
# USENIX Security '26 Artifact Appendix: Message Injection Attacks Against Signal

Kien Tuong Truong
ETH Zurich

Noemi Terzo
Max Planck Institute for Security and Privacy

Kenneth G. Paterson
ETH Zurich

## A Artifact Appendix

### A.1 Abstract

This artifact consists of a self-contained QEMU virtual machine that allows running the attacks presented in the paper against a victim Signal account. The attacks are both in the malicious server setting and allow an adversary to inject messages in a conversation between two honest, uncompromised users. We reproduce this threat model by running an Android emulator (via nested virtualization), installing the Android Signal application on it (with some modifications, see below), and proxying the connection via *mitmproxy*. This allows message injection from the server into the client running on the emulator. The VM is accompanied by a set of scripts that allow running the attacks and verifying its behaviour.

We depict the setup in Fig. 1.

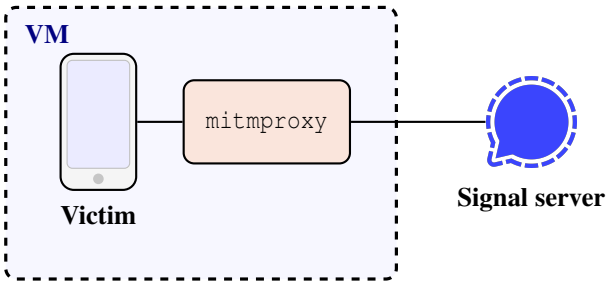


Figure 1: The setup from our artifact. The emulator communicates with the real Signal server via our proxy. Both emulator and proxy are running on the VM. The proxy has message injection capabilities that we can use from within the VM.

### A.2 Description & Requirements

For the remainder of this appendix, we will refer to the Signal account running on the emulator as "the victim".

The attacks require two changes to the Signal application on the emulator. It must accept a custom CA certificate for `mitmproxy`, and it must no longer contain the fixes that Signal deployed against our attacks. We apply both automatically during the build. Unfortunately, this means that this process may not work with future versions of the Signal application, as the patch may not apply cleanly. This is fundamentally a constraint given by the fact that the Signal server does not allow connections from clients which are not up-to-date, and therefore we cannot roll back to an unpatched version. Here, we describe all the patches that we apply to the Signal application during the build process, so that future evaluators can adapt them to future versions of the Signal application.

1. Re-enables receiving messages from a PNI address (for attack 1)
2. Re-enables receiving plaintext messages wrapped in a sealed sender envelope (for attack 2)
3. Add a trust root for the sealed sender certificate (for attack 2)
4. Modify the websocket connection to the Signal server to go through `mitmproxy` (for both attacks)
5. Trust the `mitmproxy` CA certificate (for both attacks)
6. Use a fixed secret for the application database (as a convenience feature, not strictly needed for the attacks)
7. Drop the `FLAG_SECURE` flag on the application window (as a convenience feature, not strictly needed for the attacks). This allows seeing the screen of the emulator via `ws-scrcpy`.

Note that the first three items simply revert the patches that Signal has introduced to fix our attacks. A malicious server already has the ability to intercept the connection (so already implicitly has items 4 and 5), and items 6 and 7 are simply convenience features that do not affect the security of the application. We have tested that all patches work for Signal-Android version 8.16.1.

#### A.2.1 Security, privacy, and ethical concerns

In the setup of our artifact, the victim has to connect to the production Signal server. This demands care when running the attack, as it may negatively affect Signal as a service provider. We have taken care to ensure that our scripts only inject in one direction – from the server to the client – such that no arbitrarily crafted messages are sent to the real server.

Our attacks allow message injection on a victim user. In particular, our second attack allows injecting messages from arbitrary users, given only their username. This raises two ethical concerns. The first one is that one may use our tools to maliciously craft chats with specific users in order to mislead others, or spread misinformation. We believe that such a concern is mitigated by the fact that this is possible regardless of our attack, as a well-motivated attacker may perform the same feat by simply editing their own message database. We do acknowledge that our artifact makes that easier, nonetheless. The second concern is that any reply to our injected messages do hit the Signal server. However, in this case, the reply messages are well-formed from the point of view of the Signal server as they have been created by an honest party (i.e. the victim).

