Don’t Talk All at Once!

Elevating Privileges on macOS by Audit Token Spoofing

Computest
always on.

>>

Framework

XPC

Access a low-level interprocess communication mechanism.

macOS 10.10+ Mac Catalyst 13.0+

Overview

XPC provides a lightweight mechanism for basic interprocess communication. It allows you to create lightweight helper tools, called XPC services, that perform work on behalf of your app. The `launchd` system daemon manages these services, launching them on demand, shutting them down when idle, and restarting them if they crash. Benefits of XPC services include:

- Centralize work from multiple processes or mediate access to a shared resource.
- Delegate work so it continues beyond a client's life cycle.
- Privilege isolation to narrow the scope of access for different functionality.

Hello!

I’m Thijs Alkemade

Security Researcher at Computest

Computest
always on.

>>

About me

- Thijs Alkemade (https://infosec.exchange/@xnyhps)
- Security researcher at Computest Sector 7 (The Netherlands)
- Other work:
  - 0-click RCE in Zoom, Pwn2Own Vancouver 2021
  - Winning Pwn2Own Miami 2022 with 5 ICS vulnerabilities
  - Last year's OBTS talk: "Process injection: breaking all macOS security layers with a single vulnerability"

Computest
always on.

XPC

- XPC is often used between different privilege boundaries
  - For example: privileged daemon performing requests from lower privileged processes
- These often need to do an authorisation check: “**should this process be allowed to do this?**”
  - Does it have a specific entitlement?
  - Is the client signed by a specific Team ID?
  - Does it have a specific TCC permission?
  - Is the client sandboxed?

XPC

> `xpc_connection_get_pid` is **not safe** to use here!

Don't Trust the PID!

Stories of a simple logic bug and where to find it

Samuel Groß (@5aelo)

Issue 1223: MacOS/iOS userspace entitlement checking is racy

Reported by ianbeer@google.com on Thu, Mar 23, 2017 at 9:37 PM GMT+1 **Project Member**

Learn XPC exploitation - Part 2: Say no to the PID!

@WOJCIECH REGUŁA · APR 23, 2020 · 4 MIN READ

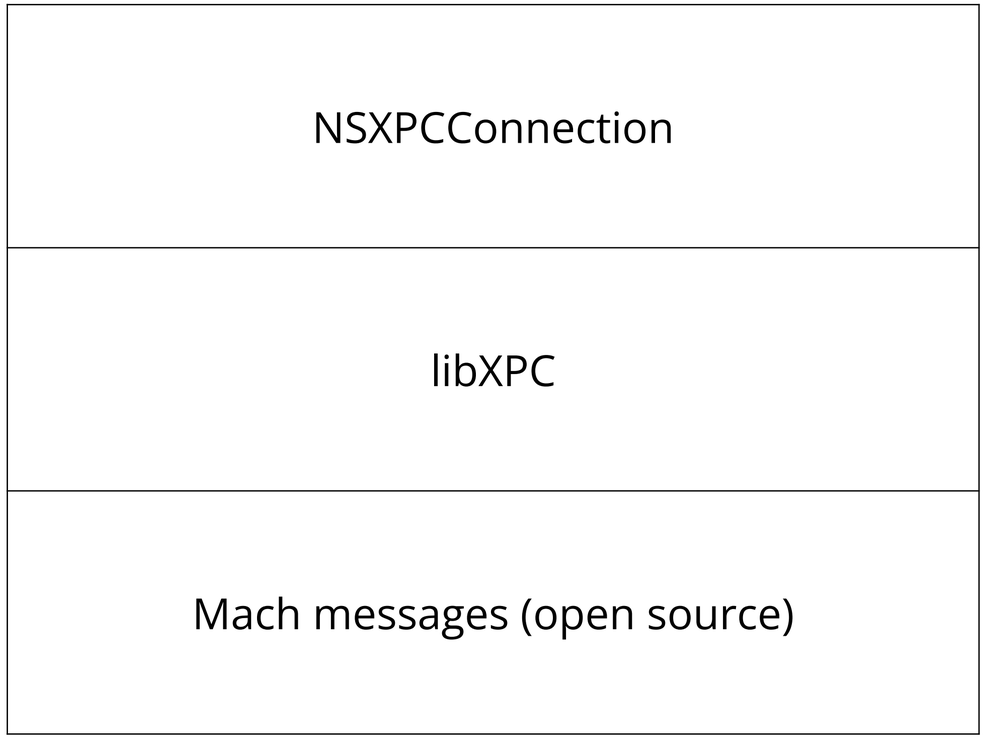
OBTS v3.0: "Abusing & Security XPC in macOS apps" - Wojciech Regula

>>

XPC

- It is well known that `xpc_connection_get_pid` is unsafe for authorising XPC clients
  - It is vulnerable to a race condition:
    - Send message
    - Exec an authorised process (without changing PID)
    - Hope the exec is done before the authorisation check
- `xpc_connection_get_audit_token` is better. Audit tokens are safe, because they contain a **PID version**
- But as I will show today, there are situations where even this function can be vulnerable to a race condition
- Better: `xpc_dictionary_get_audit_token`.

