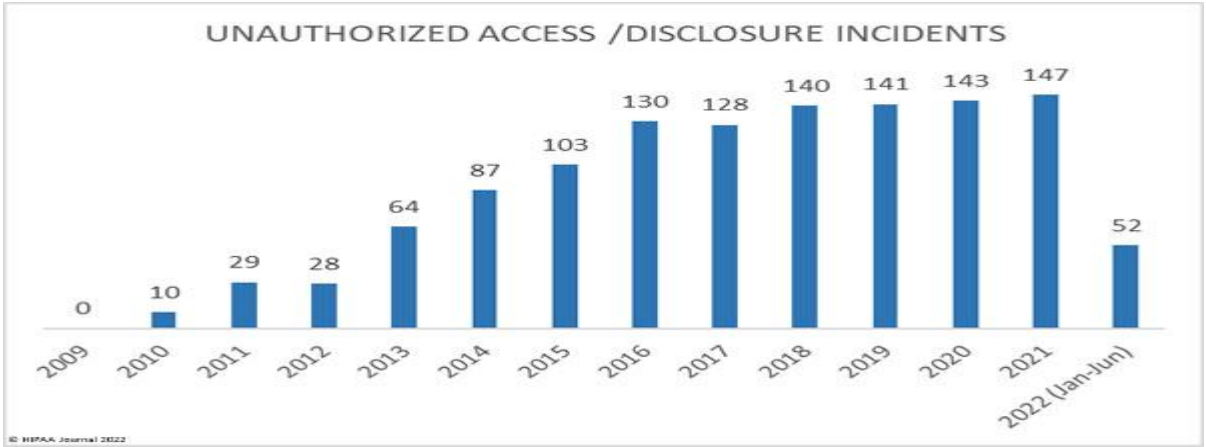
A Dataset of Protected Health Information (PHI) Leakage in Android Applications

Ashfak Md Shibli
Tahiatul Islam

PHI data leakage events

- 1500 patient data leaked from ZomoHealth
- 42% of all data breaches attributable to the healthcare industry



Source: HIPAA Journal

-http://healthfinancejournal.com/~junland/index.php/johcf/article/view/67

PHI data points

- 18 PHI points
- SSN
- Address
- DOB
- Name
- Medical report number
- Email

Source: HIPAA

Why PHI data breach is critical?

- Lead to stolen personal identity
- Face billing & treatment issue
- User trust lost
- Companies face penalties

Potential data breaches

- SQL Injection
- Cross-Site Scripting (XSS)
- Insecure webclient connection

- Logging Secure Data
- Phishing attempt to get PHI Data

```
void ProcessRequest(HttpRequest request)
{
  string name = request.Form["name"]; // <= taint source
  // now "name" contains potentially tainted data

  string sql = $"SELECT * FROM Users WHERE name='{name}'";
  ExecuteReaderCommand(sql); // tainted data passed as an argument
  ....
}

void ExecuteReaderCommand(string sql)
{
  using (var command = new SqlCommand(sql, _connection)) // <= sink
  {
    using (var reader = command.ExecuteReader()) { /*....*/ }
  }
  ....
}
```

Source: https://pvs-studio.com/en/blog/terms/6496/

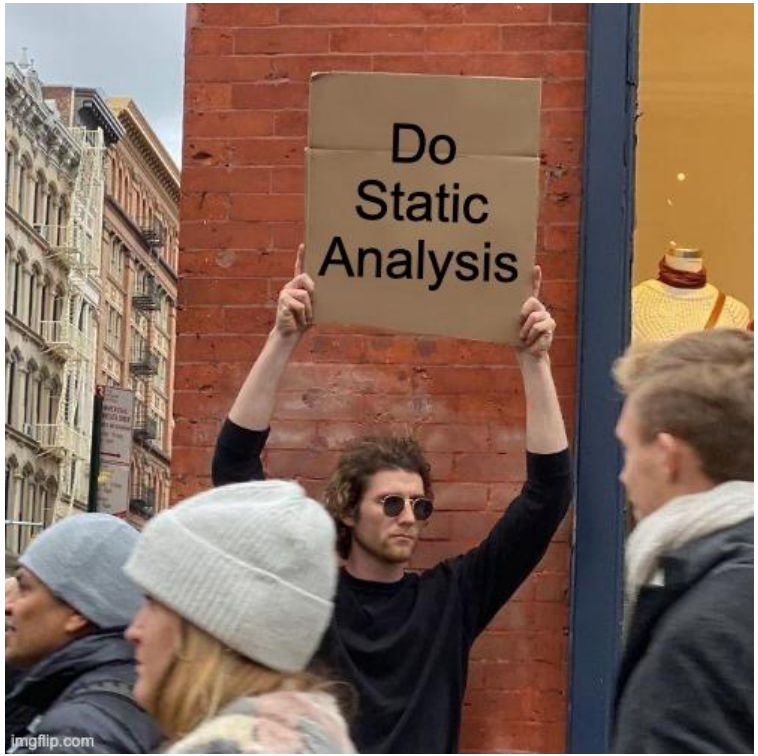
```
# Beautiful code above

user_ssn = getSsn()
user_dob = getDOB()
Logger.Log("This is ssn: " + user_ssn + "This is birthdate: "+
user_dob)
# Beautiful code beyond
```

