Smartphones (Android) in Wars: Software Assurance Challenges

Jeff Voas and Angelos Stavrou

US National Institute of Standards and Technology and George Mason University

Overall Project Goal: Software Assurance

Software Assurance is a combination of:

(1) the degree to which the *functional* requirements are defined and satisfied,

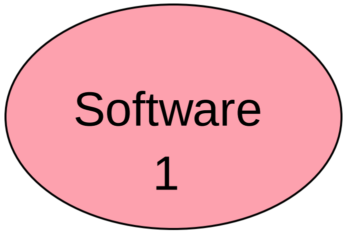
(2) the degree to which the *non-functional* requirements ("ilities") are defined and satisfied,

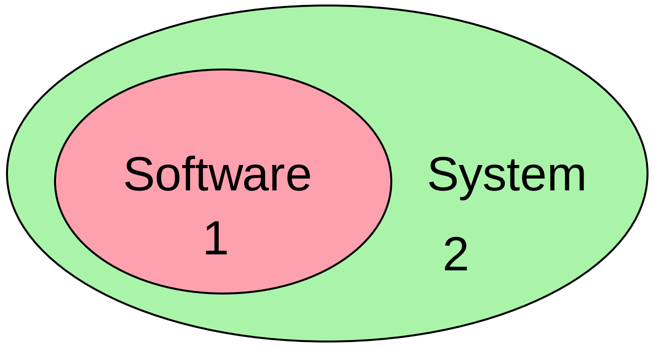
(3) and the degree to which the "*shall not*" requirements are defined and mitigated.

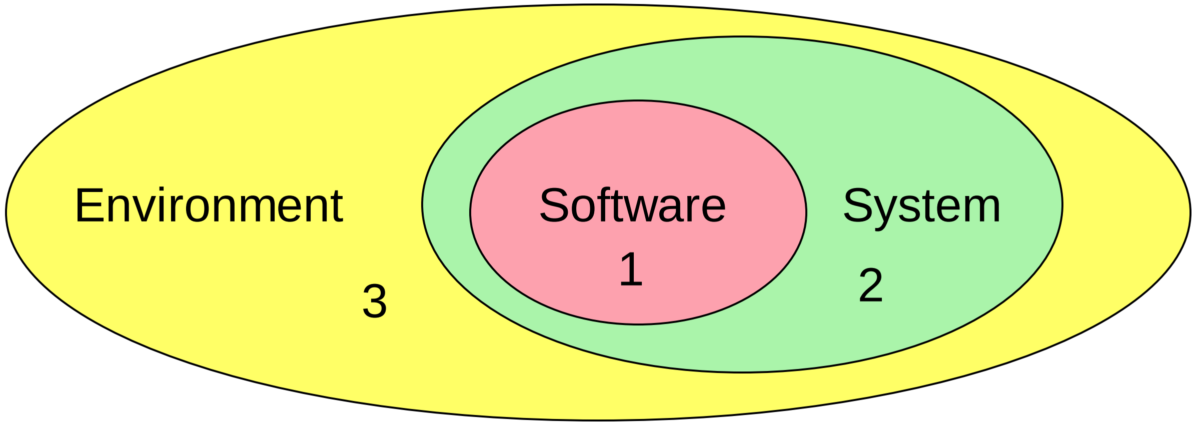
Overall Idea

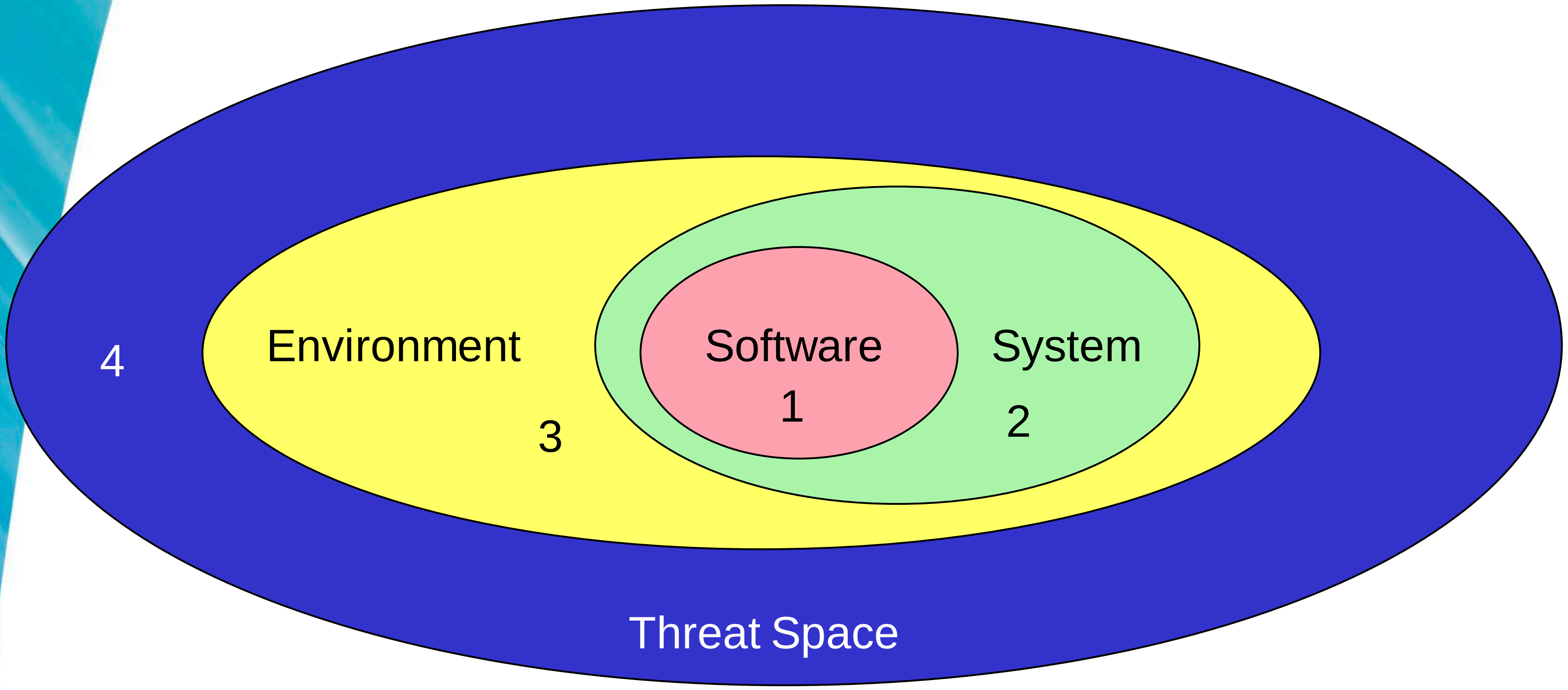
- Software assurance is <u>always</u> a function of time,
- Software assurance is a <u>non-boundable</u> problem, except in rare cases, and only partially, such as embedded systems.
- Software today is increasingly less viewed as a <u>product</u> and more viewed as a <u>service</u> , and therefore *service assurance* is vital to understand.