XPC architecture



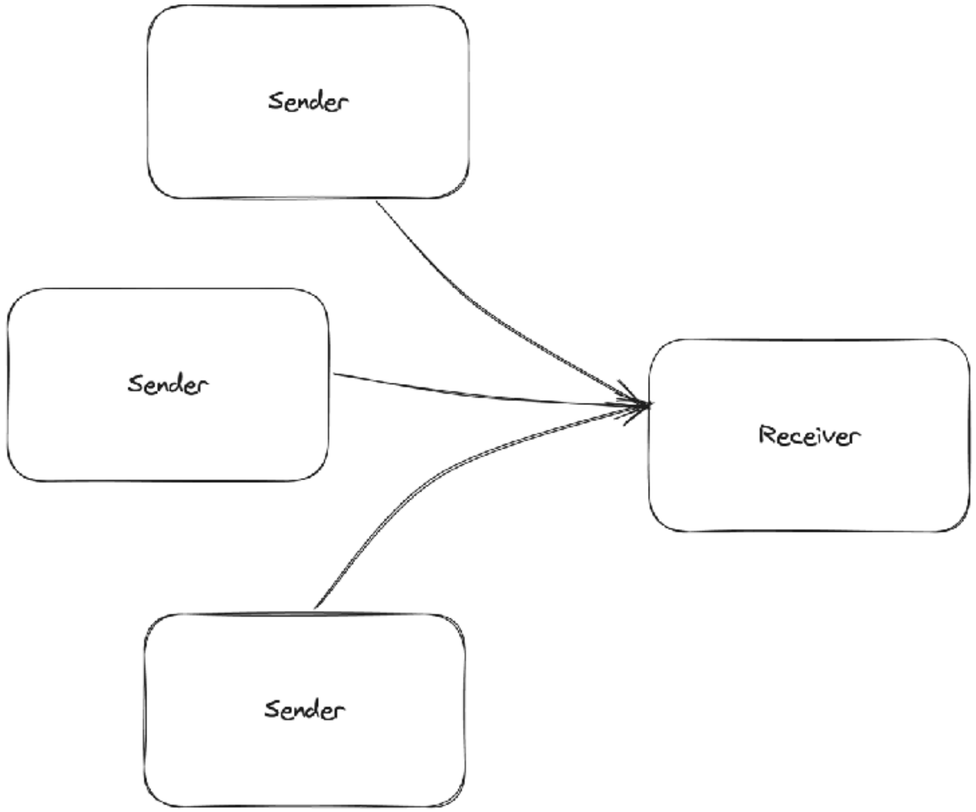
Mach messages 101

Computest
always on.

>>

Mach messages 101

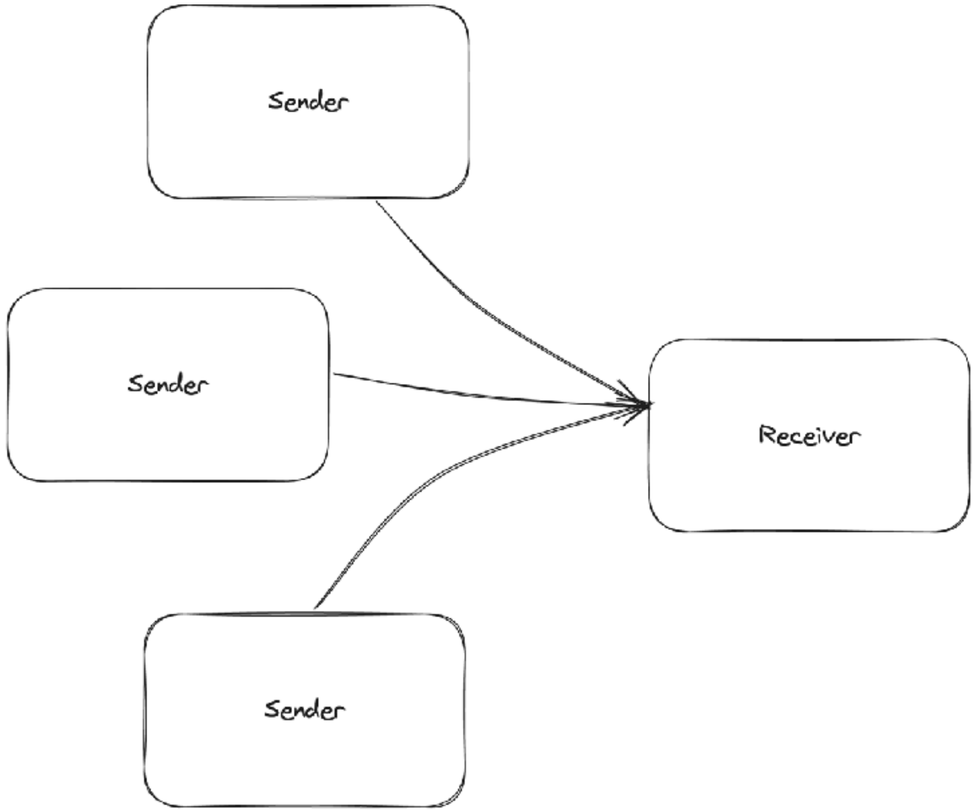
- Mach messages are sent over “mach ports”
- A mach port is a **single receiver, multiple sender** communications channel managed by the kernel
  - The actual object exists in the kernel, process only has a reference to it (the **port name**)
- Used extensively throughout the system
  - Some kernel API’s are just special mach ports



Computest
always on.

Mach messages 101

- Three types of “rights” exist for a mach port:
  - Receive right (only one process can have this)
  - Send right
  - Send-once right



Computest
always on.

Mach messages 101

- Send rights can be **duplicated** (or converted into a send-once)
- Port rights can be **transferred** with a mach message
- For example, the “local_port” field of a mach message can be used to transfer a send-once right for a reply to a message

```
typedef struct {
        mach_msg_bits_t         msgh_bits;
        mach_msg_size_t         msgh_size;
        mach_port_t             msgh_remote_port;
        mach_port_t             msgh_local_port;
        mach_port_name_t        msgh_voucher_port;
        mach_msg_id_t           msgh_id;
} mach_msg_header_t;
```

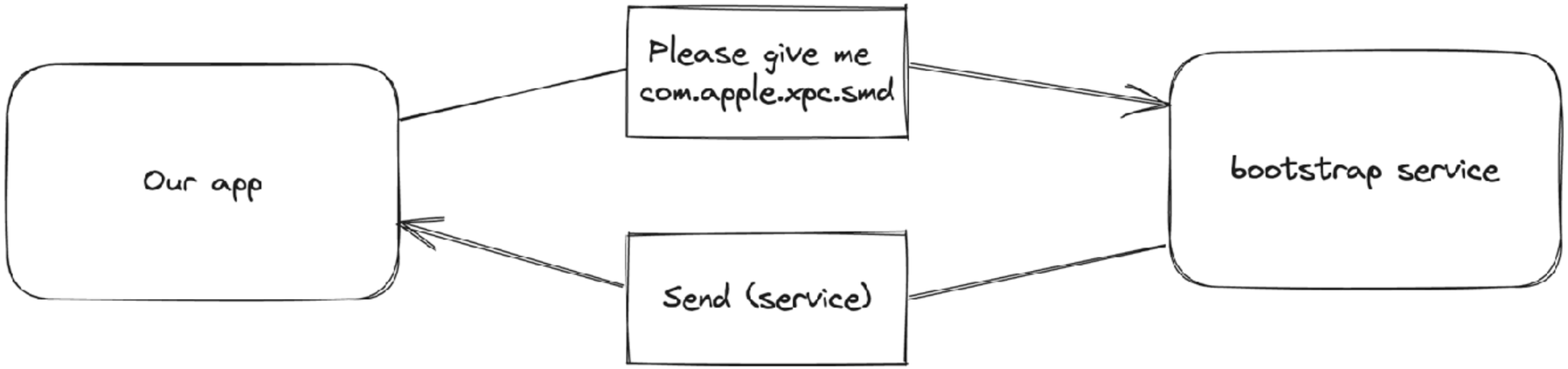
Computest
always on.

