SYNACKTIV

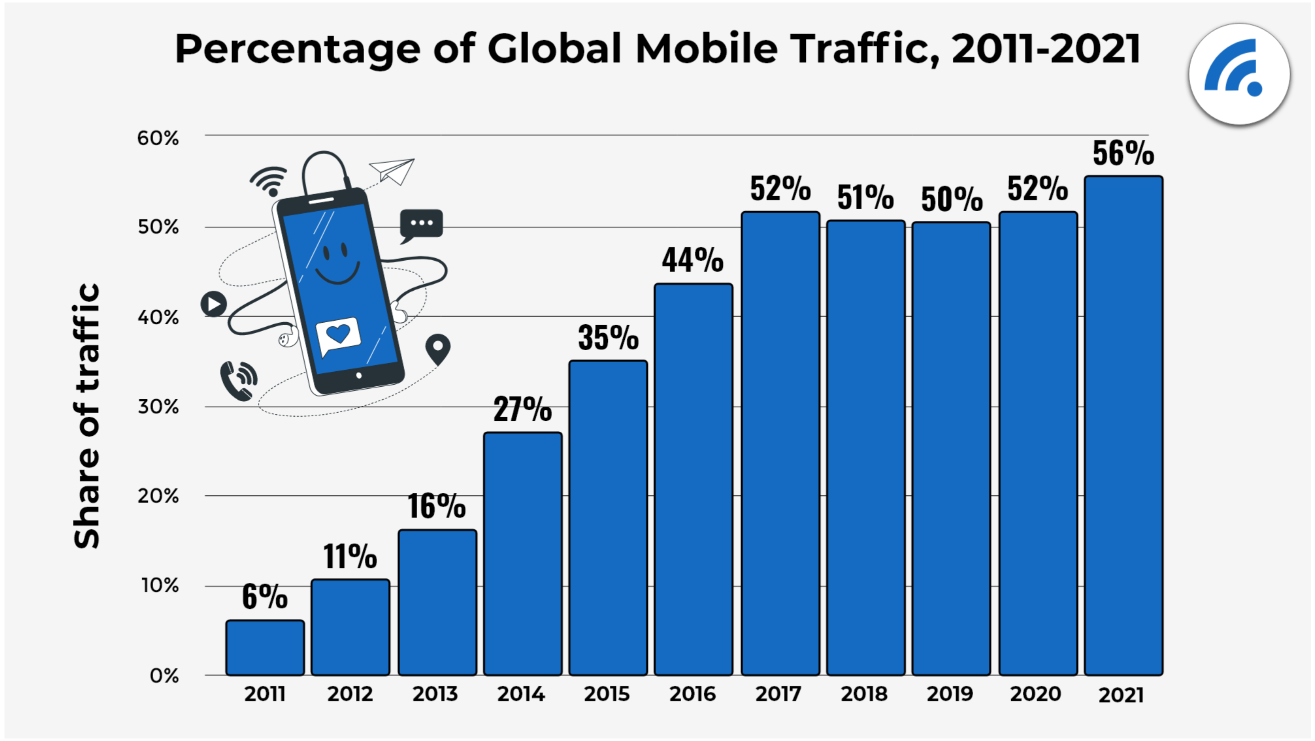
MobSF for pentetration tester

PassTheSalt 2022

Who are we?

- **Antoine Cervoise & Mickaël Benassouli**
  - Pentesters
  - Not MobSF developers / maintainers
- **Working for Synacktiv**
  - Offensive security
  - 100 ninjas: pentest, reverse engineering, development, incident response
  - We are hiring!

