Mobile Security Theater

or why **YOU** should pay $1,000,000 for two regex queries!1!

Miłosz Gaczkowski

WithSecure Consulting

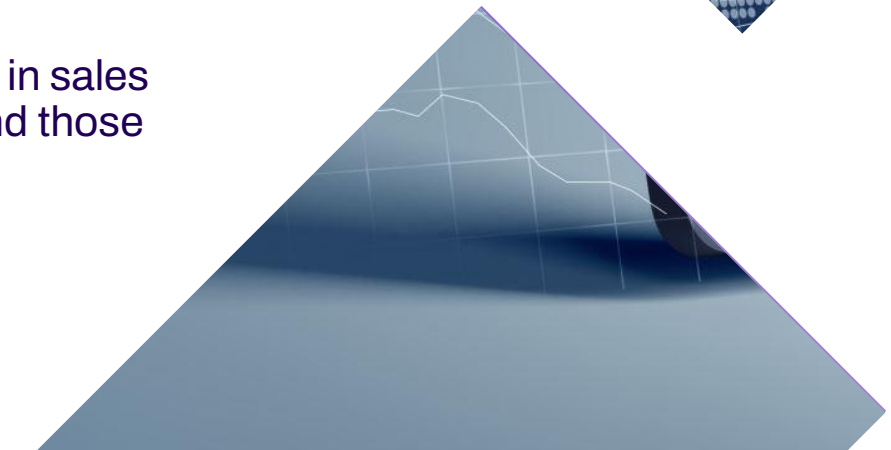
WITH secure

whoami

- Miłosz Gaczkowski
  - /ˈmi.wɔʂ/
- Past life: University teaching
  - Computer science
  - Cybersecurity
- Current life: Mobile Security Lead at WithSecure Consulting
  - Android/iOS apps
  - Android devices
  - BYOD Mobile Application Management setups

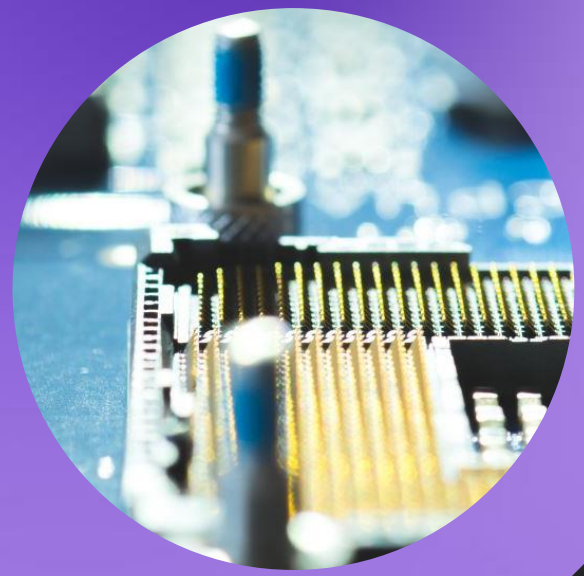
Today's talk

- We've done a lot of testing of MAM solutions, individual applications with MAM support, and additional security apps
- Some of them are better than others, but all have serious issues
- The two big mobile platforms – iOS and Android – simply do not have the tools to support good MAM approaches
  - (tbf, neither does any laptop or other device)
- However, the push from industry is strong – so MDM providers try their best to provide MAM solutions
- I want to show you some funny trends we've seen, without naming specific vendors
  - They have a really tough job here – someone in sales promised the world to large organisations, and those teams now have to deliver the impossible



Managed (and not-so-managed) Devices

What's a "MAM" anyway?



WITH secure

MDM vs. MAM

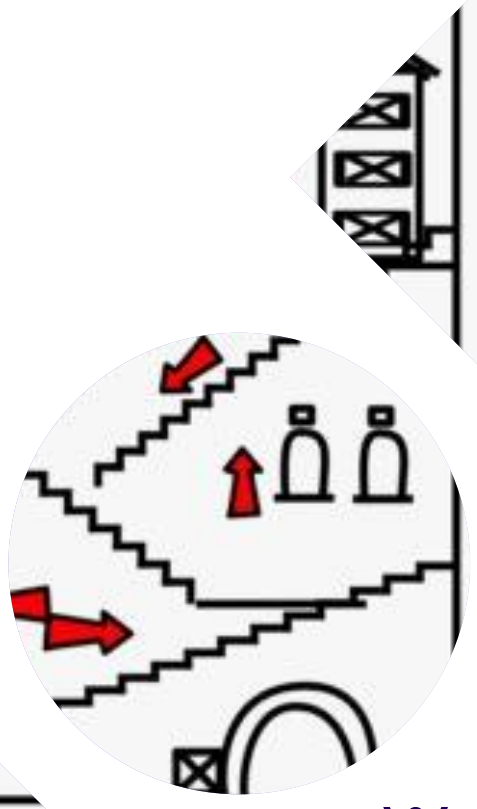
- Today’s talk is all about the use of mobile phones **at work**
- Accessing **work** e-mails, chat messages, etc., on the go
- Could be a lot of sensitive stuff in there
- Stuff your boss doesn’t want public

MDM vs. MAM

- Historically, the solution has been obvious:
- Workplace buys you a phone
- You use it, but **they own it**
- Logically: they **control** it
  - They can lock and wipe “your” device
  - They decide what apps go on the phone
  - What VPNs/proxies you connect to
  - (Maybe?) They have some level of **surveillance**, too
- Solutions that facilitate this approach are called **Mobile Device Management** solutions
  - Usually: a management profile that restricts your device
  - Usually: a set of apps with admin rights that can change device settings
  - Sometimes: additional tools, like an antivirus
  - Usually: a cloud service to co-ordinate a fleet of devices, deploy config at scale, etc.

