is your app secure?

revelations a Jedi wouldn’t tell you

Kabir Oberai

Sun Tzu:

if you know neither the enemy nor yourself, you will succumb in every battle.

if you know yourself but not the enemy, for every victory gained you will also suffer a defeat.

if you know the enemy and know yourself, you need not fear the result of a hundred battles.

⇒ **threat model**

know yourself // assets

what am I protecting?

misconception: nothing to protect?

I talk to a server:

- 🔑 api keys
- 👥 other users' data
- 🤐 company secrets
- 💯 uptime

I'm local-first:

- 🔒 file access
- 💪 data integrity
- 📵 device availability
- 💵 purchases

I… exist:

- 📱 users' devices
- 📣 your reputation
- 😶 users' reputation

know yourself // assets

what am I protecting?

misconception: nothing to protect?

I talk to a server:

- 🔑 api keys
- 👥 other users' data
- 🤐 company secrets
- 💯 uptime

I'm local-first:

- 🔒 file access
- 💪 data integrity
- 📵 device availability
- 💵 purchases

I… exist:

- 📱 users' devices
- 📣 your reputation
- 😶 users' reputation

know yourself // assets

what am I protecting?

another perspective

confidentiality

- 🔑 api keys
- 👥 other users' data
- 🤐 company secrets
- 🔒 file access

integrity

- 💪 data integrity
- 💵 purchases
- 📣 your reputation
- 😶 users' reputation

availability

- 📱 users' devices
- 💯 uptime
- 📵 device availability

know yourself // attack surface

what are my weak spots?

short answer: *all input* is untrusted
all output should be controlled.

} trust boundaries

know yourself // attack surface

what are my weak spots?

…but Swift!

Swift is a **general-purpose** programming language that's **approachable** for newcomers and **powerful** for experts.

It is **fast**, **modern**, **safe**, and a **joy** to write.

```
struct Binomial: Codable {
    var genus: String
    var species: String
    var subspecies: String?
}

let tree = Binomial(genus: "Pin", species: "oak")
let jsonData = try JSONEncoder().encode(tree)

// {"genus":"Pin","species":"oak"}
```

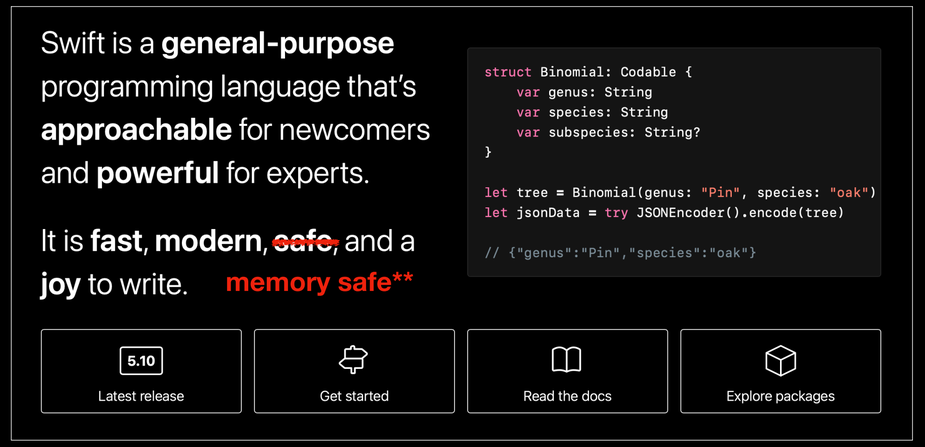
5.10 Latest release | Get started | Read the docs | Explore packages

know yourself // attack surface

what are my weak spots?

…but Swift!