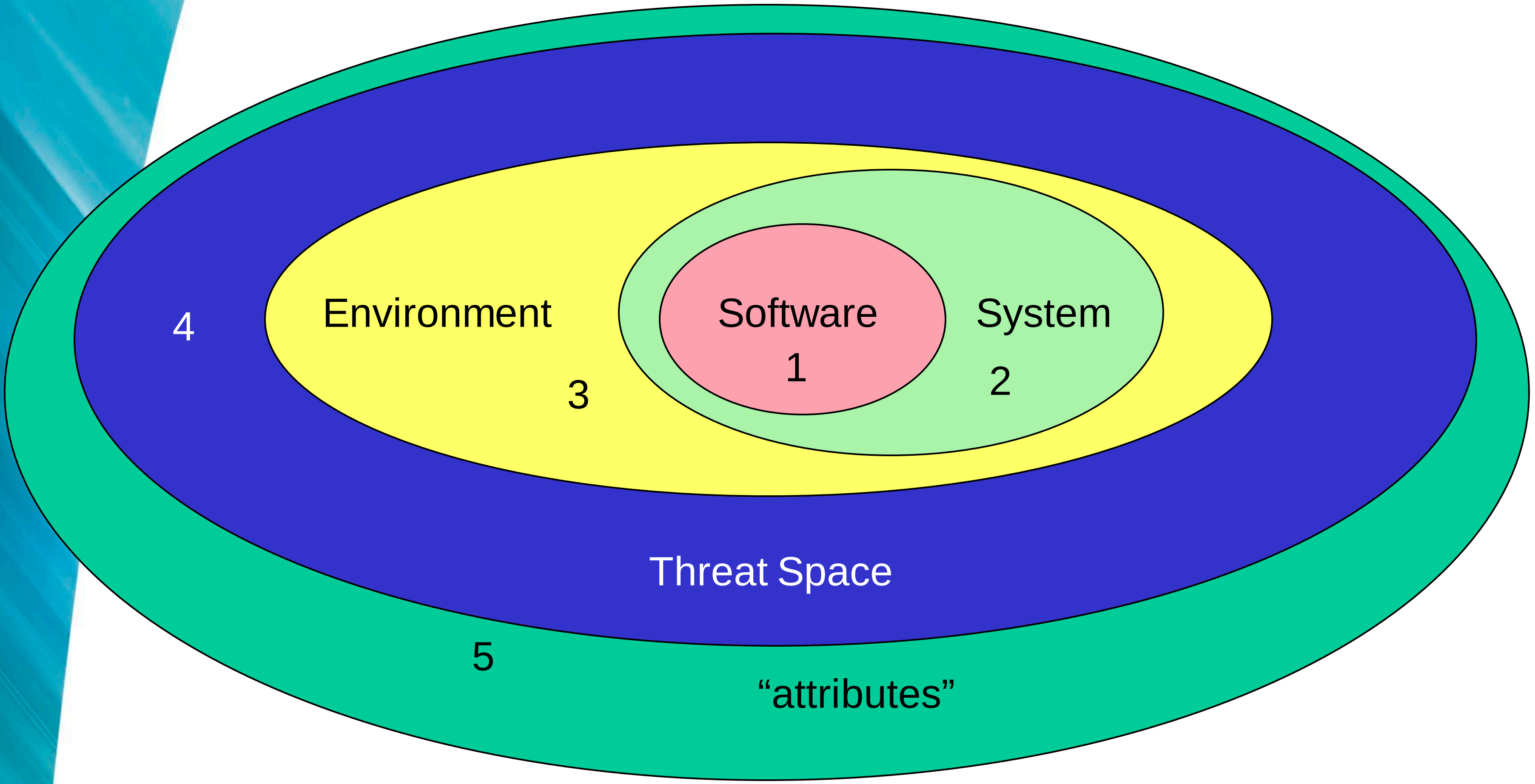
Framing the Discussion

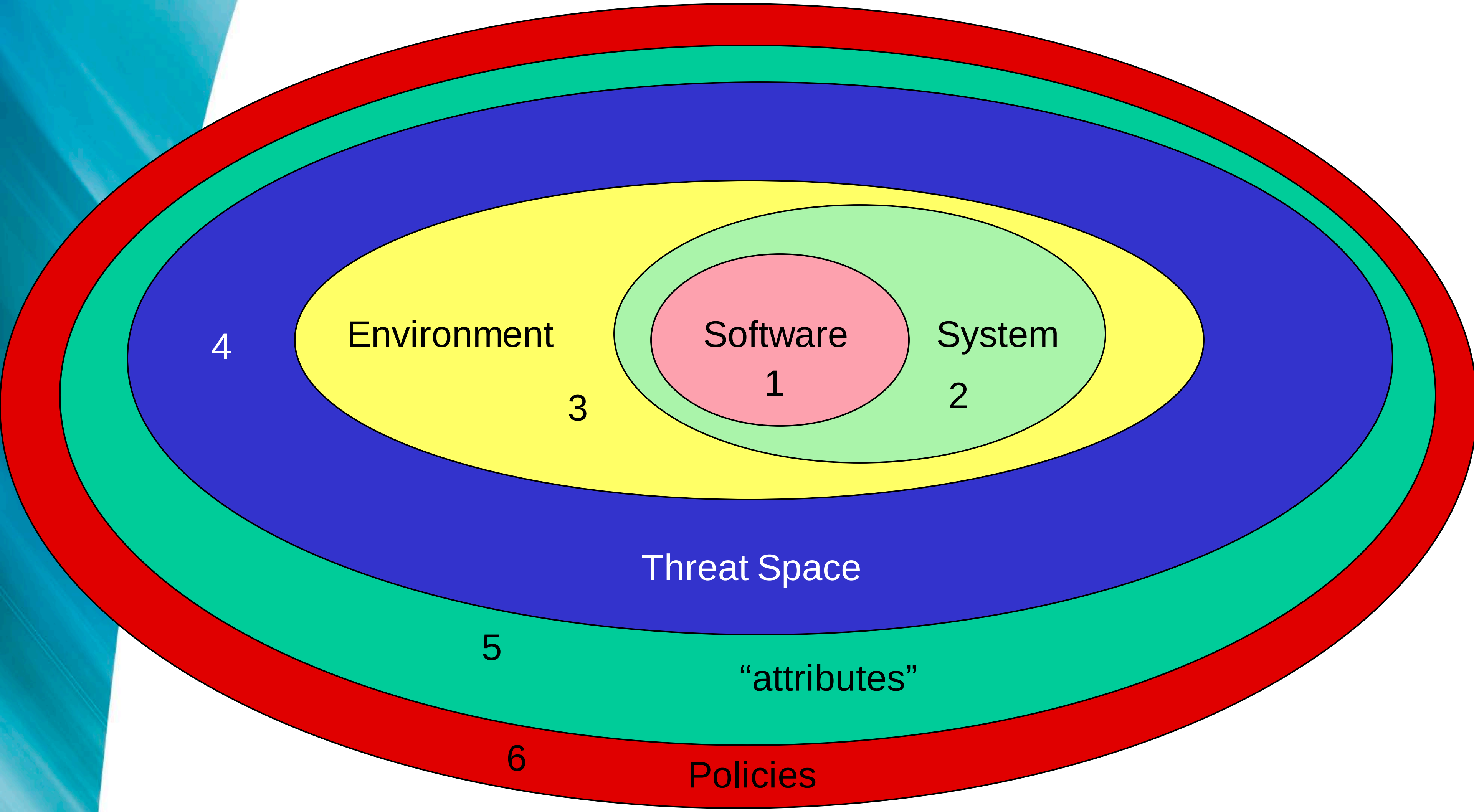


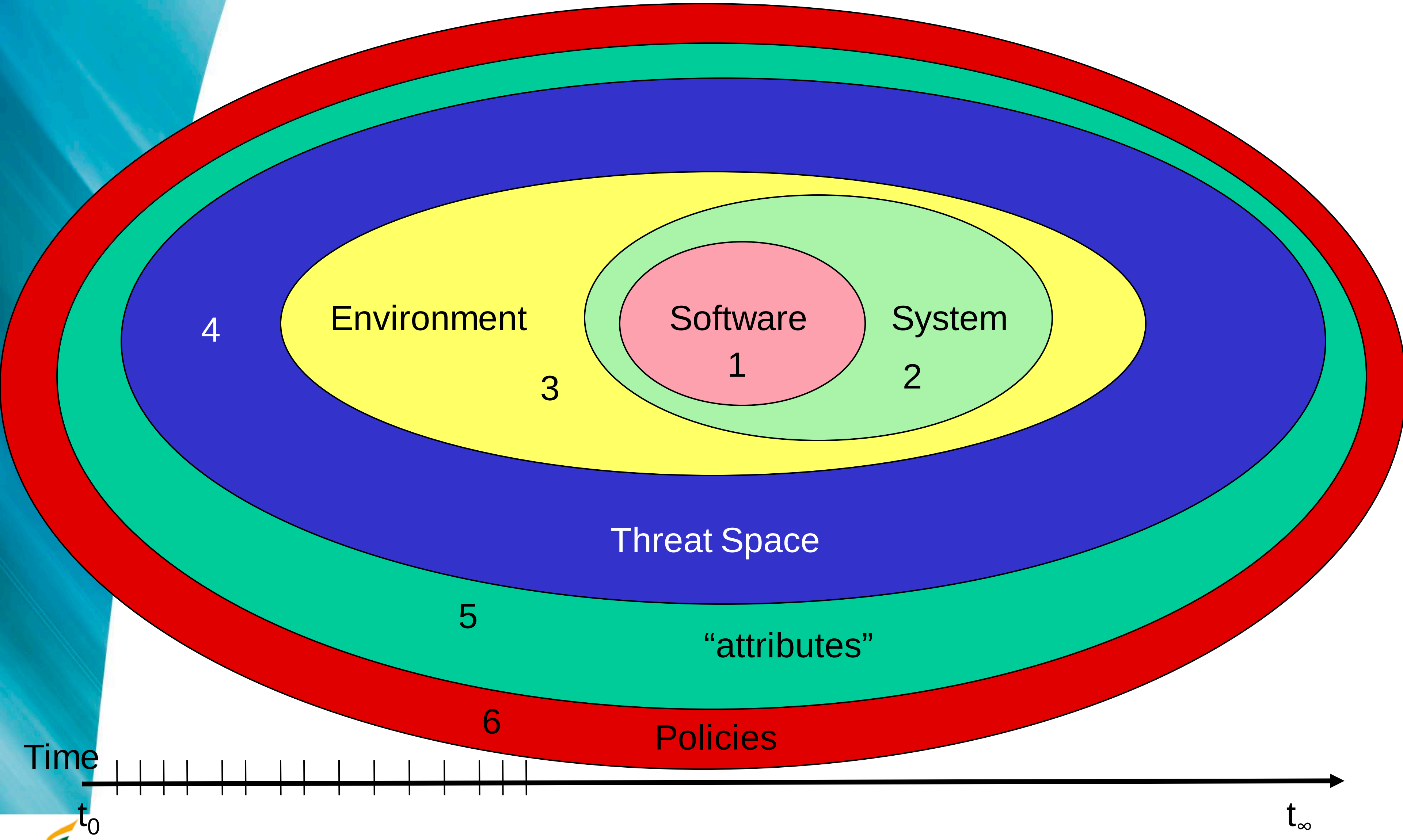

Software
1
System
2

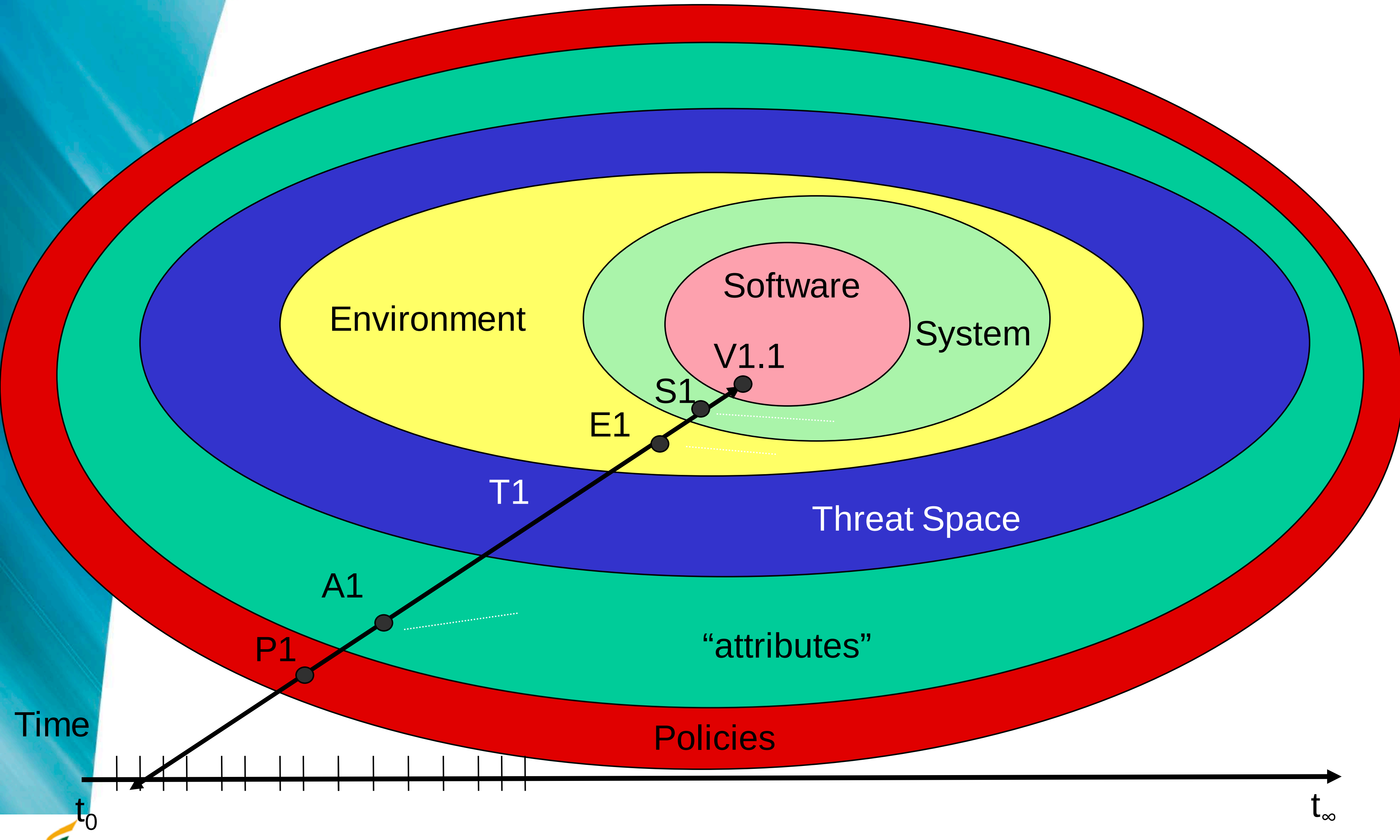

Environment
3
Software
1
System
2

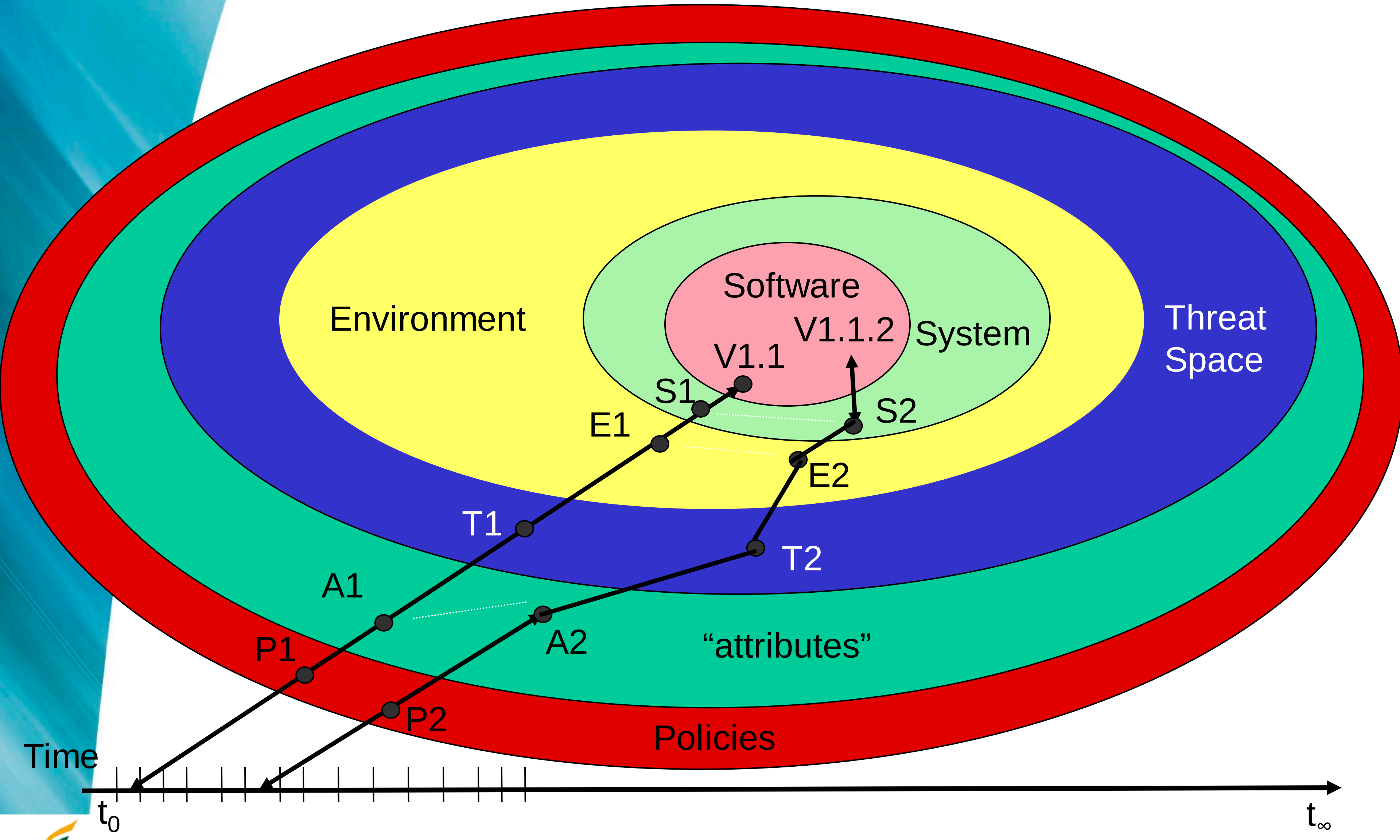

Environment
Software
1
System
2
3
4
Threat Space


4
Environment
3
Software
1
System
2
Threat Space
5
“attributes”


Environment
Software
System
4
1
3
2
Threat Space
5
"attributes"
6
Policies


Environment
Software
System
4
1
3
2
Threat Space
5
"attributes"
6
Policies
Time
t_0
t_∞


Software
Environment
System
V1.1
S1
E1
T1
Threat Space
A1
"attributes"
P1
Policies
Time
t_0
t_∞