MDM vs. MAM

- Now, MDM works very well, but it has its problems
- Company needs to own the devices
  - Can’t reasonably ask user to sign off admin rights to their own phone
  - Expensive!
- Settings usually very restrictive on users
  - If a user needs to quickly install an app, they’re out of luck
  - IT needs to change your profile
  - This probably means exceptions/approvals
  - Have fun searching for permit A38
- It’s tempting to let users bring their own devices into work
  - But what about security?



MDM vs. MAM

- MAM – Mobile **Application** Management
- Rather than controlling the whole device, we create a sandbox to protect just corporate apps
  - Typically: no copy-paste outside the sandbox
  - No screenshots
  - No file sharing between MAM and unsandboxed apps
  - Company can block access/wipe data of managed apps, but nothing else
- Started around 2010, but **really picked up** during early COVID
- In theory: best of both worlds
  - User retains control of their own device
  - Company doesn't have to buy 1000s of devices, keep up with maintenance
  - No need to deal with complex processes over whether Employee X should be allowed to install Spotify/TikTok/Google Maps
  - Company data still protected
- Approach has its problems, and it's not an uncommon opinion that "you can't secure BYOD"

Example 1

“Malicious WiFi detection”

“Malicious WiFi detection”

- Part of a mobile threat detection solution
- Problem: user may control to any WiFi – what if it does something dodgy?
  - DNS spoofing?
  - Person-in-the-Middle?
  - Evil twin attacks?
- Promise: No problem, our defensive security product will identify them!
  - (but we won’t tell you what we detect, or how)
- Our task: assess this solution and comment on its value
- Let’s get reversing!

One reverse engineering later...

- OK, so the application that does it doesn't have much privileged access to the actual networks
- Remember, this is in a MAM context – the user owns the phone, so the software can't have much control
- Application fetches detection rules from a server – so they could be dynamically updated
- Let's have a look at the config!

```
networkProtection': {
    "RogueAP": [
        {
            "id": 1,
            "OUI": "00:C0:CA"
        },
        {
            "id": 2,
            "ssid": "^Pineapple_[0-9a-fA-F]{4}$"
        },
        {
            "id": 3,
            "prviateIp": "^172\\.16\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\. (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$",
            "ssid": "^Pineapple_[0-9a-fA-F]{4}$"
        },
        {
            "id": 4,
            "ssid": "^Pineapple_[0-9a-fA-F]{4}$"
            "OUI": "00:13:37"
            "prviateIp": "^172\\.16\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\. (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$",
        },
    ],
}
```

```
networkProtection’: {
    “RogueAP”: [
        {
            “id”: 1,
            “OUI”: “00:C0:CA”
        },
        {
            “id”: 2,
            “ssid”: “^Pineapple_[0-9a-fA-F]{4}$”
        },
        {
            “id”: 3,
            “prviateIp”: “^172\\.16\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\. (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$”,
            “ssid”: “^Pineapple_[0-9a-fA-F]{4}$”
        },
        {
            “id”: 4,
            “ssid”: “^Pineapple_[0-9a-fA-F]{4}$”
            “OUI”: “00:13:37”
            “prviateIp”: “^172\\.16\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\. (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$”,
        },
    ],
}
```

That’s… it. 4 rules.

Quick summary

- Rule 1: Detect all wireless networks whose OUI is 00:C0:CA
  - OUI – Organizationally Unique Identifier – first 3 octets of MAC address, typically used to identify manufacturer
- Rule 2: Detect all networks whose SSID (name) is “Pineapple_” followed by 4 hex characters
  - e.g., Pineapple_1234, Pineapple_1a2b
- Rule 3: Same as rule 2, but with some additional matching on IP addresses
- Rule 4: Same as rule 3, but also checks if OUI is 00:13:37

- If one of these rules is met, then the network is “malicious”, and we warn the user to disconnect

Quick summary

- Rule 1: Detect all wireless networks whose OUI is 00:C0:CA
  - OUI – Organizationally Unique Identifier – first 3 octets of MAC address, typically used to identify manufacturer
- Rule 2: Detect all networks whose SSID (name) is “Pineapple_” followed by 4 hex characters
  - e.g., Pineapple_1234, Pineapple_1a2b
- Rule 3: Same as rule 2, but with some additional matching on IP addresses
- Rule 4: Same as rule 3, but also checks if OUI is 00:13:37

- If one of these rules is met, then the network is “malicious”, and we warn the user to disconnect
- Pay attention to the last 2 rules
- If rule 3 is met, then rule 2 was already met
- If rule 4 is met, then rule 3 was already met
- ...so those last 2 are redundant!