Mach messages 101

- Some terminology help:
  - A mach **port** is the entire communication channel (unlike “ports” in TCP/UDP)
  - “Holds a send/receive **right**” just means “this is a sender/the receiver”
    - Talking about rights is easier when they get transferred

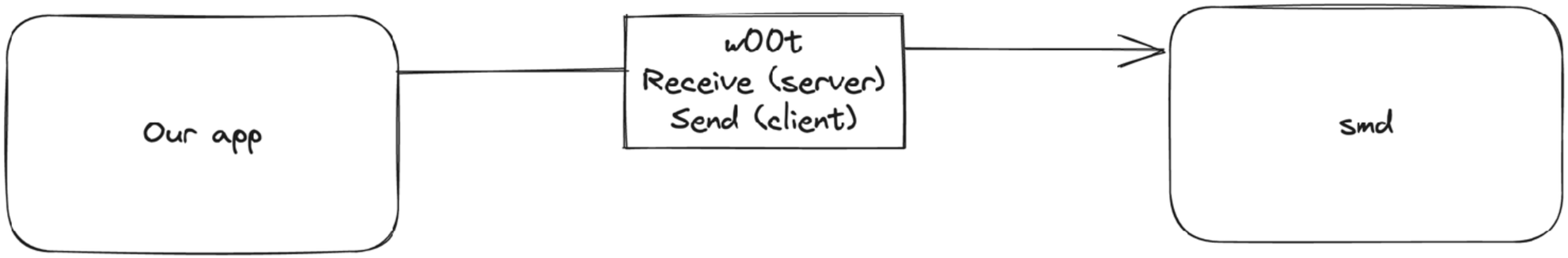
Establishing an XPC connection

- To establish an XPC connection, a client first needs (a send right for) the **service port**
- For a mach service, this is retrieved from the **bootstrap service** (in launchd)
- To get the service port:
  1. Client asks for service port for a name
  2. launchd looks up the name and duplicates the associated send right
  3. launchd sends the duplicated send right back



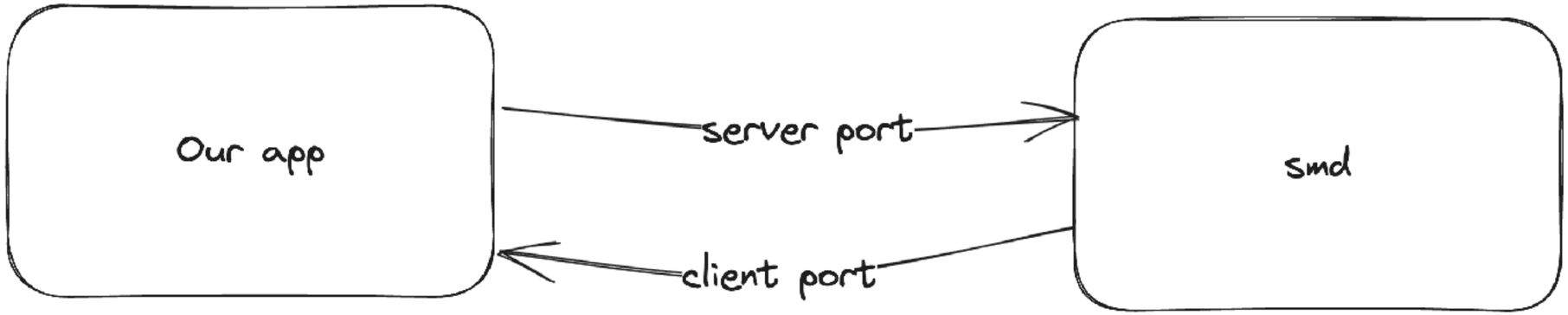
Establishing an XPC connection

- Then, the client generates two new mach ports: a **server port** and a **client port**
- It sends a mach message (id: `w00t`) to the service port. This message:
  - Transfers the **receive right** of the **server** port
  - Transfers the **send right** to the **client** port
- If the server accepts the connection, it starts listening on the server port



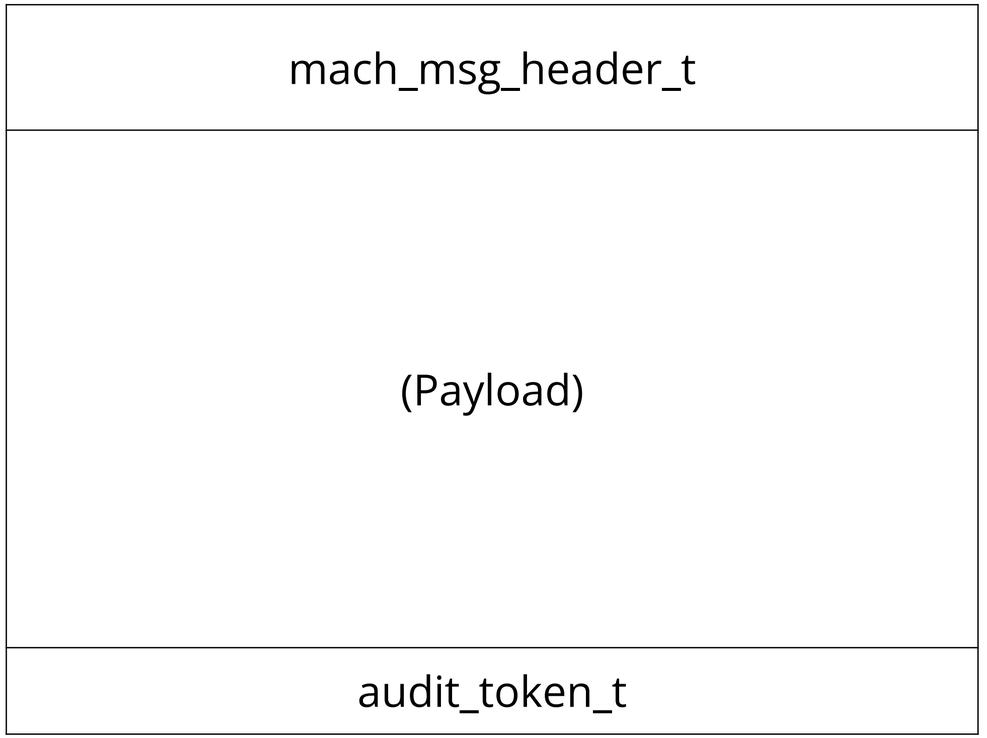
Establishing an XPC connection

- Server can send messages to the client port, client can send messages to the server port
- Note that this doesn't use the reply port mentioned earlier
  - It is used for XPC, but only for `xpc_connection_send_message_with_reply{_sync}`