#### A.2.2 How to access

The latest version of our artifacts is available at https://doi.org/10.6084/m9.figshare.31272589.

For the convenience of the artifact evaluation reviewers, we separately provided access to a zip file (on HotCRP), containing the application state of a Signal user. This allowed testing our attacks without needing two phone numbers and without needing to register a Signal account on the emulator. We did not include this zip file on FigShare as it is not strictly needed for running the attack and because malicious parties may misuse this account if it were to be openly available.

#### A.2.3 Hardware dependencies

This artifact requires no special hardware to execute. That said, we run a VM with a nested Android emulator, therefore systems that do not support nested KVM virtualization may be too slow for proper usage. Additionally, we build the Signal application within the VM. The build needs memory and CPU power, so the VM claims 16 GiB of RAM and 8 cores by default. A smaller host can shrink it: `RAM_MB=12288 CPUS=6 ./vm/run-vm.sh up` boots the VM with 12 GiB and 6 cores instead, at the cost of a longer build and a slower emulator. The VM disk is a 64 GiB sparse overlay, of which a full build fills roughly 20 GiB.

The first attack also requires another registered Signal account running on another device (presumably a phone). Note that this device will not be negatively impacted by this attack, as it will only send and receive honestly-crafted messages.

#### A.2.4 Software dependencies

Our artifact requires curl, QEMU, and a cloud-init seed builder to be installed on the host. When the VM is ready, it opens an SSH server on port 2222 to allow access.

On an Ubuntu host, all of these can be installed via:

```
apt install qemu-system-x86 \
            qemu-utils \
            cloud-image-utils \
            openssh-client \
            curl
```

#### A.2.5 Benchmarks

None

### A.3 Set-up

#### A.3.1 Installation

After installing the software dependencies mentioned above, ensure that the current user on the host is in the `kvm` group. [1]

Then, download the artifact from the FigShare link provided above and extract it to a folder of your choice. Next, start the VM: `./vm/run-vm.sh up`

This will start the VM and open an SSH server on port 2222. The user account is `ubuntu` and the password is `artefact`.

The cloud-init script will automatically provision all dependencies. Run `cloud-init status --wait` to wait for the provisioning to finish.

Finally, run `SIGNAL_REF=v8.16.1 artefact all`. This clones the Signal Android application at the version we tested and builds it. It also prepares the `mitmproxy` configuration and builds the binaries needed for the attacks. Finally, it creates the emulator and installs the Signal application on it. Note that old Signal client are regularly phased off, so this version may no longer be accepted by the Signal server in the future. Point `SIGNAL_REF` at a newer tag to try a later version. Unfortunately, the patches may no longer apply to newer versions, so this may require some manual work. Note that new releases may need a newer JDK runtime as well.

The whole process takes around 30 minutes.

#### A.3.2 Basic Test

Run `bash /opt/artefact/repl.sh`. This opens a REPL for running the attack.

Run `doctor` first. It prints one line per tool that the harness needs and ends with `all prerequisites satisfied`.

Then run the `emulators`, `stream` and `mitm` commands in sequence. `emulators` boots the victim emulator and installs the patched application. `stream` starts a ws-scrcpy web server, accessible on the host via http://localhost:8000. `mitm` starts the `mitmproxy` instance and routes the emulator through it.

At this point, the emulator should be usable via http://localhost:8000. Confirm that the Signal application is installed.

[1] If not, add the user to the group via `sudo usermod -aG kvm $USER` and log out and back in for the change to take effect.

### A.3.3 (Optional) Signal Account Provisioning

If available, you can use the provided account backup zip file to restore a Signal account on the emulator.

If using the VM, the backup has to be installed on the guest. Place the zip in `backups/` on the host before the first boot, and provisioning copies it in. If the VM is already running, copy it over directly:

```
scp -P 2222 victim-signal-db.zip \
    ubuntu@localhost:/opt/artefact/backups
```

Then run the `restore` command in the REPL and select the zip file. The password was provided to the reviewers via HotCRP. This will restore the account and allow you to run the attacks without needing to register.

Only one copy of the restored account may be online at a time. When a second copy connects, the Signal server closes the older websocket with code 4409 (`Connected elsewhere`) and that client stops receiving messages. Run `restore` again to recover.