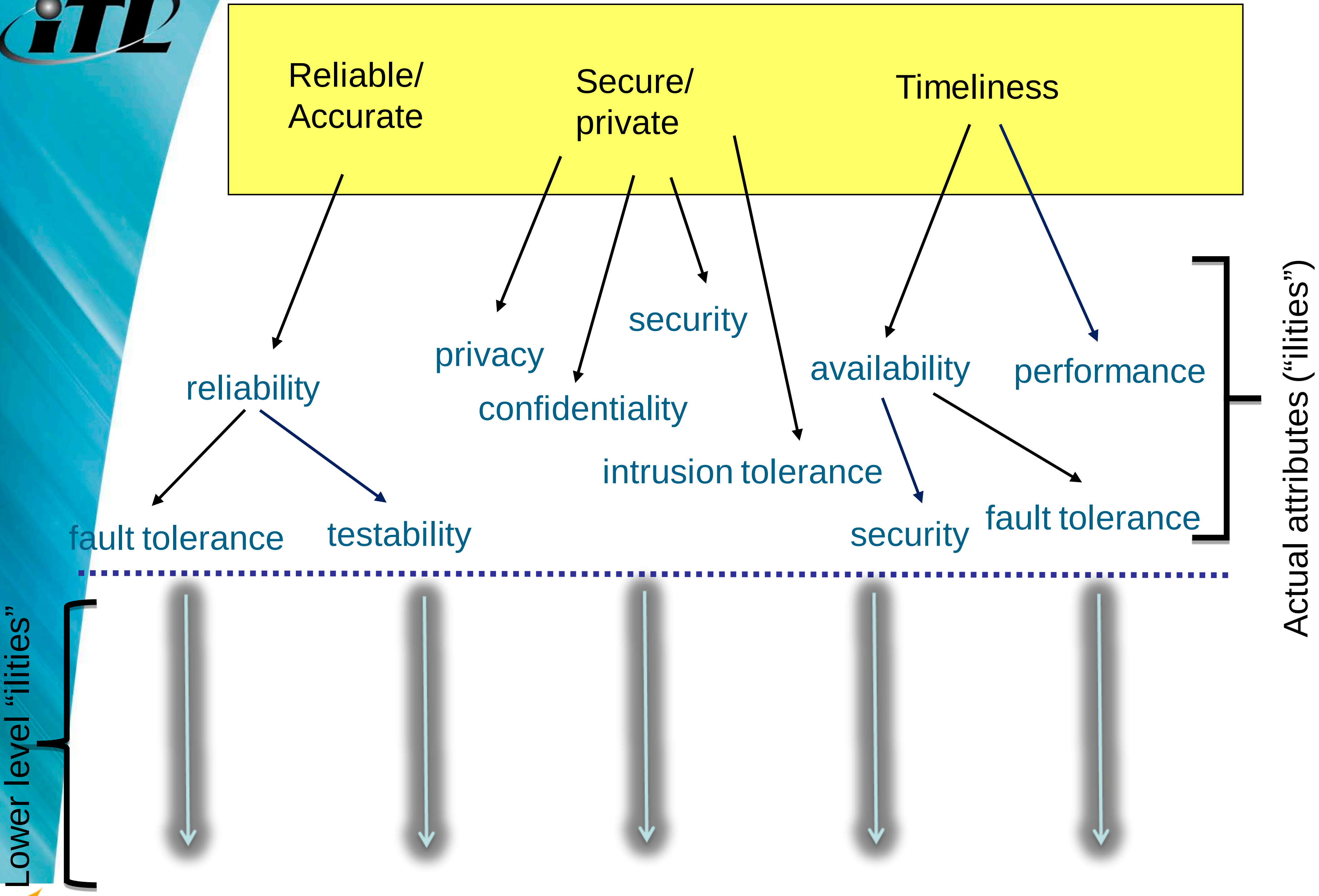

Software
V1.1.2
V1.1
System
Environment
Threat Space
S1
S2
E1
E2
T1
T2
A1
A2
"attributes"
P1
P2
Policies
Time
t_0
t_∞

QoS “attributes”

Reliable/ Accurate

Secure/ private

Timeliness


Reliable/ Accurate
Secure/ private
Timeliness
reliability
privacy
security
confidentiality
intrusion tolerance
availability
performance
fault tolerance
testability
security
fault tolerance
Actual attributes ("ilities")
Lower level "ilities"

Pre-Define Attributes

- System and Software Requirements should <u>prescribe</u>, at some level of granularity, how much of each "ility" is desired, i.e., the *non-functional attributes*.
- HOW?
- <u>Note: Ignoring non-attributes is not an option for achieving high assurance.</u>

Weighting Attributes

w_1Reliability	w_2Performance
w_3Fault-tolerance	w_4Safety
w_5Security	w_6Availability
w_7Testability	w_8Maintenance

in order to not over-design any attribute into the system.

For example, for an cloud-based transaction processing application, w_4 would probably ≈ 0.0, and w_7 would be $\leq w_8$

Tradeoffs

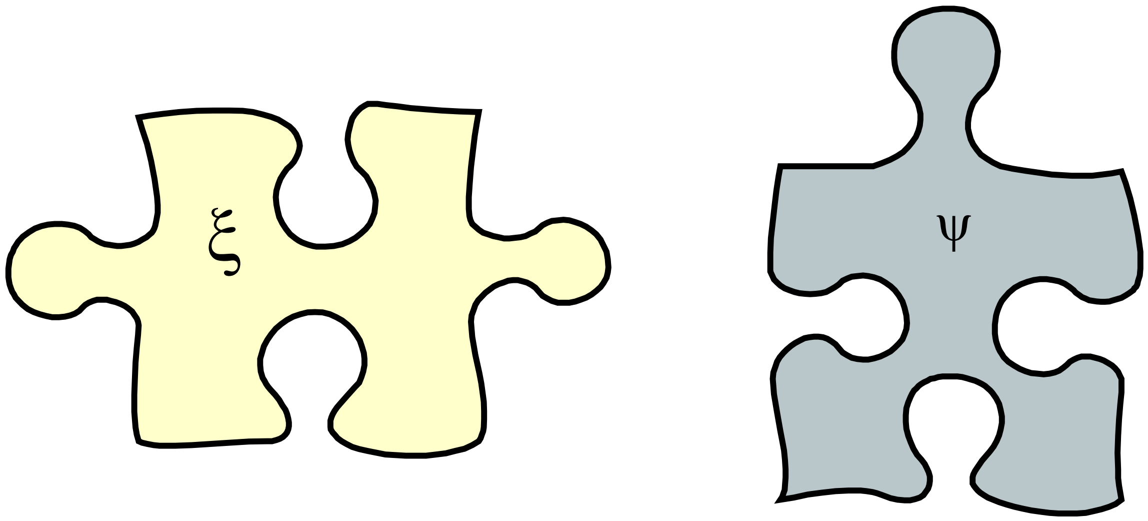
How much will you spend for increased reliability knowing that doing so will take needed, financial resources away from security or performance or …?

Mathematical Equation?

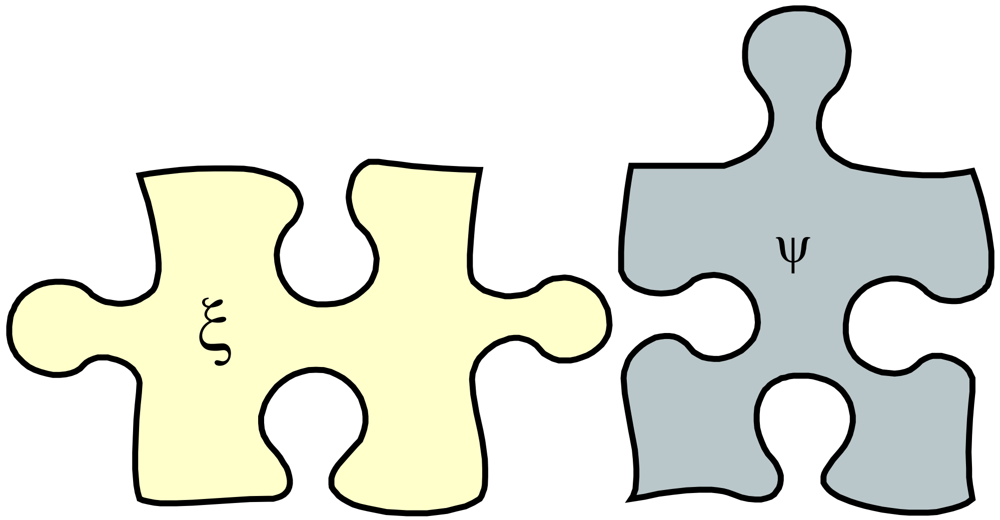
$$Assurance = \mathscr{f}(aR, bP, cF, dSa, eSe, fA, gT, hM)$$

where *a, b, c, d, e, f, g,* and *h* are quantitative or qualitative measures of particular attributes.

Two Components



With Attributes



ξ has the following properties:

(*a*R, *b*P, *c*F, *d*Sa, *e*Se, *f*A, *g*T, *h*M)

ψ has the following properties:

(*i*R, *j*P, *k*F, *l*Sa, *m*Se, *n*A, *o*T, *p*M)

Key Problem

- Attributes are only reasonable to talk about within the context of a system, i.e., it is not reasonable to talk about them and attempt to measure them as standalone component properties.

- Thus the composability of environments is as interesting of a problem as the composability of attributes.

Basic Foes

- Security *vs.* Performance
- Fault tolerance *vs.* Testability
- Fault tolerance *vs.* Performance
- etc.

