SEMIOTIC LABS

TAP security

March, 2023

Attack vectors

Possible issues/attack vectors and handling

1. Doppelganger Attack

This attack is the result of receipts and RAV’s having exact same structure

- **Attack:**
  - Indexer acts as normal and collects receipts and creates RAV requests (indexer should keep all RAV’s and receipts)
  - Indexer then makes an additional rav request with no previous RAV, instead includes all RAV’s and receipts collected as their list of receipts
- **Solution:**
  - Add a field (byte) with **message_type** (0-receipt, 1-RAV) and sign it along with the rest

2. Ludicrous Attack

This attack is the result of handling the possibility of receipt collisions with re-issue of a receipt

**Duplicate receipt is can be defined as all fields are duplicated or timestamp/nonce value is duplicated, both definitions work for this attack*

- **Attack:**
  - Indexer receives a duplicate receipts from two different gateways (with same operator)
  - Indexer responds by providing each gateway with a copy of the duplicate receipt as proof of duplication and request a re-issue of the receipt

OK

Possible issues/attack vectors and handling

2a. Ludicrous Attack

This attack is the result of handling the possibility of receipt collisions with re-issue of a receipt

**Duplicate receipt is can be defined as all fields are duplicated or timestamp/nonce value is duplicated, both definitions work for this attack*

- **Attack:**
  - Indexer receives a duplicate receipts (R_a, R_b) from two different gateways (with same operator)
  - Indexer responds by providing each gateway with a copy of the duplicate receipt as proof of duplication and request a re-issue of the receipt
  - Each gateway issues a new receipt to the indexer (R_{a*}, R_{b*}), unaware of the other gateway doing the same
  - Indexer then turns in 3 receipts for next RAV, (R_{a*}, R_{b*}) with either R_a or R_b
  - Indexer would then have gained the value contained in either R_a or R_b without providing any service
- Note: the probability of an indexer receiving duplicate receipts is ludicrously low, regardless it is still possible
- **Current* Solution:** (*still open to discussion) Don’t allow receipt re-issues at all

OK

Mar 7, 2023

Possible issues/attack vectors and handling

2b. Modified Ludicrous Attack

This attack is the same as 2a, but does not require the indexer *actually* receiving duplicate receipts

**Duplicate receipt in this case is defined specifically as fields being duplicated*

- **Attack:**
  - Indexer receives a receipt (R_a) from a gateway
  - Indexer responds to gateway claiming it is a duplicate receipt and sends R_a as the claimed duplicate receipt
  - The gateway has no way of knowing if this is a copy of the receipt it sent or the result of another gateway coincidentally sending exact same receipt values (this is a valid scenario, although ludicrously unlikely)
  - Gateway therefore just checks if receipts have duplicated fields and valid receipts, then reissues the receipt (R_{a*})
  - Indexer then turns in both receipts for next RAV, (R_a, R_{a*})
  - Indexer could repeat this continuously (claim R_{a*} is a duplicate to get R_{A**}, and so on)
- **Current* Solution:** (*still open to discussion) Don’t allow receipt re-issues at all

OK

Mar 7, 2023

Possible issues/attack vectors and handling

Discussion for the solution for attack 2a./2b. of not re-issuing receipts

- We still might need to think through a response to the **claim** of a duplicate receipt, otherwise a malicious actor could gain
  - Malicious Indexer could:
    - respond to any query request by refusing to serve the query, claiming they received a duplicate receipt previously
    - Use two copies of the receipt as “proof”
    - Even though they never served the queries the receipt is still valid and they will still get paid
    - There is no way currently to show if they are a bad actor or just unlucky, assuming gateways considered to be stateless
  - If the protocol responded by requiring Indexers to serve received queries regardless of being duplicates (since duplicates are nearly impossible and should never happen)
  - Then a malicious Gateway could purposely send duplicate receipts that must be served, but only one can be claimed

OK

Mar 7, 2023

Possible issues/attack vectors and handling

2. Collision (and replay attack)

Actually there several possible attack vectors

- **Attack:**
  - Gateway freezes the timestamp and value and resends it along with a query to be executed by the Indexer, with the goal of paying only once for all of those served queries
  - Indexer sends the same receipt many times to the Gateway for aggregation
  - …?
- **Solution:**
  - Add Gateway ID (identifying the instance of a gateway)
  - Sign the tuple (gateway ID, datetime, message_type, nounce, value)
  - Duplicates are detected on both sides
    - When a Gateway receives a batch of receipts with even one duplicate:
      - Returns an error to the Indexer
    - When an Indexer will receive a receipt and detect that it is a duplicate
      - Rejects the query?? (can an Indexer even do that??)

OK

Apr 4, 2023

Possible issues/attack vectors and handling

Insecure/evil dependency (both of TAP lib, or any other added by the final executable binary that uses TAP)

- **Attack:**
  - Read/write any memory address of the program
  - Could be triggered by a vulnerability in the dependency
  - Vulnerability/backdoor could be triggered remotely
- **Solution:**
  - Limit # of dependencies
  - Dependencies analysis
    - Github CI action for SBOM analysis / OWASP Dependency-Track **OK**
    - Github dependabot for CVE scans **OK**
    - Manual review of some of the dependencies **OK**

| In Progress |

Implementation Notes [DONE]

Implementation Considerations

Key management: key storage is outside of the scope of the TAP protocol, however the code is assessed to ensure that secret/sensitive information is not inadvertently leaked.

Side-channel attacks: attacks exploiting implementation specific details of cryptographic operations. Typically targeting e.g. signing key recovery. The code is assessed to ensure that all subroutines touching secret/sensitive material were written to provide some protection against these types of attacks.

Cryptographic Subroutines

Requiring RNG	RNG used	Notes
timeline_aggregation_protocol::receipt::Receipt::nonce	rand::thread_rng	On Linux, uses a blocking call to dev/urandom source under the hood, ✅

Cryptographic operations using secret key	Notes
timeline_aggregation_protocol::receipt::Receipt::new()	
timeline_aggregation_protocol::receipt_aggregate_voucher::aggregate_receipt()	

timeline_aggregation_protocol::receipt::Receipt::new()

k256::ecdsa::SiginingKey::sign()	**OK**
ecdsa::SignPrimitive<Secp256k1>::try_sign_prehashed()	

ecdsa::SignPrimitive<Secp256k1>::try_sign_prehashed()	**OK**
k256::Aritmetic::scalar::Scalar::invert()	Constant time operation
k256::Aritmetic::scalar::Scalar::mul()	Constant time operation
k256::ProjectivePoint::mul_by_generator()	The primary SCA target, implemented using Windowed Exponentiation w/ constant time lookups and with constant time Montgomery Ladder. ✅

Other functions

These functions are used by the signing algorithm to compute the deterministic ephemeral secret key generated during ECDSA signing. Primarily uses hash functions, which in general are much harder to exploit using SCA, especially remotely.

rfc6979::generate_k()
-> crypto_bigint::array::ArrayEncoding::to_be_byte_array()
-> crypto_bigint::array::ArrayEncoding::from_be_byte_array()
-> rfc6979::HmacDrbg::new()
-> rfc6979::HmacDrbg::fill_bytes()
rfc6979::HmacDrbg::new()
-> digest::mac::update()
-> digest::mac::finalize()
-> digest::mac::finalize_reset()
rfc6979::HmacDrbg::fill_bytes()
-> digest::mac::update()
-> digest::mac::finalize_reset()
-> digest::mac::SimpleHmac::new_from_slice()