumer checks, and notifCorreId is optional and identical for every notification of a subscription. A notification recorded on the SBI can therefore be posted to the same notifUri again and is stored as a second, new notification.

What the attacker needs. A position on the service based interface and one notification observed in transit.

What the attacker gains. The consumer acts on the same model publication repeatedly, and any notification-driven step it performs (a model download, a re-training trigger) is repeated at a time the MTLF did not choose.

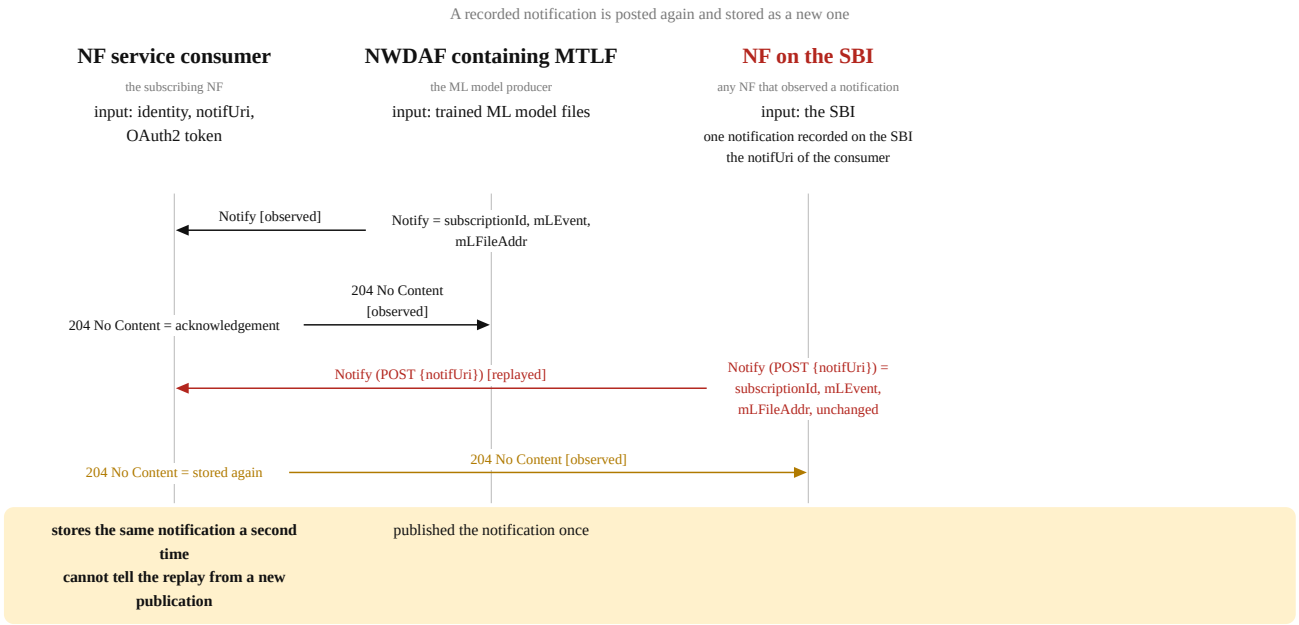


Figure 7. A recorded notification is posted again and stored as a new one. Property A04; attack query a04.

Proposed change. Add a value generated by the NWDAF containing MTLF for each notification, unique per subscription (a notification sequence number or a nonce), to NwdafMLModelProvNotif, and have the consumer reject a notification whose value it has already accepted.

Check of the change with ProVerif. Not checked: The change adds a new attribute to NwdafMLModelProvNotif, which is outside the fields the model carries; it was not built and run within this analysis.

2.4 A notification is still stored after the consumer unsubscribed

Kind	Weakness of the protocol as specified
New	Yes: the documents do not state it
Value	medium
Properties	S05 (section 3)
Shown by	ProVerif attack queries `s05`, `s05b`; Figures 8 and 9

What happens. The DELETE removes the subscription at the NWDAF containing MTLF and is acknowledged with 204, but the consumer keeps its notifUri open and the flow gives it no value with which to refuse a notification for a subscription it has removed. With the consumer unrolled to one notification before the unsubscribe and one after the acknowledged DELETE, ProVerif reaches the second acceptance and the notification is stored.

What the attacker needs. A position on the service based interface, one notification observed before the unsubscribe and the subscriptionId that the consumer used.

What the attacker gains. The consumer stores and uses a model publication after it asked to stop receiving them, and the two parties hold different states: the provider has no subscription, the consumer still acts on the identifier.

