Using Malware Analysis to Reduce Design Weaknesses

Carol Woody, Ph.D.
Technical Manager,
Cybersecurity Engineering

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Notices

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0003815

Software Realities

There is no such thing as perfect code

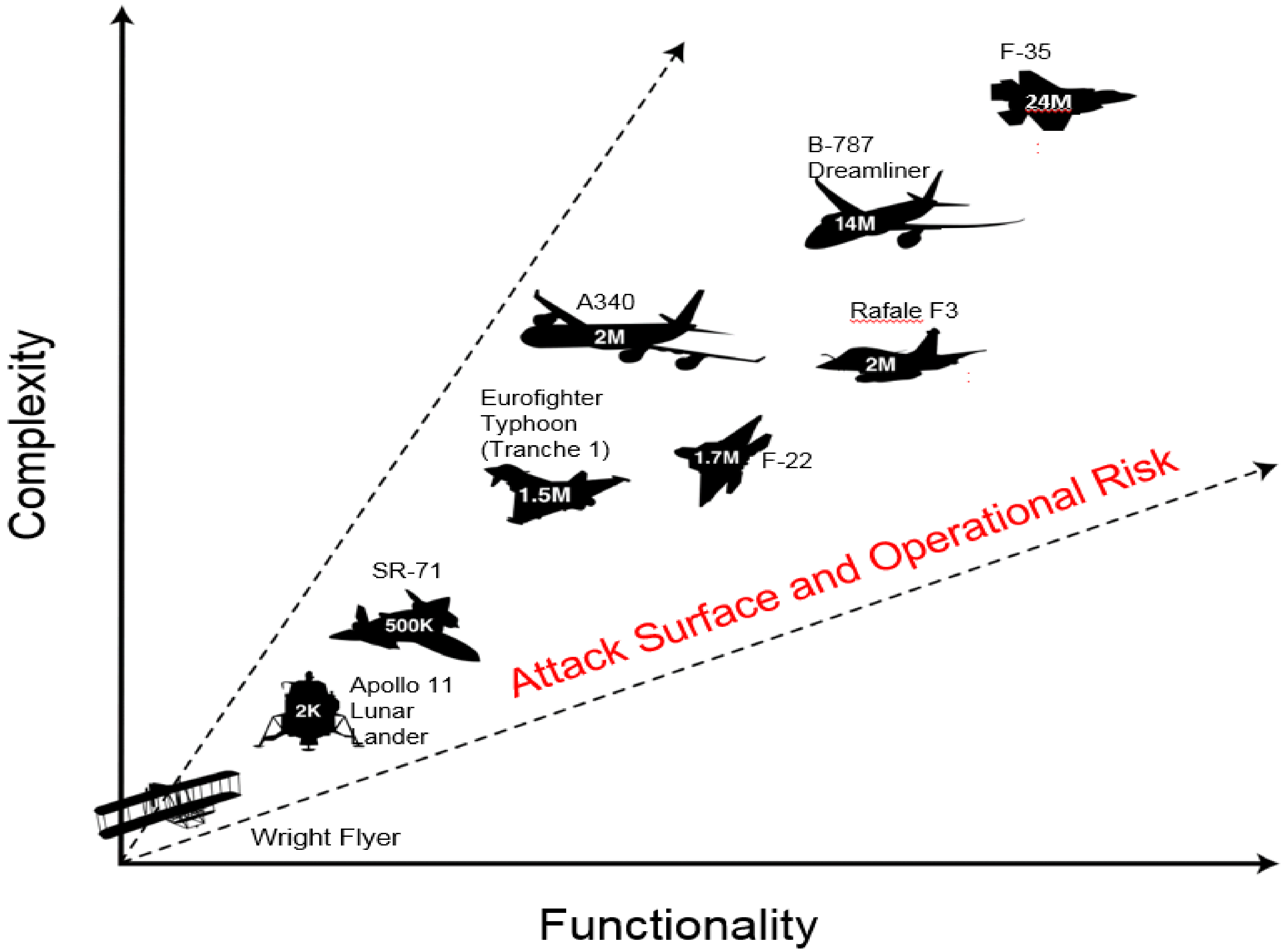
- Best in class code contains 2.5 defects per function point which is < 600 defects per MLOC
- Very good code has an estimated 600-1000 defects per MLOC
- Average quality is 4.5 defects per function point which is 6000 defects per MLOC

(reference: Capers Jones, *sqgne.org/presentations/2011-12/**Jones**-Sep-2011.pdf)*

