Scaling Application of Security Standards by Customizing a Code Analysis Tool

Software Confidence. Achieved.

John Steven
Senior Director
Office of the CTO
Cigital Inc.

Introduction

Code Analysis Market

cigital

Software Confidence. Achieved.

Static Analysis

- Definition: Any analysis of software without actually executing the code
- The term includes simple text searches
- Even advanced tools with partial modeling fall into this category

The Tools' proposition

- People have 'rules' in mind during code reviews
- A tool's proposition is:
  - 'Rules' represent a 'security expert' in a box
  - Scales code review to mammoth code bases without sacrificing consistency

cigital

State of Practice

Tools on the market for 4-6 years now

- Early adopters have all bought
- Penetration has been difficult
- Consolidation beginning to occur

Tools vary ***dramatically***:

- Results presentation and Integration
- Underlying technologies
  - Macroscopic: Parsing, modeling, ‘runtime’
  - Microscopic: how they scan for buffer overflow

Tool Knowledge Gaps

- ***Historically, tools have been sold as ‘install and run’***
- Tool vendors aren’t consultancies
- Consultants limited:
  - Ranks don’t possess deep technical expertise
  - Don’t have experience across breadth of tools

cigital

What Goals & Challenges does Customization Address?

Initial Goals

- Introduce *lightweight* code analysis to SDLC
- *Inexpensively* purchase security expertise
- *Consistently* apply expertise

Subsequent Desires

- *Scale* ‘whitebox’ code analysis
- ***Automate* checking against corporate security coding standards**
- ***Enable* developers to test powerfully**

Non-starters

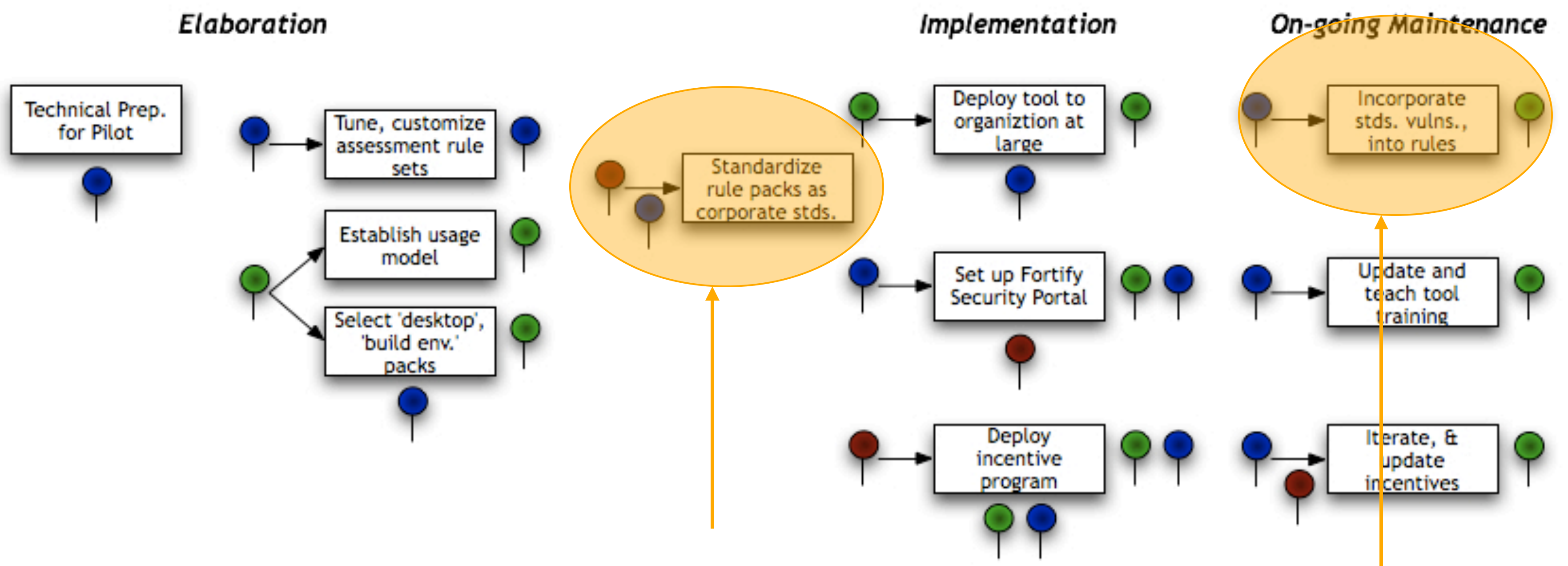
- Unwieldy build *integration*
- *Overwhelming False positives*
- Inappropriate *division of labor*: filtering findings, writing rules

Stumbling Blocks

- Unclear process/tool *ownership*, inability to *Shepherd* the tool
- **Overcoming *objections* to accuracy, alternatives**

cigital

Where Customization Fits in the Program…

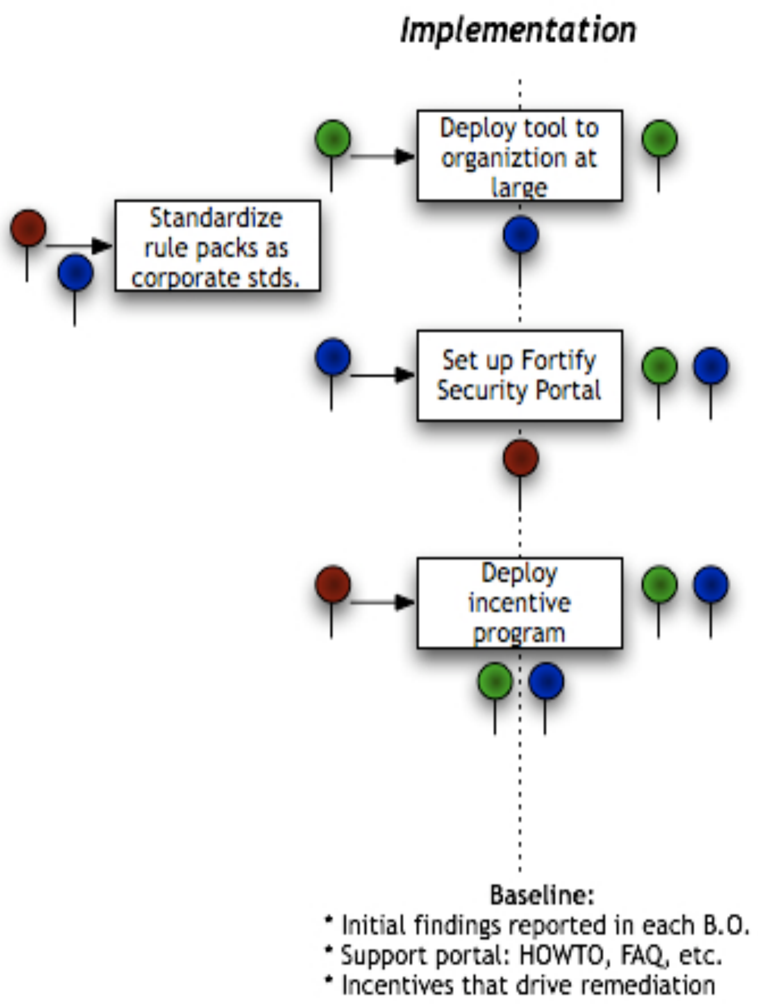


Elaboration
Implementation
On-going Maintenance
Technical Prep. for Pilot
Tune, customize assessment rule sets
Establish usage model
Select 'desktop', 'build env.' packs
Standardize rule packs as corporate stds.
Deploy tool to organiztion at large
Set up Fortify Security Portal
Deploy incentive program
Incorporate stds. vulns., into rules
Update and teach tool training
Iterate, & update incentives