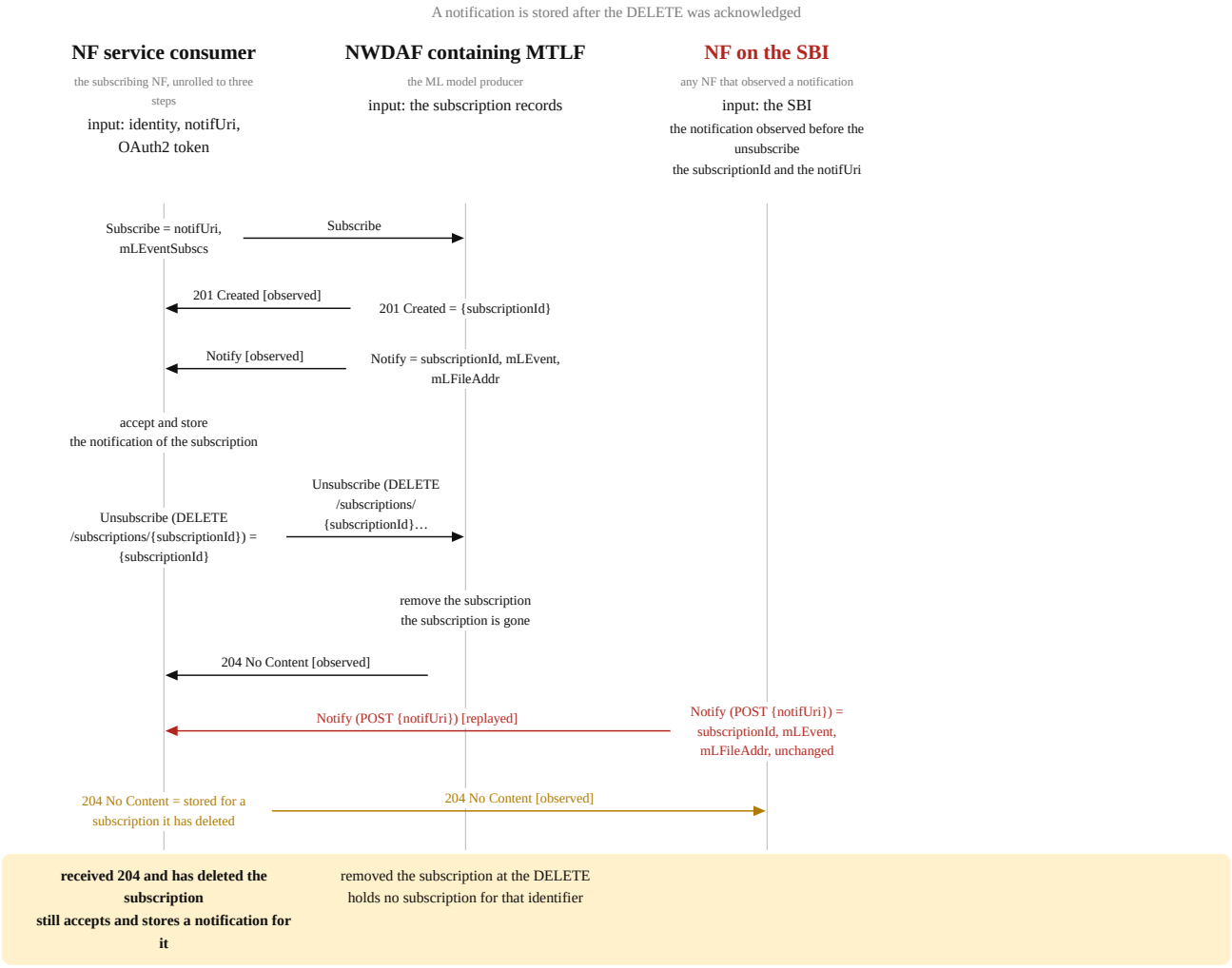


Figure 8. A notification is stored after the DELETE was acknowledged. Property S05; attack query s05b.

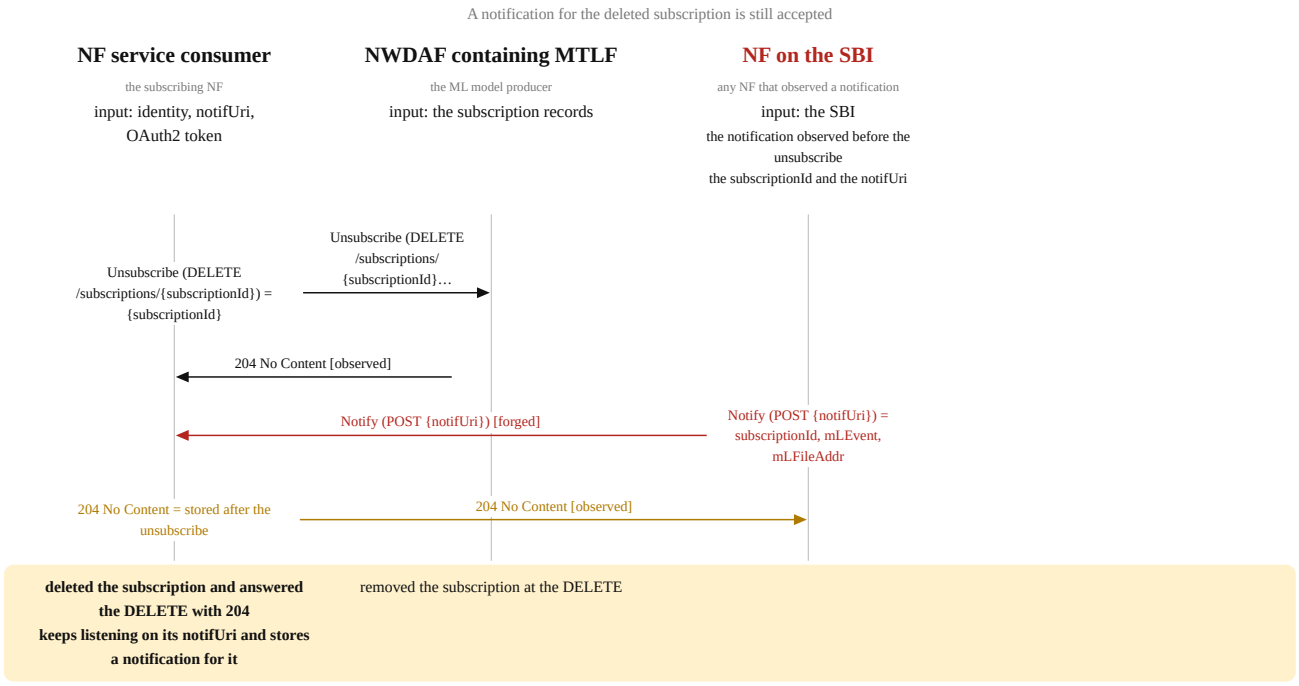


Figure 9. A notification for the deleted subscription is still accepted. Property S05; attack query s05.

Proposed change. Have the consumer keep the subscriptionId of a subscription whose DELETE it has seen acknowledged and refuse any notification carrying it, answering an error instead of 204.

Check of the change with ProVerif. Not checked: The change adds a kept value and a check to the consumer; the bounded model already unrolls the consumer, but the variant was not built and run within this analysis.