SEI research indicated an estimated 5% of the defects are vulnerabilities

(reference: Woody, Carol; Ellison, Robert; & Nichols, William. *Predicting Software Assurance Using Quality and Reliability Measures.* CMU/SEI-2014-TN-026. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=428589

Increased Software for Increased Functionality



Estimating Software Vulnerabilities

The Boeing 787 Dreamliner has 14 MLOC

- if we assume all of it is exceptional code, 8,400 defects remain in the code and approximately 420 vulnerabilities
- more likely the code is average to very good, which could have up to 84,000 defects and 4,200 vulnerabilities

The F-22 has 1.7 MLOC

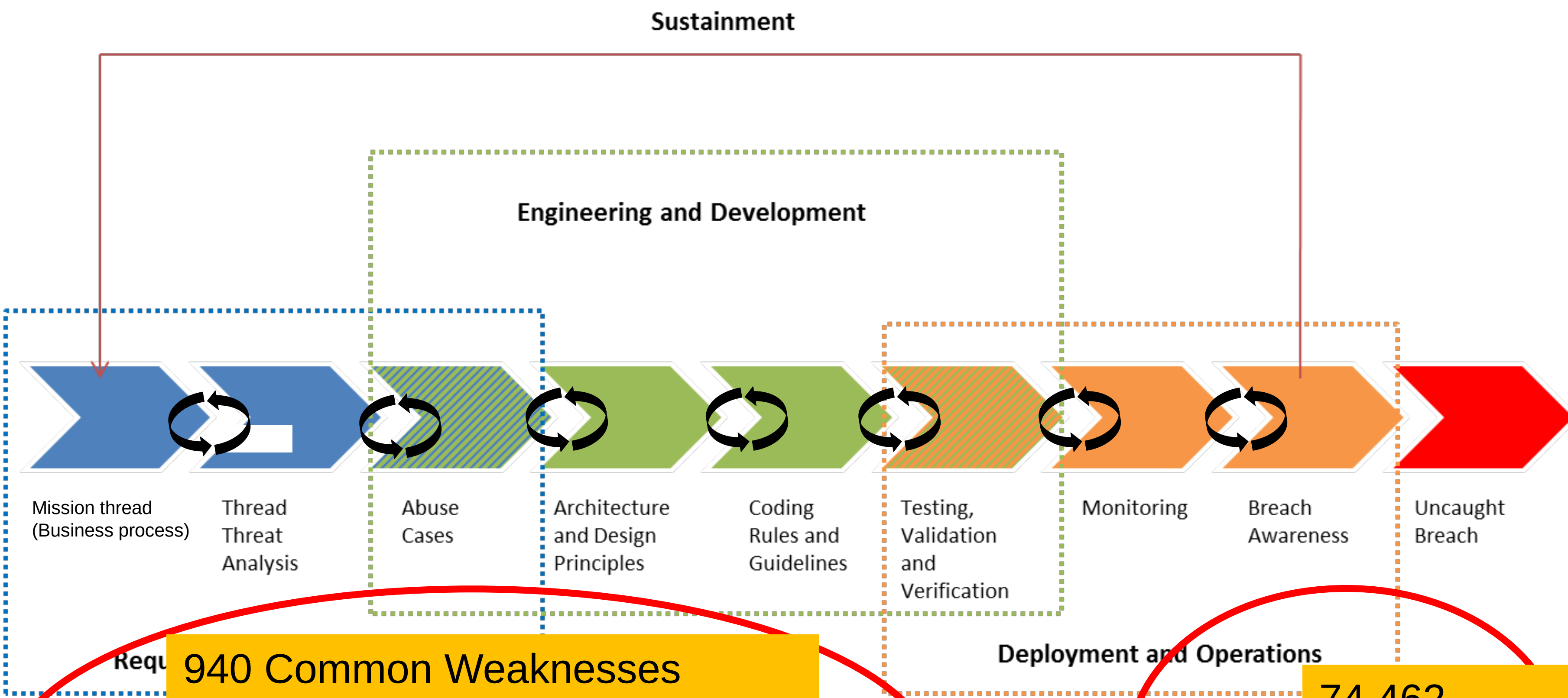
- defect range of 1,020 – 10,200
- range of vulnerabilities from 51 – 510.

The F-35 Lightning II has 24 MLOC

- 14,400 – 144,000 defects
- 720-7,200 vulnerabilities

Even more vulnerabilities if the code quality is poor!