Counterintuitive Realities

- 100% safety and 0% reliability
- 100% reliability and 0% safety
- 0% functionality/reliability and 100% security
- 100% availability and 0% reliability
- 100% availability and 0% performance
- 0% performance and 100% safety

Security Design Goals and Objectives

End-to-End Security that encompasses ALL Participating Entities

- ❖ Device Security
- ❖ Application Security
- ❖ Application Store Security
- ❖ Provisioning Security
- ❖ Identity Management of Users & Applications

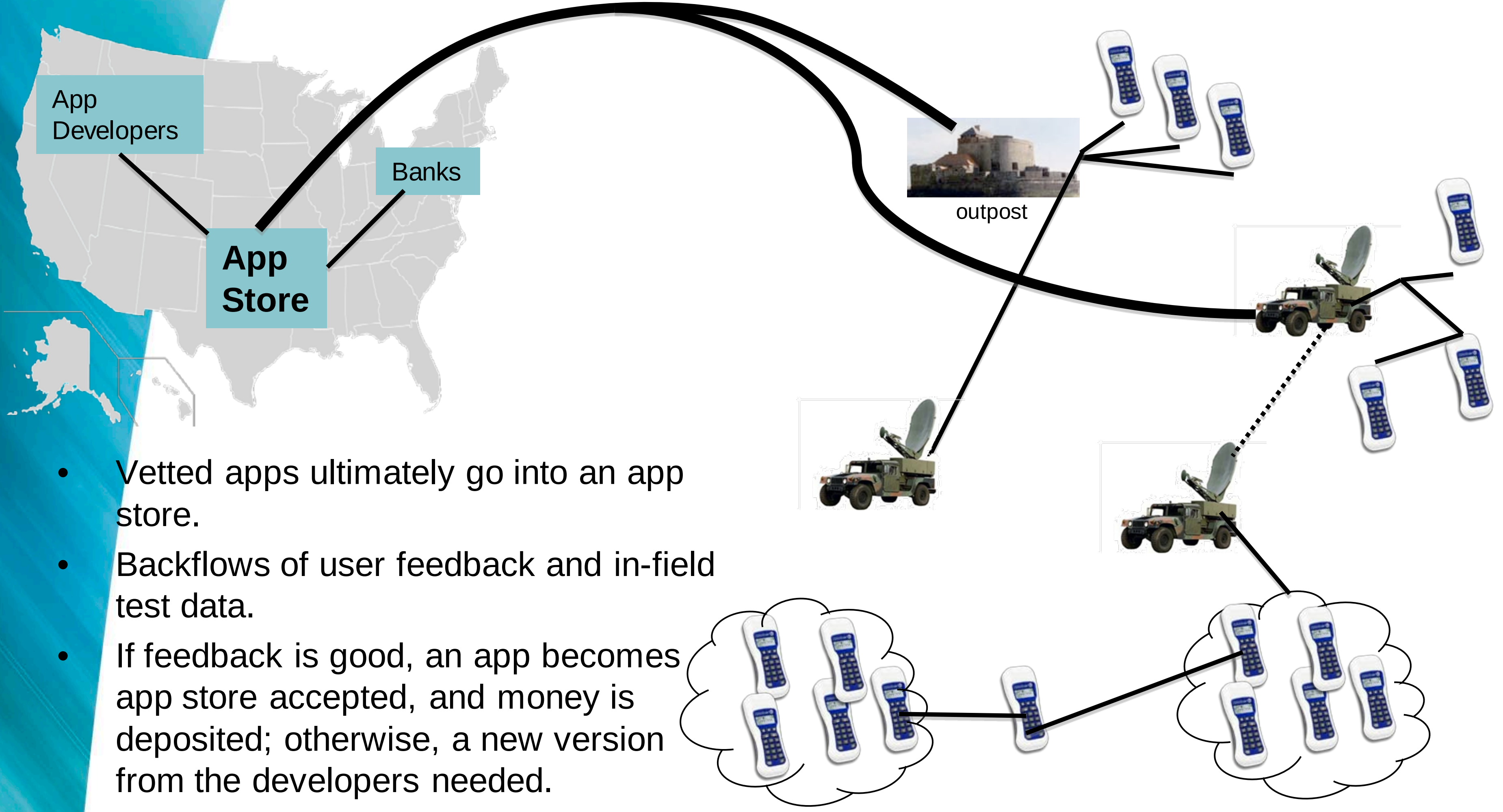
Security Customization of each device for the Mission

- ❖ Device and Application Security tailored to the Mission Objectives
- ❖ Automatic & Flexible Provisioning & Phone Reconfiguration

Defense in-Depth: Multiple Levels of Security

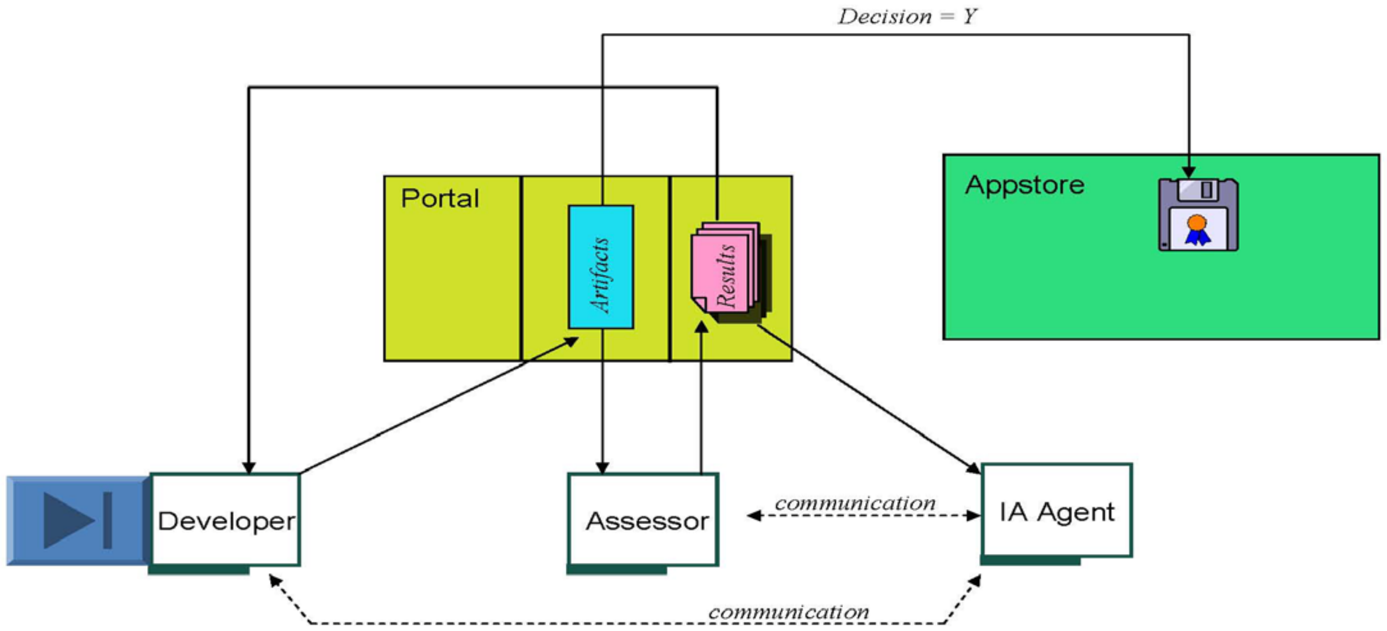
- **Application Vetting & Testing**
- Device Lock-down and Encryption of ALL Data and Communications
- Enforcement of Security Policies in the Android Framework
- Second-level Defenses placed in the Android Linux Kernel
  - Prevent Attacks that bypass Android Security Framework
  - Android has Inherited some (if not all) of the Linux Vulnerabilities
  - **Java Native Interface to Linux Libraries a potential Avenue for Exploitation**