No figure was given for these attacks of a finding: A03.9: a08, A03: a08, A08: a03, D01: 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
A03	**Notification matches a provider send.** Every time the consumer accepts a notification, there is a matching provider run: the run is triggered by the consumer's earlier subscription request, and the subscriptionId and event identifier in the notification match the values the consumer gave in the subscription.	no compromise	Fails (1 attack)	high	no
A04	**No notification accepted twice.** Every acceptance of a notification by the consumer for the same subscription corresponds to a distinct notification send run by the provider; the same notification cannot be accepted again as a new one.	no compromise	Fails (1 attack)	medium	no
A08	**Notification fields match own subscription.** In a notification the consumer accepts, the subscriptionId, event identifier and fields such as mLFileAddr correspond to the notifUri and mLEventSubscs the consumer gave in the subscription, and have not been replaced with the values of another subscription or another event.	no compromise	Fails (1 attack)	medium	no

ID	Property	Compromise assumed	Result	Value	In the documents
D01	**Notification payload is authentic.** Every notification the consumer accepts corresponds to a real send by the subscribed provider, and the notification payload (subscriptionId, eventNotifs, mLFileAddr) is the same as at send time.	no compromise	Fails (1 attack)	high	no
Z01	**Operations require a valid token.** Protected operations such as subscription creation, update, deletion and notification sending are accepted only when the requester holds a valid token issued by the authorization server that covers the nnwdaf-mlmodelprovision scope.	no compromise	Holds	low	yes
Z03	**Token is bound to its resource.** An authorization obtained for one subscription or one operation cannot be used for another subscription, another consumer identity or another protected operation.	no compromise	Fails (1 attack)	high	unclear
S05	**No notification after unsubscribe.** After the provider has successfully processed an unsubscribe (deleted the subscription), the consumer no longer accepts new notifications carrying that subscriptionId.	no compromise	Fails (2 attacks)	medium	no
A08.8	**Notification belongs to own subscription.** In a notification the consumer accepts, the subscriptionId, notifUri and event identifier correspond to a subscription it created itself, not to another subscription or another event.	no compromise	Holds	low	unclear
A03.9	**Accepted notification has a real send.** Every time the consumer accepts a notification, there is a real notification run by the subscribed provider, and the subscriptionId and event match.	no compromise	Fails (1 attack)	medium	no
U01	**No notification for a failed subscription.** After a subscription creation fails, the consumer does not accept notifications carrying that subscriptionId; every notification it accepts corresponds to a successful subscription creation.	no compromise	Holds	low	no

The questions asked, and the analyst's answers:

Question	Answer	Rests on
Q1: association of subscriptionId with creator, notifUri and subscription events	The consumer does bind a notification to a subscription identifier it holds itself, and that part holds for any number of sessions. The association is not protected beyond that: an NF on the service based interface can make the consumer believe it created a subscription it never created and can send a notification for it, and an event identifier or model address different from the ones subscribed is accepted.	A08.8, A08, A03
Q2: whether an accepted notification corresponds to a legitimate provider send event for this subscription	No. A notification is an HTTP POST to the consumer's notifUri and the flow carries no origin check, so a notification the consumer accepts need not have been sent by the MTLF at all; a recorded one can also be replayed as new.	A03.9, A03, A04
Q3: event path that distinguishes failed from successful subscription creation before a notification	The failure responses 403 and 500 are distinct, an honest producer sends no notification after them, and the model confirms that a consumer accepts a notification only for a subscription identifier it received in a creation response. What the model cannot settle is whether that response really came from the MTLF, because it puts the request and the response on one unprotected channel.	U01, A08.8

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 service based interface is modelled as one channel the attacker has. Transport protection (TLS, TS 29.500 clause 13.1) is outside the analysed text and is not modelled; an attack that needs the attacker to answer a request the consumer itself sent (a forged 201 Created) is an artifact of the model and is marked as such.
- The model has no NF identities and the NRF issues a token to whoever asks over the service based interface, so the network attacker and an authorized NF are the same attacker in the model; the answers hold for the weaker of the two readings.
- OAuth2 token issuance, validation and client authentication are represented by one signed token carrying an identity and the single service scope; the detail of TS 29.510, 29.500 and 33.501 is not modelled.
- The model file itself is not modelled: only the mLFileAddr is carried. Content, hash and download integrity are outside the task's boundary and are placed elsewhere by TS 23.288 clause 6.2A.0.
- Only the flow of one consumer with one NWDAF containing MTLF is modelled. ADRF, federated learning, model sharing, positioning, roaming, the PUT update, the redirect responses 307 and 308, expiryTime and the problem details of failures are left out.
- The subscription state is a value that changes; model.pv leaves its check out and bounded.pv unrolls the consumer to three steps. What is shown there holds for those steps only, and the state review records what lies outside them.

What the host records:

- The answers of S05 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.

A03: Notification matches a provider send

Every time the consumer accepts a notification, a matching provider run exists: that run is triggered by the consumer's earlier subscription request, and the subscriptionId and event identifier in the notification match the values the consumer gave in the subscription.

The consumer accepts a notification that no MTLF sent. An NF on the service based interface that knows the subscriptionId can post a notification with any event identifier and any model address to the consumer's notifUri, and the consumer stores it and answers 204.

- Result: Fails (1 attack); answers what is asked: yes; usable
- Value high: An unprotected acceptance path for the value the whole service exists to deliver; the documents state no such weakness and no such check.
- Stated in the documents: no
- Rests on: The service based interface is one channel the attacker has, and the model gives it no transport protection.; The consumer is modelled as comparing the subscriptionId of the notification with the one it holds and nothing else; the text does not say what it compares.
- Open: Whether a notification endpoint is required to check the identity of the sender is not stated; the documents rely on the transport layer, which authenticates a peer as some NF and not as the subscribed MTLF.
- Next step: Confirm with the editors whether a notification endpoint is expected to check the sender against the MTLF of the subscription; if the transport and network configuration are meant to do this, say so in the specification.

A04: No notification accepted twice

Every acceptance of a notification for the same subscription by the consumer corresponds to a distinct notification send run by the provider; the same notification cannot be accepted twice as a new notification.

A notification recorded on the interface can be posted again and is stored as a new notification, because the body carries no value the consumer compares for freshness. Every notification-driven step of the consumer is repeated at a time the producer did not choose.

- Result: Fails (1 attack); answers what is asked: yes; usable with caveats
- Value medium: A real gap in the message design, shown mechanically; the impact depends on what the consumer does with a notification, which the model does not represent.
- Stated in the documents: no
- Rests on: The attacker can read and resend a notification, which the model allows on the single channel.; notifCorreId is optional in the text and is modelled as absent, as the scope requires.
- Open: Whether repeated delivery of the same model publication is acceptable to the consumer is not stated.
- Next step: Add a per-notification unique value and a consumer check, then re-run the injectivity query.