Cybersecurity Is a Lifecycle Challenge



940 Common Weaknesses

Design Weaknesses

Coding Weaknesses

Implementation Weaknesses

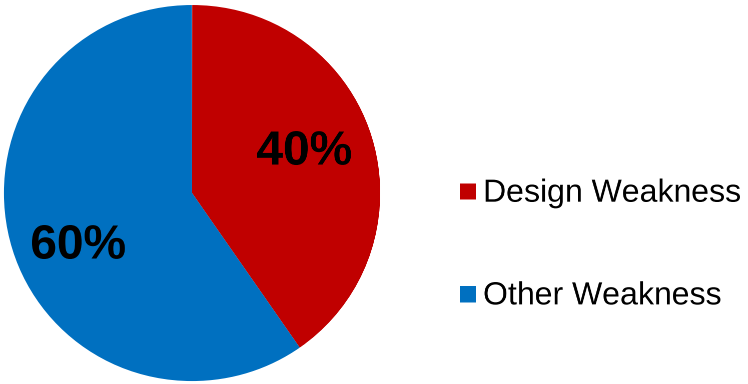
74,462 Common Vulnerability Enumerations (CVE)

CWE.Mitre.org 2014

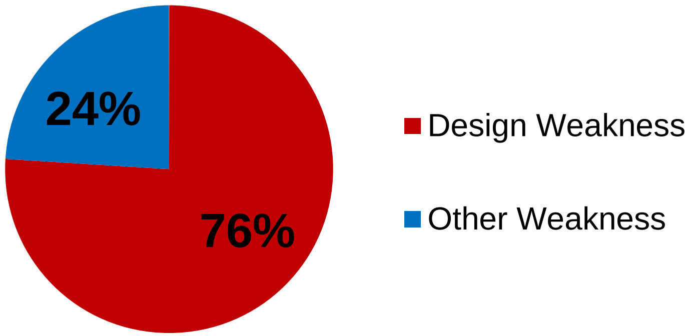
CVE.Mitre.org

Impact of Design Weaknesses

940 Total CWEs*



Top 25 CWEs
(Most Dangerous)



Source: http://cwe.mitre.org/ as of Feb 9, 2014

Causes for design weaknesses:

- Poor security requirements
- Limited understanding of the impact of security risk on mission success

Quality Processes Can Improve Security

Good quality will ensure proper implementation of specified results

- Effective code checking will identify improper implementations of specifications (11 of SANS Top 25)
- Effective design reviews will identify missing requirements (12 of SANS Top 25)
  - **if** appropriate security results are considered in the development of requirements
  - **if** requirements are effectively translated into detail designs and code specifications to support the required security results

(Reference: Woody, Carol; Ellison, Robert; & Nichols, William. *Predicting Software Assurance Using Quality and Reliability Measures.* CMU/SEI-2014-TN-026. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=428589

Security requirements must be properly specified
Are controls that address known malware in the requirements?

Why Isn't Known Malware Addressed?

Problem:

Despite the reported attacks on critical systems, operational techniques such as malware analysis are not used to inform early lifecycle activities, such as security requirements engineering

- Operational techniques like malware analysis are typically used for patch generation – there is no easy way to feed back into the development process.
- Developers of security requirements tend to either start with a blank slate or with large databases of candidate requirements and use cases based on organizational policy.
- Creation and prioritization of security requirements is largely done without the insights gained from analysis of prior attacks, especially those that are specific to a particular domain.

Proposed Solution:

Malware vulnerabilities annotated with use cases and domain specific considerations will allow improve inclusion in requirements

Malware-analysis Driven Use Case Creation

Malware already analyzed by domain expert (CWE, CAPEC)

Is it exploiting a design weakness?

If yes, additional information needed (see example in backup slides)

- Determination that requirements were overlooked
- Identification of misuse
- Creation of requirements use case that addresses the misuse
- Augment with impact analysis and domain critical criteria

Pilot Research Findings

- Structured mechanisms to include data from known malware attacks into requirements and architecture processes are nonexistent.
- When designs ignore these types of attacks, important security controls are omitted.
- Even projects that do some form of threat modeling fail to systematically consider prior successful exploits.
- Evidence indicates that projects with detailed data about successful prior attacks are more likely to appropriately create critical mitigations.

