[un]prompted

The AI Practitioner Conference

0-click exploits in an AI world

Anna Groza (@pwnlambda) - Security Researcher

Bruce Dang (@brucedang) - Sensei

It has been a very long year

It has been a very long ~~year~~ few months

Not the talk we were planning on giving

- The talk was initially meant to be about automating remote attack surface enumeration and auditing in messaging apps using LLMs
  - We started testing it on large messaging apps
  - Ended up working a **lot** better than expected
  - The results so far are much more interesting than our ‘fancy’ Markdown files
  - This will be the focus of our talk instead

Note: I am not the only researcher working on this, props to the Calif Research team for pulling all of this off :)

It started with extremely simple issues

The State of State Machines

2021-JAN-19 Natalie Silvanovich

Posted by Natalie Silvanovich, Project Zero

On January 29, 2019, a serious vulnerability was discovered in Group FaceTime which allowed an attacker to call a target and force the call to connect without user interaction from the target, allowing the attacker to listen to the target's surroundings without their knowledge or consent. The bug was remarkable in both its impact and mechanism. The ability to force a target device to transmit audio to an attacker device without gaining code execution was an unusual and possibly unprecedented impact of a vulnerability. Moreover, the vulnerability was a logic bug in the FaceTime calling state machine that could be exercised using only the user interface of the device. While this bug was soon fixed, the fact that such a serious and easy to reach vulnerability had occurred due to a logic bug in a calling state machine -- an attack scenario I had never seen considered on any platform -- made me wonder whether other state machines had similar vulnerabilities as well. This post describes my investigation into calling state machines of a number of messaging platforms, including Signal, JioChat, Mocha, Google Duo, and Facebook Messenger.

WebRTC and State Machines

The majority of video conferencing applications are implemented using WebRTC, which I've discussed in several past blog posts. WebRTC connections are created by exchanging call set-up information in Session Description Protocol (SDP) between peers, a process which is called signalling. Signalling is not implemented by WebRTC, which allows peers to exchange SDP in whatever secure communication message is available to them, usually WebSockets for web applications, and secure messaging for messaging applications.

There are a few types of SDP that can be exchanged by WebRTC peers. In a typical connection, the caller starts off by sending an SDP offer, and then the callee responds with an SDP answer. These messages contain most information that is needed to transmit and receive media, including codec support, encryption keys and much more. After the offer/answer exchange, peers can send SDP candidates to other peers. Candidates are potential network paths that the two peers can use to connect to each other, and SDP candidates contain information such as IP addresses and TURN servers. Peers usually send more than one candidate to a peer, and candidates can be sent at any time during a connection.

WebRTC connections maintain an internal state related to whether an offer or answer has been received and processed, however, applications that use WebRTC usually have to maintain their own state machine to manage the user state of the application. How the user state maps to the WebRTC state is a design choice made by the WebRTC integrator, which has both security and performance consequences. For example, some applications do not exchange any SDP until the callee user has interacted with the application to answer the call, meanwhile others set up the peer-to-peer connection, and start sending audio and video from caller to callee before the callee is even notified of the call.



[un]prompted

Calif

DESCRIPTION ti...@google.com created issue #1
Edited by ti...@google.com on Dec 2, 2025 06:40PM

Sep 2, 2025 12:24AM

Background

To prevent security issues and spam, WhatsApp for Android requires some form of user interaction to automatically download files from non-contacts:
a. After adding someone as a contact, all future received images/files will be downloaded.
b. For individual chats, if you respond to a non-contact future media/documents will be automatically downloaded.
c. For group chats, opening the group once will cause all future messages to be downloaded from that group.
d. Manually pressing download on an image from a non-contact will also download the media/document.

After downloading files they can appear in the `MediaStore` database which can open up attack surface. Whatsapp calls `MEDIA_SCANNER_SCAN_FILE` immediately after download on the file so it should show up immediately in `MediaStore`. Vulnerabilities that bypass any of these, can result in vulnerabilities like PZ-442423708 and PZ-443741909 being reachable without any of the user interaction listed above. This vulnerability requires the precondition of knowing, guessing, or leaking a contact making it lower severity than a full contact gating bypass. However it's easy to attempt this many times in quick succession, and likely easy to guess contacts in targeted attacks.