Audit tokens

- How do those audit tokens come in here?
- For the `mach_msg` syscall, it is possible to ask the kernel to append a “trailer” with the audit token using the flag `MACH_RCV_TRAILER_AUDIT`
- Every time a message or reply is received, `_xpc_connection_set_creds` copies the audit token from the trailer to the connection object



The vulnerability

Computest
always on.

>>

But wait...

> We have seen the following:

1. XPC connections use authorisation checks that assume it's a **one-to-one** communication channel
2. Mach ports are single receiver, **multiple sender** communication channels
3. `xpc_connection_get_audit_token` returns the audit token saved for the **most recently received message**

🤔

Research question

Can we set up an XPC connection where **multiple processes** can send a message?

And if so, can we use that to **spoof the audit token**?

Research question

Answer: **yes**, the client generates the mach ports for the connection, so can specifically construct them.

However, audit token spoofing only works under **limited circumstances**.

An example: smd

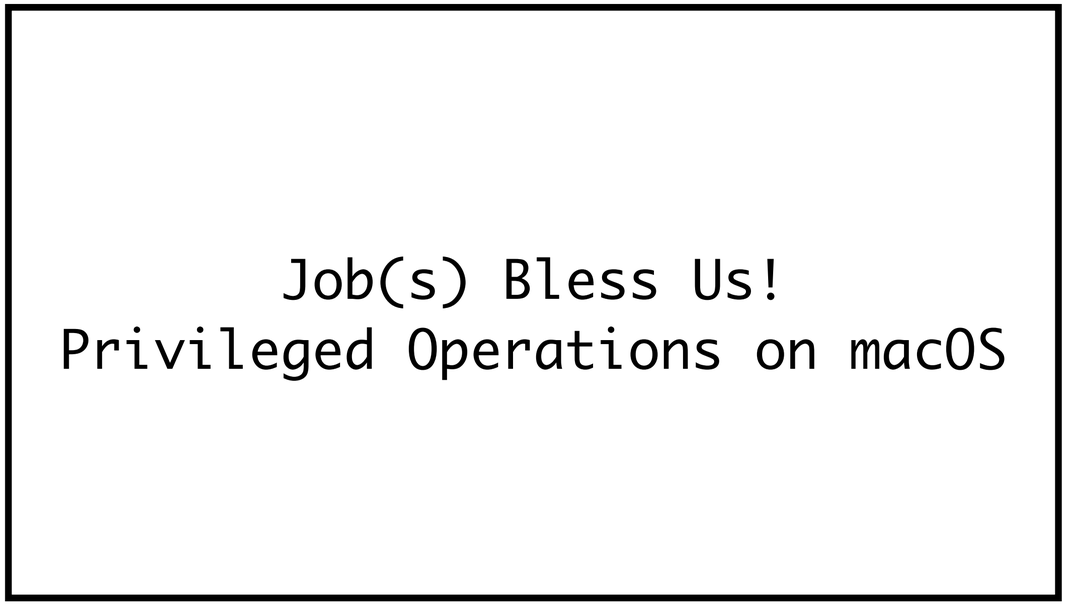
Variant 1

Computest
always on.

>>

An example

- `smd` is the **Service Manager Daemon**
- For example: `SMJobBless`, which you can use to install a privileged helper tool.
  - See also “Job(s) Bless Us! Privileged Operations on macOS” by Julia Vashchenko at OBTSv3
- Our goal: install a privileged helper tool, **without** the user entering their password



An example

- `SMJobBless` calls `smd` to perform the installation
- `smd` has multiple actions, specified by the "routine" in the request
- Routine 1004 is used by `SMJobBless`
  - This routine is handled on a different dispatch queue using `dispatch_async`