Mead, N.R., Morales, J. A., Alice, G. P., “A Method and Case Study for Using Malware Analysis to Improve Security Requirements”, *International Journal of Secure Software Engineering*, IGI Publishing, 6(1), pp.1-23, January-March 2015

Recommendation

Extend existing malware resources for each design weakness resulting from missing requirements to add the associated malware exploit analysis, malware misuse case, mitigation use case(s), and overlooked security requirement(s) needed for including them in requirements and design

Identification of the key application domains where the missing requirement is being exploited (e.g. mobile, cyber-physical, Web interfaces, Autonomy, etc.) will assist designers in making appropriate priority selections

Case Study Example from Pilot Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

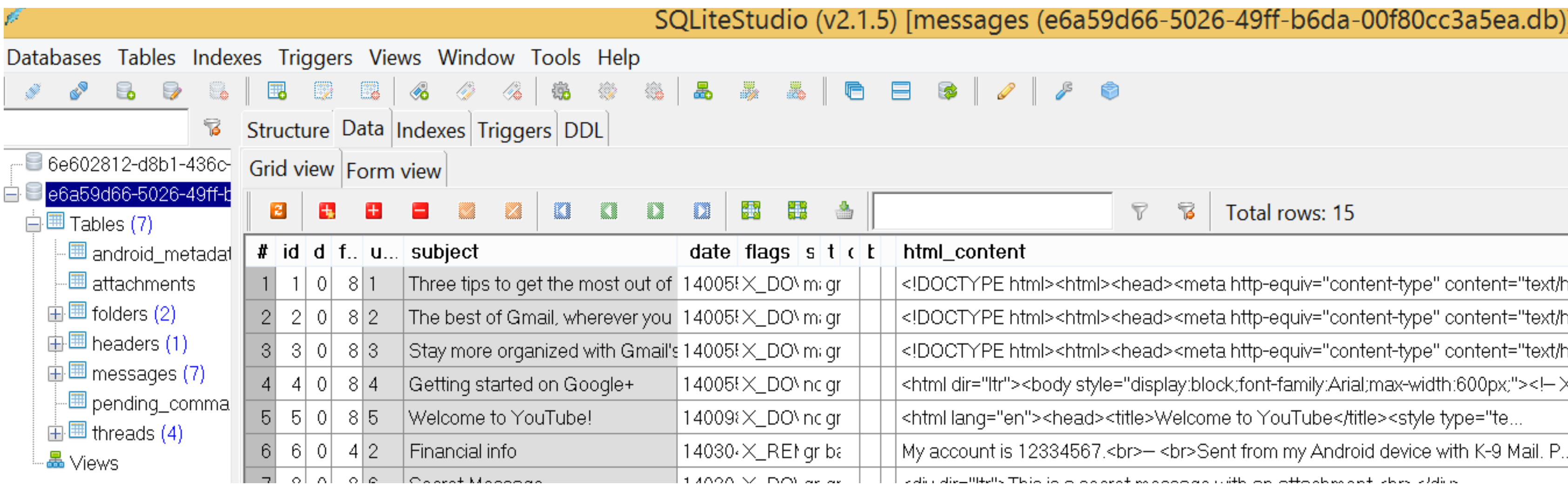
Case Study - Vulnerability

DroidCleaner

- Trojan malware
  - Claims to perform an Android tune-up.
  - Sends premium rate SMS messages.
  - Uploads data from the Android External Storage area to hacker's servers.