High-Level App Overview

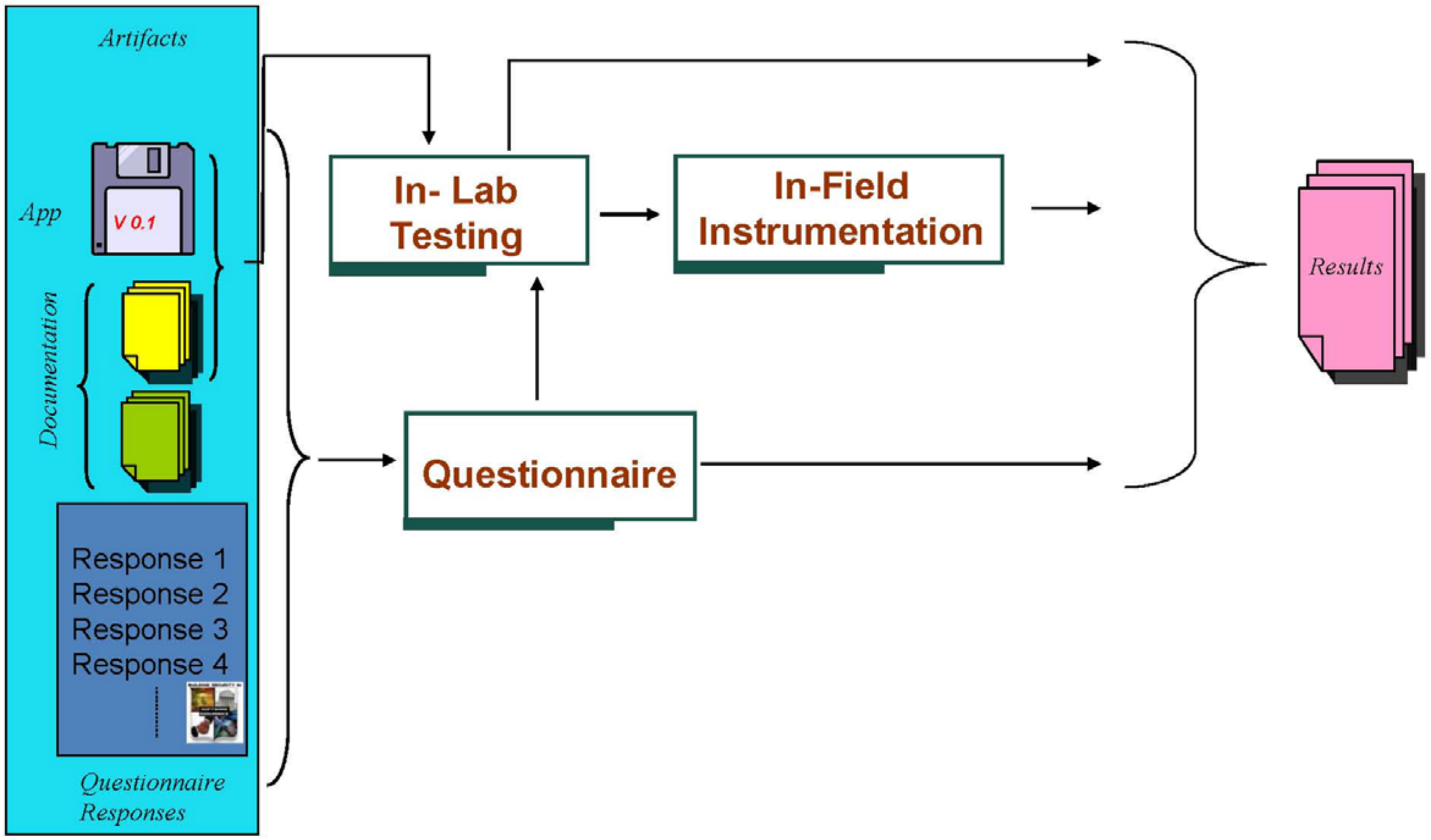


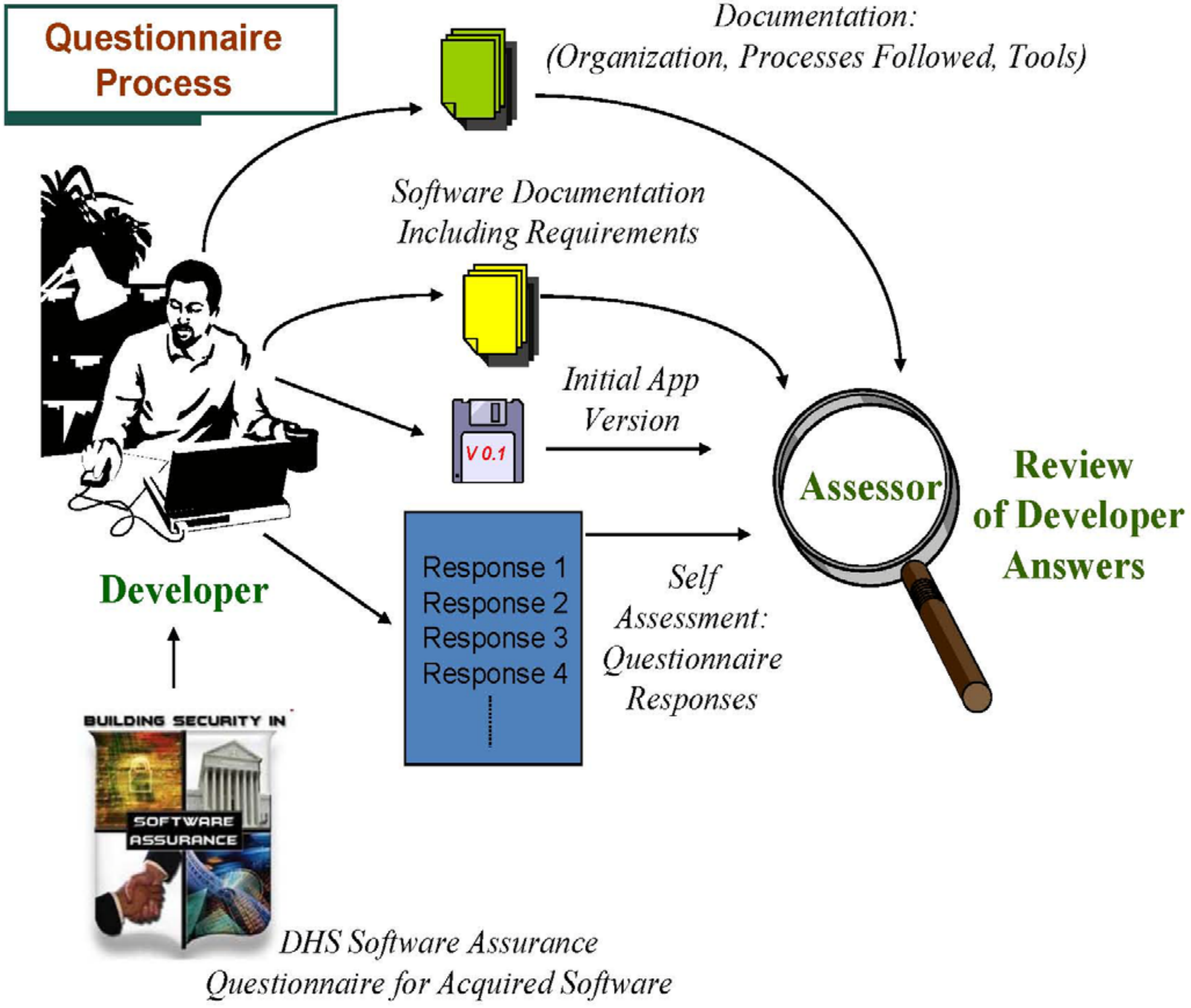
- Vetted apps ultimately go into an app store.
- Backflows of user feedback and in-field test data.
- If feedback is good, an app becomes app store accepted, and money is deposited; otherwise, a new version from the developers needed.

Application Vetting: Big Picture



Progression of Testing




Questionnaire Process
Documentation:
(Organization, Processes Followed, Tools)
Software Documentation Including Requirements
Initial App Version
V 0.1
Assessor
Review of Developer Answers
Developer
Response 1
Response 2
Response 3
Response 4
Self Assessment: Questionnaire Responses
BUILDING SECURITY IN
SOFTWARE ASSURANCE
DHS Software Assurance Questionnaire for Acquired Software

National Institute of Standards and Technology

Results

- An examiner will review the responses from the developer

- A human evaluation of trust in the responses is made

- If questionable, developer may be asked for clarifications or app recommended for re-work

- If believed, app is ready for app store inclusion or for next two assurance approaches

Questions

1. Does the app that you offer contain any security issues that need to be mitigated or revealed? For example, does the app access phone, network (wifi and/or Bluetooth), and/or GPS resources?, does the app access camera or other recording hardware or software?, does the app access operating system resources or libraries?", and "what information, if any, is transferred from the phone to a remote host (including during PC synchronization) by the app?". If so, please explain. If you believe that your app does not contain any security issues, please explain why, and then ignore remaining Questionnaire. If you answer "no" here, please give detail and good reasoning. If you answer "yes" please address as many of the following questions as are reasonable for your circumstances. Lacking information may trigger additional review of your application for app approval and thus delay adoption into the appstore.
2. How long has the software source been available? Is there an active user community providing peer review and actively evolving the software?
3. Does the license/contract restrict the licensee from discovering flaws or disclosing details about software defects or weaknesses with others (e.g., is there a "gag rule" or limits on sharing information about discovered flaws)?
4. Does software have a positive reputation? Does software have a positive reputation relative to security? Are there reviews that recommend it?
5. What are the processes (e.g., ISO 9000, CMMI, etc.), methods, tools (e.g., IDEs, compilers), techniques, etc. used to produce and transform the software (brief summary response)?
6. What security measurement practices and data does the company use to assist product planning?
7. Describe the training the company offers related to defining security requirements, secure architecture and design, secure coding practices, and security testing. Explain.
8. Are there some requirements for security that are "structured" as part of general release-ability of a product and others that are "as needed" or "custom" for a particular release?
9. What process is utilized by the company to prioritize security-related enhancement requests?
10. What review processes are implemented to ensure that nonfunctional security requirements are unambiguous, traceable and testable throughout the entire Software Development Life Cycle (SDLC)?
11. Are security requirements developed independently of the rest of the requirements engineering activities, or are they integrated into the mainstream requirements activities?
12. Are misuse/abuse cases derived from the application requirements? Are relevant attack patterns used to identify and document potential threats?
13. What threat assumptions were made, if any, when designing protections for the software and information assets processed?
14. What security design and security architecture documents are prepared as part of the SDLC process?
15. What threat modeling process, if any, is used when designing the software protections?
16. How are confidentiality, availability, and integrity addressed in the software design?
17. What are/were the languages and non-developmental components used to produce the software (brief summary response)?
18. What secure development standards and/or guidelines are provided to developers?