Key to avoiding pushback

Key to getting value
Beyond core functionality

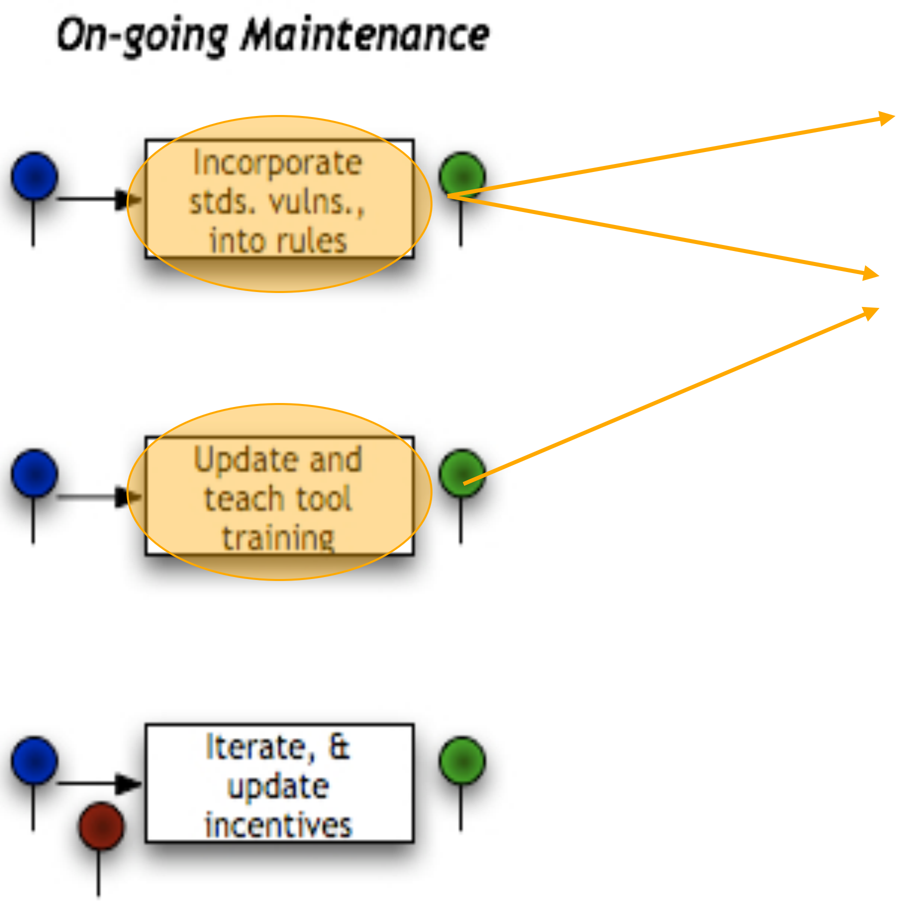
Implementation



Implementation
Deploy tool to organiztion at large
Standardize rule packs as corporate stds.
Set up Fortify Security Portal
Deploy incentive program
Baseline:
• Initial findings reported in each B.O.
• Support portal: HOWTO, FAQ, etc.
• Incentives that drive remediation

- *Baseline* **all applications**
  - Face integration issues all over again
  - Agreement rule pack essential to measurement
- **Deploy Incentives Program**
  - Measurement essential to incentives
  - *Enforce* adoption as a quality *gate*

On-going Maintenance


On-going Maintenance
Incorporate stds. vulns., into rules
Update and teach tool training
Iterate, & update incentives

Goals:

- *Scale* 'whitebox' code analysis
- *Automate* checking against corporate security coding standards
- *Enable* developers to test powerfully

Adoption

Who

How

Cost

Adoption Process

Software Confidence. Achieved.

Who Drives Adoption?

- Tools' licenses focus on developers, build envs.
- Adoption likely driven by App. Sec.

Worst case scenario:

- App Sec. 'owns' tool
- Tool thrown over the wall to dev.
- No communication bet. Development & App. Sec.

cigital

A Few Words on Cost…

- Tools cost $xx,xxx
  - $3-5k / “user”, with some exceptions

- Initial set up can take days to a week
- “ ‘Tuning’ takes 6-9mo. minimum” --jS

- Penalty paid for new:
  - Users: Analysts/Developers
  - Software projects/products

- Maintenance is real cost
  - Cost of a “finished rule” can be ~ $5k
    (~ $2,5k / week)

cigital

An Adoption Approach

- App Sec conducts initial pilot
  - On developer code bases
  - Uses developers as necessary to support
  - Remove any rules not applicable to [the Organization/environment]

- It's *very* unlikely that pilots should begin with development resources

Who runs the tool (eventually)?

- Central Security Team (App Sec.)
  - REQ: Current, deep development skill
  - Value: Risk management experience
  - Value: Broad, org-wide impact, fix
  - Risk: inflated impact, impractical fix

- Development
  - REQ: Understand sec. implications of results
  - Value: Practical fixes, quick turn-around
  - Risk: De-prioritization, Results suppression

cigital

Adoption's Challenges

Choosing

Increasing Visibility

Just Fixing It

cigital

Software Confidence. Achieved.

Choosing: Seek Experience

- Your local OWASP chapter
- Organizations within your vertical
- Similarly sized/structured organizations within your geography

- Get the war stories, but not the despair

cigital

Choosing: Eschew Deep Science

- BMW M3 or Audi SR4?

- Use representative sample of **your** apps
- Don't use a contrived test suite.
- Consider findings vs. pen-testing on same app.
  - Did new and interesting findings result?
  - Did static tool provide adequate root-cause analysis advice to fix problems earlier?
- How long did it take to on-board an app?
  - How will this scale to your portfolio of apps?
- How long did it take to triage the results?
  - How will this scale?

- Pick 3-10 Apps per 30/300

Choosing: Worry about what you *can* control

- organization’s staff size,
- skill set,
- scanning policy,
- and infrastructure

You do **not** control

- architecture,
- implementation,
- or bugs associated with the static analysis tool

cigital

Visibility

Your tool can’t find what it can’t see and it can’t see what it doesn’t parse.

- That *framework* stuff I’ve been talking about for two days?

Yeah, It don’t do that out of the box

- Demo

cigital

Visibility: Making Progress (Identifying)

- On-board apps
  - Using interface gives most feedback
- Explore scan logs for identified entry points
- Manually explore app’s:
  - Deployment descriptors
  - Critical configuration files
- Document controller logic as:
  - Framework default
  - Developer extended
- Identify key entities w/in DAO/persistence framework

cigital

Visibility: Making Progress (Codifying)

- Entry: Taking input from untrusted web sources
- Entry: Taking input from untrusted partner applications
- Exit: Placing data in a untrusted view (browser, service repsonse, etc.)
- Exit: Conducting CRUD operations on entity data

- Consider data entry/exit from 2nd and 3rd party components

cigital

Just Fix It?

- Detect consistent/thorough use of secure APIs
  - (and non-use of dangerous ones)
- Detect incorrect usage of such APIs
  - Broken call-order,
  - Un-paired functions,
  - Other bugs
- Running static tools on these security toolkits finds problems that careful review may not

Customization

Customization Process

Examples

Tool Results

cigital

Software Confidence. Achieved.

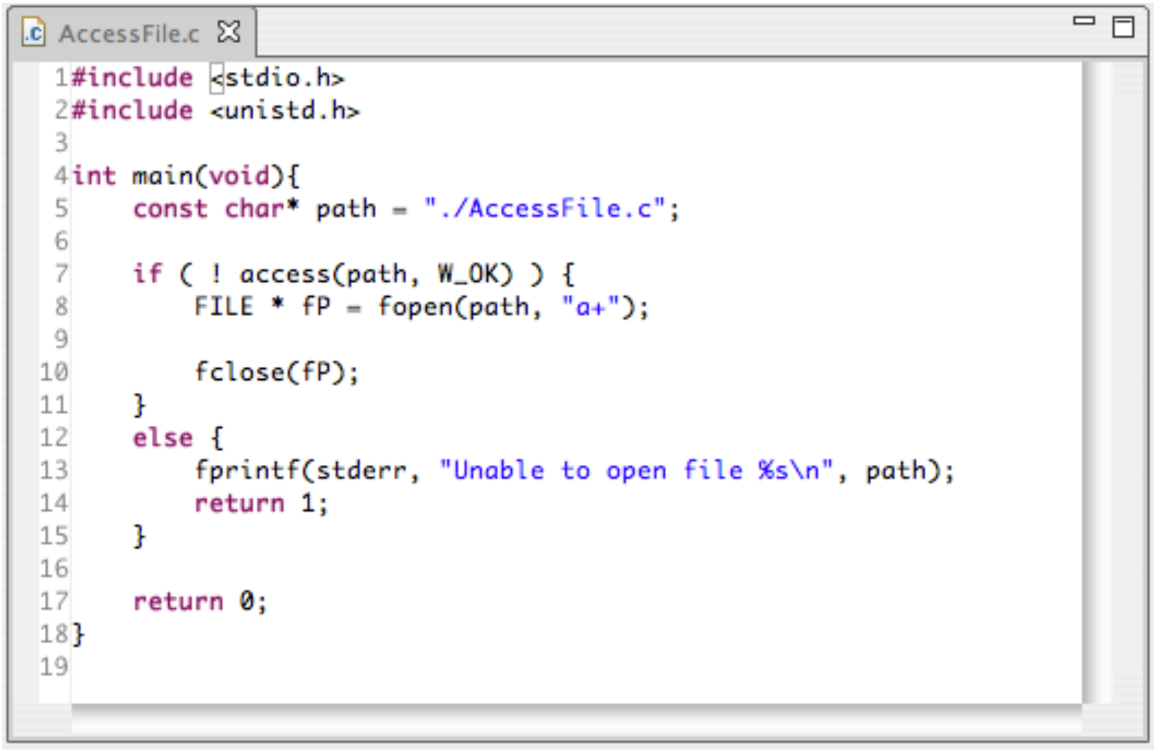
A Process for Customization

1. Begin with results set
2. Visually prune for results with security implications
3. Dig into each: classify
   1. False positive - Determine how to:
      1. Turn rule off if worthless
      2. Tweak rule/output if otherwise valuable
      3. Tune code to avoid firing rule
   2. Result worthy of remediating:
      1. Refactor code until rule doesn't fire 4)...5)...6)...