A08: Notification fields match own subscription

For a notification the consumer accepts, the subscriptionId, event identifier, mLFileAddr and other fields correspond to the notifUri and mLEventSubscs the consumer gave in the subscription, and have not been replaced with values of another subscription or another event.

The subscriptionId ties a notification to a subscription, but neither the event identifier nor the model address is checked against the subscription, so a notification that names a different event and a different model address is accepted for that subscription.

- Result: Fails (1 attack); answers what is asked: yes; usable with caveats
- Value medium: The binding the service relies on is only half present: the identifier is checked, the payload is not.
- Stated in the documents: no
- Rests on: The consumer compares only the subscriptionId; this is a modelling choice where the text is silent.; The service based interface is unprotected in the model.
- Open: The text does not say whether the consumer must compare the event identifier and the mLFileAddr with the subscription; the model has it compare only the subscriptionId.
- Next step: State in the specification which fields of the notification the consumer must compare with the subscription, and check the sender.

D01: Notification payload is authentic

Every notification the consumer accepts corresponds to a real send by the subscribed provider, and the notification payload (subscriptionId, eventNotifs, mLFileAddr) is the same as at the time of sending.

The model address in the notification can be replaced on the way to the consumer and the consumer stores the replaced value. The flow provides an address and nothing else, so the consumer cannot tell a model address published by the MTLF from one introduced by an NF.

- Result: Fails (1 attack); answers what is asked: yes; usable
- Value high: The service exists to deliver this address; an unprotected substitution redirects the consumer to a model of the attacker's choosing.
- Stated in the documents: no
- Rests on: No integrity protection is modelled on the notification; the text states none at the API level.; The model file itself is out of scope: only the address is carried.
- Open: Whether the mLFileAddr is meant to be integrity protected at the API layer at all: TS 23.288 clause 6.2A.0 sends the protection of the model file itself to TS 33.501 Annex X.
- Next step: Specify a signature over the notification body and the recipient, as in the checked recommendation.

Z01: Operations require a valid token

Protected operations such as subscription creation, update, deletion and notification sending are accepted only when the requester holds a valid token issued by the authorization server and covering the nnwdaf-mlmodelprovision scope.

The producer accepts a Subscribe or an Unsubscribe only when the request carries a token the authorization server issued for the service scope; a request without such a token is not processed. This is the only protection the flow has, and any NF can obtain such a token.

- Result: Holds; answers what is asked: yes; usable with caveats

- Value low: It confirms what the documents state (s0339 to s0342) and depends entirely on the token model, which is a simplification.
- Stated in the documents: yes. Where the documents say it: main:19218, s0339, s0340, s0342
- Rests on: The NRF issues a token to any requester on the interface, so the model does not distinguish an authorized NF from the network.; The token is modelled as a signed value carrying an identity and the scope; the certificate or client authentication of TS 33.501 is not modelled.
- Open: Token validation detail (expiry, audience, client authentication) is outside the analysed text.
- Next step: Nothing further on this claim; what the token is bound to is answered by Z03.

Z03: Token is bound to its resource

An authorization obtained for one subscription or one operation cannot be used for other subscriptions, other consumer identities or other protected operations.

A token of the service is accepted for any subscription: an NF that holds one deletes a subscription created by another NF. The API has no resource level authorization, so an identifier observed on the interface is enough to terminate another consumer's subscription.

- Result: Fails (1 attack); answers what is asked: yes; usable
- Value high: A weakness of the authorization design that the attacker reaches with a credential the network hands out as a matter of course; the documents state the rule but not the consequence.
- Stated in the documents: unclear
- Rests on: The attacker is an authorized NF that can obtain a token; the model's NRF issues to any requester.; Client authentication and certificate binding of the token to an NF instance are not modelled, though they would bind a token to an identity and not to a subscription.
- Open: Whether the single scope is a deliberate simplification for networks in which all consumers are equally trusted is not stated.
- Next step: Check the ownership check of the recommendation for the PUT as well, and against a consumer that holds a token for a second subscription.

S05: No notification after unsubscribe

After the provider has successfully processed an unsubscribe (deletion of a subscription), the consumer no longer accepts new notifications carrying that subscriptionId.

After the DELETE is acknowledged the producer has no subscription, but the consumer still accepts and stores a notification for that identifier. The two parties end in different states and the consumer acts on a model publication it had asked to stop.

- Result: Fails (2 attacks); answers what is asked: yes; usable with caveats
- Value medium: A state question the text leaves open, shown for the unrolled run; the impact is the continued use of a model after cancellation.
- Stated in the documents: no
- Rests on: The consumer is unrolled to three steps in bounded.pv; the property is shown for those steps.; The late notification is delivered by the attacker on the unprotected interface.
- Open: The text does not say what a consumer does with a notification for a subscription it has deleted.
- Next step: Confirm whether the consumer is expected to keep and enforce the state of a deleted subscription.

A08.8: Notification belongs to own subscription

Among the notifications the consumer accepts, the subscriptionId, notifUri and event identifier correspond to a subscription it created itself, not to another subscription or another event.

A consumer accepts a notification only for a subscriptionId it received in a creation response, and that holds for any number of sessions. It does not follow that the subscription exists at the MTLF: the creation response itself is not protected, so an NF can make the consumer hold an identifier the MTLF never issued.

- Result: Holds; answers what is asked: partly; usable with caveats
- Value low: The part that holds is what any implementation does with an identifier it was given; the part that fails is a transport artifact.
- Stated in the documents: unclear
- Rests on: The model puts the request and the response on one channel, so an answer can be forged; with the transport protection of TS 29.500 the response is bound to the MTLF the consumer addressed.; The consumer is modelled as comparing the subscriptionId and nothing else.
- Open: The reading in which the subscriptionId must belong to a subscription the MTLF created is not settled: the only answer there is an artifact of the model's unprotected request and response.
- Next step: State whether a consumer must correlate a notification with a subscription it created, and protect the creation response.

A03.9: Accepted notification has a real send

Every time the consumer accepts a notification, there is a real notification run of the subscribed provider, and the subscriptionId and event match.

An accepted notification need not correspond to any notification run of the MTLF: an NF on the interface posts a notification for an observed subscriptionId and the consumer accepts it as the publication of its model.