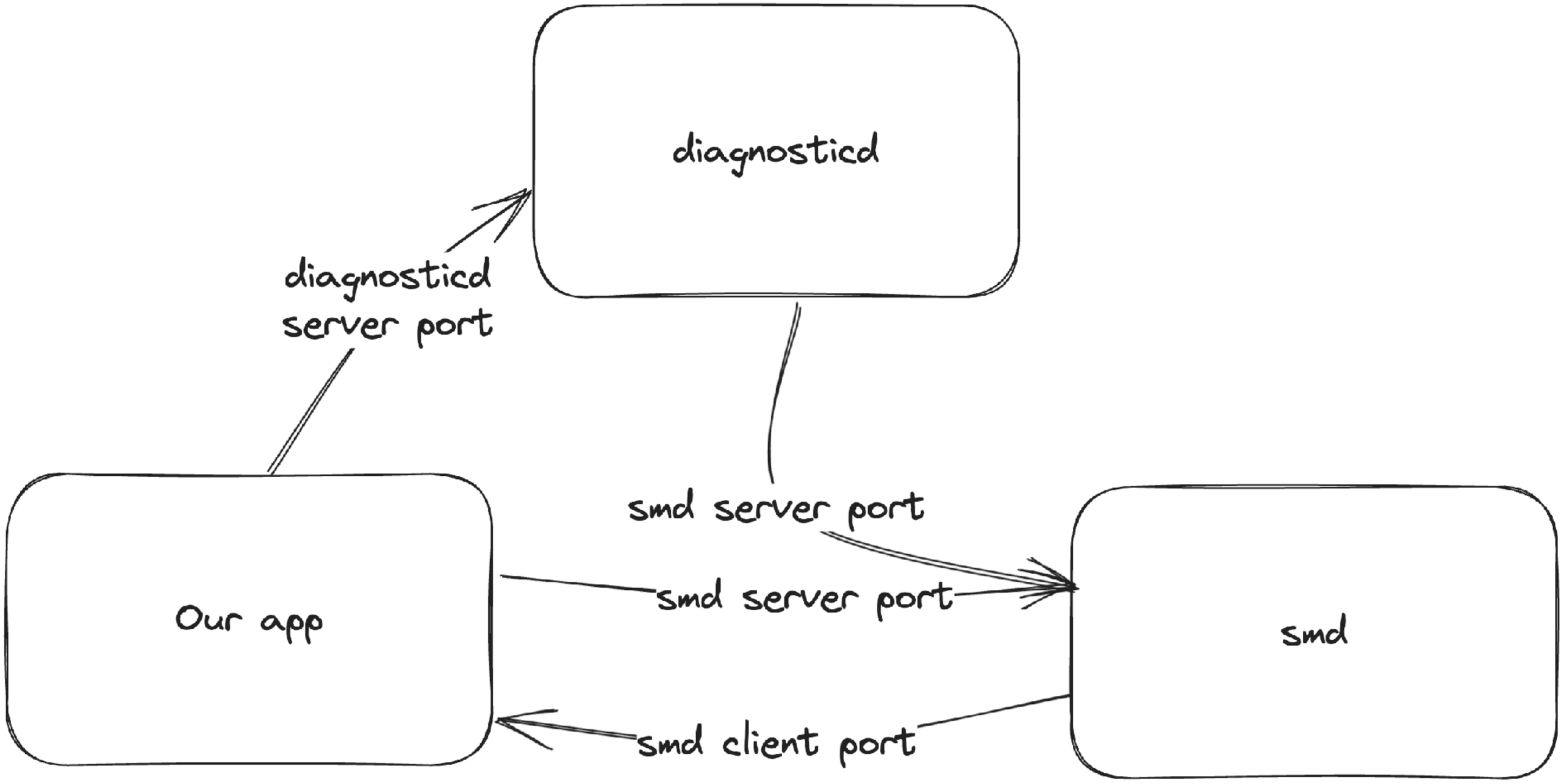
```
case 1004LL:
  buf.i64[0] = (__int64)_NSConcreteStackBlock;
  buf.i64[1] = 3254779904LL;
  buf.i64[2] = (__int64)handle_bless;
  buf.i64[3] = (__int64)&unk_100008180;
  v49 = objc_retain(v4);
  v50 = objc_retain(v5);
  dispatch_async((dispatch_queue_t)queue, &buf);
  goto LABEL_25;
```

An example

> A process is allowed to install a new privileged helper tool if it passes **one** of the following checks:
  1. It is running as root
  2. It has the entitlement `com.apple.private.xpc.unauthenticated-bless`.
  3. It has an authorisation reference for `com.apple.ServiceManagement.blesshelper`
> The third one is what is used if you call `SMJobBless`
  > Our goal is to pass the first

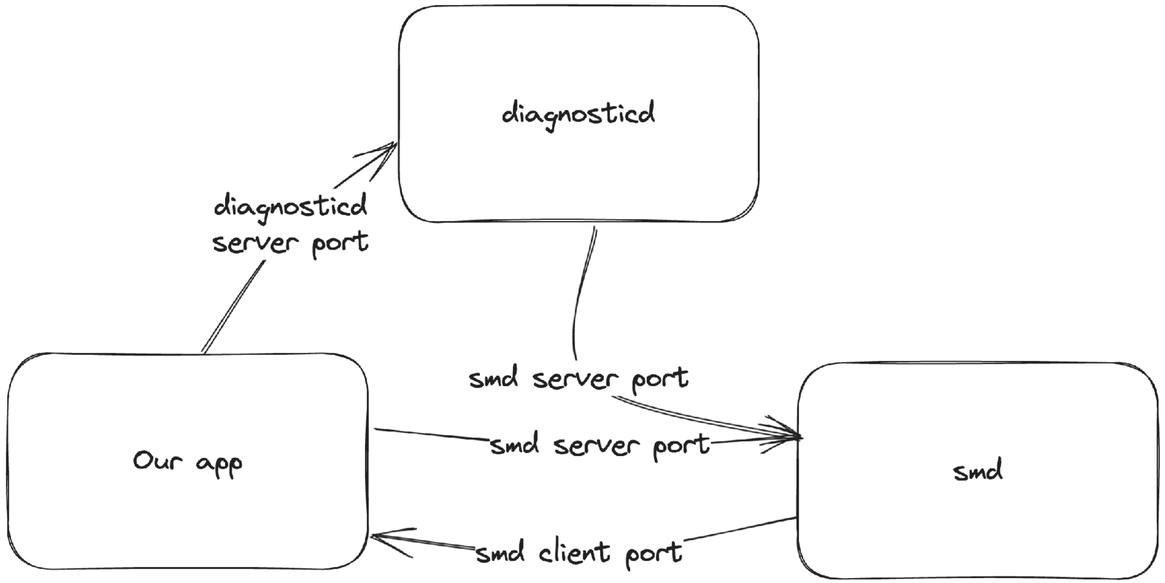
Second service: diagnosticsd

- `diagnosticd` is another daemon, which can be used to get live diagnostics of a process.
- It runs as root, so it can pass the first check in `smd`
- While monitoring a process, this daemon sends multiple messages per second about what the process is doing.


diagnosticd
diagnosticd
server port
smd server port
Our app
smd server port
smd
smd client port