10. Converge, Roll-up results

cigital

Example: File Access

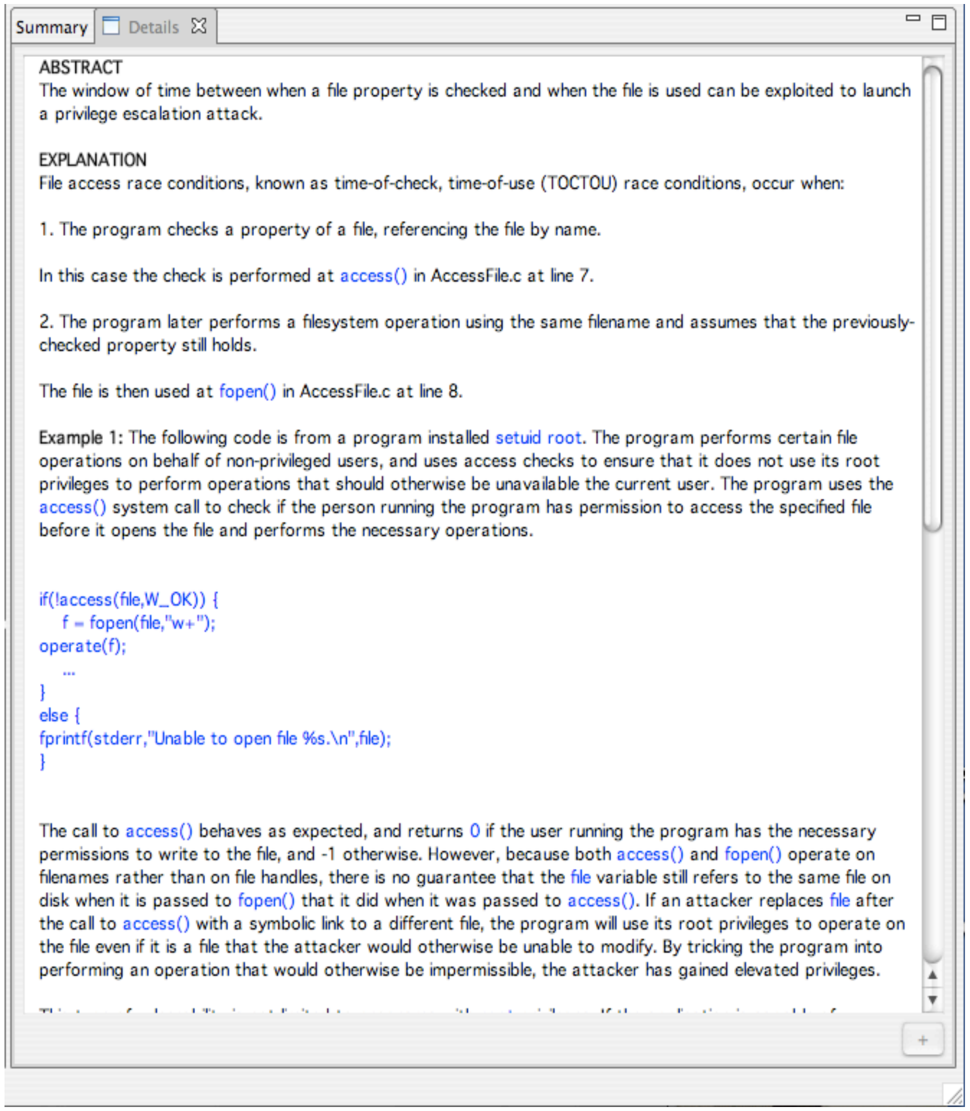

AccessFile.c

```
#include <stdio.h>
#include <unistd.h>

int main(void){
    const char* path = "./AccessFile.c";

    if ( ! access(path, W_OK) ) {
        FILE * fP = fopen(path, "a+");

        fclose(fP);
    }
    else {
        fprintf(stderr, "Unable to open file %s\n", path);
        return 1;
    }

    return 0;
}

```

- What does the tool’s explanation say about the code?
- What are the tool’s recommendations?

cigital

Example: File Access Rule Result



ABSTRACT
The window of time between when a file property is checked and when the file is used can be exploited to launch a privilege escalation attack.

EXPLANATION
File access race conditions, known as time-of-check, time-of-use (TOCTOU) race conditions, occur when:

1. The program checks a property of a file, referencing the file by name.

In this case the check is performed at access() in AccessFile.c at line 7.

2. The program later performs a filesystem operation using the same filename and assumes that the previously-checked property still holds.

The file is then used at fopen() in AccessFile.c at line 8.

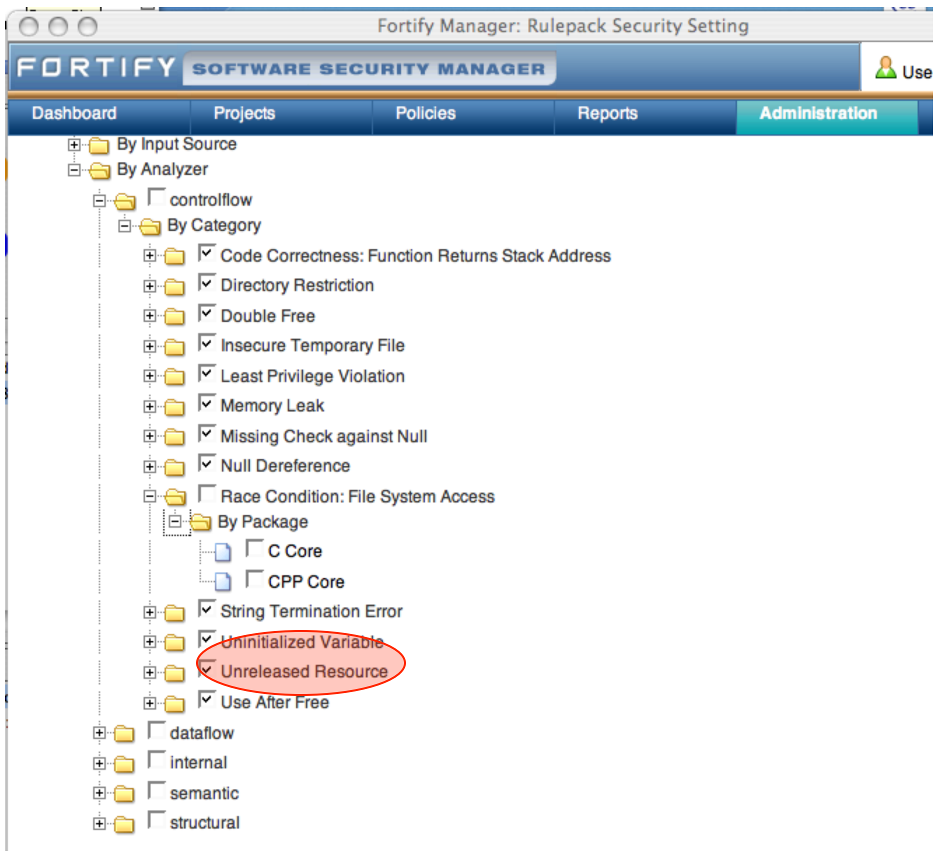
Example 1: The following code is from a program installed setuid root. The program performs certain file operations on behalf of non-privileged users, and uses access checks to ensure that it does not use its root privileges to perform operations that should otherwise be unavailable the current user. The program uses the access() system call to check if the person running the program has permission to access the specified file before it opens the file and performs the necessary operations.

```
if(!access(file,W_OK)) {
    f = fopen(file,"w+");
operate(f);
    ...
}
else {
fprintf(stderr,"Unable to open file %s.\n",file);
}
```

The call to access() behaves as expected, and returns 0 if the user running the program has the necessary permissions to write to the file, and -1 otherwise. However, because both access() and fopen() operate on filenames rather than on file handles, there is no guarantee that the file variable still refers to the same file on disk when it is passed to fopen() that it did when it was passed to access(). If an attacker replaces file after the call to access() with a symbolic link to a different file, the program will use its root privileges to operate on the file even if it is a file that the attacker would otherwise be unable to modify. By tricking the program into performing an operation that would otherwise be impermissible, the attacker has gained elevated privileges.