- Result: Fails (1 attack); answers what is asked: yes; usable with caveats
- Value medium: The same missing origin check as A03, stated as correspondence to a real notification run; the impact is the acceptance of a publication that never happened.
- Stated in the documents: no
- Rests on: The service based interface is unprotected in the model and no identity of the sender is checked at the API level.
- Open: Whether the notification endpoint is expected to check the sender is not stated.
- Next step: Specify an origin check for the notification, as in the recommendation for A03.

U01: No notification for a failed subscription

After a subscription creation fails, the consumer does not accept notifications carrying that subscriptionId; every accepted notification corresponds to a successful subscription creation.

A consumer accepts a notification only for a subscription identifier it received in a 201 Created, so a creation it processed as failed never leads to an accepted notification; that holds for any number of sessions. Whether the identifier really belongs to a subscription the MTLF created is the other reading, and there the model only shows an artifact of its unprotected response channel.

- Result: Holds; answers what is asked: partly; usable with caveats

- Value low: The proved part is what an implementation does with a creation response it processed; the failing part is a transport artifact, and the failure paths of the text were not modelled.
- Stated in the documents: no
- Rests on: The failure responses 403 and 500 are not modelled; only the successful creation is.; The attack of the second reading is an artifact of the model's single unprotected channel for requests and responses.
- Open: The failure responses 403 and 500 are not modelled, so the claim is answered for the successful path only; the second reading rests on an artifact of the unprotected response channel.
- Next step: Model the 403 and 500 responses and their effect on the consumer's subscription record.

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
A03	none				a03 (holds_in_source)	1 attack
A04	none				a04 (holds_in_source)	1 attack
A08	none				a08 (holds_in_source)	1 attack
D01	none				a03 (holds_in_source)	1 attack
Z01	proved		z01	checked		2 proved
Z03	none				z03 (holds_in_source)	1 attack
S05	none				s05 (holds_in_source); s05b (holds_in_source)	2 attack
A08.8	proved		a088_own	checked	a088_real (model_artifact)	1 proved, 1 attack
A03.9	none				a03 (holds_in_source)	1 attack
U01	proved		a088_own	checked	a088_real (model_artifact)	2 proved, 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
A03	Every notification the consumer accepts was sent by the MTLF for the subscription it created, with the same subscriptionId, event identifier and model address	`network`	attack (holds_in_source)	a03	
A04	Each accepted notification of a subscription is a distinct notification of the MTLF; a recorded notification cannot be replayed and accepted again as a new one	`network`	attack (holds_in_source)	a04	
A08	The subscriptionId, event identifier and mLFileAddr of an accepted notification belong to the subscription the consumer created for its notifUri and mLEventSubscs, and are not those of another subscription or another event	`network`	attack (holds_in_source)	a08	
D01	The notification the consumer accepts is the one the MTLF sent, with subscriptionId, eventNotifs and mLFileAddr unchanged	`network`	attack (holds_in_source)	a03	
Z01	Claim 1: a Subscribe that the MTLF accepts carries an access token the NRF issued for the nnwdaf-mlmodelprovision scope	`network`	proved	z01	checked
Z01	Claim 2: an Unsubscribe that the MTLF accepts carries an access token the NRF issued for the nnwdaf-mlmodelprovision scope	`network`	proved	z01b	checked
Z03	An authorization obtained for one subscription, one consumer identity or one operation cannot be used for another subscription, another consumer or another operation	`network`	attack (holds_in_source)	z03	
S05	After the MTLF has successfully processed the DELETE that removes a subscription, no further notification carrying that subscriptionId is accepted	`network`	attack (holds_in_source)	s05	

target	claim	attacker	answer	query	control
S05	After the MTLF has successfully processed the DELETE that removes a subscription, no further notification carrying that subscriptionId is accepted (unrolled: one notification before the unsubscribe, one after the acknowledged DELETE)	`sbi_nf`	attack (holds_in_source)	s05b	
A08.8	Reading 2: the subscriptionId in an accepted notification belongs to a subscription the MTLF really created for this consumer, not to a subscription of another consumer or to one that was never created	`network`	attack (model_artifact)	a088_real	
A08.8	Reading 1: a notification the consumer accepts carries a subscriptionId the consumer itself holds from its own subscription creation	`sbi_nf`	proved	a088_own	checked
A03.9	Every accepted notification corresponds to an actual notification run of the MTLF the consumer subscribed at, with the same subscriptionId and event identifier	`network`	attack (holds_in_source)	a03	
U01	Claim: every notification the consumer accepts belongs to a subscription whose creation succeeded; a subscription whose creation returned 403 or 500 never yields an accepted notification	`sbi_nf`	attack (model_artifact)	a088_real	
U01	Reading 1: every notification the consumer accepts belongs to a subscriptionId it received in a 201 Created, that is, to a creation it processed as successful and not to one whose creation returned 403 or 500	`sbi_nf`	proved	a088_own	checked
U01	Reading 1 (network attacker): every notification the consumer accepts belongs to a subscriptionId it received in a 201 Created, that is, to a creation it processed as successful	`network`	proved	a088_own	checked

Appendix D: How the analysis was run

An LLM analyst builds the protocol model from the documents and submits each property to ProVerif. Every result in section 3 is an answer from the prover.

ProVerif+ Agent session session-20261114T182235, 2026-11-14. Status: finished. Model: deepseek-v4-1-flash-261017. Steps: 63, prover 4 s, wall 12 min, cost 0.108 USD.

ProVerif: 100 proofs run, 100 decided, 0 left unanswered, 3 s, 29 answers recorded.