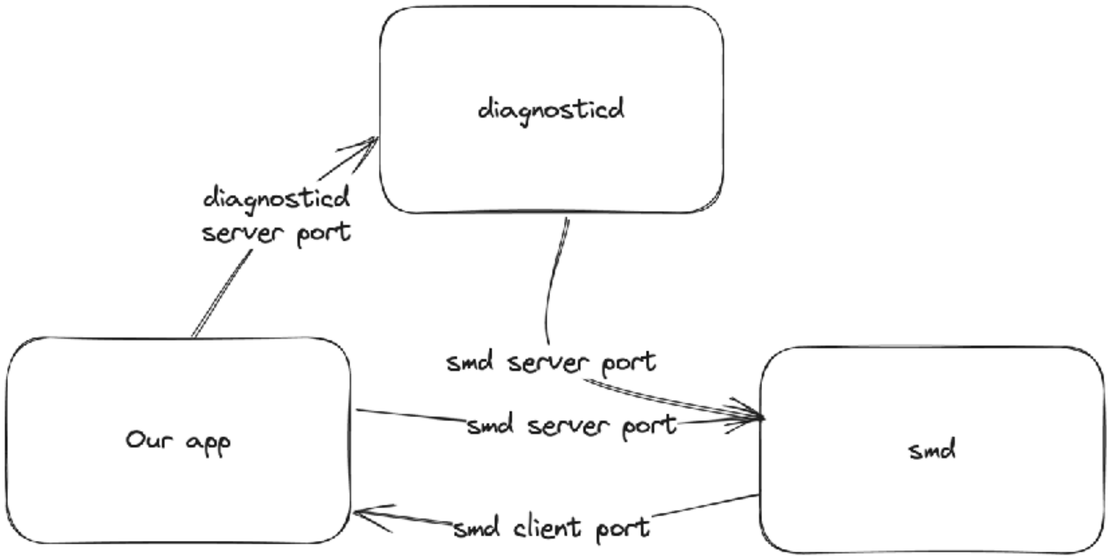
Exploiting smd

1. We establish a **normal** XPC connection to `smd`
2. Then, we set up another XPC connection to `diagnosticd`, but instead of a send right to our client port, we **duplicate our send right** to `smd`'s server port and send that.
3. Now `diagnosticd` will start streaming messages to `smd` instead of us



Exploiting smd

- Now we spam routine 1004 message to `smd`. If we are lucky, a message from `diagnosticd` overwrites the audit token at the right moment.
- If the authorisation check uses `diagnosticd`'s audit token, `smd` thinks we are running as root, so our install is allowed and we achieve privesc



Exploiting smd

- This is a race, but we can retry quickly. `smd` doesn't care if the authorisation check fails, and doesn't mind receiving many unrecognised messages from `diagnosticd`
- Within a few seconds, we install our privileged helper tool and elevate privileges!
- Since Ventura, the user gets a notification of this. But they are common, hard to identify and our payload has already executed
- Also works from a sandboxed app (kinda)

Variant 1

- To summarise the requirements:
  - Two mach services: the **exploited** service and the **impersonated** service
  - Our process must be allowed to communicate with **both**
  - The exploited service must have an authorisation check we cannot pass, but the impersonated service can
  - The impersonated service can be asked to send normal messages (not replies)
  - The authorisation check must be performed **asynchronously** from the XPC event handler