Quick summary

- Rule 1: Detect all wireless networks whose OUI is 00:C0:CA
  - OUI – Organizationally Unique Identifier – first 3 octets of MAC address, typically used to identify manufacturer
- Rule 2: Detect all networks whose SSID (name) is “Pineapple_” followed by 4 hex characters
  - e.g., Pineapple_1234, Pineapple_1a2b
- ~~Rule 3: Same as rule 2, but with some additional matching on IP addresses~~
- ~~Rule 4: Same as rule 3, but also checks if OUI is 00:13:37~~

- If one of these rules is met, then the network is “malicious”, and we warn the user to disconnect
- Pay attention to the last 2 rules
- If rule 3 is met, then rule 2 was already met
- If rule 4 is met, then rule 3 was already met
- ...so those last 2 are redundant!

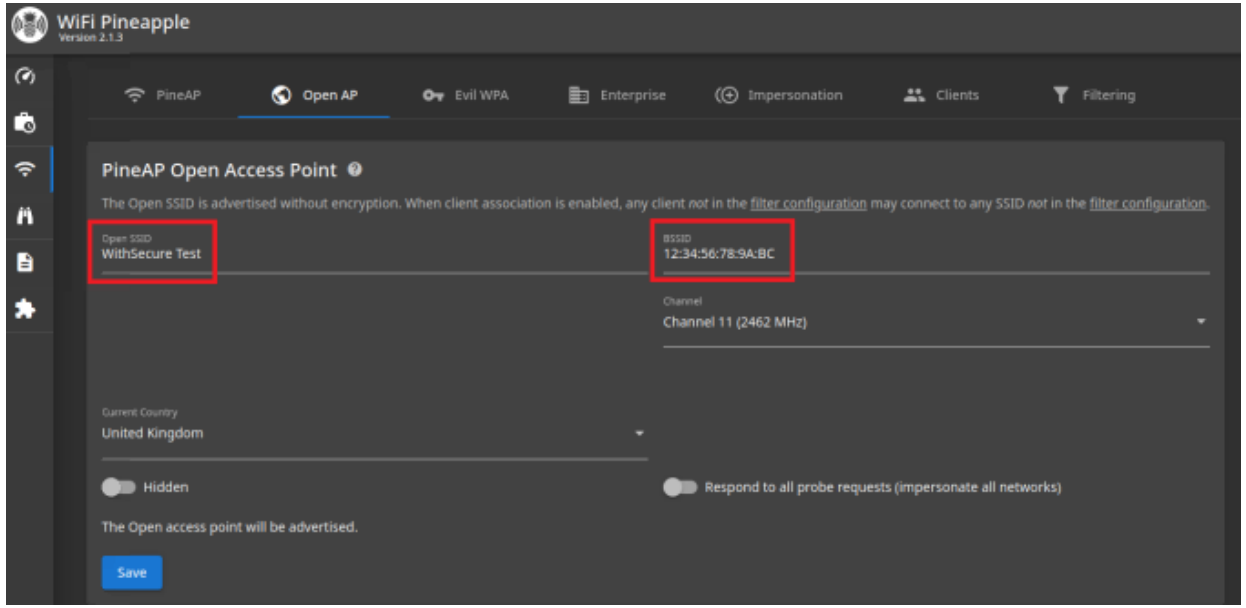
Quick summary

- Rule 1: Detect all wireless networks whose OUI is 00:C0:CA
  - OUI – Organizationally Unique Identifier – first 3 octets of MAC address, typically used to identify manufacturer
- Rule 2: Detect all networks whose SSID (name) is “Pineapple_” followed by 4 hex characters
  - e.g., Pineapple_1234, Pineapple_1a2b
- ~~Rule 3: Same as rule 2, but with some additional matching on IP addresses~~
- ~~Rule 4: Same as rule 3, but also checks if OUI is 00:13:37~~

- So, what’s actually happening here?