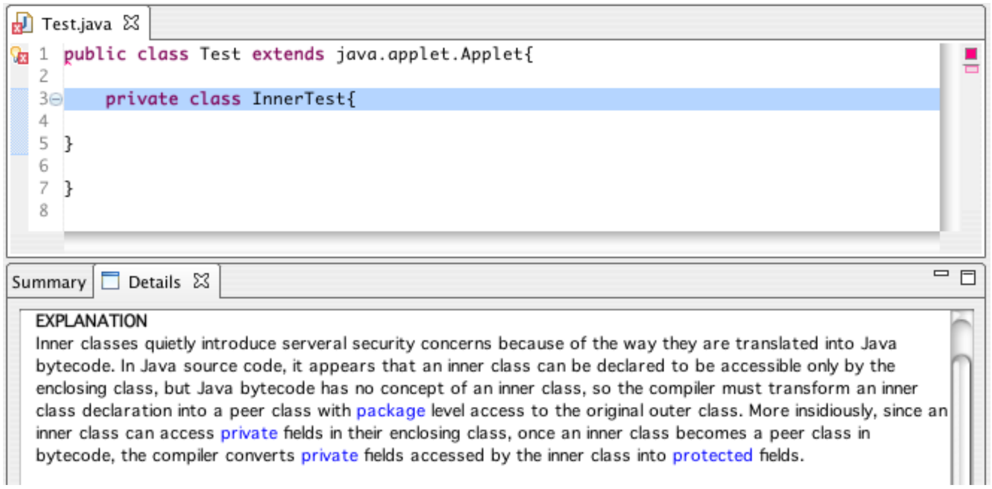
Example #1: Resolution: Turn the rule off

- Turn the rule off in SSM
- Rule will not fire again


Fortify Manager: Rulepack Security Setting

- By Input Source
- By Analyzer
  - [ ] controlflow
    - By Category
      - [✓] Code Correctness: Function Returns Stack Address
      - [✓] Directory Restriction
      - [✓] Double Free
      - [✓] Insecure Temporary File
      - [✓] Least Privilege Violation
      - [✓] Memory Leak
      - [✓] Missing Check against Null
      - [✓] Null Dereference
      - [ ] Race Condition: File System Access
        - By Package
          - [ ] C Core
          - [ ] CPP Core
      - [✓] String Termination Error
      - [✓] Uninitialized Variable
      - [✓] Unreleased Resource
      - [✓] Use After Free
  - [ ] dataflow
  - [ ] internal
  - [ ] semantic
  - [ ] structural

Example #2: Inner Classes

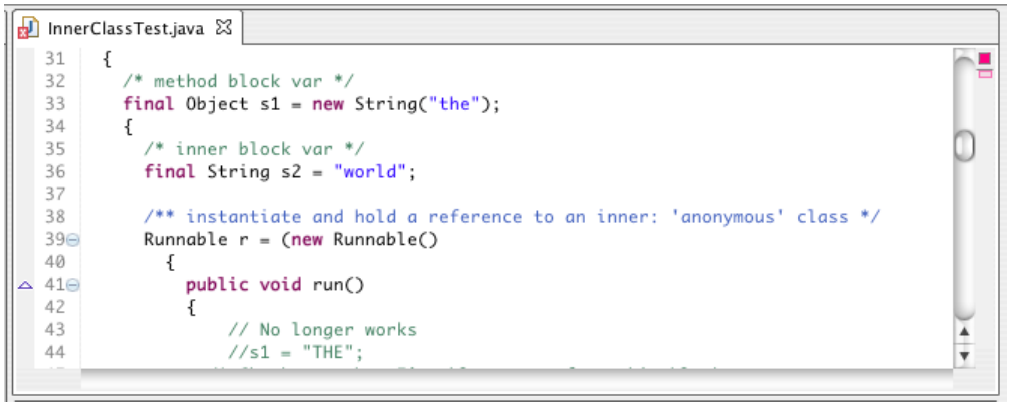


Test.java

```java
public class Test extends java.applet.Applet{

    private class InnerTest{

}

}

```

Summary | Details

EXPLANATION
Inner classes quietly introduce serveral security concerns because of the way they are translated into Java bytecode. In Java source code, it appears that an inner class can be declared to be accessible only by the enclosing class, but Java bytecode has no concept of an inner class, so the compiler must transform an inner class declaration into a peer class with package level access to the original outer class. More insidiously, since an inner class can access private fields in their enclosing class, once an inner class becomes a peer class in bytecode, the compiler converts private fields accessed by the inner class into protected fields.



InnerClassTest.java

```java
  {
    /* method block var */
    final Object s1 = new String("the");
    {
      /* inner block var */
      final String s2 = "world";

      /** instantiate and hold a reference to an inner: 'anonymous' class */
      Runnable r = (new Runnable()
        {
          public void run()
          {
              // No longer works
              //s1 = "THE";
```

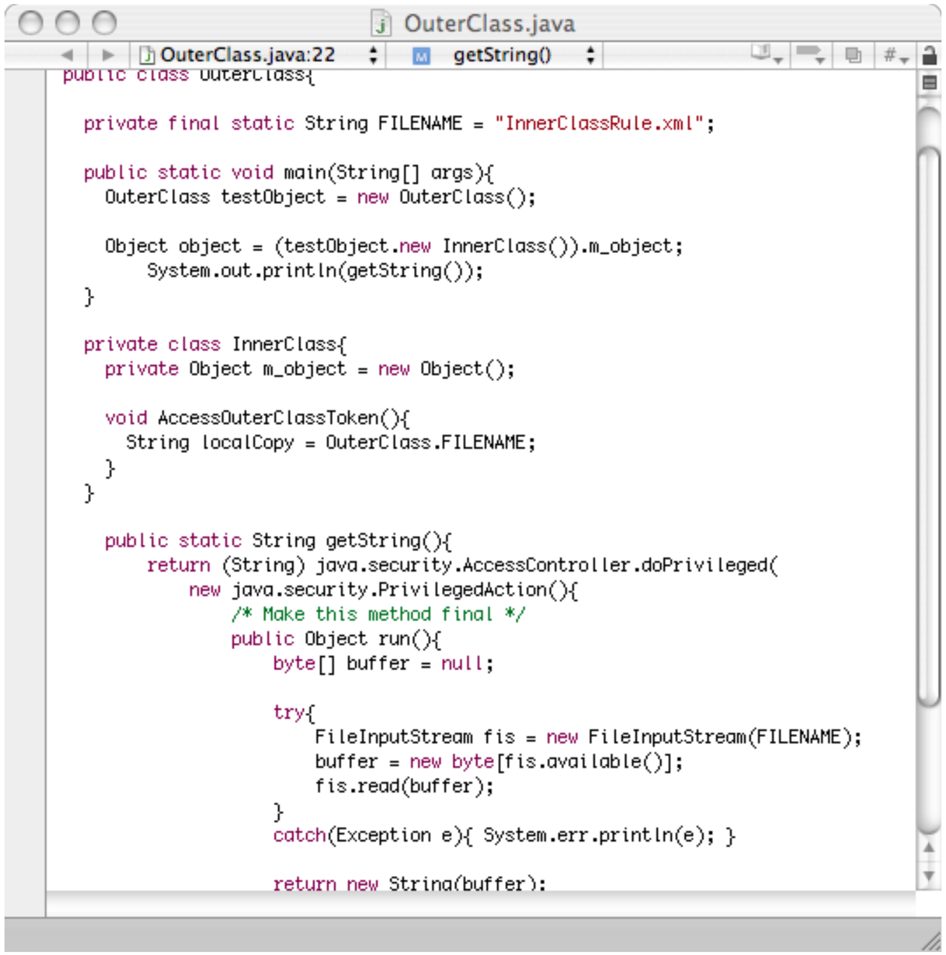
- What does the tool's explanation say about the code?
- What are the tool's recommendations?

- "Inner classes are dangerous"