Reply port forwarding

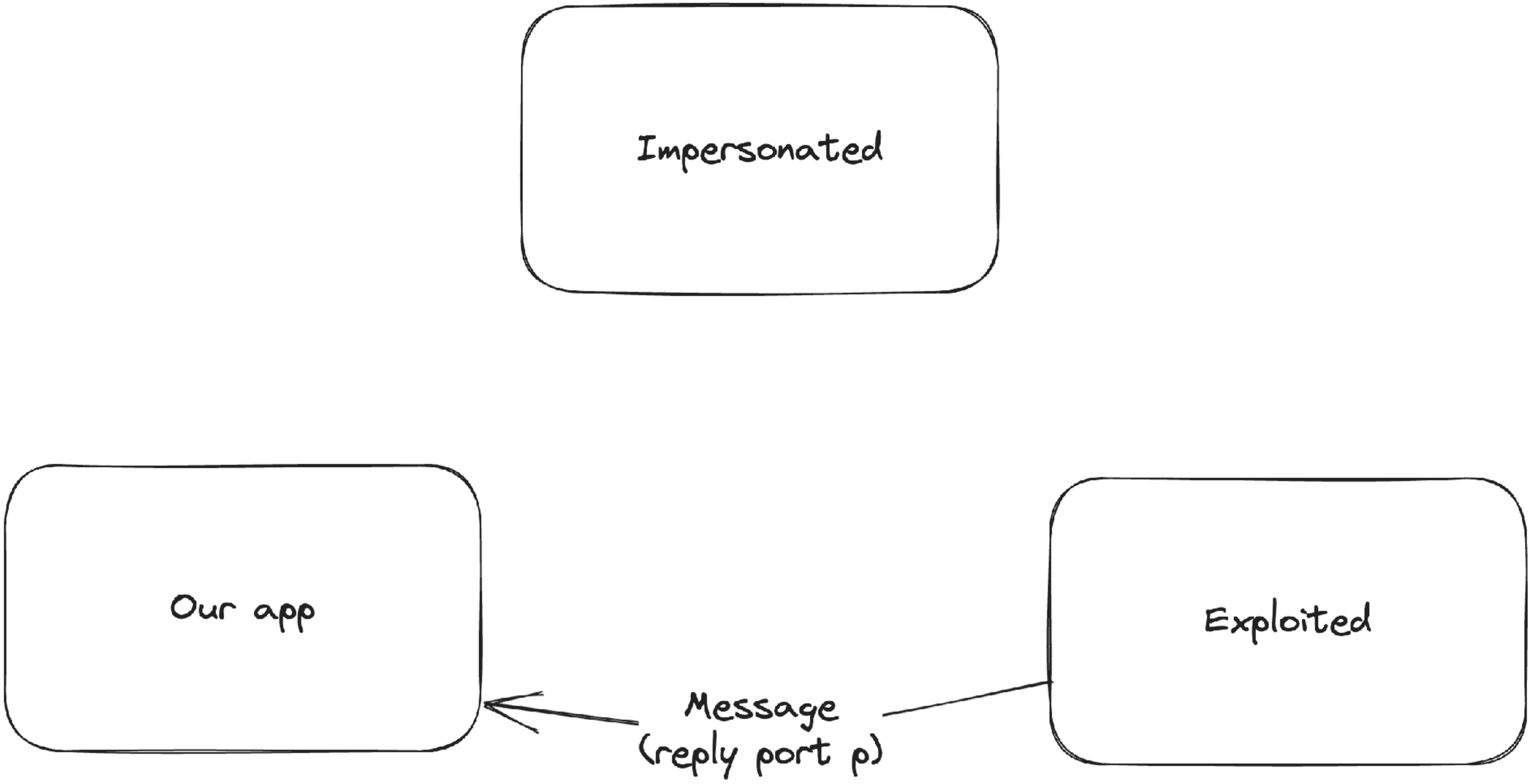
Variant 2

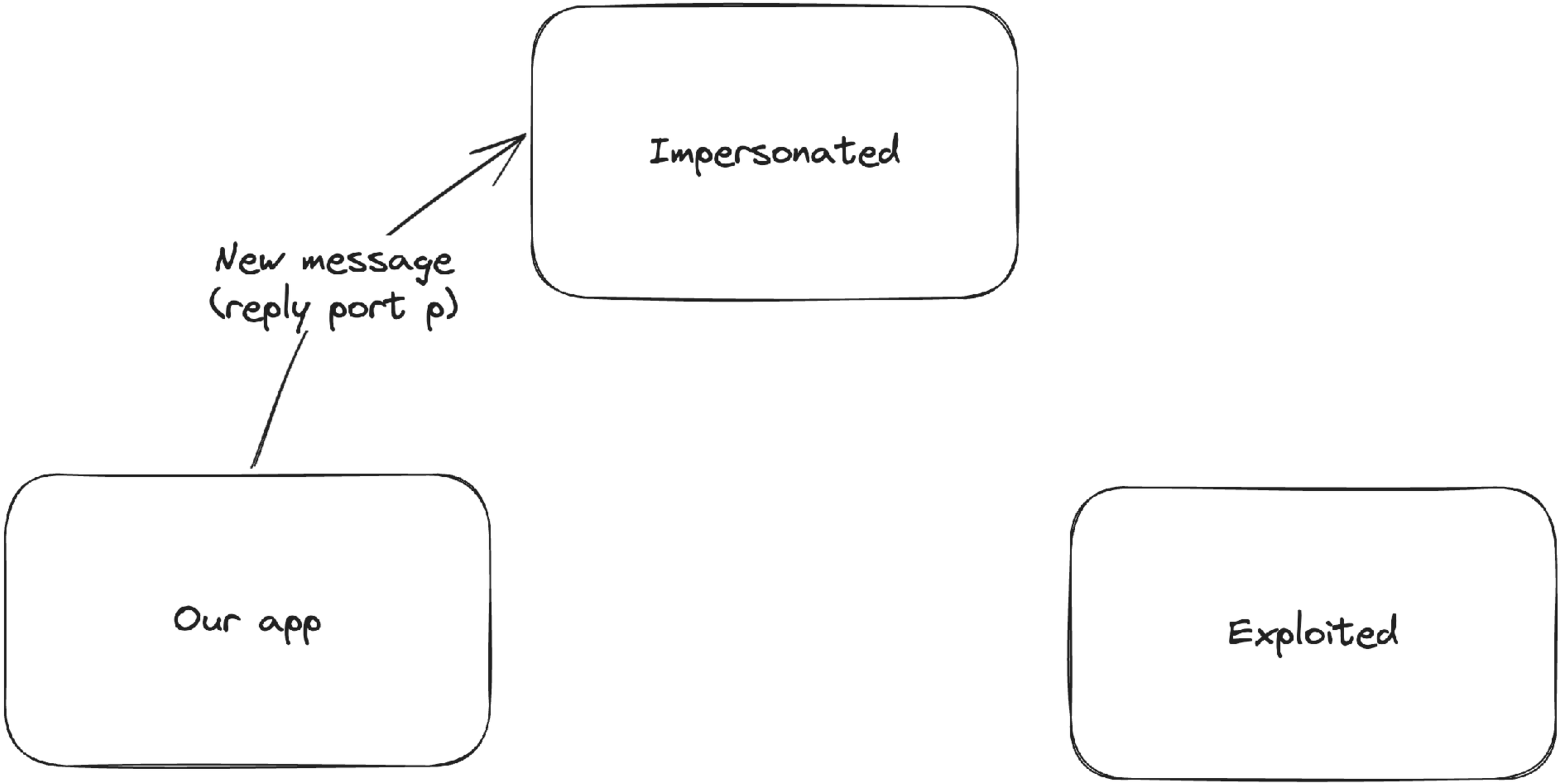
Computest
always on.