Introduction



Source: Mobile Vs. Desktop Internet Usage (Latest 2022 Data) - BroadbandSearch https://www.broadbandsearch.net/blog/mobile-desktop-internet-usage-statistics

Agenda

- **Reminder about mobile applications**
- **MobSF presentation**
- **Usecases for pentest**
  - Mobile application security review
  - Mobile application analysis for red teaming
- **MobSF limitations**

Mobiles applications

Mobile Application

- **Nowadays**
  - Android
  - iOS
- **From the past**
  - Windows Phone
  - Blackberry
  - Window Mobile
  - Symbian
  - …

Android application

- **APK (Android Package Kit)**
  - A ZIP file containing program's code (such as .dex files), libraries, resources, assets, certificates, and manifest file
  - Written in Java or Kotlin
    - Frameworks exist in order to develop application in other languages such as .NET with Xamarin
- **AAB (Android App Bundle)**
  - AAB is push to the store, a personalized APK is downloaded from the store on the device

iOS application

- **IPA**
  - A ZIP file containing application resources and binaries (machO files)

Mobile application review

- **Dedicated penetration test**
  - Vulnerabilities in the mobile application or its dependencies
    - https://owasp.org/www-project-mobile-security-testing-guide/
  - Bypass of anti-cheat measure
  - Entry points for penetration testing on the server
- **Recon on a larger scope**
  - IP / URL / emails
  - Credentials

MobSF

MobSF

- **Mobile SecurityFramework**
- **Licence: GPL 3**
- **Available on GitHub**
  - https://github.com/MobSF/Mobile-Security-Framework-MobSF
- **Online analyzer**
  - https://mobsf.live/

MobSF Features

- **Android review**
  - Application: Static and dynamic analysis
  - Source code: Static analysis
- **iOS review**
  - Application: Static analysis
  - Source code: Static analysis
- **Windows Phone App**
  - Static Analysis

