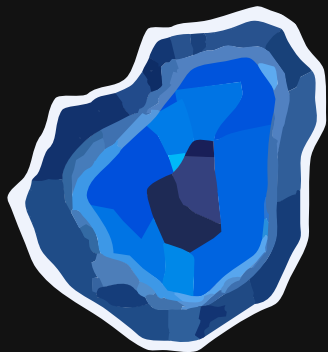


Signed, Sealed, Stolen


How We Patched Critical Vulnerabilities Under Fire

JADE, NEXUS



Jade Ellis


jade.ellis.link

 [JadedBlueEyes](#)



Nexus

timedout.uk

 [nexy7574](#)

We are not CyberSecurity professionals

Continuuity

- Continuuity is a matrix Chat server
- continuuity.org is where we keep 'Official' secondary accounts
- Continuuity is a part of an ecosystem of Conduit-based servers, forked from the same codebase
 - Conduit
 - Grapevine
 - Tuwunel

Setting the scene

This is very bad

- Jade did not send this event
- It contains a command which prints out every session token for the account
- There are a bunch of weird things going on with the event

```
17 Jade (recovering account) changed their displayname to Jade (she/her)
Monday, 29 December 2025

18 djos joined the room
Tuesday, 30 December 2025

19 Kayno joined the room

Jade (↳ thread)
\admin query raw raw-iter userdeviceid_token @jade

Jade
\admin query raw raw-iter userdeviceid_token @jade

1 ("@jade:continuwuity.org@1QYrqMkfkL", "lpxCdtMLYgVIHhTKCMbmwe07wk8LpQaZ")
2 ("@jade:continuwuity.org@7Bhn0oh8py", "3lcqT0dA13rI1lBgYAD54weiSmA2D3Gg")
3 ("@jade:continuwuity.org@9xFa5J98iU", "yWkF3nKzit0nQouCpsvF5548n0YIWTQ5")
4 ("@jade:continuwuity.org@8oJ4dk8ufz", "hL9TIArAmBzy2F14LiZcpkDs5oL8xIAk")
5 ("@jade:continuwuity.org@LHBaTo9DxP", "oTSkQJHfG0cDoWcUm1pbmd6U66zut9mI")
6 ("@jade:continuwuity.org@PqRvvrw2e", "kWLGOj7IydB3BfKX70CfJuEUBEGy1rZo")
7 ("@jade:continuwuity.org@R2WafpMsHc", "VmCyE609nXjvwwe0DVRnsiAEY8iirbad")
8 ("@jade:continuwuity.org@WMEFF1CATX", "5LcJGe5BH2gMacz8GMKQxBGYJA956Tf")
9 ("@jade:continuwuity.org@XiyiXPCI7N", "DrVrKJ7mwzXUBuGqYdtwNL0Eij405aKf")
10 ("@jade:continuwuity.org@dTMbDEidCf", "KiRBgBwb63VxYebbArSc40hUcuTtFnVS")
11 ("@jade:continuwuity.org@j8L56y0s4U", "RgwAotXXbKzrHdwoQNbWaWeBRzFLFqdo")
12 ("@jade:continuwuity.org@jWAQJCUKvZ", "zG6Ja2UueChMcb06XRmf1Ms8qmk0W5AA")
13 ("@jade:continuwuity.org@r5Q1EZh8hA", "mdHNtFd5gM4Es75Weyx2k7RsoLjcyVD5")
14 ("@jade:continuwuity.org@upGT9Q8mcg", "17X20hU6S3EodfhYTp318Vg5jihvyJ5u")
15

Query completed in 1.458309ms
```

23 seconds

What's the first thing you do when an account is compromised?