OUI rules

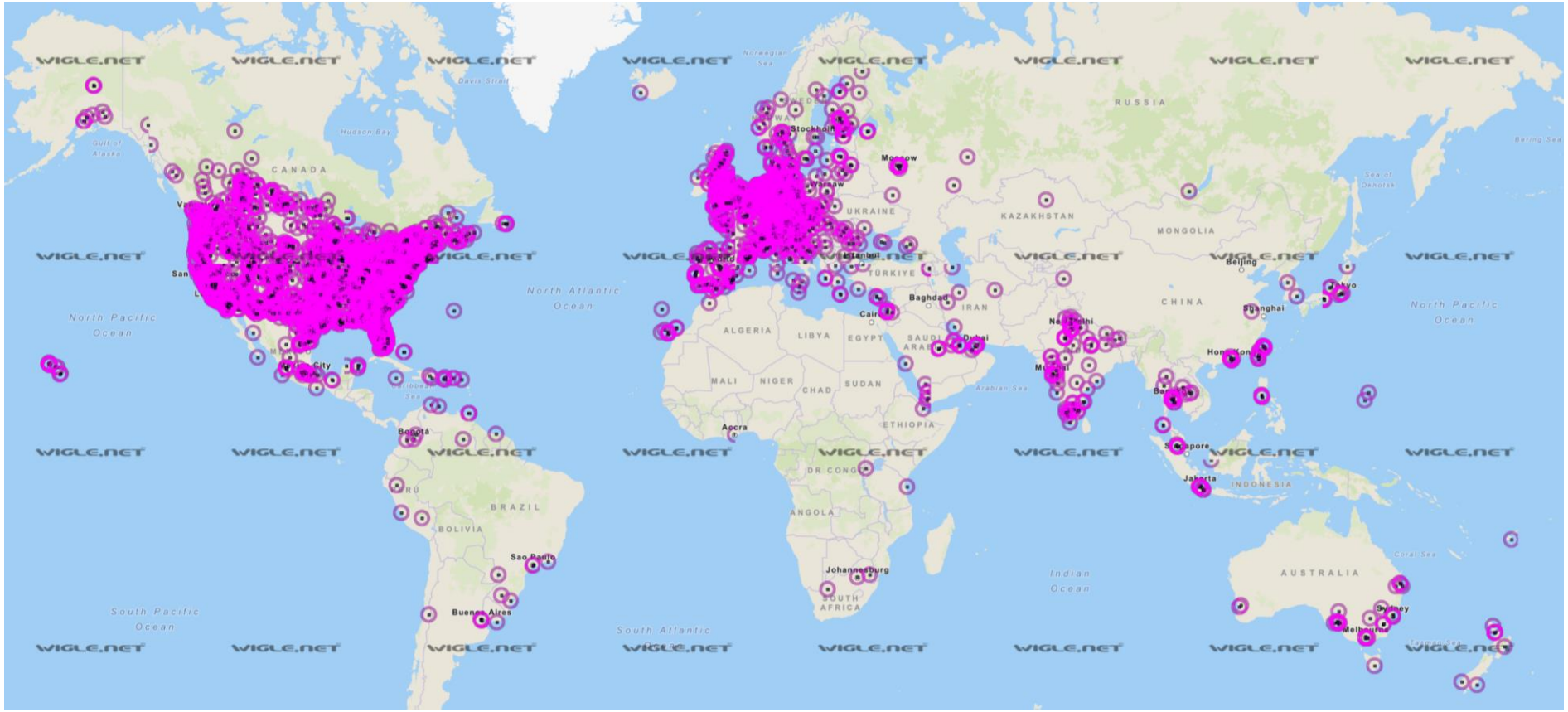
- These are meant to identify devices by 2 manufacturers
  - 00:C0:CA – Alfa, Taiwanese network equipment manufacturer. Their USB dongles are popular among security testers
  - 00:13:37 – WiFi Pineapple by Hak5 – again, common pentesting tool
- Caveat: both devices are capable of spoofing their OUI
  - No real attacker would leave these on their default settings



- Also, Alfa makes a lot of network equipment, not just pentesting tools. I wonder how many of them are out there in the wild...

OUI rules

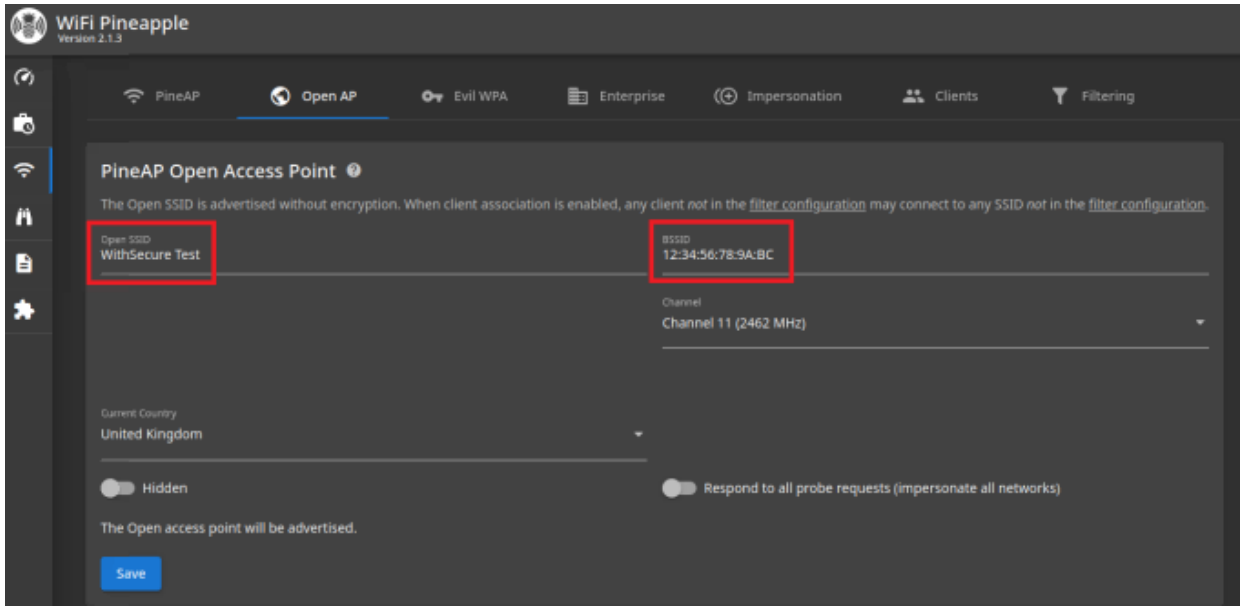
- Also, Alfa makes a lot of network equipment, not just pentesting tools. I wonder how many of them are out there in the wild...
- WIGLE.NET can help us find out!



- Oh.