- memory safety != logical safety
- not swift all the way down



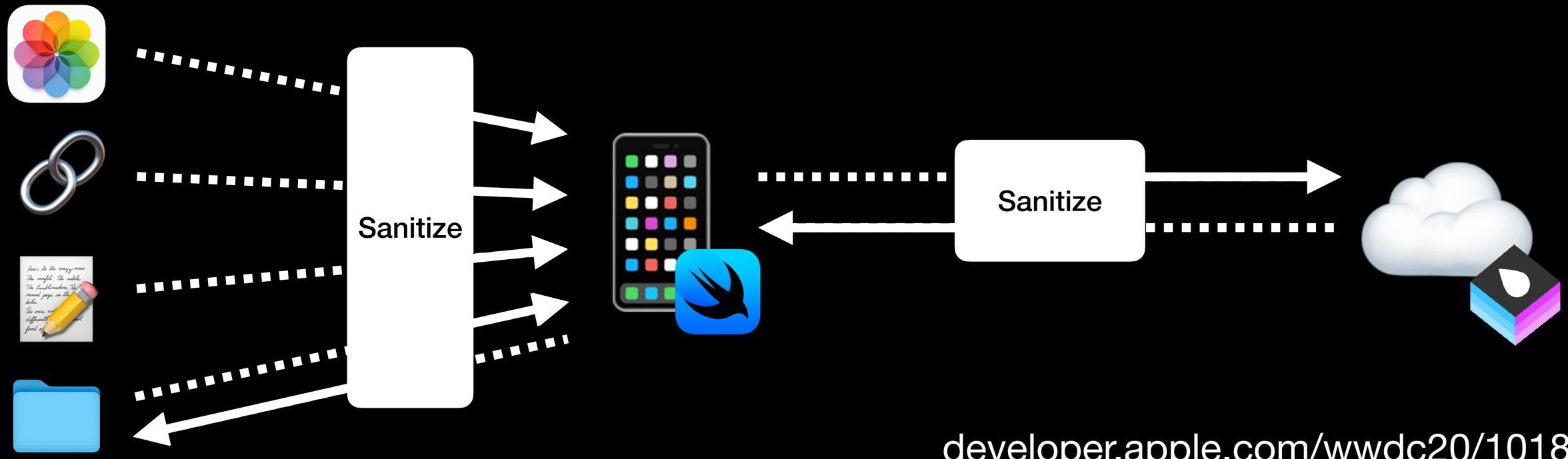
C image decoder?

filename overwrites something?

know yourself // attack surface

what are my weak spots?

sanitize ⇒ check validity, access level



developer.apple.com/wwdc20/10189

interlude

meet our star

Yelp

interlude

meet our star

Yelpepperoni*

**we forgot to hire a legal team*

interlude

meet our star

Yelpepperoni

an app for pizzeria discovery!

- view pictures and details of pizzerias
- manage pizzerias you own
- classify pizza pictures with cutting-edge pizza detection AI
- buy **YelpeppePROni*** for amazing discounts

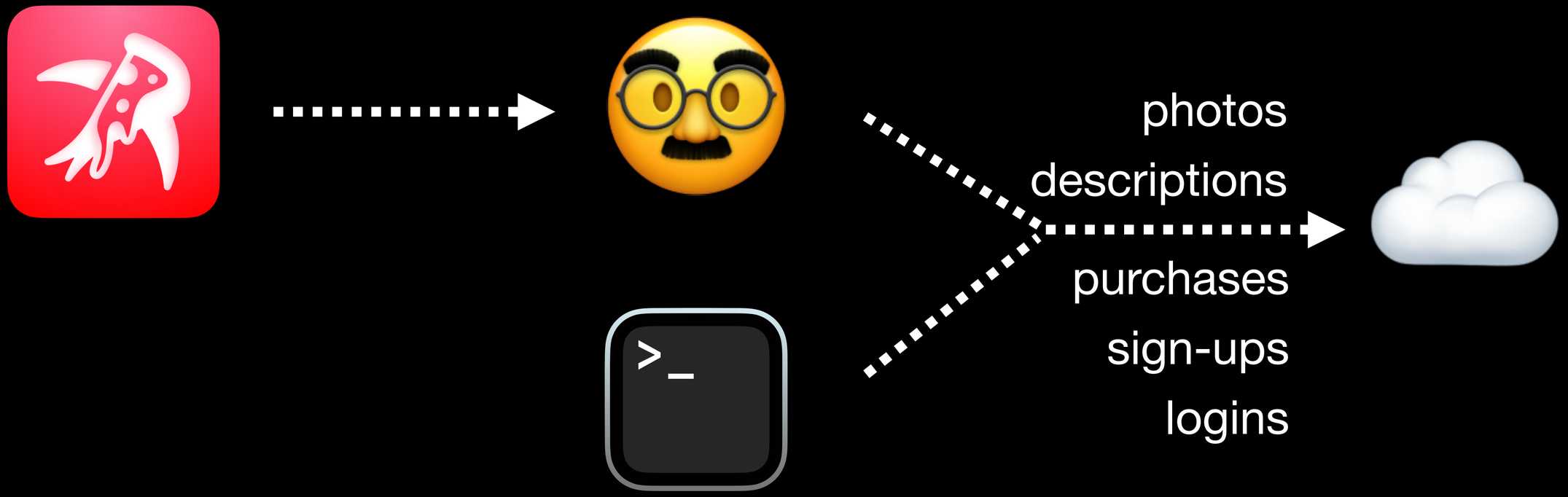
**we also forgot to hire a marketing team*