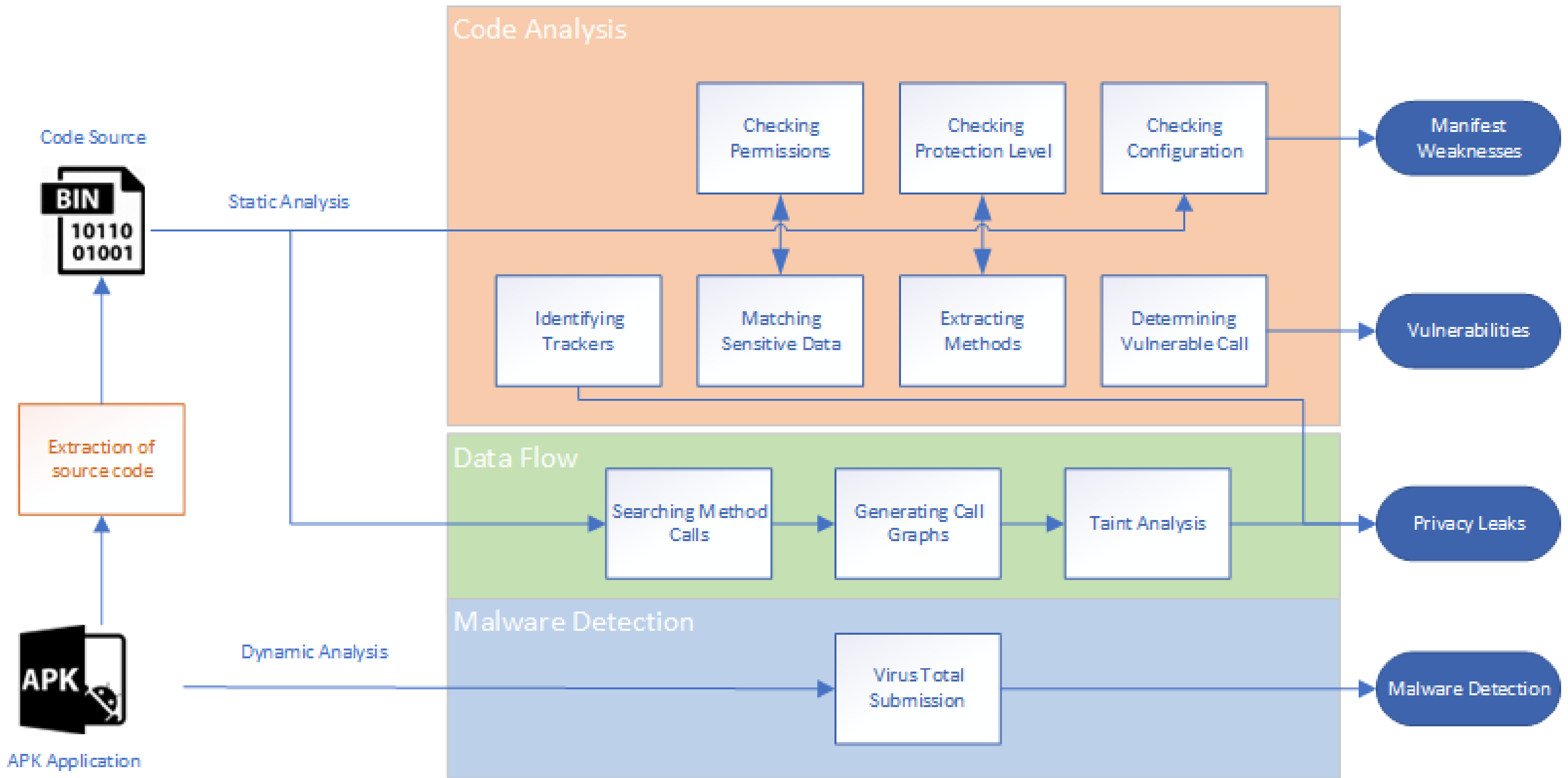
MobSF installation

- Can be launched with docker / kubernetes

```
$ docker pull opensecurity/mobile-security-framework-mobsf:latest

$ mkdir -p $1/mobsf/

$ chmod -R 777 $1/mobsf

$ docker run -it --rm --name mobsf -p 8000:8000 -v
$1/mobsf/:/home/mobsf/.MobSF/ opensecurity/mobile-security-framework-
mobsf:latest
```

- Made python / Oracle JDK / macOS, Linux, Windows
- Hosted only

MobSF architecture



What are we missing

- Android dynamic analysis
- iOS source code review
- Windows applications review
- MobSF in CI/CD

Usecases for Pentesters

Mobile application security review

- Demo time!

Mobile application security review

- **App Score**
  - Quick overview for security score
  - SDK Version and Android Code Version
- **Application Signer Record**
  - Quickly identified issuer and verify certificate
  - Here first check for countermeasure
    - Cipher Algo for signing
    - Code Signing

Mobile application security review

- **Application Permissions**
  - What they need for working.
  - Quickly identify dangerous permissions for pentester
  - Attack scenarios for red teamer
- **Manifest Analysis**
  - The manifest file record also reveals the security flaws found in the target application
  - Need to understand the architecture of the Android OS to assess its actual criticalness
  - A good starting point for analysis, but can be huge too

Mobile application security review

- **Code Analysis**
  - Analysis result of java-code by a static analyzer
  - Detect here countermeasures like
    - Anti Root
    - Pinning
  - Can be false positive and need to be check by reading code
- **NIAP Analysis**
  - Good conformity
  - Pentester? Your first free vulnerabilities

Mobile application security review

- **File / URLs / Text File**
  - Check if files is marked as infected
  - URLs tab shows where the data have been send
  - Where the information have been stored
  - Text file, is a lazy grep for searching quick pattern in code

Mobile application analysis for red teaming

- **Use cases**
  - Penetration testing on a web application that provide a mobile application
  - Red Team

Mobile application analysis for red teaming

- **What are we looking for?**
  - IP addresses / Domains
  - “hidden” folders
  - Credentials (login/password, JWT, API keys…)
    - Or just a “valid” User-Agent

Mobile application analysis for red teaming

- **MobSF feature - Reconnaissance**
  - URLs
  - Emails
  - Strings
  - Hardcoded Secrets
    - Look for specific patterns in strings names

