OWASP Top 10 Mobile Risks

Jack Mannino, nVisium Security
Mike Zusman, Carve Systems
Zach Lanier, Intrepidus Group

OWASP Mobile Security Project

Agenda

- Introductions
- Mobile Security Project
- Mobile Threat Model
- Top 10 Risks
- Wrap Up/Q&A

Introductions

Mike Zusman

- Carve Systems
- Principal Consultant
- http://www.carvesystems.com

Jack Mannino

- nVisium Security
- CEO
- https://www.nvisiumsecurity.com

Zach Lanier

- Intrepidus Group
- Principal Consultant
- https://intrepidusgroup.com

Mobile Security Project

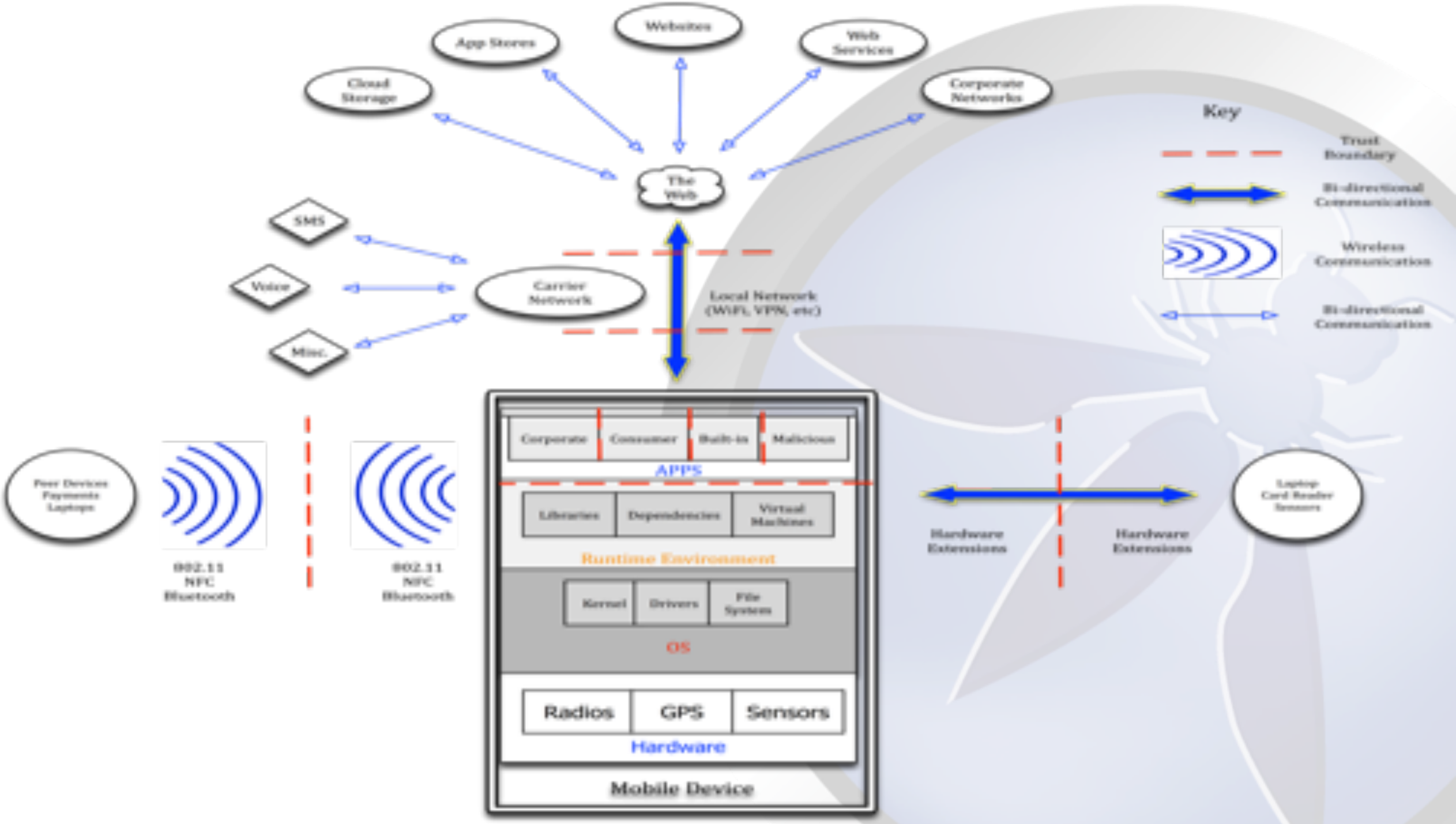
- Began Q3 2010
- ***Why*** Unique and different security risks
- ***Goal*** To build security into mobile dev. life cycle
- Interested? Contribute