VULNERABILITY DETAILS/REPRODUCTION CASE

1. `Attacker` creates a WhatsApp Group
2. `Attacker` adds `Victim` to Whatsapp Group
3. `Attacker` adds `Victim's Contact` to Whatsapp group
4. `Attacker` promotes `Victim's Contact` to admin
5. `Attacker` sends a presumably malicious image to the WhatsApp Group (WhatsApp web is the easiest to avoid errors on the sender's client)
6. `Victim's` device will automatically download the image without ever interacting with the group
   6.a. Note the image is not downloaded by the `Victim's Contact`

Note, to verify the photo is now in the MediaStore database run `adb shell content query --uri content://media/external/file --projection _data` on the Victim's device.

Note: Disabling Automatic Download or enabling WhatsApp Advance Privacy Mode prevents the file from being automatically downloaded.

VERSION

WhatsApp Version: 2.25.23.81 (stable on WhatsApp Website).
WhatsApp Version: 2.25.22.80 (stable on play store)

Credit Information

Brendon Tiszka of Google Project Zero.

This bug is subject to a 90-day disclosure deadline. If a fix for this issue is made available to users before the end of the 90-day deadline, this bug report will become public 30 days after the fix was made available. Otherwise, this bug report will become public at the deadline. The scheduled deadline is 2025-11-30.

WeWorm

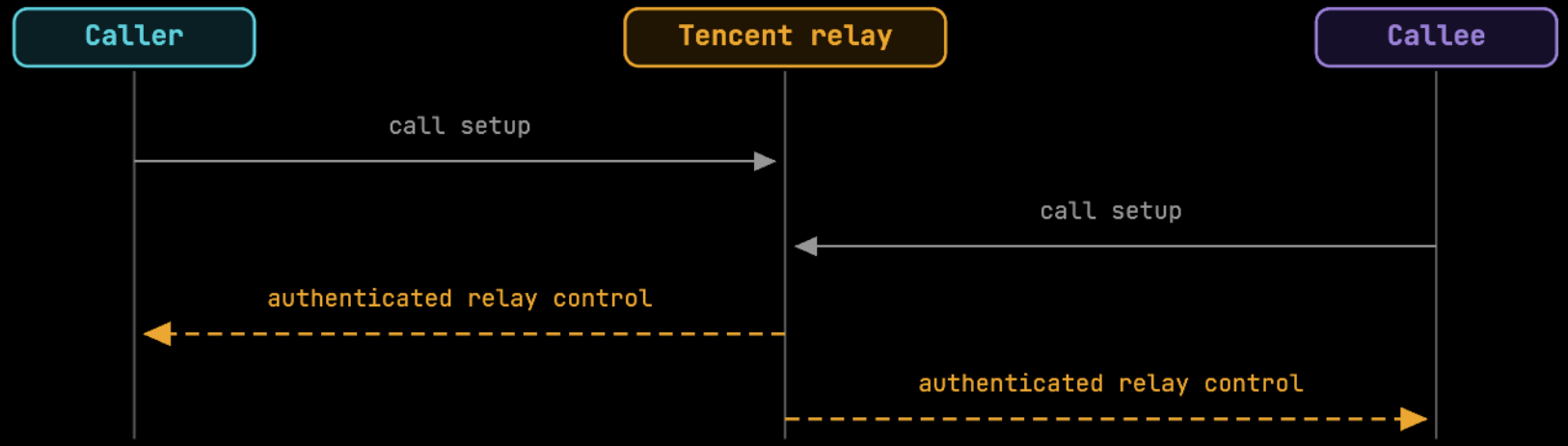


Fig 1 — Normal media topology: both endpoints check in to a Tencent-operated relay.

