DoS Mitigation in (n+1)sec protocol

January 29, 2016

Contents

1 DoS axioms:

- we assume that the joiner cannot deny access for other joiners for the following:

1.1 The line in the code which result 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

1.2 The condition that lead to PRESUME$_{\textbf{HEIR}}$

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 joing we don't need to relimbo and the room will* / ignore the action, }

1.3 The condition that lead to NEW$_{\textbf{PRIORITYSESSION}}$

RoomAction Session::shrink(std::string leaving$_{\text{nick}}$) { / *we are basically running the intention to leave message* / without broadcasting it (because it is not us who intend to do so) // make a fake intention to leave message but don't send ack 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 already shrunk and made a session logger.debug("shrunk session for leaving user " + leaving$_{\text{nick}}$ + " has already been generated. nothing to do", $\underline{\underline{\text{FUNCTION}}}$, 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$_{\text{participants}}$());

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

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

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

1.4 The joiner DOSing

- As the result, the joiner can not leave the session before joining so it can't invoke $\text{NEW}_{\text{PRIORITYSESSION}}$.
  - $\text{PRESUME}_{\text{HEIR}}$ is only called when $\text{group}_{\text{dec}}$ is successful: If the group dec is not correct then the user will not drop the other sessions.
  - If the joiner send correct message to some users and bad message to others, it can force some of participant to drop the session and some not. Therefore, whenever a user is dropping a session, they should announce it if it is because of
    - timeout
    - auth error
    - decryption error.
    - confirmation error.
    - If any user send decryption error or confirmation error then all users of the session runs the cheater detection algorithm.
  - Badly signed messages are considered just garbage/not sent/not received.
  - Scenarios:
    - Malicious joiner sends bad shares.
    - Results in decryption error. Still need investigation because it could be some insider who did it. Other joiners joins as they did before.
    - Some decrypt correctly 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 when grace period + wait period passed.

        * You accept drop if the grace period has passed.
        * If you received the message of the party being dropped you keep them. Drop the dropper.
    - Detection
        1. If it fails do to session confirmation or decryption failure, the user should inform and request for reseassion marked with dos detection.
        2. If it passes the second time then you can go ahead, take off the dos tag.
        3. If it fails and tag with dos detection. Everybody sign and send 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 with sending the proof.
    - Remedy:
        1. You drop the session with the cheaters/dosers if it is past presume heir, you send a new participant info message for the new message which tells other joiners to try again.
        2. You drop as many as you need till you are alone.
        3. When you drop someone you don't accept them as participants anymore only as joiners.
        4. Somebody wants to join as participant but you expect them as joiner, you inform them that.

1.5 No joiner is receiving priority before

**

2 List of DoS possibilities

DoS Maliciousness

- Unresponsiveness.
- Generating wrong keyshare.
- Confirming wrong session.

- Asking for people to leave without reason.
- When a participant conclude that a participant is malicious bceause of one of above reasons it request a leave for that participant.

3 How to detect

- How to detect DoS:
  - Time dependent: allow double amount of timeout to re-act but accept any re-action after the timeout period.
  - Generating wrong keyshare:
    - AES-GCM the shares sends it out to everybody.
    -

4 How to react after detection

- If DoS happens during a join process, Is it the joiner 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:
  - run a leave request which generate a session confirmation which helps the joiner.
- If DoS happens during leave process: Run a leave on the malicious participant.
- If DoS happens during re-session. Run a leave on the malicious participant.
- Authentication failure is a reason for barring join but not DoS.
- When someone get kicked out due to DoS reason he should be put on last person to join after all joiners already in line.

5 Concerns:

Timing problems. There should be acceptable delay. The messages arrived during acceptable delay period should be ordered in their hash order. But for now we assume global ordering on messages for now.

6 Proof of DoS protection

- Theorem: Suppose $U_1, ..., U_n$ are set of participant. $I_o \cap I_m = \{1, ..., n\}$.

then after running above algorithm, each participants get a list of $plist_i$. If the transport is honestly delivering messages in timely manner then for $i, j \in I_o$ we have $U_i \in plist_j$.

With enough round of running algorithm an honest joiner $U_i \in I_o$.

7 New Algorithm:

- Badly signed messages dropped treated as undelivered.
- If someone fails to contribute any message we are waiting for the $\text{grace}_{\text{period}}$ we just assume they left.
- If key generation or confirmation fails then we need to resession with the session

tagged as DoS detection. You only can do that 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 someone kick someone out (that's staring session S while U_i is not in the new session) without signed leave request by U_i then U_j start a session without U_i in it but U_j in it. That could be because we didn't leave request

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

Note: Authentication faille should be an acceptable response.

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

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

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

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

– If they receive a session shrink but the kicked out user has replied in time of the first timer. They drop the requesting user. (should we?)

The users will play the second round. setup timers and follow the same rules.

– Conflict of circles: Obviously there will be 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 drop C C thinks A,B and C are in. (A,B,C) => doesn't receive a response from A and B and drop both of them.

—- So it is obvious that we should treat cheating and transport problems totally separately. Therefore we call provable cheating

In particular if A decide that C is unresponsive while C is responsive then B can relay C info signed by C to A.

—- The protocol general rule:

– If key recovery or confirmation failed. Rekey try again, if it fails then publish keys, publish proof of cheating. start a new session with the cheater out.

– If someone failed to reply. Send the failed replied after grace period. Other participant either should send the same message or rebroadcast the failed message. When you get the message for failed delivery. You **have to** agree or rebroadcast, if you fail to do so you'll be drop as well.

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

So we break the protocol into sections:

Delivery failure/Transport delay recovery.

- The protocol agrees on $\mathrm{INTERACTIVE}_{\mathrm{GRACEPERIOD}}$.
- If U_i expect a message from U_j to establish session and it does not receive it in $\mathrm{INTERACTIVE}_{\mathrm{GRACEPERIOD}}$ as of end of last round. then U_i start $\mathrm{new}_{\mathrm{session}}(\mathrm{kick}_{\mathrm{out}}$ U_i, reason: U_i fails message type x from session sid).

- If $\mathrm{U_k}$ receive a kick out message, either they have received the failed delivery message or not: yes, they resend the message to s'id, message type x from sid by $\mathrm{U_j}$ (encrypt or not?) and does not follow up with the new session.
- If $\mathrm{U_i}$ is the only member of the session and there are more participant in the room $\mathrm{U_i}$ will rejoin the room.

Cheater detection protocol:

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

- $\mathrm{U_i}$ fails at key recovery or $\mathrm{conf_j} != \mathrm{conf_i}$. Request a cheater detection session, with reason. (reason is the set of signed shares which does not satisfies the system or the confirmation which does not matches the deduced key)
- $\mathrm{U_j}$ receive a request for cheater detection evaluate the reason. If it is legitimate, starts a cheater detection session.
- If the cheater detection succeed, it become the main session.
- If the cheater detection session fails. publish private keys signed by old private keys.
- Detect cheater and kicked them out with proof.