Mobile Threat Model

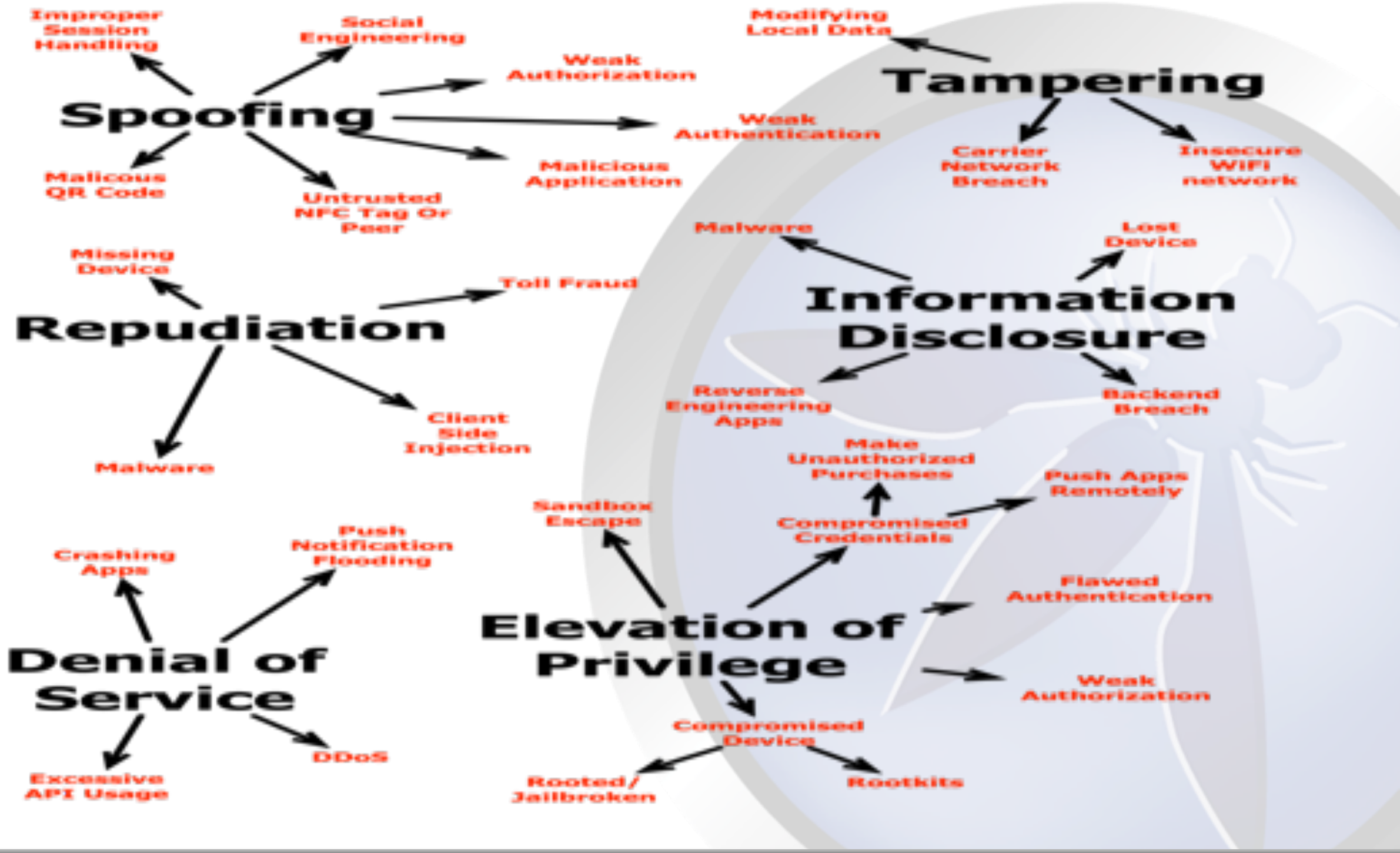
Mobile Threat Model

- Platforms vary with mileage
- Very different from traditional web app model due to wildly varying use cases and usage patterns
- Must consider more than the ‘apps’
  - Remote web services
  - Platform integration (iCloud, C2DM)
  - Device (in)security considerations

Mobile Threat Model



Mobile Threat Model



Top 10 Risks

Top 10 Risks

- Intended to be platform-agnostic
- Focused on areas of risk rather than individual vulnerabilities
- Weighted utilizing the OWASP Risk Rating Methodology
  - https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology
- Thanks to everyone who participated

Top 10 Risks

OWASP Mobile Top 10 Risks	
M1- Insecure Data Storage	M6- Improper Session Handling
M2- Weak Server Side Controls	M7- Security Decisions Via Untrusted Inputs
M3- Insufficient Transport Layer Protection	M8- Side Channel Data Leakage
M4- Client Side Injection	M9- Broken Cryptography
M5- Poor Authorization and Authentication	M10- Sensitive Information Disclosure