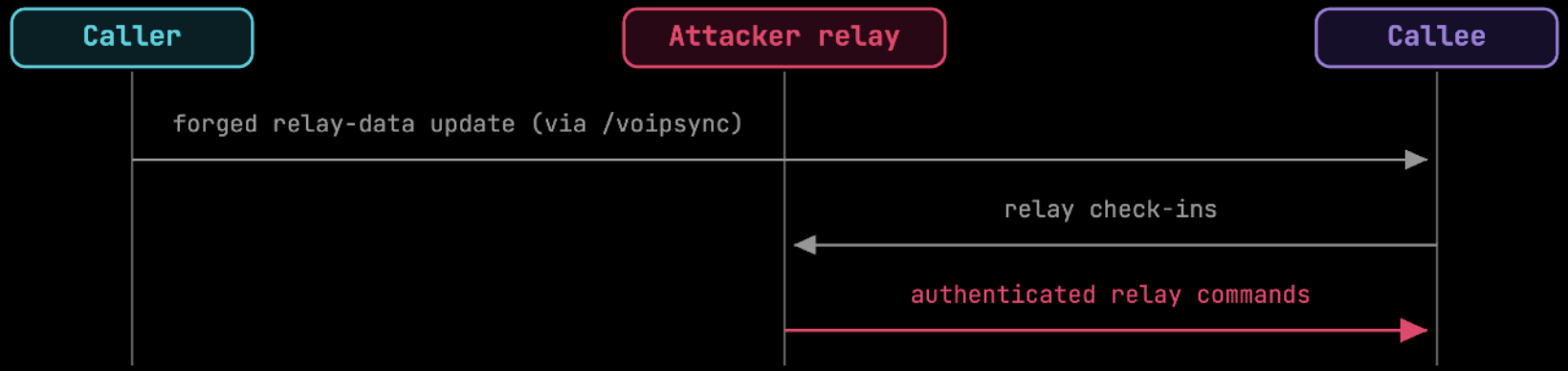


Fig 2 — After a forged `/voipsync` relay-data delta, the ringing callee checks into an attacker-controlled endpoint. The attacker still speaks the real relay protocol but now controls both directions.

WeWorm

Relay handler:

```
// Simplified from the vulnerable handler
memcpy(relayRoom + 0x6F0,
       request.clientIpStr().c_str(),
       request.clientIpStr().size());
```

Client check-in callback:

```
const char *clientIpStr =
    reinterpret_cast<const char *>(relayRoom + 0x6F0);

checkIn.set_clientipstr(clientIpStr, strlen(clientIpStr));
```