knowing ourselves

attack surface (server)

knowing ourselves

attack surface (server)



reminder: input is untrusted! *don't trust the client.*

knowing ourselves

attack surface (client)

knowing the attacker

onto the fun stuff!

knowing the attacker

securing integrity

scenario: attacker is the user. attempting to compromise in-app purchases.

```
transaction != nil
```

trust assumptions:

📱 **user hasn’t modified the system**

jailbreaking is game over for the client. but it’s getting harder, so let’s set this aside.

🔧 **user hasn’t modified your app’s code**

can the user modify your app without a jailbreak?

knowing the attacker

securing integrity

scenario: attacker is the user. attempting to compromise in-app purchases.

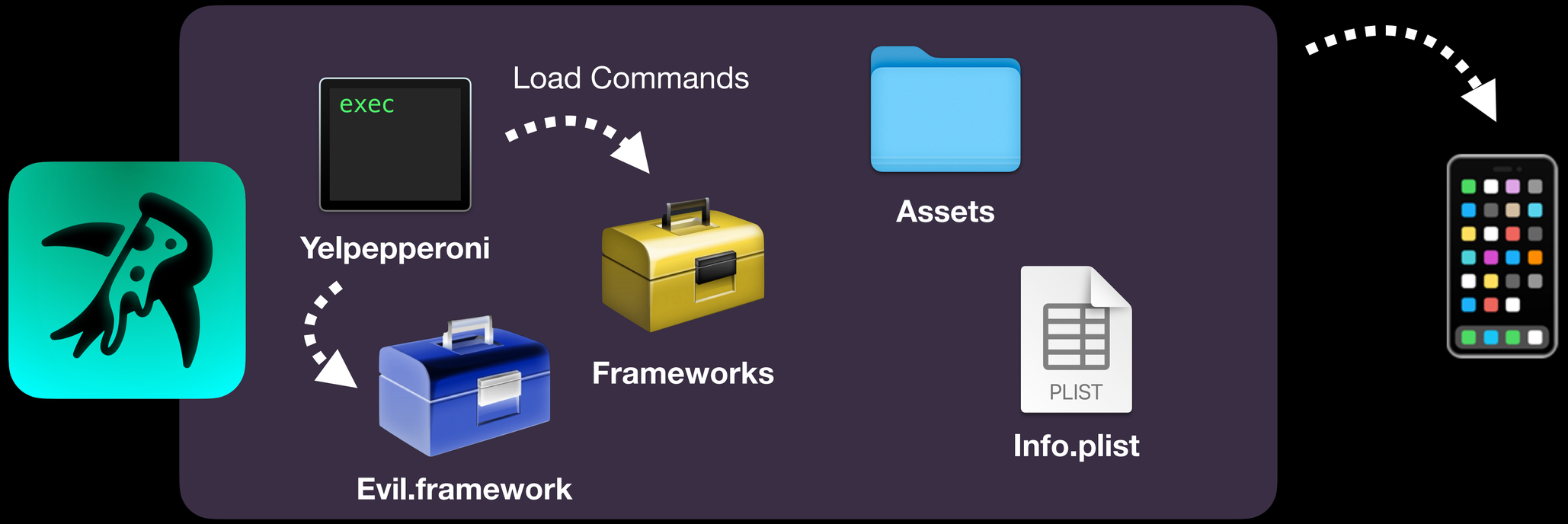
1. **extract Yelpepperoni.ipa**

   third-party service / Apple Silicon / jailbroken device (one-off)

knowing the attacker

securing integrity

scenario: attacker is the user. attempting to compromise in-app purchases.