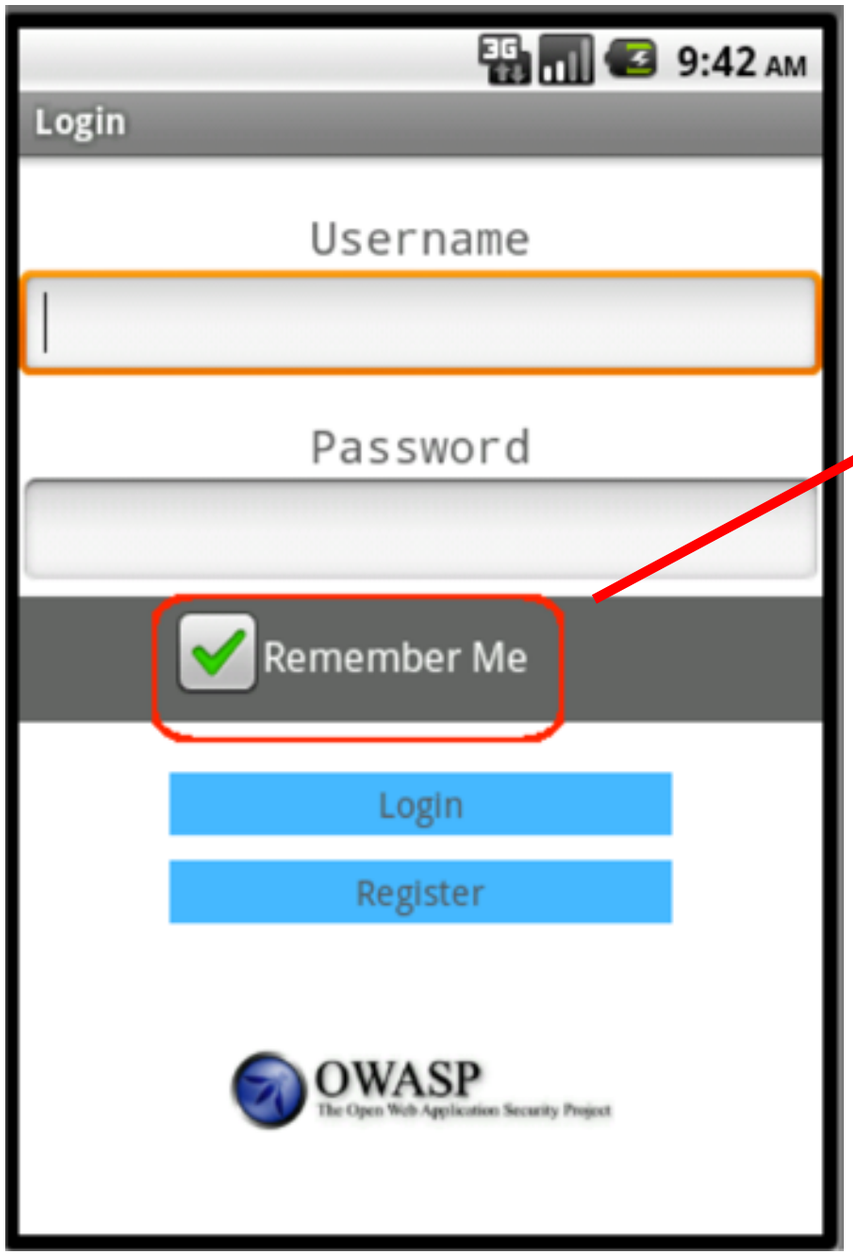
M1- Insecure Data Storage

- Sensitive data left unprotected
- Applies to locally stored data + cloud synced
- Generally a result of:
  - Not encrypting data
  - Caching data not intended for long-term storage
  - Weak or global permissions
  - Not leveraging platform best-practices

Impact

- Confidentiality of data lost
- Credentials disclosed
- Privacy violations
- Non-compliance

M1- Insecure Data Storage




```
public void saveCredentials(String userName, String password) {

    SharedPreferences credentials = this.getSharedPreferences(
            "credentials", MODE_WORLD_READABLE);
    SharedPreferences.Editor editor = credentials.edit();
    editor.putString("username", userName);
    editor.putString("password", password);
    editor.putBoolean("remember", true);
    editor.commit();
}
```

— Very Bad

— Convenient!

M1- Insecure Data Storage
Prevention Tips

- Store ONLY what is absolutely required
- Never use public storage areas (ie- SD card)
- Leverage secure containers and platform provided file encryption APIs
- Do not grant files world readable or world writeable permissions

Control #	Description
1.1-1.14	Identify and protect sensitive data on the mobile device
2.1, 2.2, 2.5	Handle password credentials securely on the device

M2- Weak Server Side Controls

- Applies to the backend services
- Not mobile specific per se, but essential to get right
- We still can't trust the client
- Luckily, we understand these issues well
- Existing controls may need to be re-evaluated (ie- out of band comms)

Impact

- Confidentially of data lost
- Integrity of data not trusted

