DoS Mitigation in (n+1)sec protocol

March 11, 2016

Contents

1 DoS axioms:

1.1 We assume that the joiner cannot deny access for other joiners because of the following:

1.1.1 The line in the code which results in dropping other join requests:

if (action$_{\text{totake}}$.action$_{\text{type}}$ = `RoomAction::NEW_PRIORITY_SESSION || action_to_take.action_type` = RoomAction::PRESUME$_{\text{HEIR}}$) { stale$_{\text{inlimbosessionspresumeheir}}$(action$_{\text{totake}}$.bred$_{\text{session}}$->session$_{\text{id}}$); } // else { //user state in the room

1.1.2 The condition that leads to PRESUME$_{\text{HEIR}}$

if (everybody$_{\text{authenticatedandcontributed}}$()) { group$_{\text{dec}}$(); // first compute the confirmation compute$_{\text{sessionconfirmation}}$(); // we need our future ephemeral key to attach to the message future$_{\text{cryptic}}$.init(); // now send the confirmation message Message outboundmessage(&cryptic);

outboundmessage.create$_{\text{sessionconfirmationmsg}}$(session$_{\text{id}}$, hash$_{\text{tostringbuff}}$(session$_{\text{confirmation}}$), public$_{\text{keytostringbuff}}$(future$_{\text{cryptic}}$.get$_{\text{ephemeralpubkey}}$()));

outboundmessage.send(room$_{\text{name}}$, us);

RoomAction re$_{\text{limboaction}}$;

re$_{\text{limboaction}}$.action$_{\text{type}}$ = RoomAction::PRESUME$_{\text{HEIR}}$; re$_{\text{limboaction}}$.bred$_{\text{session}}$ = this;

return StateAndAction(GROUP$_{\text{KEYGENERATED}}$, re$_{\text{limboaction}}$); */ if we are joining we don't need to relimbo and the room will /* ignore the action }

1.1.3 The condition that leads to NEW$_{\text{PRIORITYSESSION}}$

RoomAction Session::shrink(std::string leaving$_{\text{nick}}$) { */ we are basically building the intention-to-leave message /* without broadcasting it (because it is not us who intend to do so), // so we are making a fake intention-to-leave message without sending it RoomAction new$_{\text{sessionaction}}$;

auto leaver = participants.find(leaving$_{\text{nick}}$); if (leaver == participants.end()) { logger.warn("participant " + leaving$_{\text{nick}}$ + " is not part of the active session of the room " + room$_{\text{name}}$ + " from which they are trying to leave, already parted?"); } else if (zombies.find(leaving$_{\text{nick}}$) != zombies.end()) { // we haven't shrunk and made a session yet logger.debug("shrunk session for leaving user " + leaving$_{\text{nick}}$ + " has already been generated. nothing to do", FUNCTION, myself.nickname); } else { */ shrink now /* if everything is ok, add the leaver to the zombie list and make a // session without zombies zombies.insert(*leaver);

// raison$_{\text{detre}}$.insert(RaisonDEtre(LEAVE, leaver->id));

Session* new$_{\text{childsession}}$ = new Session(PEER, us, room$_{\text{name}}$, &future$_{\text{cryptic}}$, future$_{\text{participants}}$());

new$_{\text{sessionaction}}$.action$_{\text{type}}$ = RoomAction::NEW$_{\text{PRIORITYSESSION}}$; new$_{\text{sessionaction}}$.bred$_{\text{session}}$ = new$_{\text{childsession}}$;

/ we are as we have farewelled / my$_{\text{state}}$ = FAREWELLED; */We shouldn't change here and it is not clear why /* we need this stage: so as not to accept join? Why? Join will fail by non-confirmation // of the leavers return new$_{\text{sessionaction}}$; }

return c$_{\text{noroomaction}}$; }

1.1.4 The joiner DOSing

- As a result, the joiner cannot leave the session before joining, so she cannot invoke NEW$_{\text{PRIORITYSESSION}}$.
  - PRESUME$_{\text{HEIR}}$ is only called when group$_{\text{dec}}$ is successful: if the group dec is not correct, then the user will not drop the other sessions.
  - If the joiner sends a correct message to some users and a bad message to others, she can force some of the participants to drop the session and some not. Therefore, whenever a user is dropping a session, she should announce it if it is because of:

    - timeout
    - auth error
    - decryption error
    - confirmation error
    - If any user sends a decryption error or a confirmation error, then all users of the session run the cheater detection algorithm.
- Badly signed messages are considered just garbage/not sent/not received.
- Scenarios:
    - A malicious joiner sends bad shares.
    - Results in decryption error. This still needs investigation because it could be some insider who did it. Other joiners join as they did before.
    - Some decrypt correctly and some decrypt badly. In this situation, we still run a DoS detection process. If the shares are all signed, then it is on the sender.
- Dropping by timing:
    - You request drop in the grace period and wait for the period to pass.
    - You accept drop if the grace period has passed.
    - If you received the messages of the party being dropped, you keep them. Drop the dropper.
- Detection
    1. If it fails due to session confirmation or decryption failure, the user should inform and request for re-session marked with DoS detection.
    2. If it passes the second time, then you can go ahead, taking off the DoS tag.
    3. If it fails again while tagged with DoS detection, everybody signs and sends their new private key with their old ephemeral public key and old established p2p keys.
    4. Detect the cheater. Drop the cheater. Broadcast your proof.
    5. If someone is dropping someone else without proper proof, drop them indicating lack of proof.
- Remedy:

1. The user drops the cheaters/DOSers from the new participant list (as they have indicated their intention to leave), and sends a new participant info message for the new message which tells other joiners to try again.
2. The user drops as many as she needs till she is alone.
3. When the user drops someone, she doesn't accept them as participants anymore, only as joiners.
4. Somebody wants to join as a participant but the user expects her as a joiner, so she informs her about that.

1.1.5 No joiner is receiving priority before

**

1.2 If a set of participants cannot reach an agreement, the status quo will remain in place.

1.3 Maliciousness is relative and defined based on the agreeing subgroup. If the transport delivers different payloads for different participants, then those sets of participants cannot reach an agreement.

2 DoS goal:

2.1 A set of benign participants where the transport honestly delivers packets between them should be able to form a session.

3 List of DoS possibilities

3.1 DoS Maliciousness

1. Unresponsiveness.
2. Generating wrong keyshare.
3. Confirming wrong session.
4. Asking for people to leave without reason.

3.2 Exception: Authentication discrepancy:

3.2.1 If two participants do not agree on authenticating a new joiner, then the protocol halts without consequences because authentication is (1) deniable and (2) inherently a privilege.

4 How to detect

- When a participant concludes that another participant is malicious because of one of the above reasons, she request that participant to be kicked out of the participant list, and includes the reason for the kick.
- How to detect DoS:
  - Time dependent: allow double amount of timeout to react, but accept any reaction after the timeout period.
  - Generating wrong keyshare or confirmation share:
    - Generate a new share (not to disclose the previous secret).
    - In case of failure, encrypt the new shares using AES-GCM with the old p2p key and send it to everybody.

5 How to react after detection

- If DoS happens during a join process, If it is the joiner who is malicious: The maliciousness is happening in session confirmation phase:
  - just drop the session in limbo. Send re-join message with participant info without DoSer.
  - The maliciousness is happening before sending session confirmation phase. Just drop the joining session.

  A current participant is malicious:
  - send a kick request which generates a session confirmation that helps the joiner to know that a new session is generated.
- If DoS happens during leave process: Send a kick request for the malicious participant.
- If DoS happens during re-session. Send a kick request for the malicious participant.

- Authentication failure is a reason for barring join but not DoS.
- When someone gets kicked out due to DoS reasons, she should become the last person to join after all the other joiners already in line.

6 Concerns:

- Timing problems. There should be an acceptable delay. The messages arrived within an

acceptable delay period should be ordered in their hash order. But we assume global ordering on messages for now.

- Multiple sessions in the same room. This is a natural consequence of the end-to-end protocol, as a different subgroup might have agreed on different views. This is not a problem with current participants, as they are ignoring the session id for which they do not have an established session.
- For a joining participant, the UI will present a choice of sessions and participants which the joiner can choose to join.
- For the sake of simplicity, we assume that each room is divided into mutually exclusive sessions.

7 Proof of DoS protection

- Theorem: Suppose $U_1, ..., U_n$ are sets of participants. $I_h \cap I_m = \{1, ..., n\}$.

where I_o is the set of honest and I_m the set of malcious participants, then, after running the above algorithm, each participant gets a list of $plist_i$. If the transport is honestly and consistently delivering messages in timely manner, then for $i, j \in I_h$ we have $U_i \in plist_j$.

Proof: TBD.

8 New Algorithm:

- Badly signed messages are dropped and treated as undelivered.

- If someone fails to contribute any message we are waiting for, we wait for the $\text{grace}_{\text{period}}$ and then we just assume they left.
- If key generation or confirmation fails, then we need to re-session with the session tagged as DoS detection. Users only can do this by sharing the evidence of cheating.
- If we fail key generation or confirmation with Dos detection tag, then we publish all private key encrypted by p2p keys signed by non-DoS-tagged authenticated private key. The cheater will be detected and kicked out.
- If U_i kicks U_j out (that is starting session S while U_j is not in the new session) without cheating evidence signed by alleged cheater U_j, then U_k simply ignore the request for the new session.