What was analysed

Protocol	NWDAF model provision: Subscribe → Notify → Unsubscribe
Version	19.8.0
Analysed	Fixed at Rel-19; one consumer and one MTLF provider. Direct mLFileAddr notification is used; ADRF, FL, roaming, model sharing and positioning extensions are not enabled. Subscription creation and one notification are verified first; unsubscribe is extended as a separate stage.
Not analysed	This flow provides the model address; it does not guarantee file content or hash integrity, and model download is outside the first-round boundary. The optional notifCorreId or modelUniqueId is not forced to be a mandatory field. Global notification prohibition after unsubscribe, expiry and re-subscription require state unfolding or a tool better suited to state. Only properties under the explicit model and attacker conditions are proved; missing checks, a fully secure channel or all-honest NFs cannot substitute for the conclusions to be proved.
Source text	342 numbered passages (87586 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	4.5	Nnwdaf_MLModelProvision Service
main	4.5.1	Service Description
main	4.5.1.1	Overview
main	4.5.1.2	Service Architecture
main	4.5.1.3	Network Functions
main	4.5.1.3.1	Network Data Analytics Function (NWDAF)
main	4.5.1.3.2	NF Service Consumers
main	4.5.2	Service Operations
main	4.5.2.1	Introduction
main	4.5.2.2	Nnwdaf_MLModelProvision_Subscribe service operation
main	4.5.2.2.1	General
main	4.5.2.2.2	Subscription for event notifications
main	4.5.2.2.3	Update subscription for event notifications
main	4.5.2.3	Nnwdaf_MLModelProvision_Unsubscribe service operation
main	4.5.2.3.1	General

Document	Section	Title
main	4.5.2.3.2	Unsubscribe from event notifications
main	4.5.2.4	Nnwdaf_MLModelProvision_Notify service operation
main	4.5.2.4.1	General
main	4.5.2.4.2	Notification about subscribed event
main	5.4	Nnwdaf_MLModelProvision Service API
main	5.4.1	Introduction
main	5.4.2	Usage of HTTP
main	5.4.2.1	General
main	5.4.2.2	HTTP standard headers
main	5.4.2.2.1	General
main	5.4.2.2.2	Content type
main	5.4.2.3	HTTP custom headers
main	5.4.3	Resources
main	5.4.3.1	Resource Structure
main	5.4.3.2	Resource: NWDAF ML Model Provision Subscriptions
main	5.4.3.2.1	Description
main	5.4.3.2.2	Resource definition
main	5.4.3.2.3	Resource Standard Methods
main	5.4.3.2.3.1	POST
main	5.4.3.2.4	Resource Custom Operations
main	5.4.3.3	Resource: Individual NWDAF ML Model Provision Subscription
main	5.4.3.3.1	Description
main	5.4.3.3.2	Resource definition
main	5.4.3.3.3	Resource Standard Methods
main	5.4.3.3.3.1	PUT
main	5.4.3.3.3.2	DELETE
main	5.4.3.3.4	Resource Custom Operations
main	5.4.4	Custom Operations without associated resources
main	5.4.5	Notifications
main	5.4.5.1	General
main	5.4.5.2	Event Notification
main	5.4.5.2.1	Description
main	5.4.5.2.2	Operation Definition
main	5.4.6	Data Model
main	5.4.6.1	General
main	5.4.6.2	Structured data types

Document	Section	Title
main	5.4.6.2.1	Introduction
main	5.4.6.2.2	Type NwdafMLModelProvSubsc
main	5.4.6.2.3	Type MLEventSubscription
main	5.4.6.2.4	Void
main	5.4.6.2.5	Type NwdafMLModelProvNotif
main	5.4.6.2.6	Type MLEventNotif
main	5.4.6.2.7	Type FailureEventInfoForMLModel
main	5.4.6.2.8	Type MLModelAddr
main	5.4.6.2.9	Void
main	5.4.6.2.10	Void
main	5.4.6.2.11	Type MLRepEventCondition
main	5.4.6.2.12	Type InputDataInfo
main	5.4.6.2.13	Type ModelProvisionParamsExt
main	5.4.6.2.14	Type AdditionalMLModelInformation
main	5.4.6.2.15	Type MLModelAdrf
main	5.4.6.2.16	Type TrainInputDataInfo
main	5.4.6.2.17	Type InferenceDataForModelTrain
main	5.4.6.2.18	Type VflInfo
main	5.4.6.3	Simple data types and enumerations
main	5.4.6.3.1	Introduction
main	5.4.6.3.2	Simple data types
main	5.4.6.3.3	Enumeration: FailureCode
main	5.4.6.3.4	Enumeration: MLModelMetric
main	5.4.6.3.5	Enumeration: VflTrainingStatus
main	5.4.6.3.6	Enumeration: PositioningCase
main	5.4.7	Error handling
main	5.4.7.1	General
main	5.4.7.2	Protocol Errors
main	5.4.7.3	Application Errors
main	5.4.8	Feature negotiation
main	5.4.9	Security

The purpose of the protocol as the source step located it in the documents: Fixed at Rel-19; one consumer and one MTLF provider. Direct mLFileAddr notification is used; ADRF, FL, roaming, model sharing and positioning extensions are not enabled. Subscription creation and one notification are verified first; unsubscribe is extended as a separate stage.