M2- Weak Server Side Controls

OWASP Top 10

- https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

OWASP Cloud Top 10

- https://www.owasp.org/images/4/47/Cloud-Top10-Security-Risks.pdf

M2- Weak Server Side Controls
Prevention Tips

- Understand the additional risks mobile apps introduce into existing architectures
- Leverage the wealth of knowledge that is already out there
- OWASP Web Top 10, Cloud Top 10, Web Services Top 10
- Cheat sheets, development guides, ESAPI

Control #	Description
5.1-5.8	Keep the backend APIs (services) and the platform (server) secure

M3- Insufficient Transport Layer Protection

- Complete lack of encryption for transmitted data
  - Yes, this unfortunately happens *often*
- Weakly encrypted data in transit
- Strong encryption, but ignoring security warnings
  - Ignoring certificate validation errors
  - Falling back to plain text after failures

Impact

- Man-in-the-middle attacks
- Tampering w/ data in transit
- Confidentiality of data lost

M3- Insufficient Transport Layer Protection

Real World Example: Google ClientLogin Authentication Protocol

- Authorization header sent over HTTP
- When users connected via wifi, apps automatically sent the token in an attempt to automatically synchronize data from server
- Sniff this value, impersonate the user
  - http://www.uni-ulm.de/in/mi/mitarbeiter/koenings/catching-authtokens.html

M3- Insufficient Transport Layer Protection
Prevention Tips

- Ensure that all sensitive data leaving the device is encrypted
- This includes data over carrier networks, WiFi, and even NFC
- When security exceptions are thrown, it's generally for a reason…*DO NOT* ignore them!