2. inject

3. sideload

knowing the attacker

securing integrity

scenario: attacker is the user. attempting to compromise in-app purchases.

```
+[Transaction all] { ... }
```

ObjC runtime swizzling /
bespoke Swift hackery

Yelpepperoni

```
return [Transaction(purchased: true)]
```

Evil.framework

knowing the attacker

securing integrity

scenario: attacker is the user. attempting to compromise in-app purchases.

```
+[Transaction all] { return [Transaction(purchased: true)] }
```

Yelpepperoni

Evil.framework

knowing the attacker

securing integrity

scenario: attacker is the user. attempting to compromise in-app purchases.

knowing the attacker

securing integrity

scenario: attacker is the user. attempting to compromise in-app purchases.

see also: developer.apple.com/documentation/appstoreservernotifications

demo

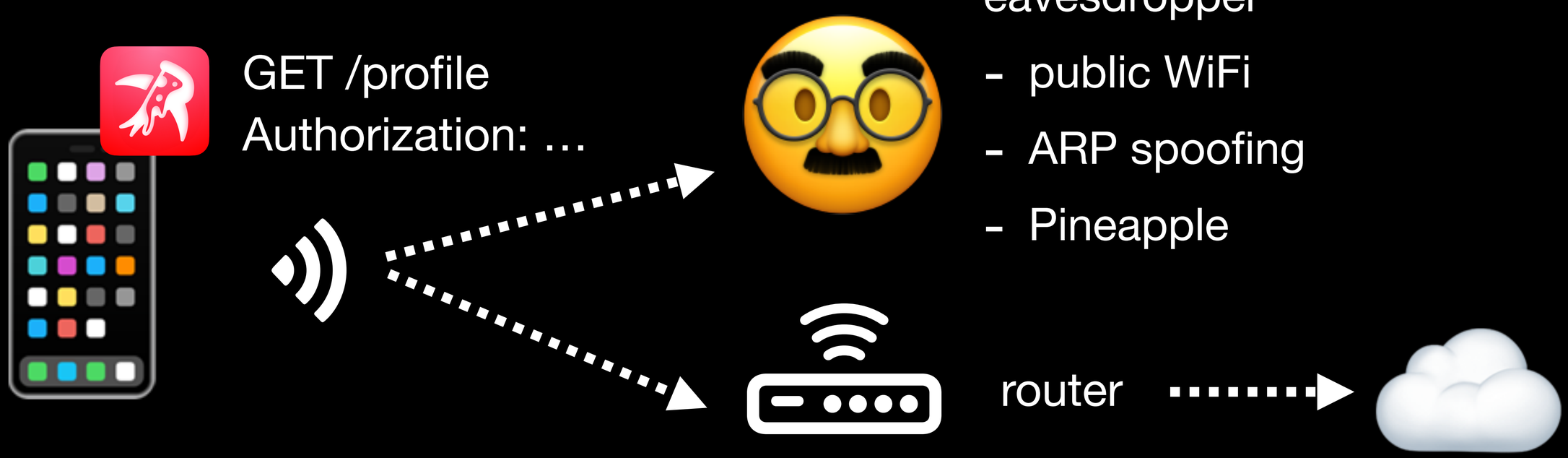
receipt validation

knowing the attacker

securing confidentiality

scenario: user is a pizzeria owner.

adversary is nearby; no access to user device.