- Alert your team
- Panic
- Disable the account's admin access
- Suspend the account
- Reset the password
- Disable all login tokens
- Pull out the plugs on other possible targets

**OK, so we have to be safe now.
But how did the attacker do that?**

Time to investigate

- Identifying the attacking server is easy
- But the event seems to check out, there's nothing suspicious from a remote server
- I have a hunch

The incriminating event

```
{
  "auth_events": [
    "$R-PRPFumZ6zfsmW4Ek0YkvpeCdtAWRVLMWzol5aNqQA",
    "$HMbtQIwqzbMKJCHM4LxhYY31byrU3mj2SKphMJhEnSQ",
    "$CeVk90j0crDU47l098iD0IRDME4AiNwV0bkfw7F3jyo"
  ],
  "content": {
    "body": "\\!admin query raw raw-iter userdeviceid_token @jade",
    "m.relates_to": {
      "event_id": "$R-PRPFumZ6zfsmW4Ek0YkvpeCdtAWRVLMWzol5aNqQA",
      "rel_type": "m.thread"
    },
    "msgtype": "m.notice"
  },
  "depth": 70,
  "event_id": "$f8r_Zq-hVfKSgLMkeNB7IXTtVb085DFELqpT5R6N8kU",
  "hashes": {
    "sha256": "B+1hum2uRiJ/hnX+sHTlGP+yWPNHyNyPvHjzHrJo9Qo"
  }
}
```

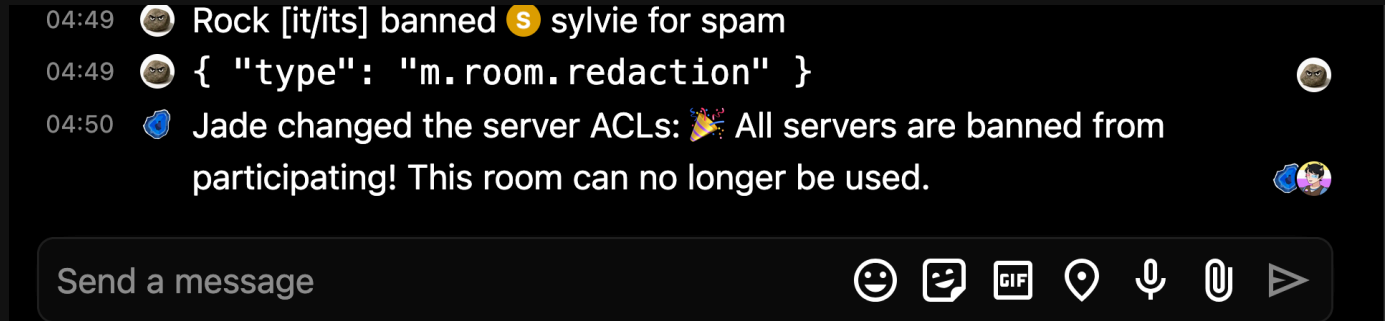
```
...
"origin": "continuwuity.org",
"origin_server_ts": 1767067175446,
"prev_events": [
  "$Cabj9hUltCn2vDgAonj7tLzwR4eSE6e58N2bjh7n0bE"
],
"room_id": "!3og6vwG8x7aZeCXJvC:ellis.link",
"sender": "@jade:continuwuity.org",
"signatures": {
  "continuwuity.org": {
    "ed25519:PwHlNsFu": "0yd4XlIgjbrQ/HMevRVHyWj oZXnd0QCZFXiKRjXdNbtjyZKugHyGHNSRCRjQ4aH3EiPJ"
  }
},
"type": "m.room.message"
}
```



that only happens for events not sent via /send 04:30



The event must be forged over federation.




In the meantime...








The attacker is causing as much damage as possible, now they're locked out of Jade's account



04:49  Rock [it/its] banned  sylvie for spam

04:49  { "type": "m.room.redaction" } 

04:50  Jade changed the server ACLs:  All servers are banned from participating! This room can no longer be used. 

Send a message       

One by one, our public chatrooms are made unusable

The time is now 5:00. It has been one hour.