cigital

Example #2: Remediation: Tune the Rule

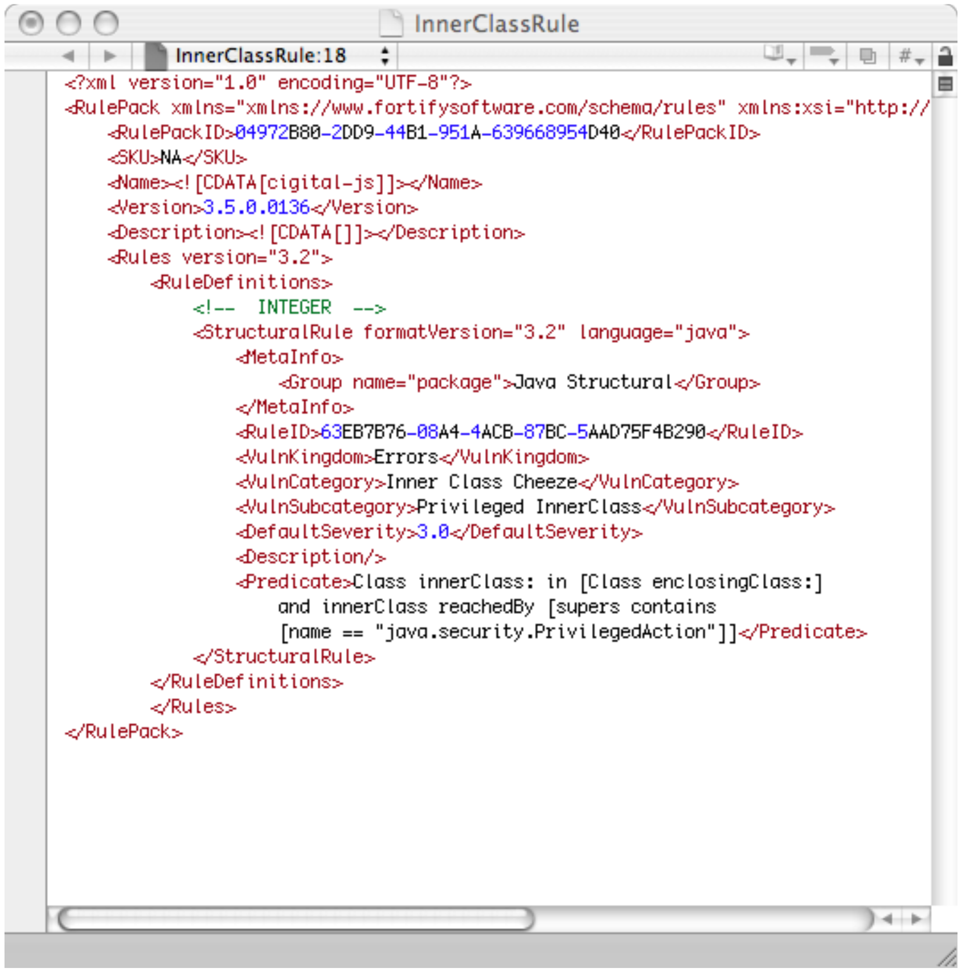
- Turn rule off (avoid FPs)
- “Fine tune” rule:
  - Model threat
  - Illuminate attack vectors
  - Brainstorm source code constructs
  - Mature into axioms
  - Test
    - Validate results
    - Loop back to 2, 3



```java
public class OuterClass{

  private final static String FILENAME = "InnerClassRule.xml";

  public static void main(String[] args){
    OuterClass testObject = new OuterClass();

    Object object = (testObject.new InnerClass()).m_object;
        System.out.println(getString());
  }

  private class InnerClass{
    private Object m_object = new Object();

    void AccessOuterClassToken(){
      String localCopy = OuterClass.FILENAME;
    }
  }

    public static String getString(){
        return (String) java.security.AccessController.doPrivileged(
            new java.security.PrivilegedAction(){
                /* Make this method final */
                public Object run(){
                    byte[] buffer = null;

                    try{
                        FileInputStream fis = new FileInputStream(FILENAME);
                        buffer = new byte[fis.available()];
                        fis.read(buffer);
                    }
                    catch(Exception e){ System.err.println(e); }

                    return new String(buffer);
```

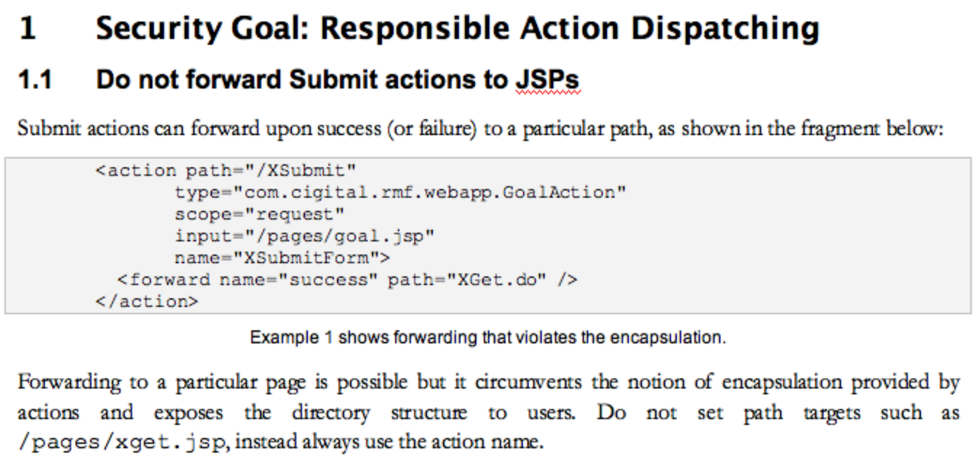
Example #2: Rule, First cut

- Axioms:
  - Inner class definition
  - Implements `PrivilegedAction`
- What might you do next?



```
<?xml version="1.0" encoding="UTF-8"?>
<RulePack xmlns="xmlns://www.fortifysoftware.com/schema/rules" xmlns:xsi="http://
    <RulePackID>04972B80-2DD9-44B1-951A-639668954D40</RulePackID>
    <SKU>NA</SKU>
    <Name><![CDATA[cigital-js]]></Name>
    <Version>3.5.0.0136</Version>
    <Description><![CDATA[]]></Description>
    <Rules version="3.2">
        <RuleDefinitions>
            <!--  INTEGER  -->
            <StructuralRule formatVersion="3.2" language="java">
                <MetaInfo>
                    <Group name="package">Java Structural</Group>
                </MetaInfo>
                <RuleID>63EB7B76-08A4-4ACB-87BC-5AAD75F4B290</RuleID>
                <VulnKingdom>Errors</VulnKingdom>
                <VulnCategory>Inner Class Cheeze</VulnCategory>
                <VulnSubcategory>Privileged InnerClass</VulnSubcategory>
                <DefaultSeverity>3.0</DefaultSeverity>
                <Description/>
                <Predicate>Class innerClass: in [Class enclosingClass:]
                    and innerClass reachedBy [supers contains
                    [name == "java.security.PrivilegedAction"]]</Predicate>
            </StructuralRule>
        </RuleDefinitions>
        </Rules>
</RulePack>
```

Example #3: Enforcing Conventions



1 Security Goal: Responsible Action Dispatching

1.1 Do not forward Submit actions to JSPs

Submit actions can forward upon success (or failure) to a particular path, as shown in the fragment below:

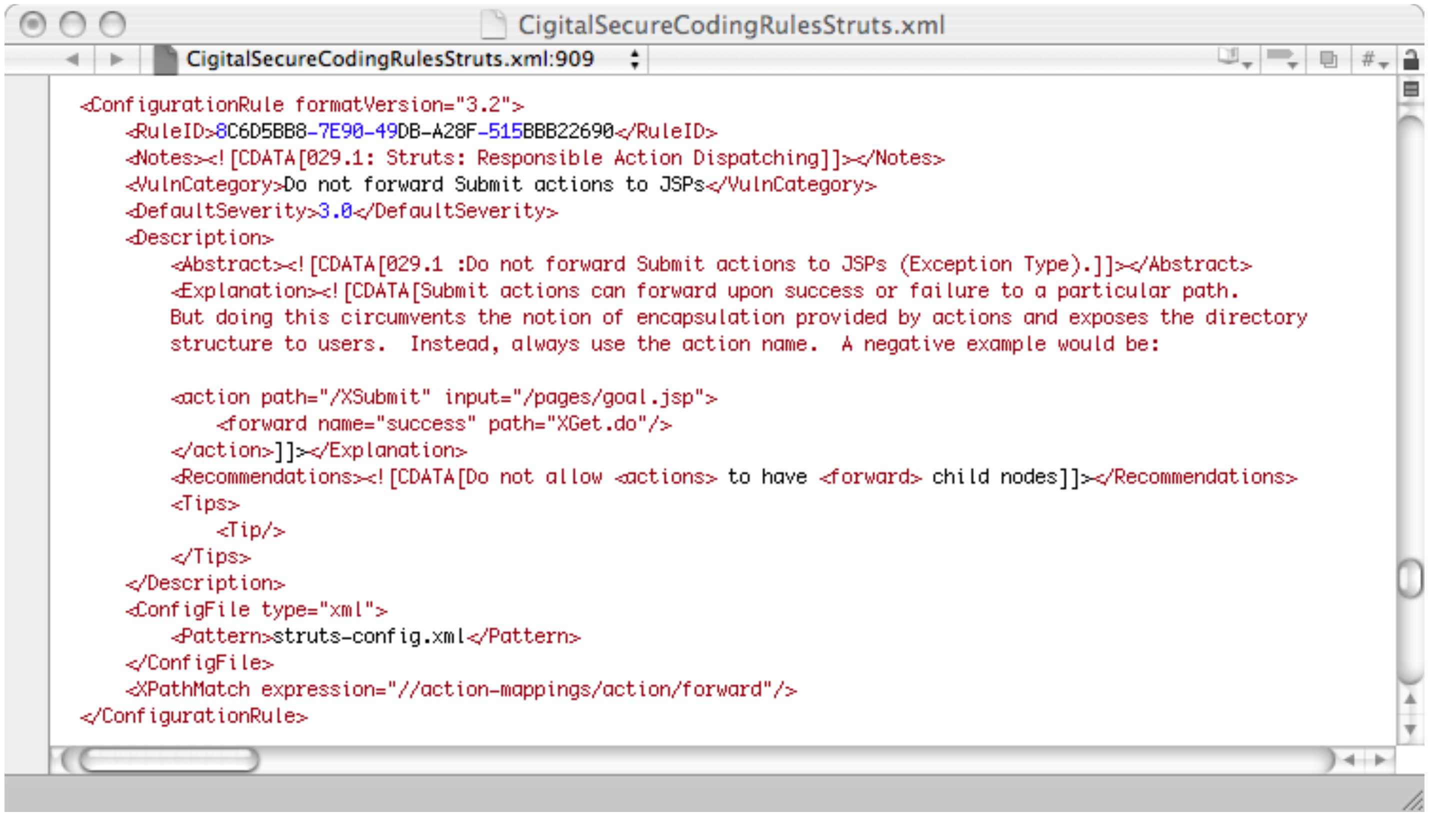
```
<action path="/XSubmit"
        type="com.cigital.rmf.webapp.GoalAction"
        scope="request"
        input="/pages/goal.jsp"
        name="XSubmitForm">
  <forward name="success" path="XGet.do" />
</action>
```

Example 1 shows forwarding that violates the encapsulation.