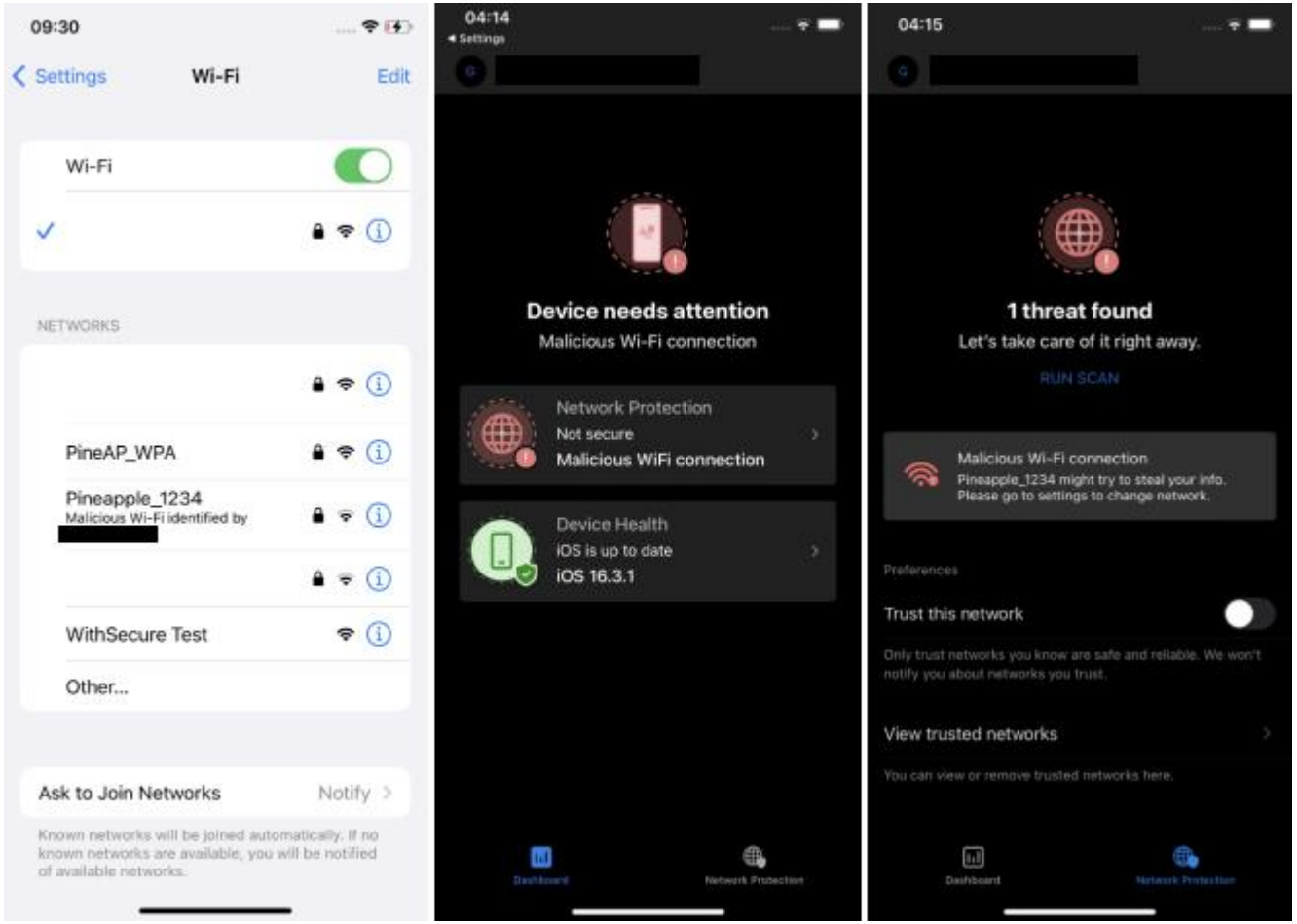
SSID rule

- The only other rule that actually matters is the SSID (network name) one
- Pineapple_{4 hex chars} is the default SSID of the WiFi Pineapple
- Can you just change the defaults?
  - Yup



SSID rule

- Would it produce false positives for any network with that name?
  - Yup



So, what does this all do?