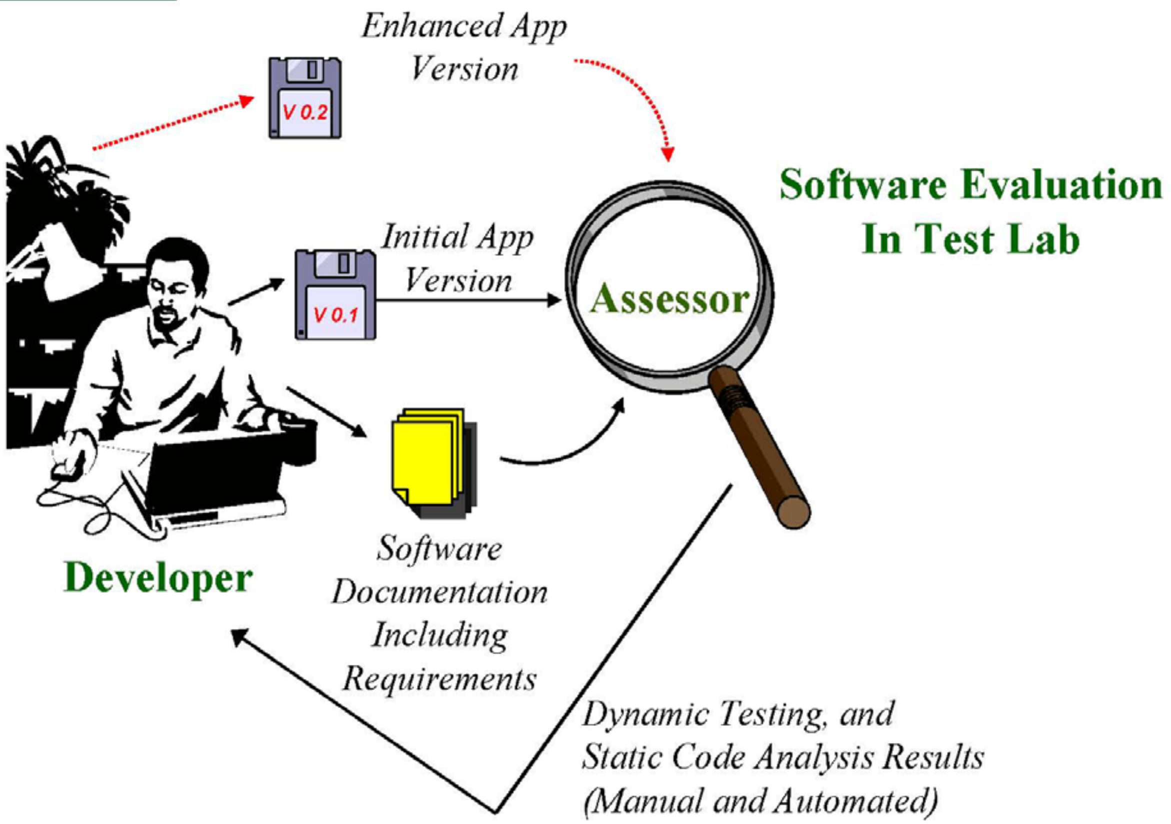
Questions (cont.)

19. Are tools provided to help developers verify that the software they have produced has a minimal number of weaknesses that could lead to exploitable vulnerabilities? What are the tools, and how have they been qualified? What is the breadth of common software weaknesses covered (e.g., specific CWEs)?
20. Does the company have formal coding standards for each language in use? If yes, how are they enforced?
21. Does the software development plan include security peer reviews?
22. Does the organization incorporate security risk management activities as part of the software development methodology? If yes, will a copy of the documentation of this methodology be available or information on how to obtain it from a publicly accessible source?
23. Does the software"s exception-handling mechanism prevent all faults from leaving the software, its resources, and its data (in memory and on disk) in a vulnerable state? Does the exception-handling mechanism provide more than one option for responding to a fault? If so, can the exception-handling options be configured by the administrator or overridden?
24. Does the software validate (e.g., filter with white listing) inputs from potentially un-trusted sources before being used? Is a validation test suite or diagnostic available to validate that the application software is operating correctly and in a secure configuration following installation?
25. Has the software been designed to execute within a constrained execution environment (e.g., virtual machine, sandbox, chroot jail, single-purpose pseudo-user, etc.) and is it designed to isolate and minimize the extent of damage possible by a successful attack?
26. Does the documentation explain how to install, configure, and/or use the software securely? Does it identify options that should not normally be used because they create security weaknesses?
27. Where applicable, does the program use run-time infrastructure defenses (such as address space randomization, stack overflow protection, preventing execution from data memory, and taint checking)?
28. How does the company minimize the risk of reverse engineering of binaries? Are source code obfuscation techniques used? Are legal agreements in place to protect against potential liabilities of non-secure software?
29. Does the software default to requiring the administrator (or user of a single-user software package) to expressly approve the automatic installation of patches/upgrades, downloading of files, execution of plug-ins or other "helper" applications, and downloading and execution of mobile code?
30. Does the software have any security critical dependencies or need additional controls from other software (e.g., operating system, directory service, applications), firmware, or hardware? If yes, please describe.
31. Does the software include content produced by suppliers other than the primary developer? If so, who?
32. What are the policies and procedures for verifying the quality and security of non-developmental components used?
33. What types of functional tests are/were performed on the software during its development (e.g., spot checking, component-level testing, integrated testing)?
34. Who and when are security tests performed on the product? Are tests performed by an internal test team, by an independent third party, or by both?
35. What degree of code coverage does testing provide?
36. Are misuse test cases included to exercise potential abuse scenarios of the software?

Questions (cont.)

37. Are security-specific regression tests performed during the development process? If yes, how frequently are the tests performed? Are regression test scripts available?
38. Does the company‟s defect classification scheme include security categories? During testing what proportion of identified defects relate to security?
39. How has the software been measured/assessed for its resistance to identified, relevant attack patterns? Are Common Vulnerabilities & Exposures (CVEs) or Common Weakness Enumerations (CWEs) used? How have exploitable flaws been tracked and mitigated?
40. Has the software been evaluated against the Common Criteria, FIPS 140-2, or other formal evaluation process? What evaluation assurance was achieved? If the product claims conformance to a protection profile, which one(s)? Are the security target and evaluation report available?
41. Are static or dynamic software security analysis tools used to identify weaknesses in the software that can lead to exploitable vulnerabilities? If yes, which tools are used? What classes of weaknesses are covered? When in the SDLC (e.g., unit level, subsystem, system, certification and accreditation) are these scans performed? Are SwA experts involved in the analysis of the scan results?
42. Are there current publicly-known vulnerabilities in the software (e.g., an unrepaired CWE entry)?
43. Has the software been certified and accredited? What release/version/configuration? When? By whom? What criteria or scheme was used to evaluate and accredit the software?
44. Is there a Support Life cycle Policy within the organization for the software in question? Does it outline and establish a consistent and predictable support timeline?
45. How will patches and/or Service Packs be distributed to the purchasing/using organization?
46. How extensively are patches and Service Packs tested before they are released?
47. How are reports of defects, vulnerabilities, and security incidents involving the software collected, tracked, and prioritized?
48. What policies and processes does the company use to verify that software components do not contain unintended, "dead", or malicious code? What tools are used?
49. How frequently are major versions of the software released?
50. Are configuration/change controls in place to prevent unauthorized modifications or additions to source code and related documentation? Do these controls detect and report unexpected modifications/additions to source code? Do they aid in rolling back an affected artifact to a pre-modified version?
51. Does the company perform background checks on members of the software development team? If so, are there any additional "vetting" checks done on people who work on critical application components, such as security? Explain.
52. Please provide company names of all 3rd party entities (foreign and domestic) with whom you, the supplier, contracts software development for this app.