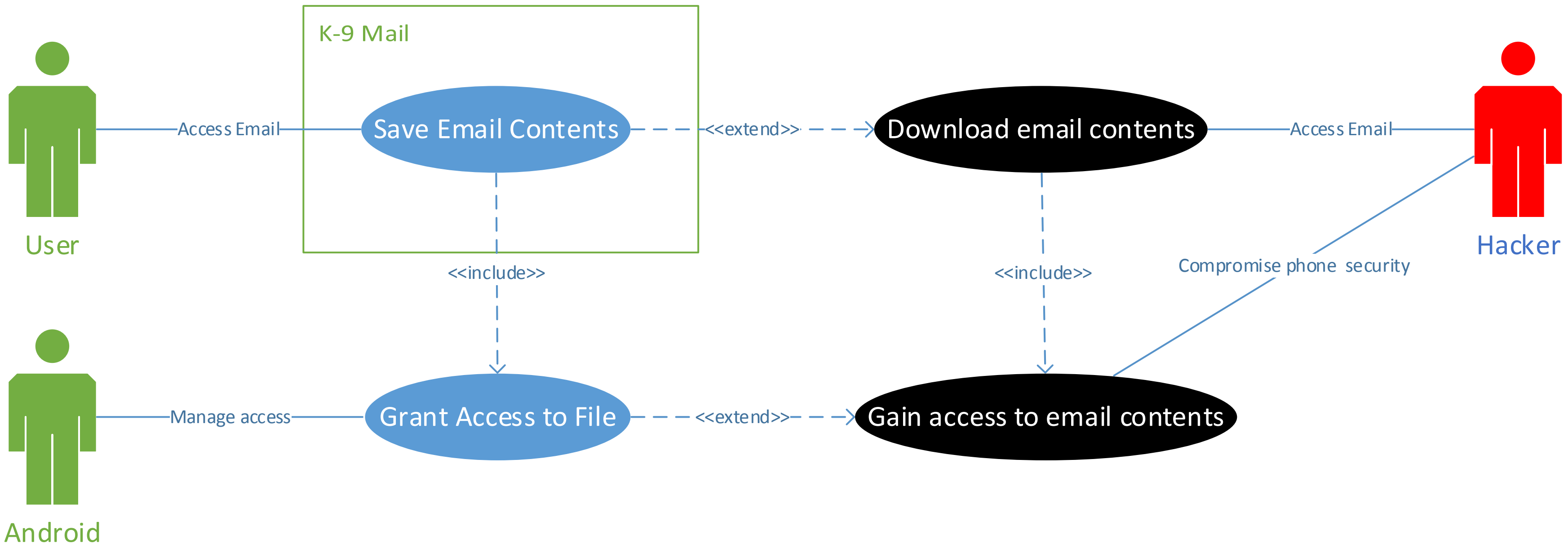
Case Study – Exploitation Scenario

- Trojan
  - Social Engineering to trick user into installing DroidCleaner:
    - Install software
    - Grant access to external storage, internet access
- K-9 Mail configured to store email in External Storage
- DroidCleaner uploads External Storage to hacker server.
- Hacker examines contents. Email contents disclosed:



Case Study – Misuse Case

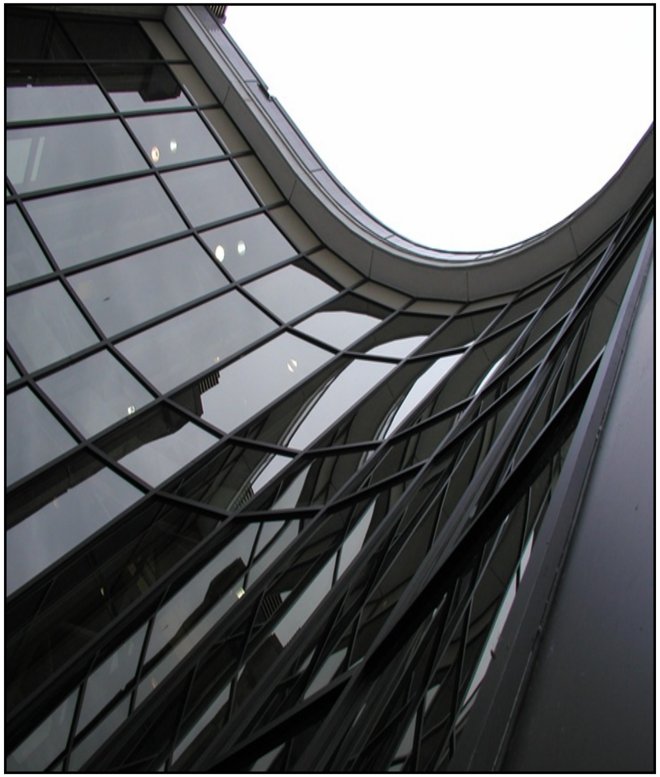
Gain Access to Email Contents



Case Study – New Requirement

Requirement Number: 1	
Requirement	1.1 Email contents shall be protected from unauthorized access. Email contents shall be stored in an area only available to the application (Android Internal Storage default configuration) – and/or – protected through encryption which cannot be decrypted using data available in Android External Storage. 1.2 Processes with access to External Storage shall not have the ability to view K-9 Mail contents in clear text. If external storage is selected, a warning message or mitigation, such as encryption is recommended.
Category	Data Protection
Priority	High
Cost	Medium
Misuse Case	MUC2
Rationale	Due to the high risk of data theft malware on Android, it is not safe to assume data kept on the phone is private, therefore the email contents must be kept in a form which cannot be read even if the Hacker has access to the storage location.

Contact Information



Carol Woody, Ph.D.

cwoody@cert.org

Web Resources (CERT/SEI)

http://www.cert.org/cybersecurity-engineering/

http://www.sei.cmu.edu/