Control #	Description
3.1.3.6	Ensure sensitive data is protected in transit

M4- Client Side Injection

- Apps using browser libraries
  - Pure web apps
  - Hybrid web/native apps
- Some familiar faces
  - XSS and HTML Injection
  - SQL Injection
- New and exciting twists
  - Abusing phone dialer + SMS
  - Abusing in-app payments

Impact

- Device compromise
- Toll fraud
- Privilege escalation

M4- Client Side Injection

Garden Variety XSS….

```
@Override
public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.demo);
    context = this.getApplicationContext();
    webView = (WebView) findViewById(R.id.demoWebView);
    webView.getSettings().setJavaScriptEnabled(true);
    webView.addJavascriptInterface(new SmsJSInterface(this),
            "smsJSInterface");
    GetSomeInfo getInfo = new GetSomeInfo();
    getInfo.execute(null, null);
}

public String generateHTML(String untrustedData) {

    return "<b>Check this out!</b><br>" + untrustedData;
}
```

With access to:

```
public class SmsJSInterface implements Cloneable {

    Context mContext;

    public SmsJSInterface(Context context) {

        mContext = context;
    }

    public void sendSMS(String phoneNumber, String message) {

        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(phoneNumber, null, message, null, null);
    }
```

M4- Client Side Injection
Prevention Tips

- Sanitize or escape untrusted data before rendering or executing it
- Use prepared statements for database calls…concatenation is still bad, and always will be bad
- Minimize the sensitive native capabilities tied to hybrid web functionality

Control #	Description
6.3	Pay particular attention to validating all data received from and sent to non-trusted third party apps before processing
10.1-10.5	Carefully check any runtime interpretation of code for errors

M5- Poor Authorization and Authentication

- Part mobile, part architecture
- Some apps rely solely on immutable, potentially compromised values (IMEI, IMSI, UUID)
- Hardware identifiers persist across data wipes and factory resets
- Adding contextual information is useful, but not foolproof

Impact

- Privilege escalation
- Unauthorized access

M5- Poor Authorization and Authentication

```
if (dao.isDevicePermanentlyAuthorized(deviceID)) {
    int newSessionToken = LoginUtils.generateSessionToken();
    dao.openConnection();
    dao.updateAuthorizedDeviceSession(deviceID,
            sessionToken, LoginUtils.getTimeMilliseconds());
    bean.setSessionToken(newSessionToken);
    bean.setUserName(dao.getUserName(sessionToken));
    bean.setAccountNumber(dao.getAccountNumber(sessionToken));
    bean.setSuccess(true);
    return bean;
}
```

M5- Poor Authorization and Authentication
Prevention Tips

- Contextual info can enhance things, but only as part of a multi-factor implementation
- Out-of-band doesn't work when it's all the same device
- Never use device ID or subscriber ID as sole authenticator

Control #	Description
4.1-4.6	Implement user authentication/authorization and session management correctly
8.4	Authenticate all API calls to paid resources

M6- Improper Session Handling

- Mobile app sessions are generally MUCH longer
- Why? Convenience and usability
- Apps maintain sessions via
  - HTTP cookies
  - OAuth tokens
  - SSO authentication services
- Bad idea= using a device identifier as a session token

Impact

- Privilege escalation
- Unauthorized access
- Circumvent licensing and payments

M6- Improper Session Handling
Prevention Tips

- Don't be afraid to make users re-authenticate every so often
- Ensure that tokens can be revoked quickly in the event of a lost/stolen device
- Utilize high entropy, tested token generation resources

Control #	Description
1.13	Use non-persistent identifiers
4.1-4.6	Implement user authentication/authorization and session management correctly