Forwarding to a particular page is possible but it circumvents the notion of encapsulation provided by actions and exposes the directory structure to users. Do not set path targets such as `/pages/xget.jsp`, instead always use the action name.

- Coding conventions (quality, some security) are hard to enforce
  - Manual checking untenable
- What ‘signature’ does this have in the code, deployment?

Example #3: Rules Enforcing Conventions



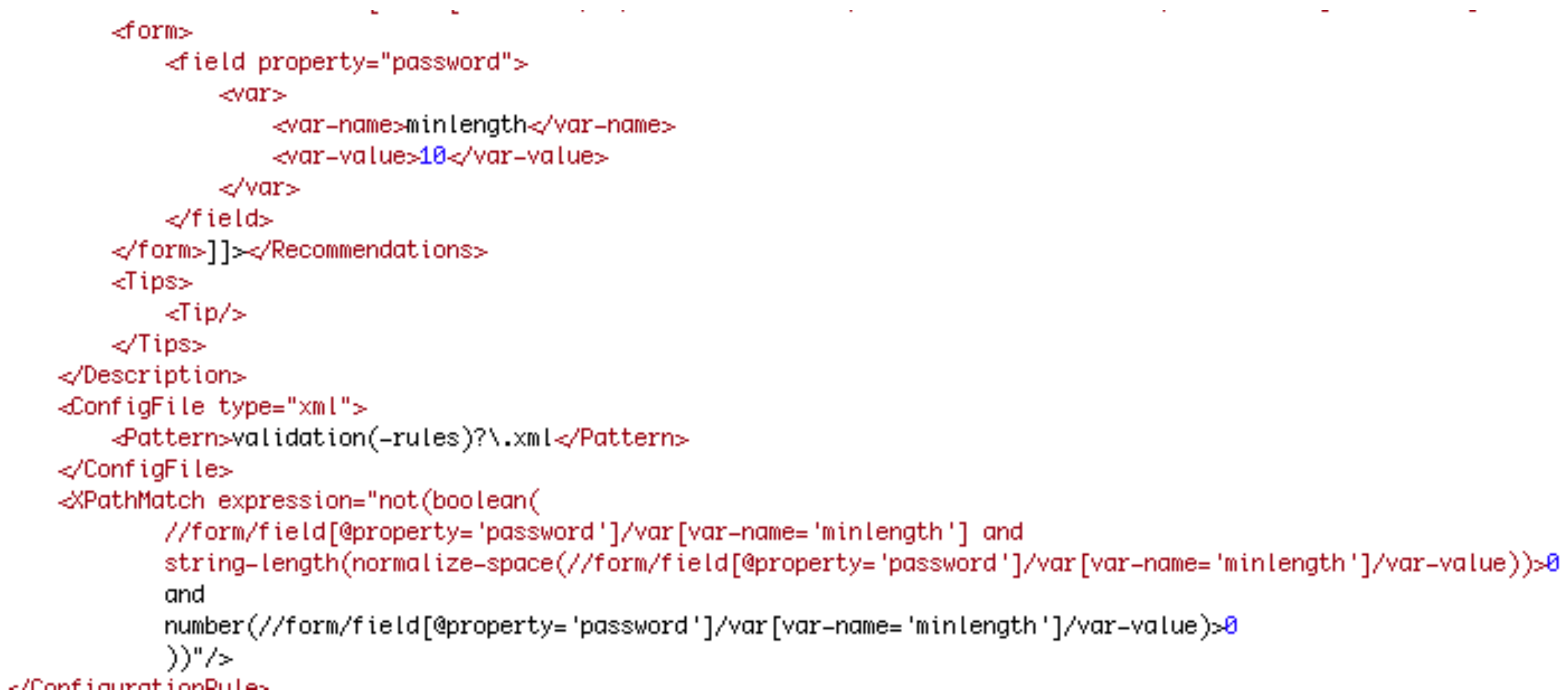
CigitalSecureCodingRulesStruts.xml

CigitalSecureCodingRulesStruts.xml:909

```xml
<ConfigurationRule formatVersion="3.2">
    <RuleID>8C6D5BB8-7E90-49DB-A28F-515BBB22690</RuleID>
    <Notes><![CDATA[029.1: Struts: Responsible Action Dispatching]]></Notes>
    <VulnCategory>Do not forward Submit actions to JSPs</VulnCategory>
    <DefaultSeverity>3.0</DefaultSeverity>
    <Description>
        <Abstract><![CDATA[029.1 :Do not forward Submit actions to JSPs (Exception Type).]]></Abstract>
        <Explanation><![CDATA[Submit actions can forward upon success or failure to a particular path.
        But doing this circumvents the notion of encapsulation provided by actions and exposes the directory
        structure to users.  Instead, always use the action name.  A negative example would be:

        <action path="/XSubmit" input="/pages/goal.jsp">
            <forward name="success" path="XGet.do"/>
        </action>]]></Explanation>
        <Recommendations><![CDATA[Do not allow <actions> to have <forward> child nodes]]></Recommendations>
        <Tips>
            <Tip/>
        </Tips>
    </Description>
    <ConfigFile type="xml">
        <Pattern>struts-config.xml</Pattern>
    </ConfigFile>
    <XPathMatch expression="//action-mappings/action/forward"/>
</ConfigurationRule>
```

- What 'signature' does this rule detect?

Example #3: Rules Enforcing Convention (2)



```
        <form>
            <field property="password">
                <var>
                    <var-name>minlength</var-name>
                    <var-value>10</var-value>
                </var>
            </field>
        </form>]]></Recommendations>
        <Tips>
            <Tip/>
        </Tips>
    </Description>
    <ConfigFile type="xml">
        <Pattern>validation(-rules)?\.xml</Pattern>
    </ConfigFile>
    <XPathMatch expression="not(boolean(
            //form/field[@property='password']/var[var-name='minlength'] and
            string-length(normalize-space(//form/field[@property='password']/var[var-name='minlength']/var-value))>0
            and
            number(//form/field[@property='password']/var[var-name='minlength']/var-value)>0
            ))"/>
```

- What ‘signature’ does this rule detect?

Example #4: Implementing Security Policy

- What security policies does your organization have?
  - Regulation-driven
    - Crypto
    - Logging
    - Auth/Authorization

cigital

Example #4: Heuristics for Potential Rules

- Bad Call
  - *Never* call `foo()`
  - *Never* call `gets()`

- Bad configuration:
  - Anything XPath can do…
  - Do not map multiple URLs onto one Servlet:
    - `XPathMatch expression="boolean(//servlet-mapping [servlet-name=following::servlet-name]`
    - Are there any `auth-constraints` referring to a non-existent `security-role?`