The properties against the documents

Before writing any 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
A03	Notification matches a provider send	The NF service consumer is assured that every notification it accepts was sent by the NWDAF containing MTLF for the subscription it created, with the subscriptionId and the event identifier it subscribed to.	The notification carries the subscriptionId (s0109) and, per event, the event identifier (s0111); the response to the subscription carries the subscriptionId in the Location header (s0065). The text states the correspondence but no means of checking it.	s0065, s0108, s0109, s0110, s0111, s0123
A04	No notification accepted twice	The consumer treats each notification of a subscription as a distinct notification; a recorded notification cannot be replayed and accepted as a new one.	The text requires the consumer to store the notification and answer 204 (s0123) and announces one-time, periodic or event-detected notification (s0024), but defines no notification sequence number, nonce or any other freshness value in NwdafMLModelProvNotif (s0109, s0110).	s0024, s0109, s0110, s0123
A08	Notification fields match own subscription	The consumer is assured that the subscriptionId, the event identifier and the mLFileAddr in the notification belong to the notifUri and mLEventSubscs it put in its own subscription, and are not those of another subscription or another event.	notifUri identifies the recipient of notifications (s0041), mLEventSubscs the subscribed events (s0042), and the notification repeats the subscriptionId (s0109) and event (s0111); the text does not say what the consumer compares.	s0041, s0042, s0108, s0109, s0111, s0112
D01	Notification payload is authentic	The consumer is assured that the notification it accepts is the one the MTLF sent, with subscriptionId, eventNotifs and mLFileAddr unchanged; the flow supplies a model address and no content or hash integrity.	The notification body is defined field by field in s0108 to s0112 and the consumer stores it (s0123); the selected text states no integrity or origin protection of the notification. TS 29.500 clause 13.1 (line 948) places message protection at the transport layer, outside this text.	s0108, s0109, s0110, s0112, s0123
Z01	Operations require a valid token	Access by an NF service consumer to Subscribe, Update, Unsubscribe and Notify is only granted when the requester holds a valid OAuth2 access token issued by the NRF for the nnwdaf-mlmodelprovision scope.	s0339 states that access 'may be authorized by means of the OAuth2 protocol' with the NRF as authorization server, and s0340 to s0342 give the token request and the single service scope. The token issuance and validation detail is in TS 29.510, 29.500 and 33.501, outside this text.	s0339, s0340, s0341, s0342
Z03	Token is bound to its resource	An authorization obtained for one subscription, one consumer identity or one operation cannot be used for another subscription, another consumer or another operation on the Nnwdaf_MLModelProvision API.	s0342 states that the API defines a single scope for the entire service and 'does not define any additional scopes at resource or operation level'; the DELETE resource is identified only by {subscriptionId} in the URI (s0188, s0192, s0095).	s0342, s0188, s0192, s0095
S05	No notification after unsubscribe	After the NWDAF has successfully processed the DELETE that removes a subscription, no further notification carrying that subscriptionId is acted on, in particular not at the consumer that had the subscription.	s0096 to s0098 require the NWDAF to remove the subscription and answer 204 No Content; the text says nothing about notifications already in flight or about the consumer's own record of a deleted subscription.	s0095, s0096, s0097, s0098, s0109
A08.8	Notification belongs to own subscription	The consumer is assured that a notification it accepts carries the subscriptionId of a subscription it created itself and the event identifier of that subscription, not those of another consumer's subscription.	The Location header returns the subscriptionId to the consumer that created the subscription (s0065) and the notification repeats it (s0109); the text does not require the consumer to compare it against a stored subscription.	s0065, s0108, s0109, s0111

ID	Name	About	What the documents say	Where
A03.9	Accepted notification has a real send	Every notification the consumer accepts corresponds to an actual notification run of the MTLF it subscribed at, with the same subscriptionId and event identifier.	The Notify service operation is invoked by the NWDAF to notify the subscribing consumer instance (s0103, s0108); the consumer stores the notification on receipt (s0123). No origin check is specified.	s0103, s0108, s0109, s0111, s0123
U01	No notification for a failed subscription	A consumer accepts no notification carrying a subscriptionId whose creation failed (403 Forbidden for VFL, 500 Internal Server Error for unavailable ML model), and every accepted notification corresponds to a successful creation.	s0065 makes the 201 Created with the subscriptionId the only successful outcome; s0068 and s0069 give the failure responses. The text does not say how a consumer treats a notification for a subscriptionId it never received.	s0065, s0066, s0068, s0069, s0123

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

The parties and the protected values, one by one

The host listed the 16 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: 4. 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	`sbi_nf`	
S02	NWDAF (containing MTLF) provider	party	Everything lost by definition	The NWDAF containing MTLF is the authority for the model it notifies; if the attacker controls it, the model address and the event it reports are whatever the attacker decides, which is the compromise the notification properties assume away. The selected text asks nothing of the flow under a dishonest provider; the model file itself is protected elsewhere (TS 33.501 Annex X, cited in TS 23.288 clause 6.2A.0).
S03	NRF	party	Everything lost by definition	The NRF is the authorization server (s0339). If the attacker controls it, every token is one it issued, so Z01 and Z03 are lost by definition and no other target speaks about token issuance.
S04	subscriptionId	value	Nothing gained	The subscriptionId travels in the Location header of the 201 response and in every notification (s0065, s0109); the network already reads it, so holding it adds nothing.
S05	notifUri	value	Nothing gained	The notifUri is in the Subscribe request body (s0041) and is read on the SBI; holding it adds nothing to what the network has.
S06	mLEvent event identifier	value	Nothing gained	The mLEvent is in the Subscribe request and in the notification (s0043, s0111); it is public on the SBI.
S07	mLFileAddr	value	Nothing gained	The mLFileAddr is the payload of the notification (s0112) and is read on the SBI; knowing it is exactly what D01 and A08 test for, and D01 already assumes the attacker sees it.
S08	Notification content	value	Nothing gained	The notification content is the mLFileAddr and the event identifier (s0110 to s0112); it is on the SBI, so the network attacker holds it already.
S09	mLEvent	value	Nothing gained	The mLEvent identifier is public on the SBI (s0043, s0111).
S10	Notification payload	value	Nothing gained	The notification payload is the body of an HTTP POST on the SBI (s0108); the network reads it.
S11	OAuth2 access token	value	`sbi_nf`	

ID	Party or value	Kind	Attacker	Why none
S12	nnwdaf-mlmodelprovision scope	value	Nothing gained	The scope name nnwdaf-mlmodelprovision is a fixed public string (s0342); holding it is holding nothing.
S13	Subscription resource	value	`sbi_nf`	
S14	Access token	value	`sbi_nf`	
S15	Subscription state	value	Nothing gained	The subscription state held by the provider (which subscriptionIds exist and the DELETE that removes them, s0064, s0096 to s0098) adds nothing to the subscriptionIds and notifUris the network already reads from the SBI; S05 asks how the state behaves, not who holds it.
S16	Error response	value	Nothing gained	The failure responses 403 and 500 with their problemDetails causes (s0068, s0069) carry no value the network does not already have; no target asks for indistinguishability of failures.

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		
`sbi_nf`	A position on the service based interface as an NF service consumer, and a valid OAuth2 access token for the single scope nnwdaf-mlmodelprovision, obtained from the NRF. It reads, injects, replays and modifies SBI messages and can run Subscribe/Unsubscribe itself.	s0339 makes OAuth2 with the NRF as authorization server the authorization mechanism, and s0342 states that the API defines a single scope for the entire service with no resource or operation level scopes, so any authorized NF holds a token that covers every subscription and every operation.	s0339, s0340, s0342

ID	`network`	`sbi_nf`
A03	Fails	Fails
A04	Fails	Fails
A08	Fails	Fails
D01	Fails	Fails
Z01	Holds	Holds
Z03	Fails	Fails
S05	Fails	Fails
A08.8	Holds	Holds
A03.9	Fails	Fails
U01	Holds	Holds

Source and targets

Source: 342 units, 87586 bytes, frozen in source-delivery-01.json. Targets: 8 approved (A03, A04, A08, D01, Z01, Z03, S05, U01); 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 the source package protocol-8c6f560bedbfa26c000a198dd5f3

This step as it reports itself: 1 requests (1 failed), 0.000 USD, 55291 tokens in and 4125 out, 156 s.