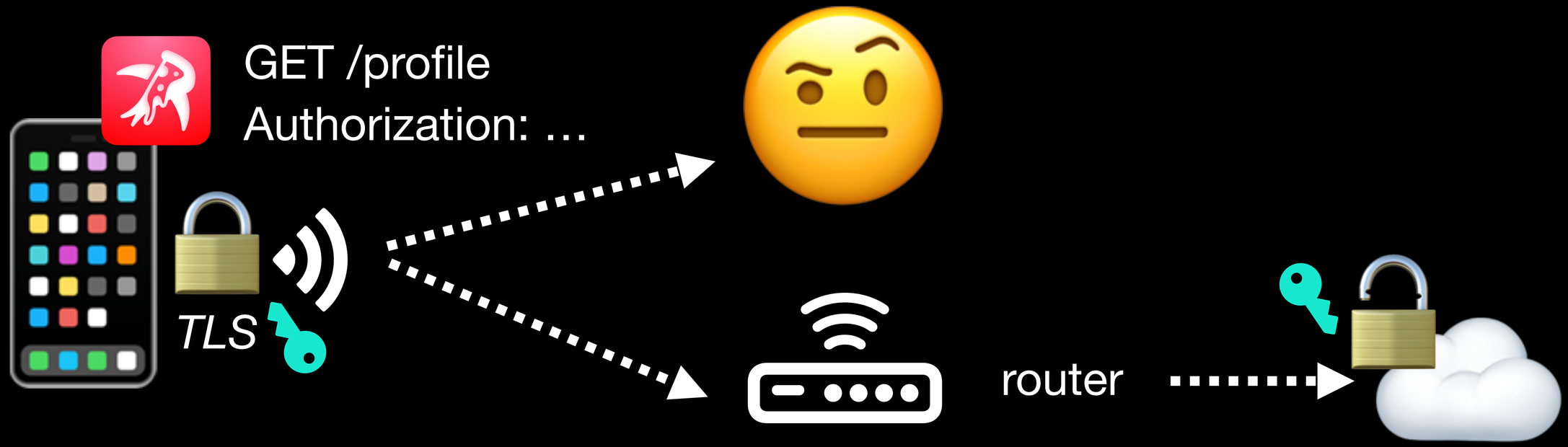


knowing the attacker

securing confidentiality

scenario: user is a pizzeria owner.

adversary is nearby; no access to user device.