WeWorm demo

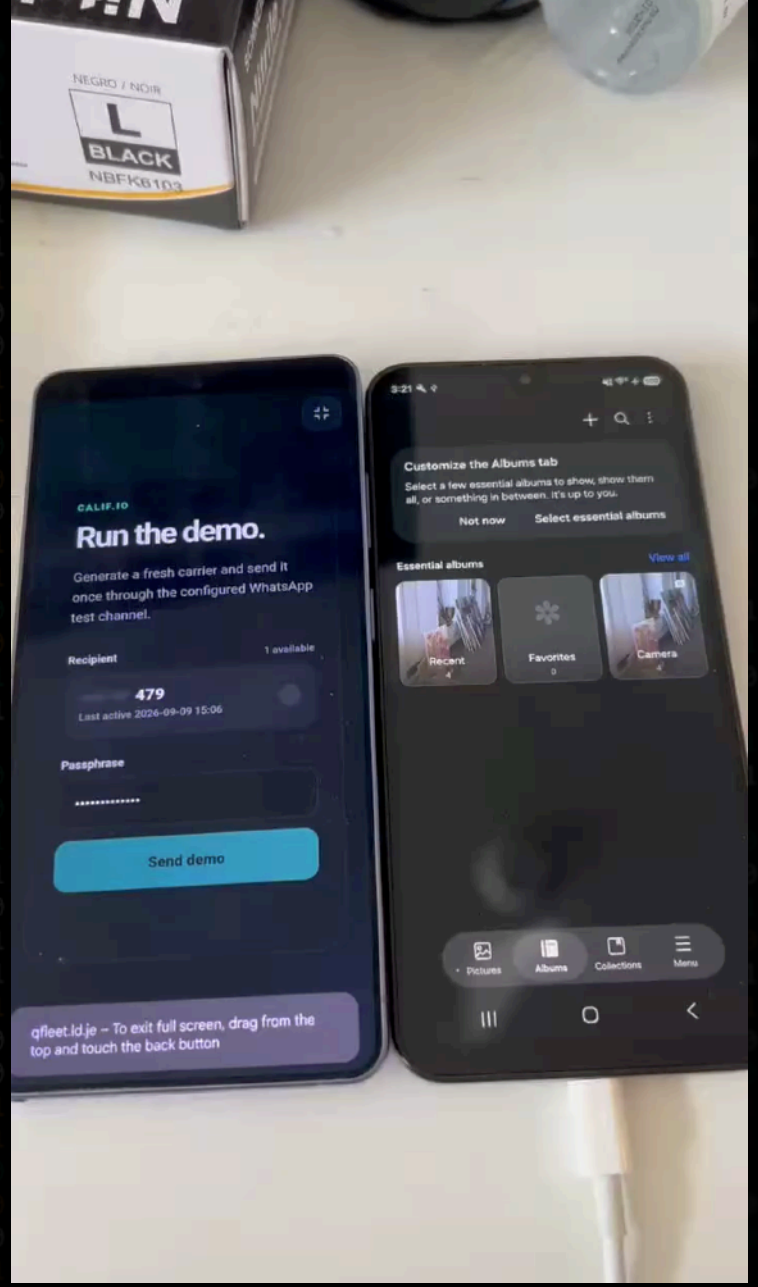
Calif

Then we broadened our search

(RE)DACTED

CVE-2026-?????

- Bug in Samsung's [REDACTED] parser
- Developing a reliable exploit required substantial guidance and steering
- We thought they fixed it, turns out that is not the case 🤦‍♀️
- We paired this with a way to reach it 0click from a trusted contact via WhatsApp
  - Thanks to Meta's Product Security team for their collaboration :)



[un]prompted

Calif

We’re living through exponential change
Newer models made this much easier

FORCEDENTRY

CVE-2021-30860

A deep dive into an NSO zero-click iMessage exploit: Remote Code Execution

2021-DEC-15 Ian Beer

Posted by Ian Beer & Samuel Groß of Google Project Zero

We want to thank Citizen Lab for sharing a sample of the FORCEDENTRY exploit with us, and Apple's Security Engineering and Architecture (SEAR) group for collaborating with us on the technical analysis. The editorial opinions reflected below are solely Project Zero's and do not necessarily reflect those of the organizations we collaborated with during this research.

Earlier this year, Citizen Lab managed to capture an NSO iMessage-based zero-click exploit being used to target a Saudi activist. In this two-part blog post series we will describe for the first time how an in-the-wild zero-click iMessage exploit works.

Based on our research and findings, we assess this to be one of the most technically sophisticated exploits we've ever seen, further demonstrating that the capabilities NSO provides rival those previously thought to be accessible to only a handful of nation states.

The vulnerability discussed in this blog post was fixed on September 13, 2021 in iOS 14.8 as CVE-2021-30860.

NSO

NSO Group is one of the highest-profile providers of "access-as-a-service", selling packaged hacking solutions which enable nation state actors without a home-grown offensive cyber capability to "pay-to-play", vastly expanding the number of nations with such cyber capabilities.

For years, groups like Citizen Lab and Amnesty International have been tracking the use of NSO's mobile spyware package "Pegasus". Despite NSO's claims that they "[evaluate] the potential for adverse human rights impacts arising from the misuse of NSO products" Pegasus has been linked to the hacking of the New York Times journalist Ben Hubbard by the Saudi regime, hacking of human rights defenders in Morocco and Bahrain, the targeting of Amnesty International staff and dozens of other cases.

Last month the United States added NSO to the "Entity List", severely restricting the ability of US companies to do business with NSO and stating in a press release that "[NSO's tools] enabled foreign governments to conduct transnational repression, which is the practice of authoritarian governments targeting dissidents, journalists and activists outside of their sovereign borders to silence dissent."

Citizen Lab was able to recover these Pegasus exploits from an iPhone and therefore this analysis covers NSO's capabilities against iPhone. We are aware that NSO sells similar zero-click capabilities which target Android devices; Project Zero does not have samples of these exploits but if you do, please reach out.

From One to Zero

In previous cases such as the Million Dollar Dissident from 2016, targets were sent links in SMS messages:

[un]prompted

Calif

Received 15 August 2017; revised 3 November 2017; accepted 11 December 2017.
Date of publication 19 December 2017; date of current version 9 June 2020.