### A.3.4 Victim Account Extraction

Once the victim account is in place, whether restored or registered by hand, run the `extract` command in the REPL. It reads the victim's databases, prints its ACI and PNI, and writes `attack-script/config.json`. Attack 1 reads that file, so (E1) requires this step. Note that this step only extracts information that a malicious server already has access to, since it is the one that assigns the ACI and PNI addresses to clients.

## A.4 Evaluation workflow

### A.4.1 Major Claims

**(C1):** A malicious server can inject messages by exploiting the ACI-to-PNI protocol in Signal. This leads to a single "Safety number changed" message, without actually affecting the safety numbers (e.g. if the users compare their QR codes out-of-band, they will succeed). This is the attack described in Section 3 of the paper. This is proven by experiment (E1).

**(C2):** A malicious server can inject messages by exploiting a weakness in Signal Sealed Sender (SSS) and a validation error for Plaintext messages wrapped using SSS. No "Safety number changed" message appears and safety numbers are not affected. This is the attack described in Section 4 of the paper. Experiment (E2) proves this for a one-on-one conversation. The attack can be extended to groups, though this artifact does not contain the tooling to do so.

### A.4.2 Experiments

Inbetween the experiments, it may be convenient to restore the Signal application state to a clean state. If you have access to the provided account backup zip file, you can run the `restore` command in the REPL. Alternatively, create such a backup by using the `backup <name>` command in the REPL and then use the `restore` command.

**(E1):** *[Attack 1] [10 human-minutes]*

In the following description, we refer to the victim as Alice and to the other device as Bob. Some of the following actions will require interaction from Bob.

**Preparation:** Set up the environment up to and including the steps described in Section A.3.2. Make sure to have a browser open on the host with access to `http://localhost:8000` and a REPL running. If not using the provided account backup zip file, register a Signal account on the emulator. Either way, run the `extract` command once Alice's account is in place: Attack 1 reads the `config.json` that it writes. Ensure that Bob is also registered on Signal. Do not exchange messages between Alice and Bob before running the attack.

**Execution:** The following steps run the attack and verify its effects.

1. From Alice's device in the emulator, open Signal and open a new chat (using the pencil icon on the bottom right) with Bob *by using the "Find by phone number" option*.
2. Send a message to Bob.
3. On Bob's device, accept the message from Alice and send a message to Alice. At this point, Alice's device has linked the ACI and PNI addresses of Bob. This fulfils the preconditions for the attack.
4. Run the `attack1` command in the REPL.
5. Select the message to be injected and use Bob's PNI address as the sender. This will inject the message from the server to Alice's device, appearing as if it were sent by Bob. This will also trigger a "Safety number changed" message on Alice's device, but the safety numbers will not actually change.
6. On both devices, click on the name of the contact and press on "View safety number". Confirm that the safety numbers match on both devices.
7. Optionally, send more messages using the REPL. These messages will not trigger any new "Safety number changed" message.
8. Optionally, send messages from Bob to Alice and confirm that they are received correctly, without triggering any new "Safety number changed" message.

**(E2):** *[Attack 2] [5 human-minutes]*

In the following description, we refer to the victim as Alice. Select a Signal user with a registered username and refer to them as Bob. This attack does not require any

interaction from Bob. For ethical reasons, we strongly recommend using an account under the evaluator's control for Bob.

**Preparation:** Set up the environment up to and including the steps described in Section A.3.2. Make sure to have a browser open on the host with access to `http://localhost:8000` and a REPL running. If not using the provided account backup zip file, register a Signal account on the emulator. Messages may be exchanged between Alice and Bob prior to the attack.

**Execution:** The following steps run the attack and verify its effects.

1. Run the `attack2` command in the REPL.
2. Select the message to be injected and use Bob's username as the sender. This will inject the message from the server to Alice's device, appearing as if it were sent by Bob. No "Safety number changed" message will appear, and safety numbers will not be affected.
3. Optionally, send more messages using the REPL. These messages will not trigger any "Safety number changed" message.
4. Optionally, send messages from Bob to Alice and confirm that they are received correctly, without triggering any new "Safety number changed" message.

## A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2026/`.