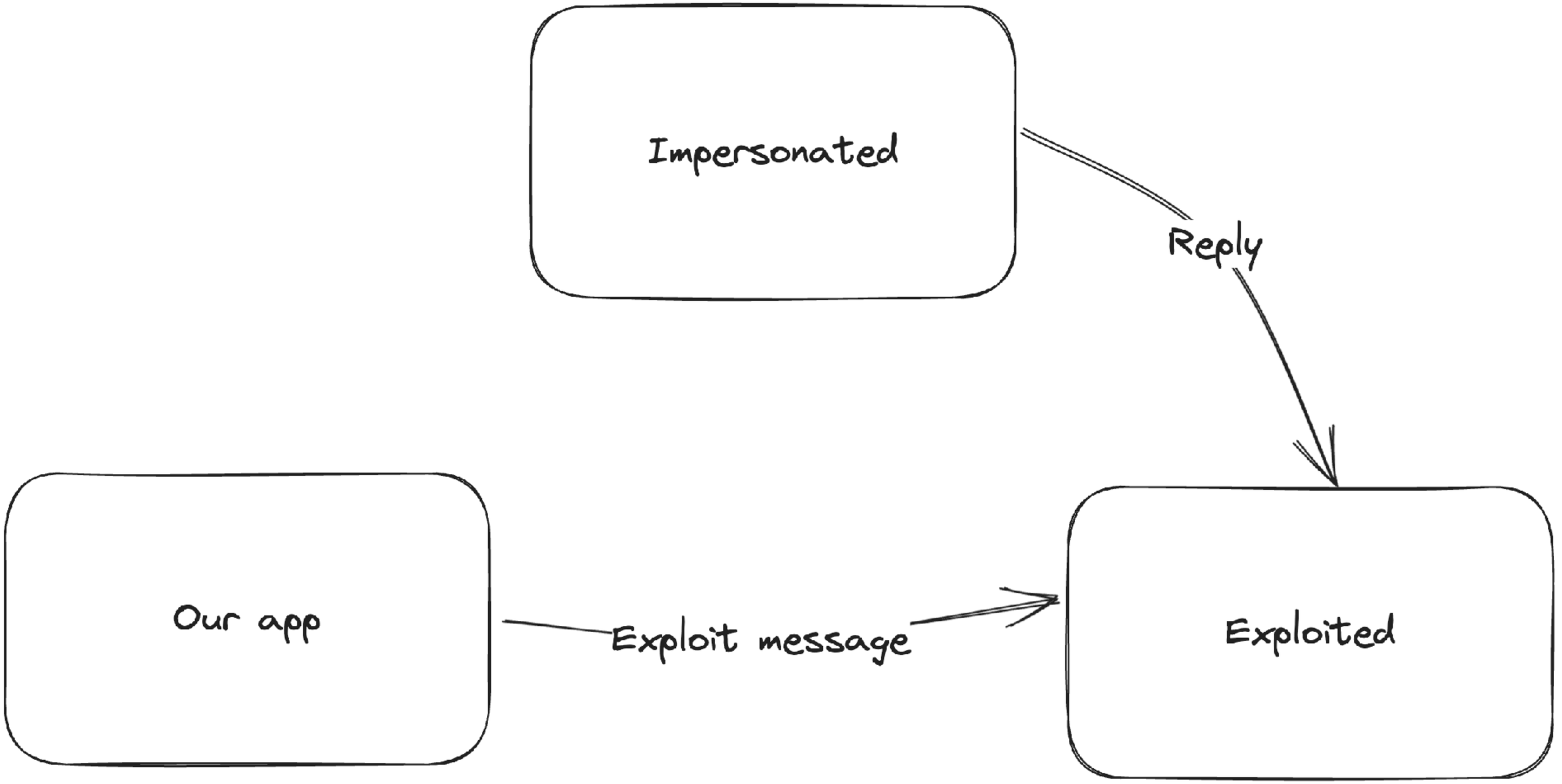
>>

Other instances?

- Second variant: **reply** overwriting the audit token of a **normal message**
- XPC reply messages (typically) get handled on a different dispatch queue than normal messages
- Needs the mach service to send a message to my app which expects a reply
- Concurrently receiving a normal message and a reply can overwrite the audit token of the message with that of the reply, **even if the message is handled synchronously**


Impersonated
Our app
Exploited
Message
(reply port p)


Impersonated
New message
(reply port p)
Our app
Exploited


Impersonated
Reply
Our app
Exploit message
Exploited

Other instances?

> No instances found, but replicated with two custom launch agents

test_mach_service	Reply received: <OS_xpc_dictionary: dicti
test_mach_service	*** AUDIT TOKEN CHANGED!!!
test_mach_service	Got message: <OS_xpc_dictionary: dictiona
second_mach_service	Message received: <OS_xpc_dictionary: dic
test_mach_service	Reply received: <OS_xpc_dictionary: dicti
test_mach_service	Got message: <OS_xpc_dictionary: dictiona
second_mach_service	Message received: <OS_xpc_dictionary: dic
test_mach_service	Reply received: <OS_xpc_dictionary: dicti
test_mach_service	*** AUDIT TOKEN CHANGED!!!
test_mach_service	Got message: <OS_xpc_dictionary: dictiona
second_mach_service	Message received: <OS_xpc_dictionary: dic

Summary

- Verifying a connection **before accepting** is not vulnerable (e.g. `-listener:shouldAcceptNewConnection:`)
- XPC event handlers for a connection are never called concurrently
  1. But, if `xpc_connection_get_audit_token` is called **asynchronously** (i.e. not from an event handler), it may get the wrong audit token
  2. Receiving a **reply** concurrently with a **message** can also trigger the audit token to be wrong

More instances

Maybe in iOS?

Computest
always on.

>>

Other instances?

- Impact on macOS was demonstrated
- But one issue doesn’t warrant a structural fix, does it also affect iOS?
- Spent many days (spread out over more than a year) trying to find an instance where it is exploitable on iOS

Other instances?

- Used **Frida** to look for `xpc_connection_get_audit_token`, used the backtrace to check if not in an XPC event handler
  - (Using a **Security Research iPhone**, if you're interested, sign-ups are open!)
  - Only finds functionality that is used and in the monitored daemon
- Also decompiled a lot of iOS mach services, very time intensive
- Tried writing a **Ghidra script** to search for paths from `xpc_connection_set_event_handler` to `dispatch_async` and then from that block to `xpc_connection_get_audit_token`…
  - Parrsing blocks and the shared dyld cache made this very challenging
- Gave up, submitted what we had

The fix

- Suggested structural fix: **drop a message if the audit token is not equal to the saved audit token**
  - UID may change, but PID and PID version really should **never** change
- Apple's fix: changed the vulnerable call to `xpc_connection_get_audit_token` to `xpc_dictionary_get_audit_token` in `smd`
  - `xpc_dictionary_get_audit_token` uses the audit token from the mach message's trailer, which is safe
- Issue may still affect other services, so… good luck anyone else who wants to look at this!

Hardening advice for XPC services

- Drop the connection after receiving an unrecognised message
- Perform authorisation checks **before accepting** a connection wherever possible
  - If not, use `xpc_dictionary_get_audit_token`
  - Alternatively, save the audit token during the accept handler and use that (works for `NSXPCConnection` too)
- There are new APIs to automatically perform a codesigning check before accepting:
  - `-[NSXPCConnection setCodeSigningRequirement:]` (since macOS 13.0)
  - `xpc_connection_set_peer_code_signing_requirement` (since macOS 12.0)

Summary

- XPC is a one-to-one communication channel, built on mach messages, which are a multiple sender, single receiver communication channel
- We can create an XPC connection with multiple senders. The audit token for such a connection becomes unreliable
- We have shown how we exploited this to achieve privilege escalation on macOS using `smd`
- Apple has not addressed this structurally, so this may still affect other services and platforms

Want to read our full write-up?

SECTOR7
powered by Computest

sector7.computest.nl

Bonus: sandbox escape

- The `smd` exploit still works if sandbox is enabled! 🥳
  - But it does require including the to be installed helper tool in the app's bundle and Info.plist…
- Exploiting existing software won't work
- Mac App Store review will almost certainly reject this (easy static check)
- Would only help to convince users to download and run the app: "it's safe, just look at the sandbox entitlement!"

Bonus: sandbox escape

- More generally: the sandbox prevents mach service **lookups**, so if you transfer the mach ports of an XPC connection to a sandboxed process, it can use it fine!
- Suppose you have:
  - Unsandboxed code execution as a normal user
  - Sandboxed code execution as root
- Then opening a connection as the normal user and moving it to the root process could offer certain opportunities
  - In `smd` routine 1004 doesn't work anymore, but maybe 1000 does something interesting?