Digital Object Identifier 10.1109/TETC.2017.2785299

Weird Machines, Exploitability, and Provable Unexploitability

THOMAS DULLIEN, **(Member, IEEE)**

The author is with the Google's Project Zero, Zurich 8002, Switzerland
CORRESPONDING AUTHOR: T. F. DULLIEN (thomas.dullien@gmail.com)

ABSTRACT The concept of *exploit* is central to computer security, particularly in the context of *memory corruptions*. Yet, in spite of the centrality of the concept and voluminous descriptions of various exploitation techniques or countermeasures, a good theoretical framework for describing and reasoning about exploitation has not yet been put forward. A body of concepts and folk theorems exists in the community of exploitation practitioners; unfortunately, these concepts are rarely written down or made sufficiently precise for people outside of this community to benefit from them. This paper clarifies a number of these concepts, provides a clear definition of exploit, a clear definition of the concept of a *weird machine*, and how programming of a weird machine leads to exploitation. The papers also shows, somewhat counterintuitively, that it is feasible to design some software in a way that even powerful attackers—with the ability to corrupt memory once—cannot gain an advantage. The approach in this paper is focused on *memory corruptions*. While it can be applied to many security vulnerabilities introduced by other programming mistakes, it does not address *side channel attacks*, *protocol weaknesses*, or security problems that are present *by design*.

INDEX TERMS Computer security, computer hacking, computation theory, information security, language-theoretic security

Received 15 August 2017; revised 3 November 2017; accepted 11 December 2017.
Date of publication 19 December 2017; date of current version 9 June 2020.

Digital Object Identifier 10.1109/TETC.2017.2785299

Weird Machines, Exploitability, and





protocol weaknesses, or security problems that are present *by design*.

INDEX TERMS Computer security, computer hacking, computation theory, information security, language-theoretic security

[un]prompted

Calif

FORCEDENTRY++

CVE-2026-86950

- Our exploit for the iOS CG antialiasing bug recently exploited in the wild
  - Very reliable even with MIE and PAC on the latest vulnerable iOS version (pre 26.7.1)
  - Constructs a 32-bit full adder out of a PDF/ TrueType weird machine
  - The Great Glyph Grift blogpost explains the root cause of the bug and the primitive it yields
  - The finer details of this exploit will be discussed at a future conference



FORCEDENTRY++

CVE-2026-86950

- An inconsistent conversion from double to a signed fixed-point integer makes the glyph rasterizer allocate a buffer that is too small
- Small buffers use alloca, placing them on the stack, while larger buffers go on the heap and may be protected by per-allocation MTE tags
- On our target the stack is canonically tagged so crossing the buffer's boundary does not itself trigger a tag fault
- Carefully constructed glyphs give us a repeatable relative 16-bit addition on the stack
- Nested PDF streams let us deepen the call stack and reach useful saved interpreter state

Bug writeup:



FORCEDENTRY++

```
› I have heard https://calif.io/research/the-great-glyph-grift is exploitable under MTE and PAC. Recompile the harness
  to have them then check for any useful stack primitives. Make sure your exploit works reliably across different
  executable boothashes and does not hardcode any addresses but instead calculates them all at runtime. Use the
  xzone_malloc skill and check our previous attempts to reach it at ./glyph-research
```

```
/goal demonstrate repeated r/w with logic or RCE
```

FORCEDENTRY++



FORCEDENTRY++

Goal achieved! ~3h

FORCEDENTRY++

- I struggled to believe it was real at first
  - After carefully examining the exploit and repeatedly testing it across different boots and environments i was proven wrong
  - Astra Cyber’s exploit was more sophisticated than the ITW sample since that reportedly worked only with PAC and MTE disabled
    - Note: No PAC/MTE is a good enough exploit if triggered in many 3rd party apps(i.e WhatsApp iOS)
  - Works in Signal/WhatsApp/iMessage/Gmail/Slack/anything that parses a PDF on iOS/macOS using native APIs either as 0 or 1 click
  - (·̆ ◠̆)

Closing thoughts

- In my experiments given a very strong primitive models perform better at exploitation under stronger mitigations*
- The trusted/untrusted contact gating in messaging apps feels more fragile than ever
  - Your own security should not depend on all of your trusted contacts being secure
  - Citizen Lab's recent reports showed how offensive actors have been misusing the ad ecosystem for exploit delivery
    - I wonder if a target's social graph could be obtained by similar means