Our Study

→ Analyze the application leakage
→ Empirically check the False Positives
→ Check the intent of developers
→ Analyze the possible exploit of Attacker
→ Determine relationships of Intents
→ Propose the countermeasure

Before you get hit



Static Analysis - Taint - FlowDroid

→ Decompile the APK (Application Package)
→ Analyze the flow from Manifest+XML
→ Build call graph using IFDS inter-procedural, finite, distributive, subset
→ Taint analysis from Source to Sinks
→ Out of the box tool FlowDroid

Leakage sources in Android

Source Method	Returns	Data Carries
getSubscriberId	String[]	Sim card Subscriber Info
getIpAddress	int	IP address
getAuthToken	AccountManagerFuture	Authentication token
getAllCellInfo	List	Information
getCellLocation	CellLocation	Cell tower information
getDeviceId	String	DeviceId
getLine1Number	String	Sim1 number
getVoiceMailNumber	String	Voice Mail Number
getLastKnownLocation	Location	GPS cordinates

Sinks in Android

Source Method	Returns	Data Carries
getSubscriberId	String[]	Sim card Subscriber Info
getIpAddress	int	IP address
getAuthToken	AccountManagerFuture	Authentication token
getAllCellInfo	List	Information
getCellLocation	CellLocation	Cell tower information
getDeviceId	String	DeviceId
getLine1Number	String	Sim1 number
getVoiceMailNumber	String	Voice Mail Number
getLastKnownLocation	Location	GPS cordinates

We Mapped PHI → Source/Sinks

PHI Data Point	Source/Sink Function
name	getAccounts| android.util.Log | sendTextMessage
address	android.util.Log | sendTextMessage
dates	android.util.Log | sendTextMessage|
phone number	getLine1Number|getVoiceMailNumber
email address	sendTextMessage|android.util.Log|
Social Security number	android.util.Log
medical record number	android.util.Log
health plan beneficiary number	android.util.Log
account number	android.util.Log
certificate or license number	android.util.Log
vehicle identifiers, such as serial numbers, license plate numbers	android.util.Log
device identifiers and serial numbers;	getDeviceId
web URL	getHost | android.util.Log | sendTextMessage
Internet Protocol (IP) address	getHost | getPort
biometric IDs, such as a fingerprint or voice print	getKey
full-face photographs and other photos of identifying characteristics	getKey
any other unique identifying characteristic	android.util.Log

Outcome of our study

- A dataset having 50 apps
- Analyze every app using FlowDroid on
- Store leakage and processed flow in dataset
- Determine relationship with leaks
- Formalize developer intent & attacker exploit
- Tabular representation as organized dataset

Developer Intent	Attacker Exploit
It just works that way.	No attack possible. Platform protects.
It was a mistake.	Attacker can get access and keep data to find backdoor to threat and try any gain.
Don’t know it has vulnerability.	
Missed to protect the module with best practice.	Hit to exploit or make more secure [from research or testing]

Example

- “Universal Remote Control” - 10M Downloads
- News says it has leakage
- We found leakage in location data
- Wait what? Why it is leaking that data?
- May be a mistake/disguised malware Developer Intent
- Attacker can misuse user location Attacker Exploit

Final Dataset

→ Dataset with application insights and patterns

→ Empirically defined intent specifications

→ Data for future research as there is scarcity of organized data for research on this topic.

→ Data will help users and regulatory bodies.

APK	Type	Leak Count	Place	Probable Developer Intent	Probable Attacker Exploit
com.universalremote.app	Utility	4	Location Module	Mistake	Attack User using location point
com.abc.malware [example]	Heart Rate Monitor	6	Location Module	Potential Malicious Intent	User data broker