Jade's hot tip: Don't talk to hackers

The next insight

- We have access to logs from other servers
- Based on failed attempts to access private rooms, the attacker requires knowledge of 'auth events' (ginger)
- These are previous messages in a room that control permissions
- The attacker must have an account already in the room
- The attacker must handcraft the events

[05:24] One maintainer down

[06:10] Two maintainers down

[06:30] One maintainer up

Free data

[07:13] The attacker starts probing a server we have debug logging and instrumentation on.

We now know the vulnerability.

[07:57] One maintainer down.

Before we get to the vulnerability: how do you leave a room?

- To leave a room, your server must construct a leave event
- Making this requires full knowledge of the current state of the room
- But the leaving server doesn't always have it
- The leaving server selects a server it knows is in the room, and asks it to make the leave event for it
- `GET /_matrix/federation/v1/make_leave/{roomId}/{userId}` returns the leave event.
- The leaving server then signs the event and returns it to the same server

```
for remote_server in servers {
let make_leave_response = services
  .sending
  .send_federation_request(
    remote_server.as_ref(),
    federation::membership::prepare_leave_event::v1::Request {
      room_id: room_id.to_owned(),
      user_id: user_id.to_owned(),
    },
  )
  .await;
```

```
let mut leave_event_stub = serde_json::from_str::<CanonicalJsonObject>(
    make_leave_response.event.get(),
)
.map_err(|e| {
    err!(BadServerResponse(warn!(
        "Invalid make_leave event json received from {remote_server} for {room_id}: {e:?}"
    )))
})?;

// TODO: Is origin needed?
leave_event_stub.insert(
    "origin".to_owned(),
    CanonicalJsonValue::String(services.globals.server_name().as_str().to_owned()),
);
leave_event_stub.insert(
    "origin_server_ts".to_owned(),
    CanonicalJsonValue::Integer(
        utils::millis_since_unix_epoch()
            .try_into()
            .expect("Timestamp is valid js_int value"),
    )
);
```

```
// In order to create a compatible ref hash (EventID) the `hashes` field needs
// to be present
services
  .server_keys
  .hash_and_sign_event(&mut leave_event_stub, &room_version_id)?;
```

```
services
  .sending
  .send_federation_request(
    &remote_server,
    federation::membership::create_leave_event::v2::Request {
      room_id: room_id.to_owned(),
      event_id: event_id.clone(),
      pdu: services
        .sending
        .convert_to_outgoing_federation_event(leave_event.clone())
        .await,
    },
```

Yeah.

validate membership events returned by remote servers

[Browse source](#) [Operations](#)

This fixes a vulnerability where an attacker with a malicious remote server and a user on the local server can trick the local server into signing arbitrary events. The attacker issue a remote leave as the local user to a room on the malicious server. Without any validation of the make_leave response, the local server would sign the attacker-controlled event and pass it back to the malicious server with send_leave.

The join and knock endpoints are also fixed in this commit, but are less useful for exploitation because the local server replaces the "content" field returned by the remote server. Remote invites are unaffected because we already check that the event returned from /invite has the same event ID as the event passed to it.

```
Co-authored-by: timedout <git@nexy7574.co.uk>
Co-authored-by: Jade Ellis <jade@ellis.link>
Co-authored-by: Ginger <ginger@gingershaped.computer>
```

...

 Olivia Lee last month • committed by  Jade Ellis

parent [19372f0b15](#) commit [12aecf8091](#)