Limits

- **Hardcoded Secrets**
  - does not check into *plist* files (IPA)
  - does not check for specific patterns in strings values
    - BASIC BASE64
    - proto://user:pass@domain

Let’s use the API

- **Check for plist files**
  - Get plist files

```
$ curl -s -X POST --url http://MOBSF/api/v1/report_json --data "hash=IPA_HASH" -H
"Authorization:$token" |jq ".file_analysis" |grep ".plist\"" |grep file_path |cut -d
"\"" -f 4
```

  - Grep for “password”

```
$ curl -s -X POST –url http://MOBSF/api/v1/view_source --data
"hash=IPA_HASH&type=ipa&file=$plist" -H "Authorization:$token" |grep -i password
```

Let’s use the API

- **Check for patterns in strings values**
  - This can be done using
    - APKLeaks (https://github.com/dwisiswant0/apkleaks) and Super (https://github.com/SUPERAndroidAnalyzer/super)
    - They are dedicated to APK
    - Super requires Java to run

Automation

- **Put everything in a (dirty) script**

```
$ bash mobydeep.sh
Version: 1.0
Usage: mobydeep.sh http(s)://mobsf
Args:
  -h / --help            : this help
  --get-hashes           : get applications hashes from MobSF
  --plist IPA_hash       : check for credentials in plists files
  --check-strings hash   : check for credentials in strings values
  --check-secrets hash   : return MobSF check for secrets in APP
```

Find credentials and keep digging

- **Check for secrets in strings**

```
$ mobydeep.sh http://localhost:8000 --check-strings
18*************************************42


"\"**BasicAuth\" : \"Basic UG************************************c=\"",
```

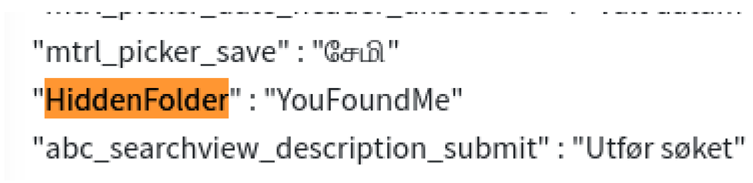
Find credentials and keep digging

- Looking for the secret usage into the source code

```
      if (new Connectivity(context).isNetworkAvailable()) {
                 try {
                     [...]
                     Uri.Builder builder = new Uri.Builder();

     builder.scheme("https").authority("webapp.customer.tld").appendPath(context.getR
     esources().getString(R.string.HiddenFolder));
                     [...]
                 } catch (Exception e) {
                     e.toString();
                 }
```

Find credentials and keep digging

- **Find the hidden folder**
  - Solution 1: Decompile the whole app and go look into res/values/strings.xml
  - Solution 2: Search it in MobSF



```
"mtrl_picker_save" : "சேமி"
"HiddenFolder" : "YouFoundMe"
"abc_searchview_description_submit" : "Utfør søket"
```

Automation issues

- **False positive**
  - Auth BASIC detection
  - Plist analysis
  - Maybe more
- **Patterns are handle into the script**
  - no external database/JSON file/whatever

Scan multiple applications

- **Upload them all**
  - https://github.com/MobSF/Mobile-Security-Framework-MobSF/blob/master/scripts/mass_static_analysis.py
- **Scan them all**

```
$ for app in $(bash mobydeep.sh http://127.0.0.1:8000 --
get-hashes); do
   echo $elmt; bash mobydeep.sh http://127.0.0.1:8000 --
check-strings $app;
done
```

MobSF limitations (as a pentester)

MobSF Limitations

- Development of new features needs to be able to develop them
- No support for AAR (Android Archive) → libraries files
- Android dynamic analysis is not easy to configure

SYNACKTIV

https://www.linkedin.com/company/synacktiv
https://twitter.com/synacktiv
Our publications: https://synacktiv.com