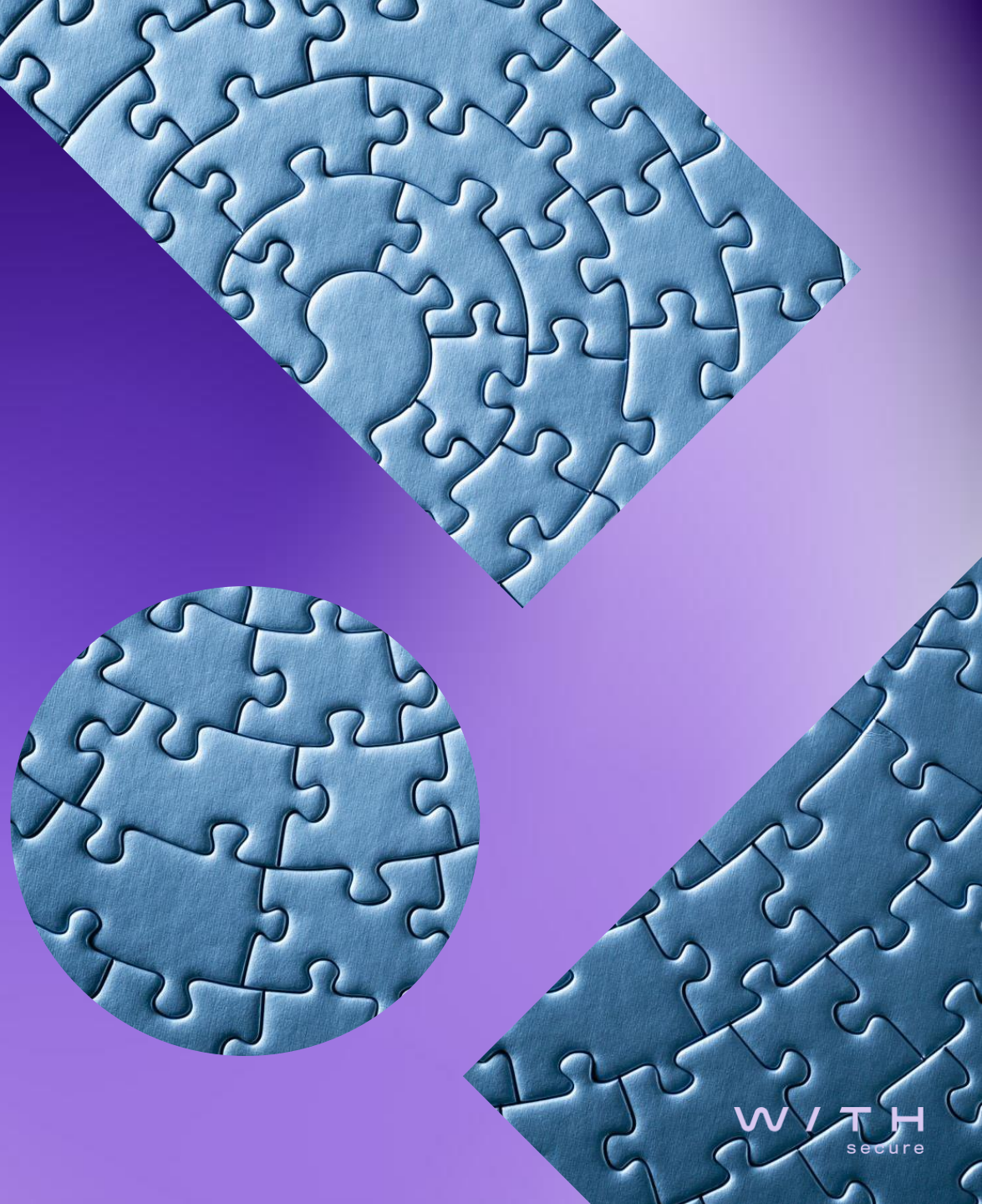
- Best case scenario: nothing
- Worst case scenario: lets you prank people by setting your phone's hotspot name to Pineapple_1234

What should it do?

- Realistically, very little
- Detecting a malicious network from this perspective would be very hard

Example 2

“Keyboard allow-listing”



“Keyboard allow-listing”

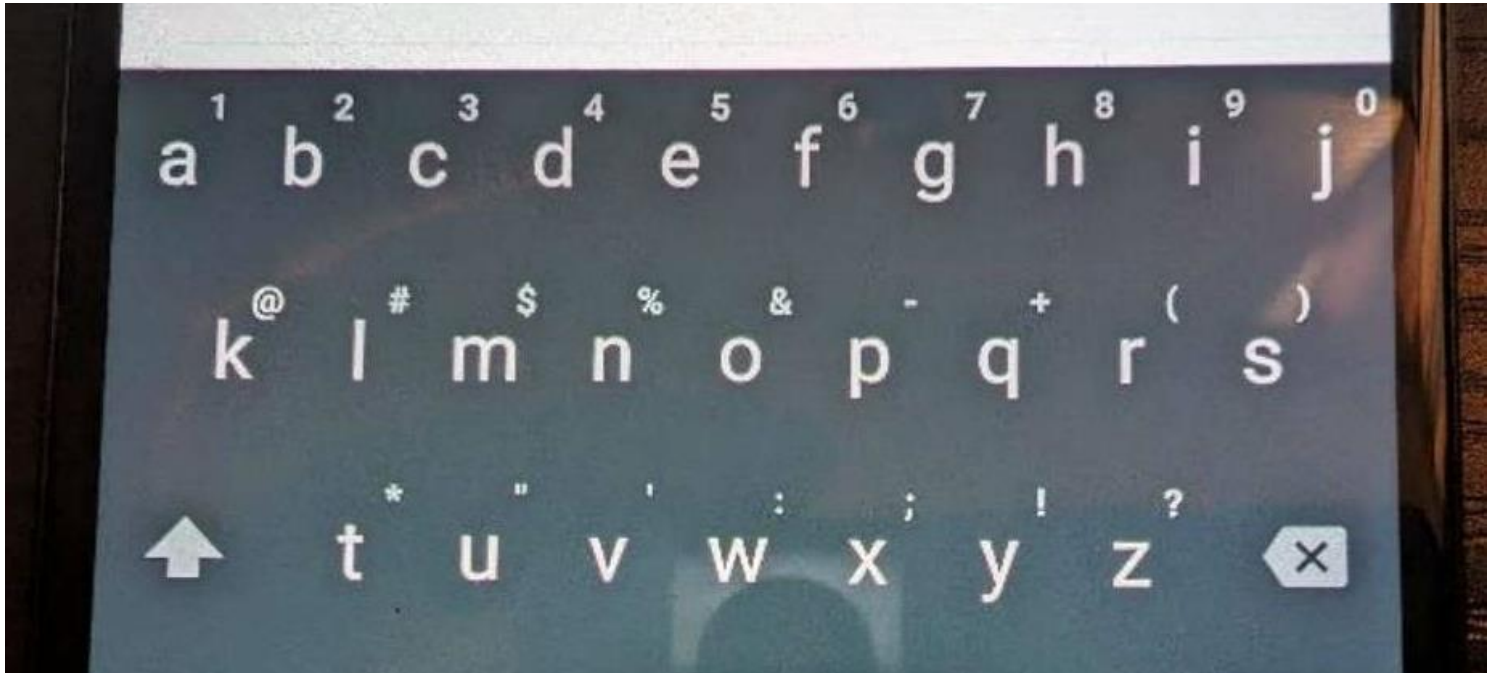
- Android phones let you install custom keyboard apps.
- These **might** allow for data leakage.
  - Technically, a 3rd party keyboard app *could* be a keylogger
  - Or it might send data to a server somewhere for analytics
- So: some businesses want to restrict 3rd party keyboards in their MAM
- Our product lets you specify which keyboards are allowed in your protected apps, and anything else is forbidden
- Administrator just specified application package identifiers (e.g., `com.android.inputmethod.latin` for GBoard)