In-Lab Testing Process


Enhanced App Version
V 0.2
Software Evaluation In Test Lab
Initial App Version
V 0.1
Assessor
Developer
Software Documentation Including Requirements
Dynamic Testing, and Static Code Analysis Results (Manual and Automated)

Results

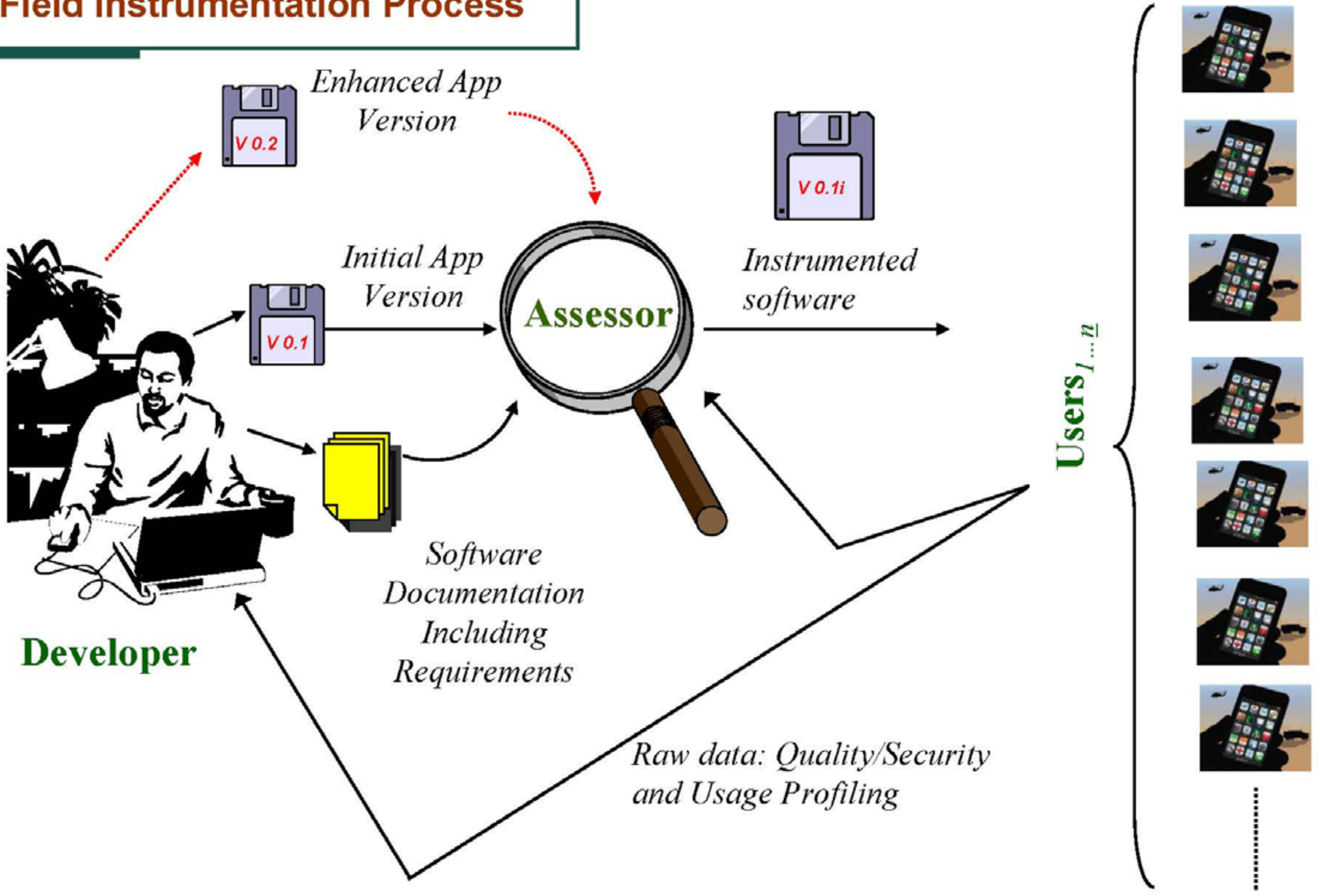
- Dynamic *reliability* and *performance* measurement of the product in the lab under assumed operational profiles.

- Static analysis of the source code using COTS and open source tools that search for programming errors such as buffer overflows.
  - Top 25 Common Programming Weaknesses CWEs) [http://cwe.mitre.org/top25/#ProfileAutomatedManual]
  - Capability to identify Application Bugs and Unwanted Functionality

Common Weakness Enumerations (CWEs)

- Detects hundreds of CWEs
- Detects 23 of the top 25 CWEs:
  - 1. CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
  - 2. CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
  - 3. CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
  - 4. CWE-352 Cross-Site Request Forgery (CSRF)
  - 5. CWE-285 Improper Access Control (Authorization)
  - 6. CWE-807 Reliance on Untrusted Inputs in a Security Decision
  - 7. CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
  - 8. CWE-434 Unrestricted Upload of File with Dangerous Type
  - 9. CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
  - 10. CWE-311 Missing Encryption of Sensitive Data
  - 11. CWE-798 Use of Hard-coded Credentials
  - 12. CWE-805 Buffer Access with Incorrect Length Value
  - 13. CWE-98 Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
  - 14. CWE-129 Improper Validation of Array Index
  - 15. CWE-754 Improper Check for Unusual or Exceptional Conditions
  - 16. CWE-209 Information Exposure Through an Error Message
  - 17. CWE-190 Integer Overflow or Wraparound
  - 18. CWE-131 Incorrect Calculation of Buffer Size
  - 19. CWE-306 Not supported
  - 20. CWE-494 Not supported
  - 21. CWE-732 Incorrect Permission Assignment for Critical Resource
  - 22. CWE-770 Allocation of Resources Without Limits or Throttling
  - 23. CWE-601 URL Redirection to Untrusted Site ('Open Redirect')
  - 24. CWE-327 Use of a Broken or Risky Cryptographic Algorithm
  - 25. CWE-362 Race Condition

In-Field Instrumentation Process



Results

- Data collected may include:
  - ❖ Amount of time an app is executed
  - ❖ Type and amount of data transmitted
  - ❖ Feature usage within an app
  - ❖ Number of exception calls

- Benefits include:
  - ❖ Usage data that can be used for billing,
  - ❖ Reducing number of apps/functionality
  - ❖ Additional app testing

Note: Instrumentation can be turned on and off easily, and done selectively as well. Also, instrumentation does incur performance and footprint hits.

Static & Dynamic Analysis not enough…

- Static & Dynamic Analysis have limitations
  - Cannot guarantee complete coverage of the application code
  - Remote Content Exploitation still possible

- Applications can cause Power Exhaustion
  - Static & Dynamic analysis do not measure the Application Behavior
  - Badly Designed or Malicious code can deplete the battery quickly

There is a need for Power Metering and Behavior Analysis

Application Testing Framework

Application Static Analysis does not cover Program Functionality

Fortify, Coverity, and other application testing tools cover regular, non-Android **specific Bugs:**

- No Security Analysis of the Code Functionality
- No Power Analysis of the Application components and code
- No **Profiling** of the resource consumption of individual applications
- **Cannot Regulate/Deny** the access and use of phone subsystems (Camera, Microphone, GPS..)

Application Testing Framework

Android Specific Analysis includes analysis of the Application Security Manifest (not supported by third-party vendors)

- **Tailored to the Android Permission Model**
- Verify if the requested permissions **are warranted** by the submitted code
- Curtails excessive permissions and enforces a tighter security model

Modifications on the Android Engine to enable dynamic policies

- Control the underlying Dalvik engine to report **absence/depletion of resources** instead of lack of permissions
- Regulate access to critical/restricted resources

Application Testing Evaluation

Analyzed ~130,000 Applications from the Google Android Market / Asian Markets

- Thousands with incorrect/permissive manifest
- Hundreds with excessive functionality that can be constituted as malicious
- Many Applications with:
    - Access to Camera/GPS/Microphone
    - Access to Sdcard Data/Contacts
    - Network/Reach-back Functionality (Updates/???)
    - Persistent Presence (Service)
    - Deviation in Functionality from what was included in the Application Description

Challenges for Power Metering

- A process can evade energy metering
  - Outsource the "expensive operations" to the Kernel
    - Network operations
    - Storage operations
  - Use Devices that themselves cause power drain
    - Wi-Fi, GPS, Bluetooth
    - **Display**
  - Spawn other sub-processes

- Changing Energy Consumption
  - **Over Time**
  - **Per User**
  - **Based on Location**

Power Metering Framework

- Design & Implement an accurate model for accounting and policing energy consumption

- Two-pronged approach
  - Meter the per-process CPU & Device utilization over time
  - Identify the relative impact of each device component on energy consumption

- Design an Android kernel subsystem to estimate energy
  - Meter energy consumption for each App/process
  - Use for characterizing application behavior
  - This behavior is Application dependent
  - Sometimes the behavior is also User dependent

Conclusions

Assuring the Secure Operation of Smart Devices has a wide-range of requirements

- Application Testing
  - Static & Dynamic
  - In-Field Instrumentation
  - Power Behavior Metering & Policing
- Physical Device Security
  - Lock-Down of the Device I/O (USB, WiFi, etc.)
  - Encryption of Data both on the Phone & Network
  - Securing Provisioning Process