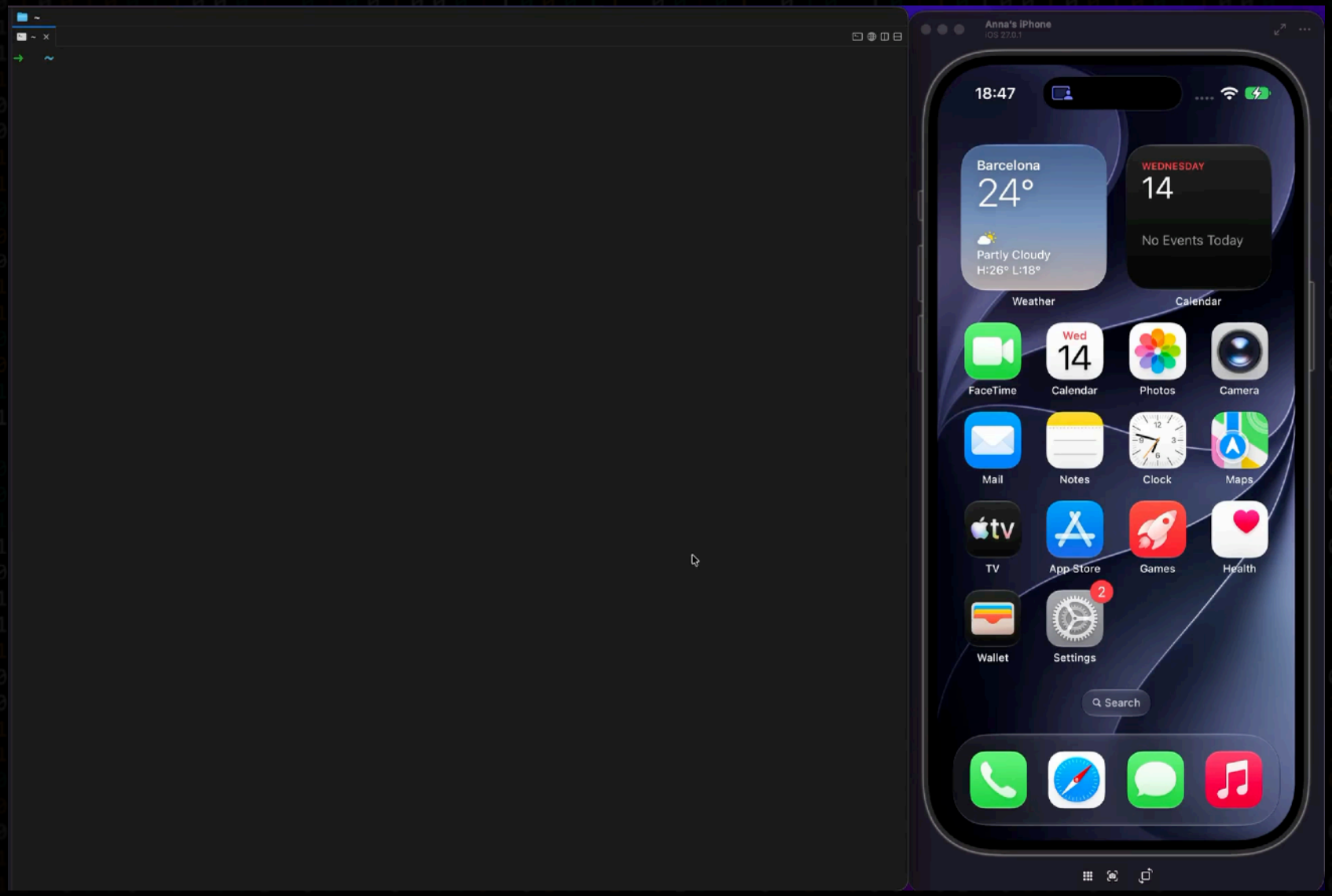
But, what happens if

- Target does not use a smartphone
- Target frequently changes his phone
- Target's phone number is unknown
- Target is highly aware and alert



Close-circle infection

One more thing...



One more thing…



[un]prompted

Calif


THIS IS
FINE

Thanks

Anna Groza - @pwnlambda
Bruce Dang - @brucedang

[un]prompted

The AI Practitioner Conference