8.1 Sub protocol for unresponsiveness on join or re-session or any other part of the key agreement protocol.

If U_i fails to reply, U_j sends a kick request (for failure to reply) after the grace period has passed. Other participants either should agree with the kick and respond by participant-info message or re-broadcast the failed message (if they have received it despite the fact that U_j has not). When you get the message for failed delivery, you **have to** agree or rebroadcast. If you fail to do so, you will be dropped as well.

- The main idea is to have two timers: A replies before timer 1 ends, B asks to kick out A=> kickout B. A replies after timer 1 but before timer 2 ends, B asks to kick out A => kickout A. A replies after timer 2 but B does not ask to kickout A => ask to kickout A.

– Current users start a timer as soon as they get a join request for all users in the room to respond with authentication.

Note: Authentication failed should be an acceptable response.

– If the timer times out, they mark the user as unresponsive and start another timer to report the user as unresponsive.

– If they receive the message before the timer times out, and no other user requests the user to leave, they continue with the session establishment.

– If somebody requests a new session without the user, they accept the request and can drop the session.

– If they don't receive the message before the second timer times out, they request a session without the unresponsive users.

– If they receive a session shrink but the kicked-out user has replied in time for the first timer, they drop the requesting user. (We should because the receiving user and the requesting user have different views of the room.)

The users will play the second round, set up timers and follow the same rules.

– Conflict of circles: Obviously there will be a circle conflict because:

A thinks B is in and C is not. (A,B) => A eventually doesn't receive a response from B and drops B. B thinks C is in and A is not. (B,C) => B doesn't receive a response from C and drops C. C thinks A, B and C are in. (A,B,C) => C doesn't receive a response from A and B and drops both of them.

- In the end only users whose views are completely in agreement will stay in the same session. The room might be divided into different mutually exclusive sessions. (View agreement is an equivalence relationship.) The new joiner will be presented the option of choosing which session to join.

So it is obvious that we should treat cheating and transport problems separately, because the latter are provable.

in particular, if A decides that C is unresponsive while C is responsive, then B can relay C info signed by C to A.

When B receives a DoS re-session request from A to drop C: either B agrees with A on the reason of dropping C or she doesn't. If yes, she continues with the kick protocol on C. If no, B re-send the missing info to A in hope of reaching agreement if A doesn't reply to requested info in timely manner, then B's view does not agree with A's view, and therefore they cannot be in the same session. As such, B sends a kick request for A and all other participants whose view agrees with B drop A and start a new session.

8.1.1 Delivery failure/Transport delay recovery.

- The protocol agrees on $\text{INTERACTIVE}_{\text{GRACEPERIOD}}$.
- If U_i expects a message from U_j to establish a session and she does not receive it in $\text{INTERACTIVE}_{\text{GRACEPERIOD}}$ as of the end of the last round, then U_i starts $\text{new}_{\text{session}}(\text{kick}_{\text{out}}\ U_i$, reason: U_i fails message type x from session sid).
- If U_k receives a kick message, they either have received the failed delivery message or they have not: if they have, they resend the message

to s'id, message type x from sid by U_j (encrypt or not?) and do not follow up with the new session.

- If U_i is the only member of the session and there are more participants in the room, U_i will rejoin the room.

8.2 Sub protocol for generating wrong keyshare or confirming wrong session.

The protocol general rule:

– If key recovery or confirmation fails, then the key agreement protocol runs again with new ephemeral keys. If it fails, then the new keys and proof of cheating are published. A new session starts when the cheater has been kicked out.

– If someone fails to reply, then run sub-protocol for unresponsive user.

The failed message has the share for the new subgroup, so finally the responsive parties will make a successful subgroup.

So we break the protocol into sections:

8.2.1 Cheater detection protocol:

If the key fails to recover or the session confirmation does not match, then a special session with new ephemeral keys will be distributed and session establishment will be attempted. If the cheater detection session fails at the same stages, then the participants will reveal their private key signed by their old key and the cheater will be detected and kicked out.

- U_i fails at key recovery or $conf_j != conf_i$. Request a cheater detection session, with reason. (Reason is the set of signed shares which does not satisfy the system or the confirmation which does not match the deduced key.)
- U_j receives a request for cheater detection and evaluates the reason. If it is legitimate, a cheater detection session starts.
- If the cheater detection succeeds, it becomes the main session.
- If the cheater detection session fails, private keys signed by old private keys are published.
- Detect cheater and kick them out with proof.
- Do not join sessions which you do not agree with on the view.