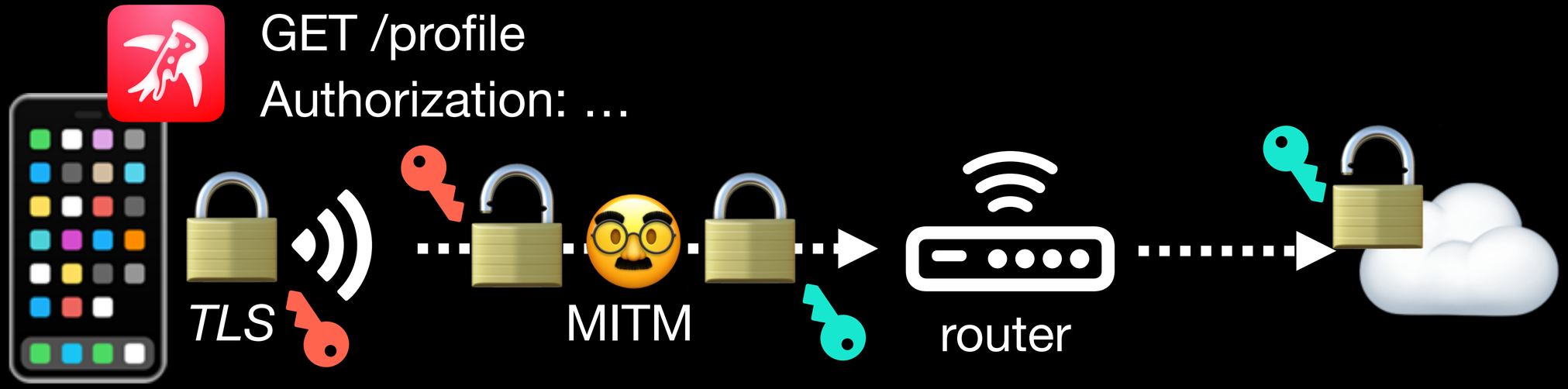


knowing the attacker

securing confidentiality

bypass TLS? 1. manually trust 🔑

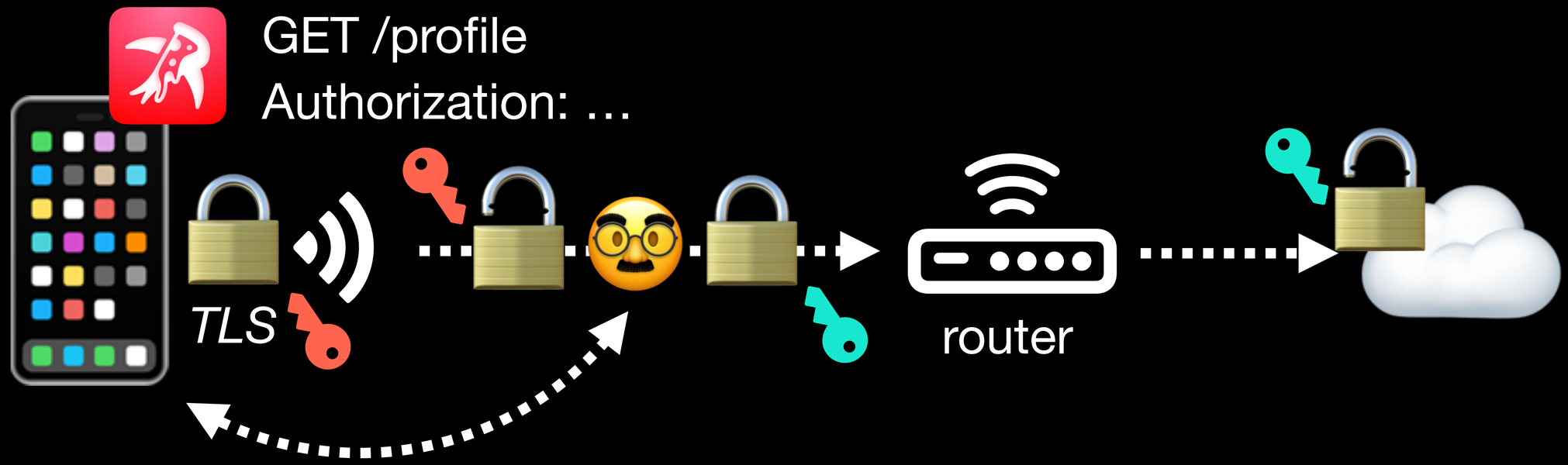
2. intercept requests: 🔑 ⇔ 🔑



knowing the attacker

securing confidentiality

scenario: attacker == user



knowing the attacker

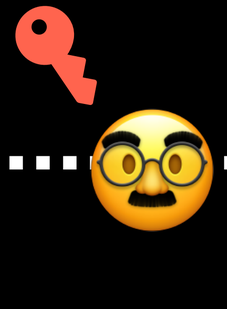
securing confidentiality

*how do we secure against **this**?*

swizzling!

SSL Pinning? `if key != 🔑 { throw InsecureError() }`

API key = sk-…



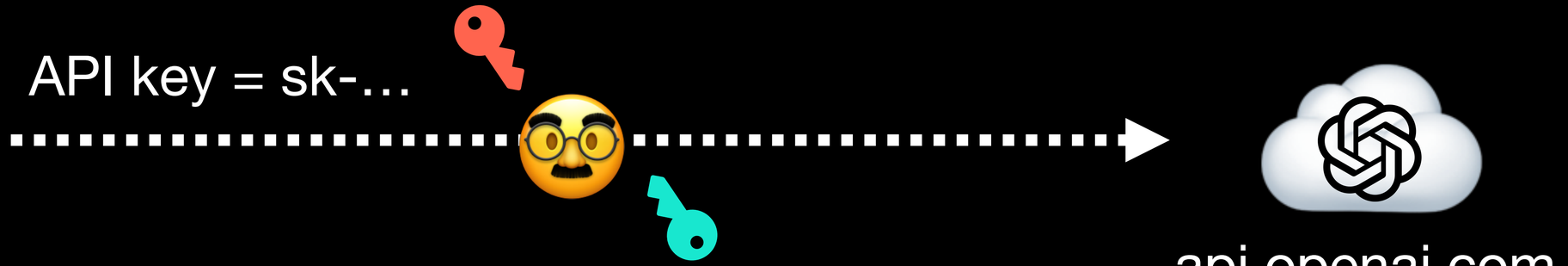
api.openai.com

knowing the attacker

securing confidentiality

*DO we secure against **this**?*

Kirchoff's Principle: *one ought to design systems under the assumption that the enemy will immediately gain full familiarity with them.* security through obscurity is bad.