“Keyboard allow-listing”

- Administrator just specified application package identifiers (e.g., `com.android.inputmethod.latin` for Gboard)
- This seemed a little strange:
  - The package identifier is just a string you specify when writing an app – it’s basically just the app’s name
  - Is there anything stopping me from grabbing one of the approved names and making my own keyboard that uses it?
  - I decided to take `com.android.inputmethod.hindi` for the Google Indic Keyboard

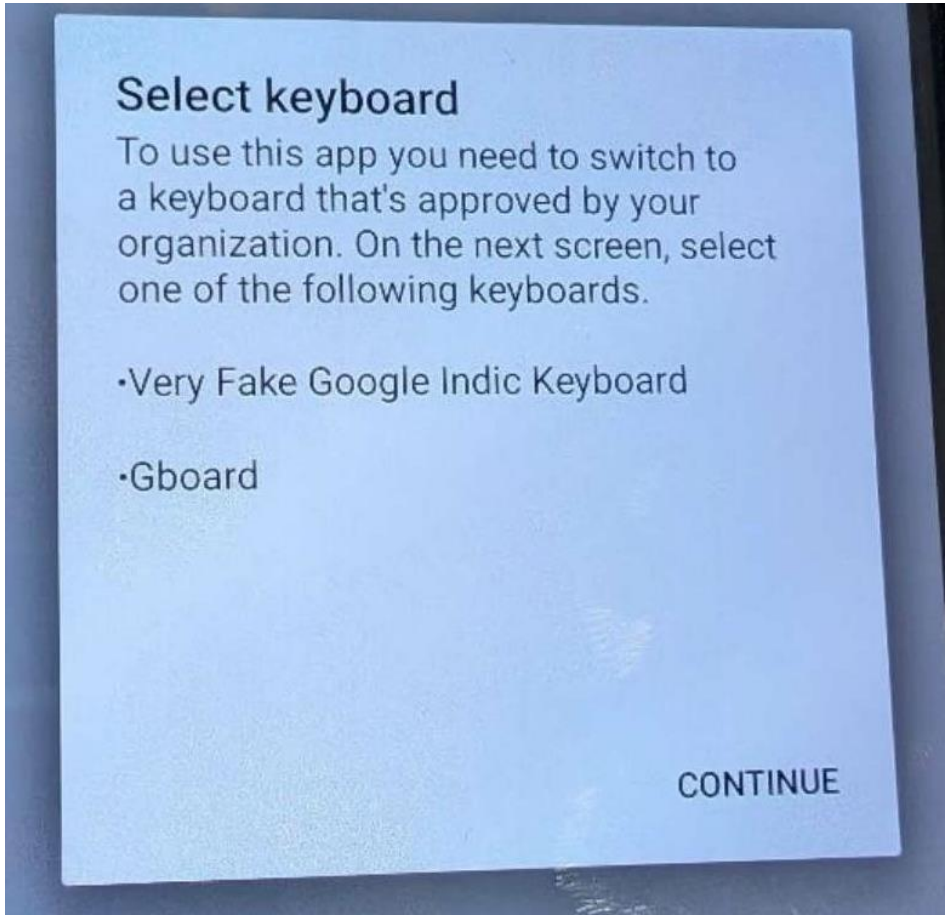
“Keyboard allow-listing”

- Administrator just specified application package identifiers (e.g., `com.android.inputmethod.latin` for GBoard)
- This seemed a little strange:
  - The package identifier is just a string you specify when writing an app – it’s basically just the app’s name
  - Is there anything stopping me from grabbing one of the approved names and making my own keyboard that uses it?
  - I decided to take `com.android.inputmethod.hindi` for the Google Indic Keyboard
  - I modified an open-source keyboard to make it into a basic keylogger, and I made it an ABC keyboard for fun



“Keyboard allow-listing”

- In theory, since the MAM seems to only check for keyboard names, I should now be able to use it
- Let’s see the list of allowed keyboards…
  - Yup!