cigital

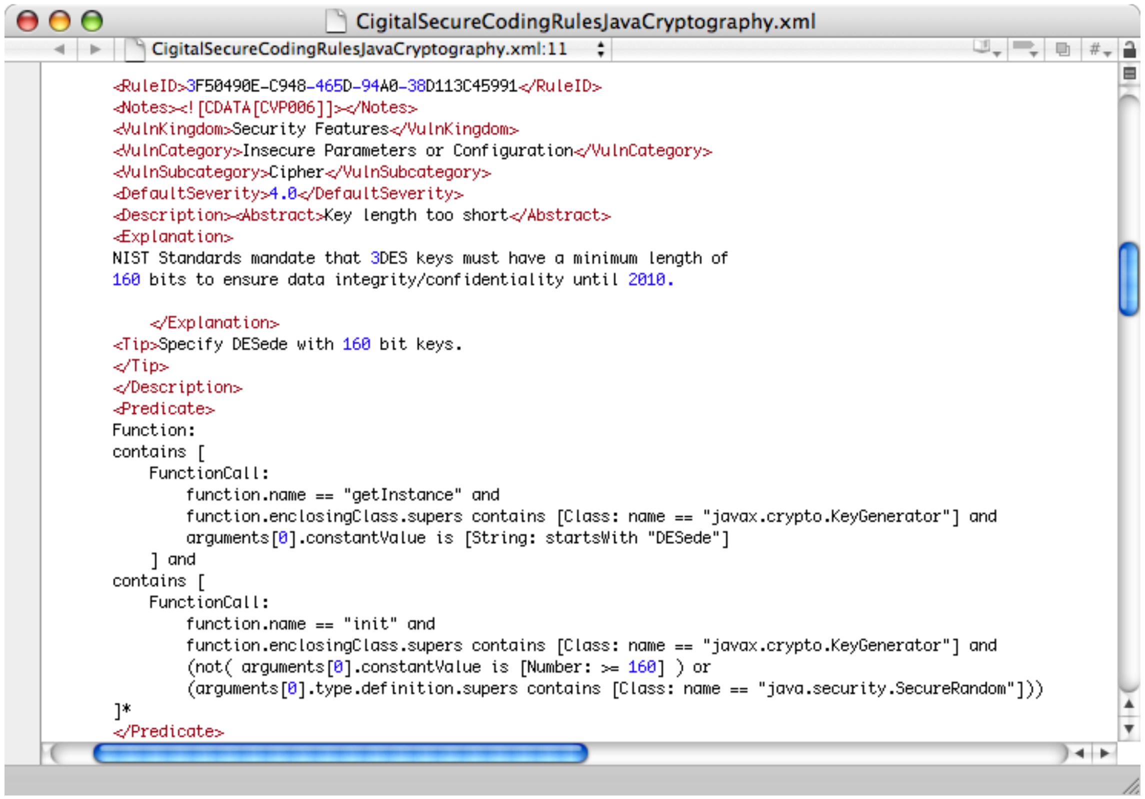
Example #4: Heuristics (II) for Potential Rules

- Call ordering, state
  - You must call `foo()` before `bar()`
  - Call `sanitize()` before `copy()`

- Data flow:
  - Data from <Foo> reaches <Bar>
  - Data tagged “ssn” gets to my logger

Example #4: (Finally) Conforming to Policy

- The *most* important rule
  - demands security standards compliance
  - Coded in a technology-specific way


CigitalSecureCodingRulesJavaCryptography.xml
CigitalSecureCodingRulesJavaCryptography.xml:11

```
<RuleID>3F50490E-C948-465D-94A0-38D113C45991</RuleID>
<Notes><![CDATA[CVP006]]></Notes>
<VulnKingdom>Security Features</VulnKingdom>
<VulnCategory>Insecure Parameters or Configuration</VulnCategory>
<VulnSubcategory>Cipher</VulnSubcategory>
<DefaultSeverity>4.0</DefaultSeverity>
<Description><Abstract>Key length too short</Abstract>
<Explanation>
NIST Standards mandate that 3DES keys must have a minimum length of
160 bits to ensure data integrity/confidentiality until 2010.

    </Explanation>
<Tip>Specify DESede with 160 bit keys.
</Tip>
</Description>
<Predicate>
Function:
contains [
    FunctionCall:
        function.name == "getInstance" and
        function.enclosingClass.supers contains [Class: name == "javax.crypto.KeyGenerator"] and
        arguments[0].constantValue is [String: startsWith "DESede"]
    ] and
contains [
    FunctionCall:
        function.name == "init" and
        function.enclosingClass.supers contains [Class: name == "javax.crypto.KeyGenerator"] and
        (not( arguments[0].constantValue is [Number: >= 160] ) or
        (arguments[0].type.definition.supers contains [Class: name == "java.security.SecureRandom"]))
]*
</Predicate>
```

cigital

Then what?

- Rule results are NOT the finish line
  - Continue to refine, iterate
    - Capture more false negatives
    - Reduce more false positives
- ALWAYS
  - Document your rules as standards
  - Test rules thoroughly with unit tests
  - BONUS: Develop positive/negative code examples

Scaling SCR

cigital

Software Confidence. Achieved.

State of Demand: SCR Volume

- Central
  - 13.5 MLoC
  - 200 Apps / yr.
  - 50 MLoC
  - 100 MLoC
- Self Service (per year)
  - 550 Apps (23MLoC)
  - 300 Apps (35 MLoC)
  - 350 Apps (14 MLoC)

- Aspirations
  - **100+ MLoC / day**
  - 1000s Apps / yr

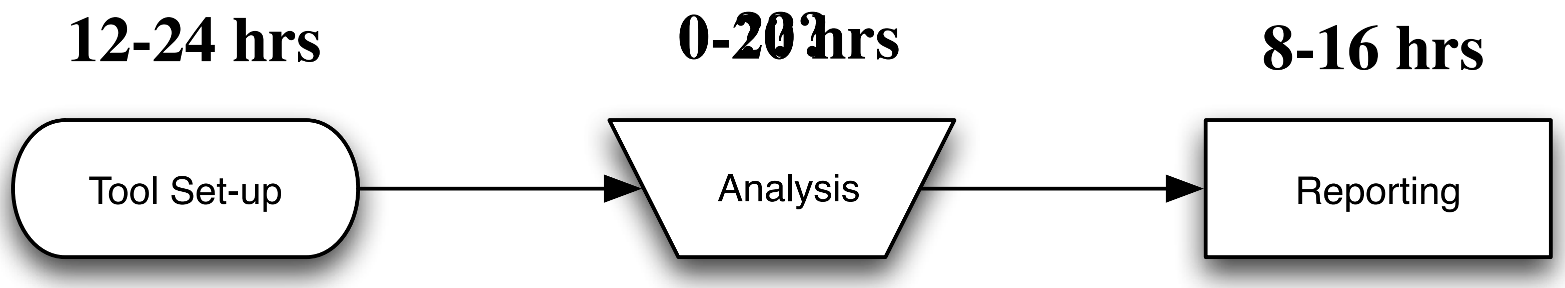
cigital

Pain Points / ESP Drivers

- Deployment Cost
- Configuration Management
- Developer Acceptance
- False Positive and False Negatives
- .NET Application Analysis
- Multi-tool Support
  - Hybrid Analysis
  - Findings Aggregation/Correlation
- Integration with Bug Tracking systems

cigital

State of the Practice – Code Assessments



- It takes a day and a half to get results
- It takes a day or two to report
- That leaves very little time for thinking, which is what we’re paid to do.

cigital

Solution – Deployment Costs

- Minimize build integration
- No developer training required / BlackBox approach
- Faster rule tuning
- Do not need to be a SCA tool expert to write custom rules

cigital

Solution - Configuration Management

- Store and maintain rule packs for each application
- Alert SSG if an application is dramatically changed
- Repeatable configuration

cigital

Workflow - Roadmap Components

- Document assessment policy
- Pilot implementation
  - Rules management
- Integrate assessment tools
  - Solution topology
- Measure, iterate
  - Reporting

cigital

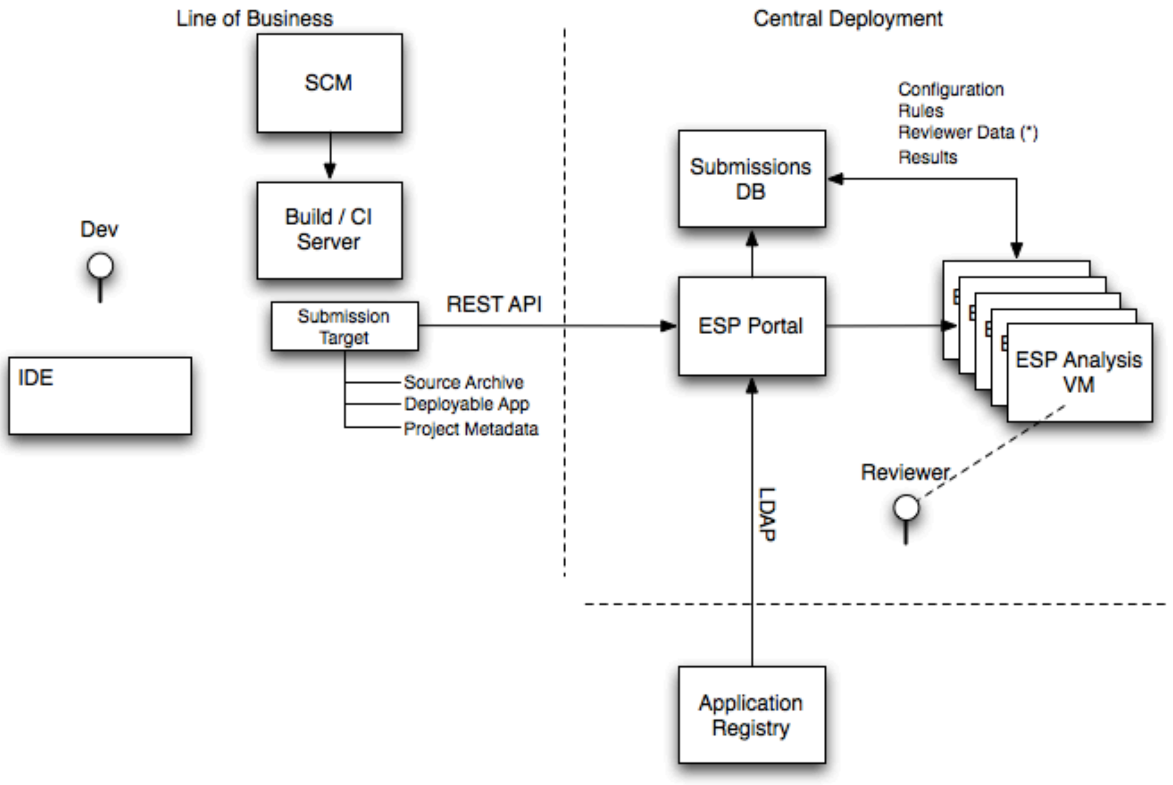
Solution Topology

cigital

Software Confidence. Achieved.