M7- Security Decisions Via Untrusted Inputs

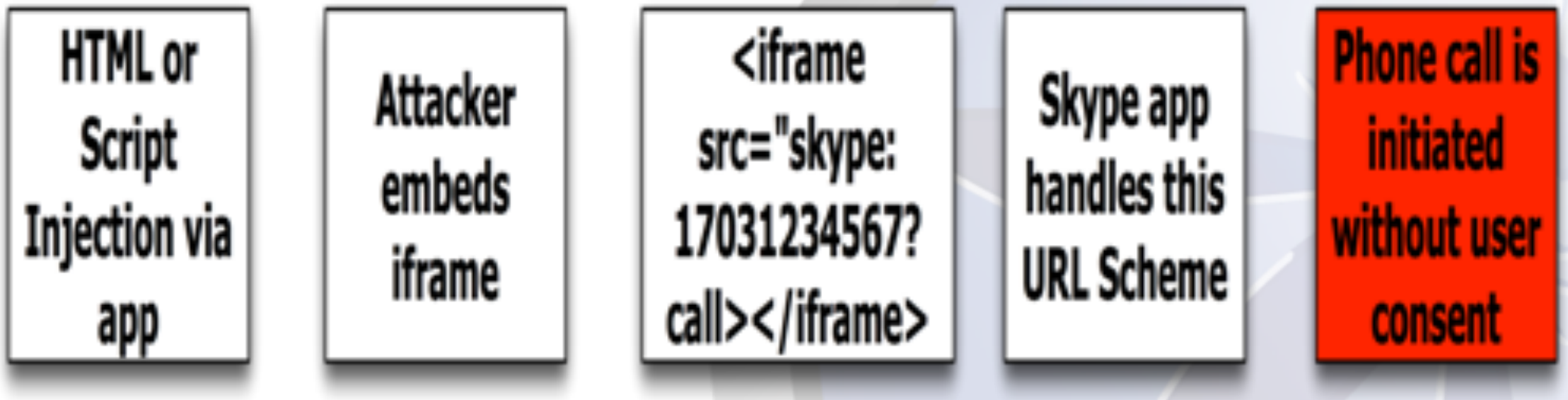
- Can be leveraged to bypass permissions and security models
- Similar but different depending on platform
  - iOS- Abusing URL Schemes
  - Android- Abusing Intents
- Several attack vectors
  - Malicious apps
  - Client side injection

Impact

- Consuming paid resources
- Data exfiltration
- Privilege escalation

M7- Security Decisions Via Untrusted Inputs

Skype iOS URL Scheme Handling Issue



- http://software-security.sans.org/blog/2010/11/08/insecure-handling-url-schemes-apples-ios/

M7- Security Decisions Via Untrusted Inputs
Prevention Tips

- Check caller's permissions at input boundaries
- Prompt the user for additional authorization before allowing
- Where permission checks cannot be performed, ensure additional steps required to launch sensitive actions

Control #	Description
10.2	Run interpreters at minimal privilege levels

M8- Side Channel Data Leakage

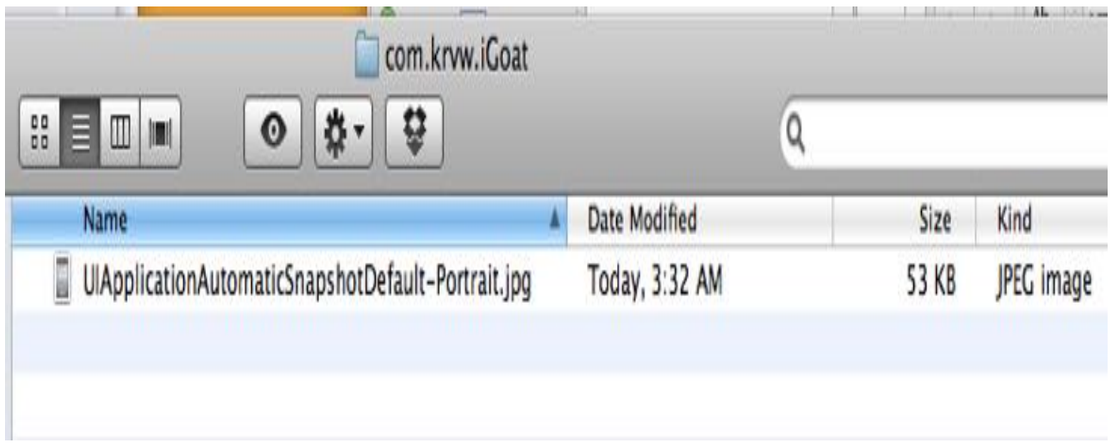
- Mix of not disabling platform features and programmatic flaws
- Sensitive data ends up in unintended places
  - Web caches
  - Keystroke logging
  - Screenshots (ie- iOS backgrounding)
  - Logs (system, crash)
  - Temp directories
- Understand what 3rd party libraries in your apps are doing with user data (ie- ad networks, analytics)