So, what does this all do?

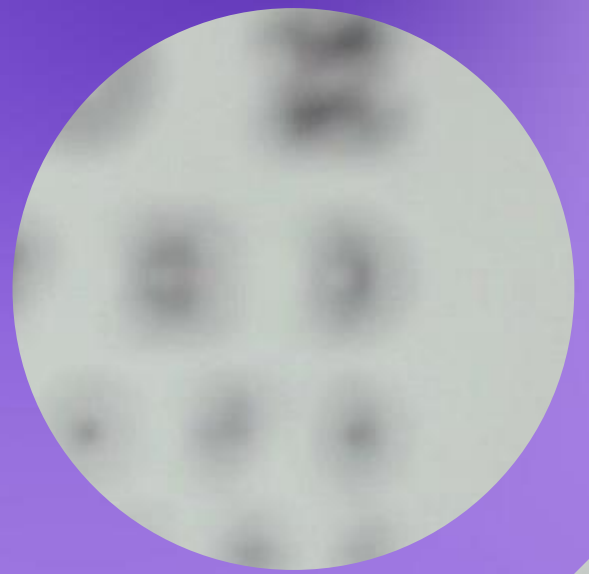
- Very little.
- Best case scenario: it prevents people from accidentally using the “wrong” keyboard.
- Worst case scenario: provides a completely false sense of security.

What should it do?

- It could try to validate the app’s signature (was it actually made by the right organisation?)
- Alternatively: implement your own keyboard and only allow that.

Example 3

“Website allow-listing”



“Website allow-listing”

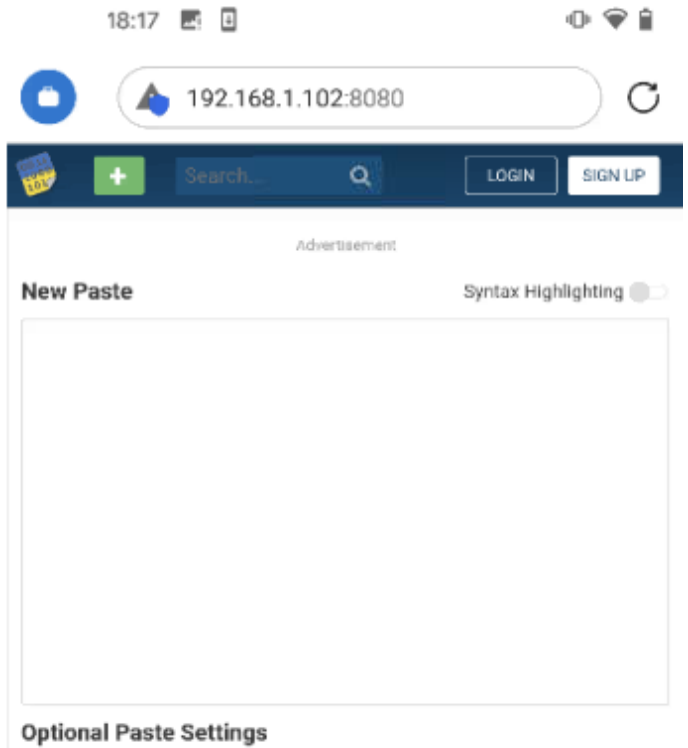
- This time we’re looking at a Web browser within MAM
- Remember: in an ideal world, we can’t copy-paste things out of “work” and into the rest of the device
- But how does that work with a Web browser, that might be used for both?
- In theory: we provide a list of “work” websites that are treated as “inside” the sandbox. Everything else is “outside” of the sandbox (so we can’t copy-paste into it)
- A little bit of a burden on the IT admins, but, in theory, this should work...

“Website allow-listing”

- This time we’re looking at a Web browser within MAM
- Remember: in an ideal world, we can’t copy-paste things out of “work” and into the rest of the device
- But how does that work with a Web browser, that might be used for both?
- In theory: we provide a list of “work” websites that are treated as “inside” the sandbox. Everything else is “outside” of the sandbox (so we can’t copy-paste into it)
- A little bit of a burden on the IT admins, but, in theory, this should work…
- Unfortunately, things are never quite so simple

“Website allow-listing”

- Unfortunately, things are never quite so simple
- Reverse-engineering the app revealed a bunch of undocumented behaviour
- Including the fact that all connections to local IP addresses (e.g., 192.168.1.xxx) were always treated as “inside” the sandbox
- So: you can either host your own info-stealer website or proxy local traffic to a website of your choice, and paste any corporate data you want into that site!



Conclusions!

Conclusions!

- We are still a long way away from “good” MAM environments
- Many of them make loud promises, and claim to deliver on the idea of personal devices with “secure” corporate sandboxes
- In reality, they rarely stand up to scrutiny…
- …and this scrutiny is **very rare**.

- Independent validation of vendors’ claims is essential here

Keep in touch!

- e-mail: milosz.gaczkowski@withsecure.com
- Twitter: @cyberMilosz
- LinkedIn: https://www.linkedin.com/in/milosz-gaczkowski/

WITH®

secure