Integration Submission - Push

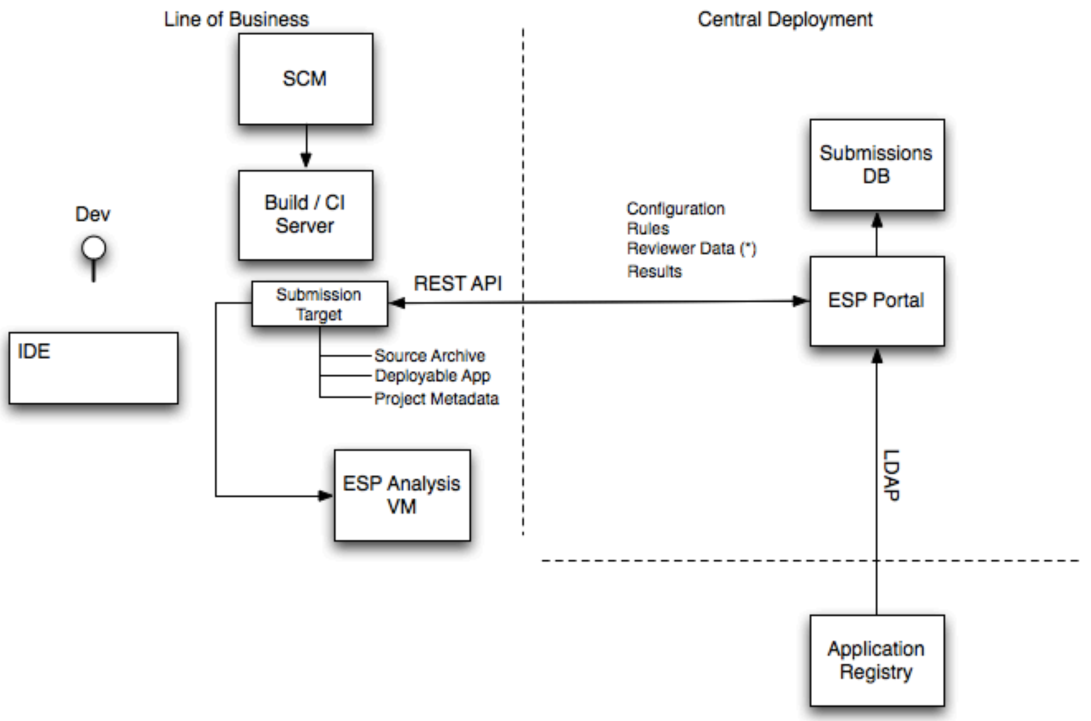
- Integrate with Lob
  - ESP CI shim in build/CI/QA environment
  - Target archives
    - source,
    - deployable binary
    - project meta
    - SCR meta
  - Submits using REST
- ESP Portal
  - Saves
    - Configuration
    - Rules
    - Reviewer data
    - Results



cigital

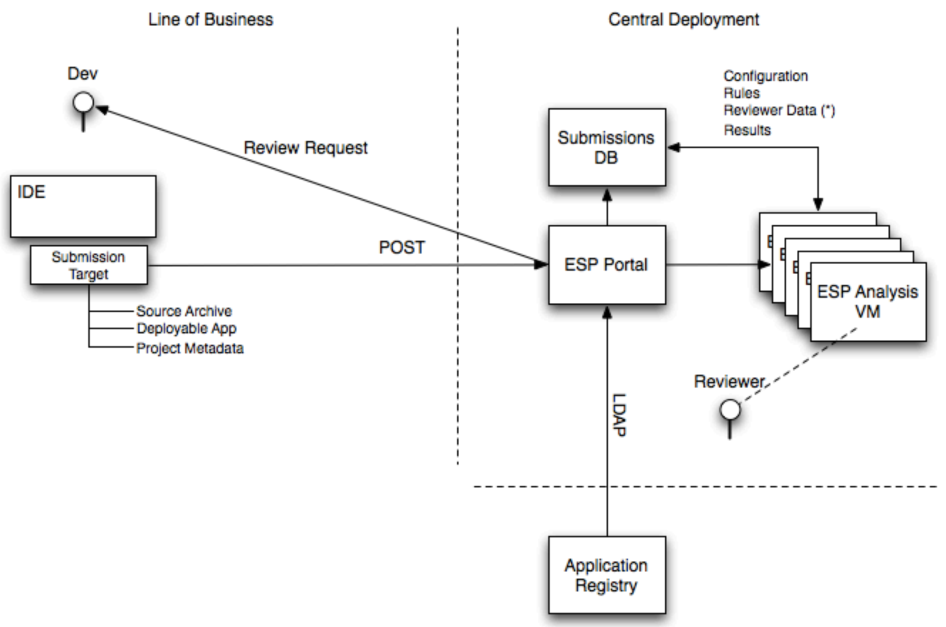
Integration Submission – Push - Distributed

- Integrate with Lob
  - ESP CI shim in build/CI/QA environment
  - Build target as usual
- ESP Portal
  - Saves
    - Configuration
    - Rules
    - Reviewer data
    - Results
  - Pushes config @ LoB
  - LoB runs ESP slave
  - Slave will likely remain separate from build server



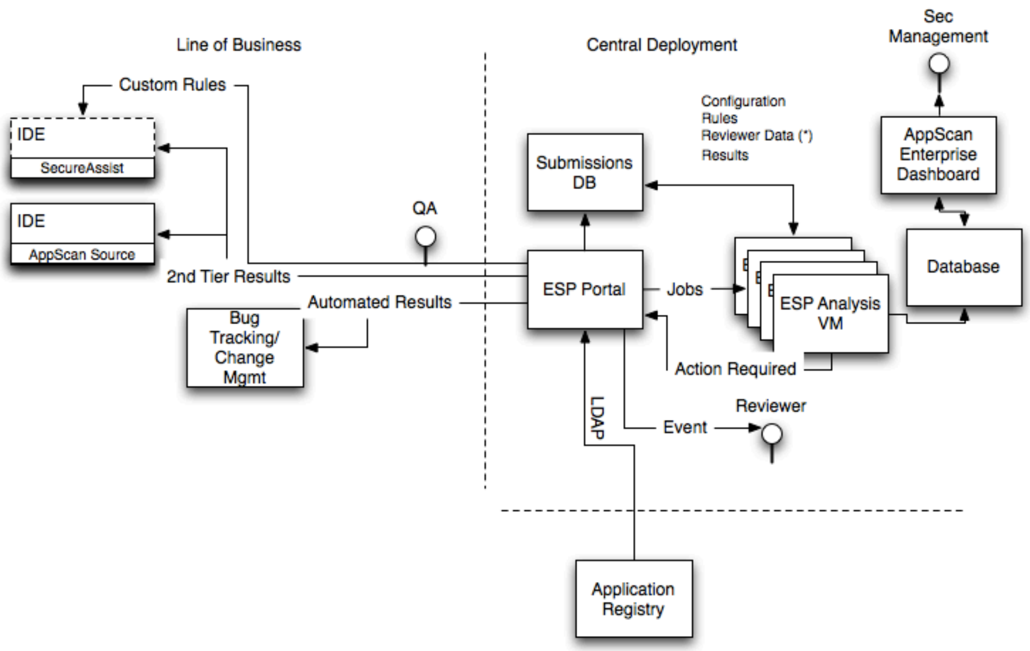
Integration Submission – Pull

- Reviewer Assigns App
  - Project / SCR IDs
  - Requests review
- Developer
  - Interacts with Submission Portal
- Analysis
  - Runs as in push model



Integration Results

- Reviewer
  - Notified of need to update SCR config
  - Escalated SCRs
- Developer
  - Receives automated results from bug tracking
  - Receives 2nd tier of results in plug-in
  - Later, will receive custom desktop-based rules based on results
- QA
  - Triages 2nd tier results, makes assignments


Line of Business
Central Deployment
Sec Management
Custom Rules
IDE
SecureAssist
IDE
AppScan Source
2nd Tier Results
QA
Automated Results
Bug Tracking/ Change Mgmt
Configuration
Rules
Reviewer Data (*)
Results
Submissions DB
ESP Portal
Jobs
ESP Analysis VM
AppScan Enterprise Dashboard
Database
Action Required
Reviewer
Event
LDAP
Application Registry