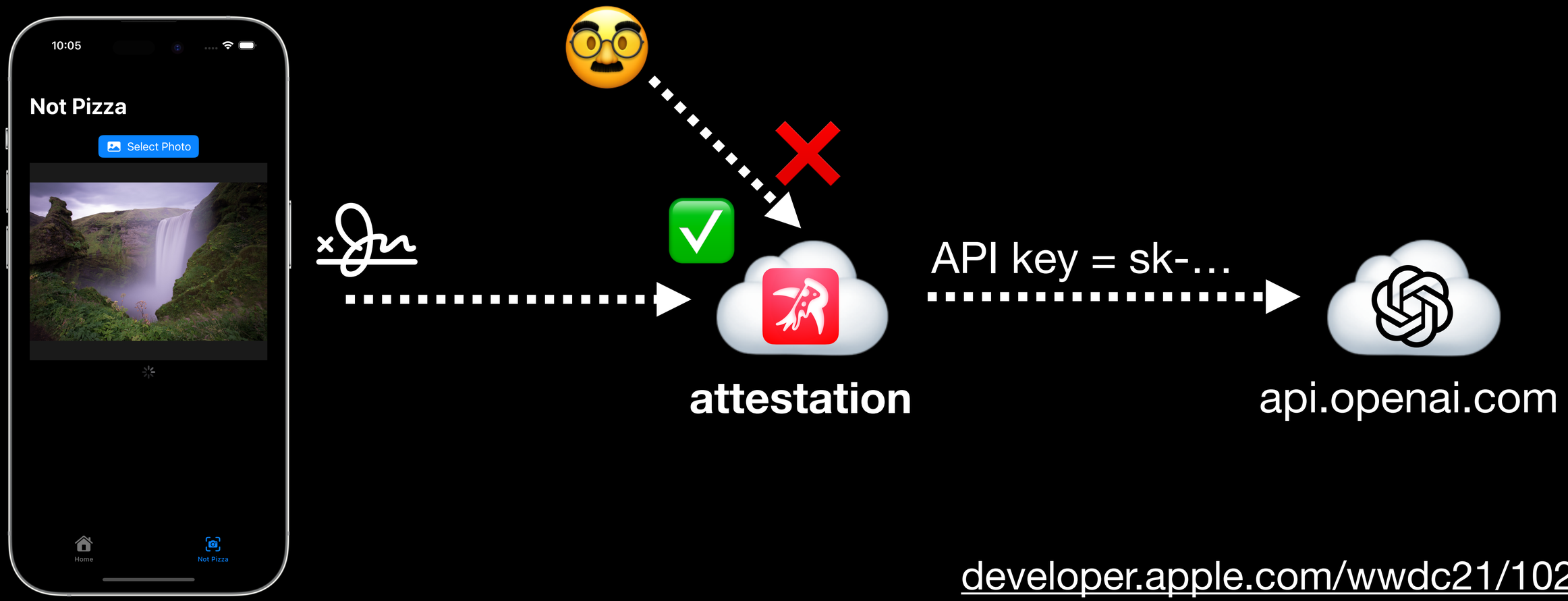
knowing the attacker

securing confidentiality

developer.apple.com/wwdc21/10244

knowing the attacker

securing confidentiality



developer.apple.com/wwdc21/10244

knowing the attacker

“don’t trust the client”

then why bother with client-side security?

different threat models!

knowing the attacker

securing availability

scenario: user is non-malicious. attacker can upload pizzerias.


```
let dest = photosDir.appending(
  path: photo.filename
)

download(photo.id, to: dest)
```

```
{
  name: "My Pizzeria",
  photos: [
    {
      filename: "photo.png",
      id: "..."
    }
  ]
}
```

knowing the attacker

securing availability

scenario: user is non-malicious. attacker can upload pizzerias.

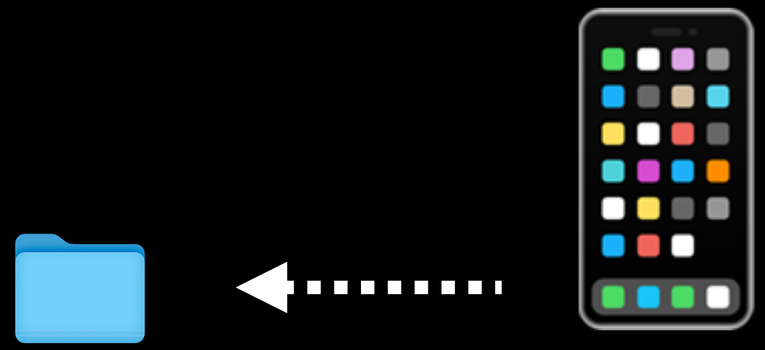

```
let dest = photosDir.appending(
  path: "../../Preferences.plist"
)

download(photo.id, to: dest)
```

```
{
  name: "My Pizzeria",
  photos: [
    {
      filename: "../../Preferences.plist",
      id: "..."
    }
  ]
}
```