Signed by:  Jade

GPG key ID: 8705A2A3EBF77BD2

± 4 changed files with 109 additions and 3 deletions

⋮ ↻ ⋮

```
validate_remote_member_event_stub(  
    &MembershipState::Leave,  
    user_id,  
    room_id,  
    &leave_event_stub,  
)?;
```

```
/// Validates that an event returned from a remote server by `/make_*`  
/// actually is a membership event with the expected fields.  
///  
/// Without checking this, the remote server could use the remote membership  
/// mechanism to trick our server into signing arbitrary malicious events.  
pub(crate) fn validate_remote_member_event_stub(  
    membership: &MembershipState,  
    user_id: &UserId,  
    room_id: &RoomId,  
    event_stub: &CanonicalJsonObject,  
) -> Result<()> {
```

```
let Some(event_type) = event_stub.get("type") else {
    return Err!(BadServerResponse(
        "Remote server returned member event with missing type field"
    ));
};
if event_type != &RoomMemberEventContent::TYPE {
    return Err!(BadServerResponse(
        "Remote server returned member event with invalid event type"
    ));
}
```

```
let Some(sender) = event_stub.get("sender") else {
    return Err!(BadServerResponse(
        "Remote server returned member event with missing sender field"
    ));
};
if sender != &user_id.as_str() {
    return Err!(BadServerResponse(
        "Remote server returned member event with incorrect sender"
    ));
}
```

```
let Some(state_key) = event_stub.get("state_key") else {
    return Err!(BadServerResponse(
        "Remote server returned member event with missing state_key field"
    ));
};
if state_key != &user_id.as_str() {
    return Err!(BadServerResponse(
        "Remote server returned member event with incorrect state_key"
    ));
}
```

```
let Some(event_room_id) = event_stub.get("room_id") else {
    return Err!(BadServerResponse(
        "Remote server returned member event with missing room_id field"
    ));
};
if event_room_id != &room_id.as_str() {
    return Err!(BadServerResponse(
        "Remote server returned member event with incorrect room_id"
    ));
}
```

```
let Some(content) = event_stub
    .get("content")
    .and_then(|content| content.as_object())
else {
    return Err!(BadServerResponse(
        "Remote server returned member event with missing content field"
    ));
};
let Some(event_membership) = content.get("membership") else {
    return Err!(BadServerResponse(
        "Remote server returned member event with missing membership field"
    ));
};
if event_membership != &membership.as_str() {
    return Err!(BadServerResponse(
        "Remote server returned member event with incorrect membership"
    ));
}
```

OK, so how did they trigger the exploit?

@bot:continuity.org

```
@event.on(EventType.ROOM_MEMBER)
async def handle_invite(self, evt: StateEvent) -> None:
    # i'm allergic to and statements
    if evt.state_key == self.client.mxid:
        if evt.content.membership == Membership.INVITE:
            if self.is_user_trustworthy(evt.sender):
                await self.client.join_room(evt.room_id)
            else:
                await self.client.leave_room(evt.room_id)
```

Distributing the patch

- 3 other projects are vulnerable in the same way
- Both the spec and another project are vulnerable to a lesser degree
- Some projects we have channels with, some we don't

Release complete

Time to completion: 13h45m

Let's travel back in time

03:12:02: The attacker joins a Matrix room dedicated to a side project of mine.

03:59:35: The attacker sends a forged message as my continuwuity.org account with the content `\!admin query raw raw-iter userdeviceid_token @jade`. This invocation of the “escaped admin commands” feature—enabled by default—causes that account to respond with the contents of the table: namely, all of the login tokens for `@jade:continuwuity.org`. This is the first visible evidence of an attack.

04:00:44: The attacker redacts the result with curl

04:01:32: The attacker uses the stolen token to log in with Element Web

04:03:00: The attacker begins exporting messages from the moderation room

04:03:39: I leave the continuwuity.org admin room. 04:03:41: Nex suspends the account. 04:03:53: Nex force-resets the password for the account. This doesn't log out any devices.

04:05:40: The attacker stops exporting messages from the moderation room

04:06:24-44: Attacker tries to view direct messages between @jade:continuwuity.org and other accounts, is stopped by e2ee

04:06:56: Attacker opens team room and begins exporting messages

04:07:47: Attacker is stopped mid-export by session deactivation

How we're securing continuity against similar attacks

- Account locking
- Admin command to forcefully log out all of a user's existing sessions
- Configuration-defined admins
- Disabling account logins
- Hardening admin command escapes
- Restricting certain admin commands
- Project Drawbridge: over 100 commits hardening federation APIs