Impact

- Data retained indefinitely
- Privacy violations

M8- Side Channel Data Leakage

Screenshots



Logging

```
try {
    userInfo = client.validateCredentials(userName, password);
    if (userInfo.get("success").equals("true"))
        launchHome(v);
    else {
        Log.w("Failed login", userName + " " + password);
    }

} catch (Exception e) {
    Log.w("Failed login", userName + " " + password);
}
```

M8- Side Channel Data Leakage
Prevention Tips

- Never log credentials, PII, or other sensitive data to system logs
- Remove sensitive data before screenshots are taken, disable keystroke logging per field, and utilize anti-caching directives for web content
- Debug your apps before releasing them to observe files created, written to, or modified in any way
- Carefully review any third party libraries you introduce and the data they consume
- Test your applications across as many platform versions as possible

Control #	Description
7.3	Check whether you are collecting PII, it may not always be obvious
7.4	Audit communication mechanisms to check for unintended leaks (e.g. image metadata)

M9- Broken Cryptography

- Two primary categories
  - Broken implementations using strong crypto libraries
  - Custom, easily defeated crypto implementations
- Encoding != encryption
- Obfuscation != encryption
- Serialization != encryption

Impact

- Confidentiality of data lost
- Privilege escalation
- Circumvent business logic

M9- Broken Cryptography

```
ldc literal_876:"QlVtT0JoVmY2N2E="
invokestatic byte[] decode( java.lang.String )
// Base 64
invokespecial_lib java.lang.String.<init> //
pc=2
astore 8

private final byte[]
com.picuploader.BizProcess.SendRequest.routine_
12998
    (com.picuploader.BizProcess.SendRequest,
byte[], byte[] );
 {
   enter
   new_lib
net.rim.device.api.crypto.TripleDESKey
```

M9- Broken Cryptography
Prevention Tips

- Storing the key with the encrypted data negates everything
- Leverage battle-tested crypto libraries vice writing your own
- Take advantage of what your platform already provides!

Control #	Description
1.3	Utilize file encryption API's
2.3	Leverage secure containers

M10- Sensitive Information Disclosure

- We differentiate by stored (M1) vs. embedded/hardcoded (M10)
- Apps can be reverse engineered with relative ease
- Code obfuscation raises the bar, but doesn't eliminate the risk
- Commonly found "treasures":
  - API keys
  - Passwords
  - Sensitive business logic

Impact

- Credentials disclosed
- Intellectual property exposed

M10- Sensitive Information Disclosure

```
if (rememberMe)
    saveCredentials(userName, password);
//our secret backdoor account
if (userName.equals("all_powerful")
        && password.equals("iamsosmart"))
launchAdminHome(v);
```

```
public static final double SECRET_SAUCE_FORMULA = (1.2344 * 4.35 - 4 + 1.442) * 2.221;
```

M10- Sensitive Information Disclosure
Prevention Tips

- Private API keys are called that for a reason…keep them off of the client
- Keep proprietary and sensitive business logic on the server
- Almost never a legitimate reason to hardcode a password (if there is, you have other problems)

Control #	Description
2.10	Do not store any passwords or secrets in the application binary

Wrap Up

Going Forward

- 60 day review period open to the public
- RC1 then becomes ‘Final v1.0’
- 12 month revision cycle
  - Rapidly evolving platforms
  - Stale data = not as useful
- If you have suggestions or ideas, we want them!

Conclusion

- This is a good start, but we have a long way to go
- We've identified the issues…now we have to fix them
- Platforms must mature, frameworks must mature, apps must mature
- The OWASP Mobile body of knowledge must grow

Q&A

Thanks for listening!

- Jack Mannino jack@nvisiumsecurity.com
  http://twitter.com/jack_mannino
- Zach Lanier zach.lanier@intrepidusgroup.com
  http://twitter.com/quine
- Mike Zusman mike.zusman@carvesystems.com
  http://twitter.com/schmoilito