Notes on the prover and the model

- S05: 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.

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 subscription record: subscriptionId, notifUri and the subscribed mLEvent set	NWDAF containing MTLF (provider)	created by the POST on /subscriptions (s0064); modified and stored again by the PUT on /subscriptions/{subscriptionId} (s0077, s0078); removed by the DELETE (s0096, s0097)	when selecting the subscription to notify, the provider uses the notifUri and the events of the stored subscription (s0108, s0110); when a DELETE arrives it looks the subscription up by {subscriptionId} (s0095)	s0064, s0077, s0078, s0095, s0096, s0097, s0100, s0108
the consumer's record of its live subscriptions	NF service consumer	on the 201 Created that returns the Location with {subscriptionId} (s0065); the text does not say whether the 204 of the DELETE (s0098) changes it	not specified: on receipt of a notification the consumer stores it and answers 204 (s0123) with no stated comparison of the subscriptionId against a live subscription	s0065, s0098, s0123
subscriptionId	provider as the resource identifier, consumer as the handle of its subscription	issued by the provider when the resource is created (s0065)	repeated in every notification (s0109) and used as the {subscriptionId} path segment of the DELETE and PUT (s0095, s0075)	s0065, s0095, s0098, s0109

pattern	result	target	shown by ProVerif	steps
replay	candidate	A04	no: not tried with ProVerif	1. The consumer subscribes and the MTLF sends its notification over the SBI (s0108). / 2. The attacker records the notification POST, which carries subscriptionId, event and mLFileAddr and no sequence number, nonce or other freshness value (s0109 to s0112). / 3. The attacker posts the same body to the same notifUri again. / 4. The consumer stores it as a new notification and answers 204 (s0123); nothing in the recorded body tells the second receipt from the first.
desynchronisation	candidate	S05	yes: ProVerif falsifies `s05b` on bounded.pv	1. The consumer subscribes, receives the 201 with {subscriptionId} and accepts one notification (s0065, s0123). / 2. The consumer sends DELETE .../subscriptions/{subscriptionId} and receives 204 No Content; the provider has removed the subscription (s0095 to s0098). / 3. The two parties now hold different states: the provider has no subscription, the consumer has no state to refuse one. / 4. A notification carrying the same subscriptionId is posted to the notifUri; the consumer accepts and stores it (s0123). ProVerif finds this run with the consumer unrolled to three steps.
no_rollback_on_failure	nothing found			A DELETE that the provider does not accept is answered with an error (s0100) and the subscription stays; the text gives no path in which the record is removed and the error returned, so no rollback question arises.
update_before_check	nothing found			The provider stores the subscription only after the POST has been accepted (s0064, s0065), and the PUT modifies it only if the request is successfully processed (s0076 to s0078). The token is checked before the record is created, as s0339 and s0340 require.
repeat_one_time_action	nothing found			A repeated DELETE on a subscription that no longer exists is answered with an HTTP error per clause 5.4.7 (s0100); the text gives no message by which a removal or a creation happens twice.
race_between_sessions	nothing found			Two consumers hold different subscriptionIds issued by the provider (s0065) and the provider looks a subscription up by {subscriptionId} (s0095), so one session's DELETE does not reach another session's record. The value shared between sessions is the token, which is Z03, not a race on the record.
distinguishable_failure	nothing found			The provider does answer different failure causes (403 VFL_MODEL_TO_BE_USED_FOR_ALL_EVENTS and 500 UNAVAILABLE_ML_MODEL_FOR_ALLEVENTS, s0068, s0069, with the problem details of 5.4.7), but no target asks who learns what from them, and the difference reveals nothing about a kept value beyond the request itself.
wrap_or_window	nothing found			subscriptionId has no order and no acceptance window; expiryTime (s0050) is an absolute time, no target rests on it, and the text specifies no wrap or window check.

What the analyst named for targets:

step	targets	features	reason
30	S05	changing_state, many_sessions	S05 asks what happens after the provider has removed a subscription: the subscription state (exists / deleted) is a value that changes across steps, which model.pv leaves out and bounded.pv unrolls. The other targets are about correspondence between notifications and subscriptions over any number of consumer and provider sessions, which ProVerif covers for any number of sessions.

The analyst's closing note

The model runs: model.pv has the NF service consumer, the NWDAF containing MTLF and the NRF with the plain Subscribe -> 201 Created -> Notify -> 204 -> DELETE -> 204 run; bounded.pv unrolls the consumer to three notification steps for the changing subscription state. All ten targets have an answer and every recorded answer was re-checked on the final text.

Established: Z01 is proved for both claims (a Subscribe or an Unsubscribe is accepted only with a token the NRF issued for the service scope), each with a control that fails. A08.8 and U01 are proved in the reading in which the consumer binds a notification to a subscriptionId it received in a 201 Created. Every other target fails: A03, A03.9 and D01 (a notification the consumer accepts with no MTLF send behind it and an arbitrary mLFileAddr), A08 (the event identifier is not compared with the subscription), A04 (a recorded notification is replayed and stored as new), S05 (also on the unrolled model), and Z03 (a token of the service deletes another consumer's subscription). The second reading of A08.8 and U01 fails only through a forged response to the consumer's own request, which the transport layer prevents; it is recorded as a model artifact and is not a finding.

Four findings, two of them with a change checked by the prover: signing the notification body including the notifUri removes the notification finding, and authorizing the DELETE against the subscription's owner removes the authorization finding; in both cases the normal run survives and no other answer changes. The replay and post-unsubscribe findings carry recommendations that were not checked on a model.

Blocking the rest: the model gives the attacker the whole service based interface and does not model TLS, OAuth2 token validation or the model file, so attacks that need the attacker to answer a request the consumer itself sent cannot be separated from transport artifacts; expiry, the PUT update, the redirects and the 403/500 failure paths are not modelled; and the two unchecked recommendations would need their own model runs.