knowing the attacker

securing availability



```swift
let dest = photosDir.appending(
  path: photo.filename
)       // ^ or use UUID here if possible

if !dest.isContained(in: photosDir) { throw }

download(photo, to: dest)
```

```swift
extension URL {
  func isContained(in parent: URL) -> Bool {
    let sanitizedParent = URL(
      filePath: parent.path(),
      directoryHint: .isDirectory
    ).standardized

    let sanitizedPath = URL(
      filePath: path()
        .replacingOccurrences(of: "//", with: "/")
    ).standardized

    return sanitizedPath.absoluteString
      .hasPrefix(sanitizedParent.absoluteString)
  }
}
```

```swift
struct FilePath {
  func lexicallyResolving(_ subpath: FilePath) -> FilePath?
}
```

knowing the attacker

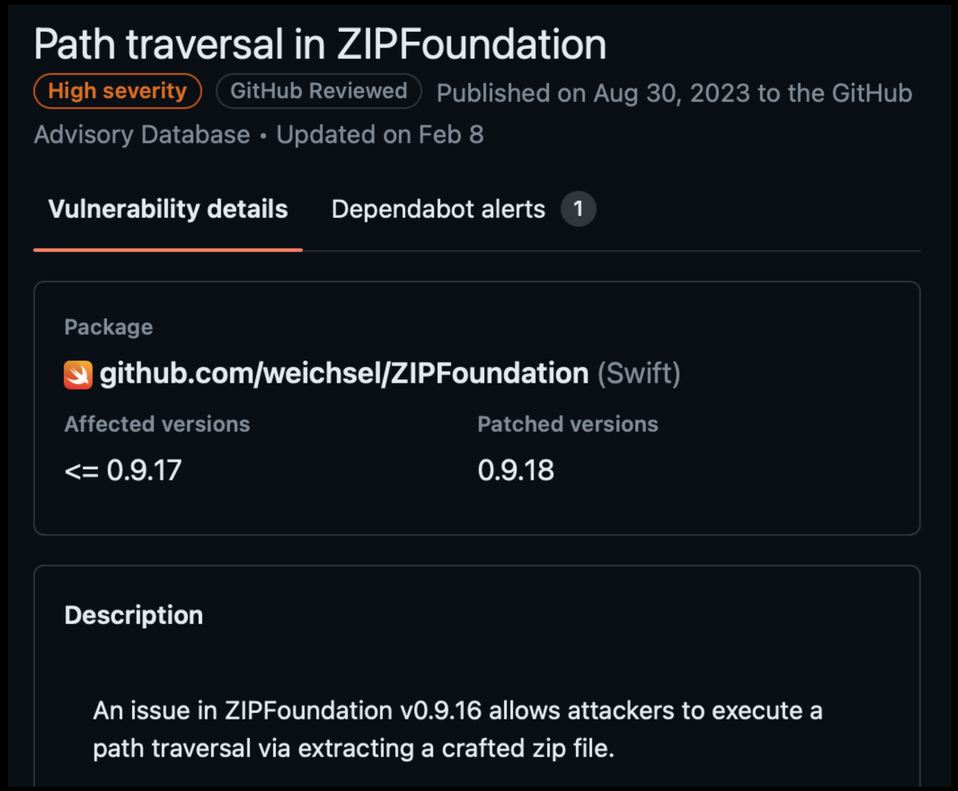
securing availability

not theoretical: this one's from ZIPFoundation!

```swift
extension URL {
  func isContained(in parent: URL) -> Bool {
    let sanitizedParent = URL(
      filePath: parent.path(),
      directoryHint: .isDirectory
    ).standardized

    let sanitizedPath = URL(
      filePath: path()
        .replacingOccurrences(of: "//", with: "/")
    ).standardized

    return sanitizedPath.absoluteString
      .hasPrefix(sanitizedParent.absoluteString)
  }
}
```



demo

path traversal

conclusion

knowledge is power

🦺 what are you protecting?

📥 what goes in, what comes out?

❗ how can different adversaries violate your assumptions?

get in touch!

kabiroberai@mastodon.social
@kabiroberai
/in/kabiroberai
me@kabiroberai.com

slides & code
ober.ai/dds