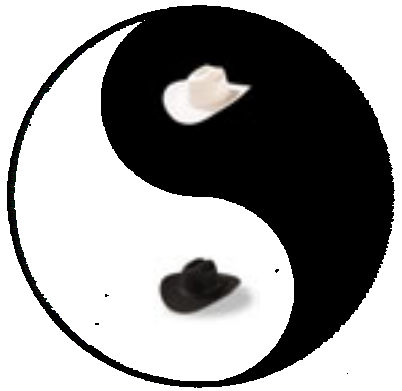
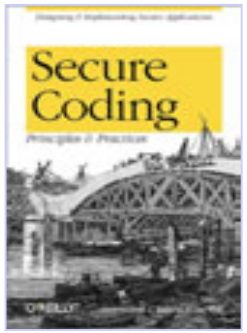
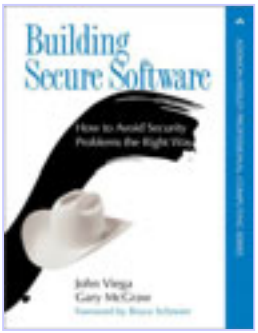
Next Steps in Bridging the Gap

Between Incident Handling and Software Development

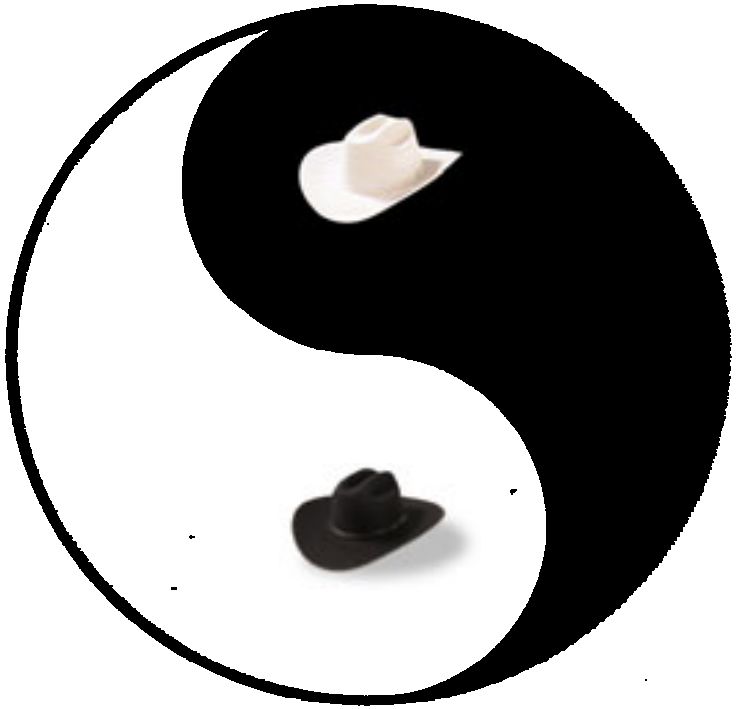






Outline

- The Problem
- Security touchpoints and collaboration opportunities



The Problem

Quiz: What's wrong with this code snippet?

```
int main(char **argv, int argc)
{
        char buf[10];
        strcpy(buf, argv[1]);
}
```

Dev answer: No input bounds checking

CSIRT answer: Buffer overflow that can lead to execution of arbitrary code

Both answers are correct, but quite different…

Let's explore those differences a bit

- Two valid perspectives
  - Dev's answer describes the code issue
  - CSIRT's describes the resulting attack issue
- Fundamentally different ways of viewing things
  - Build vs. break
- And it only gets worse from here

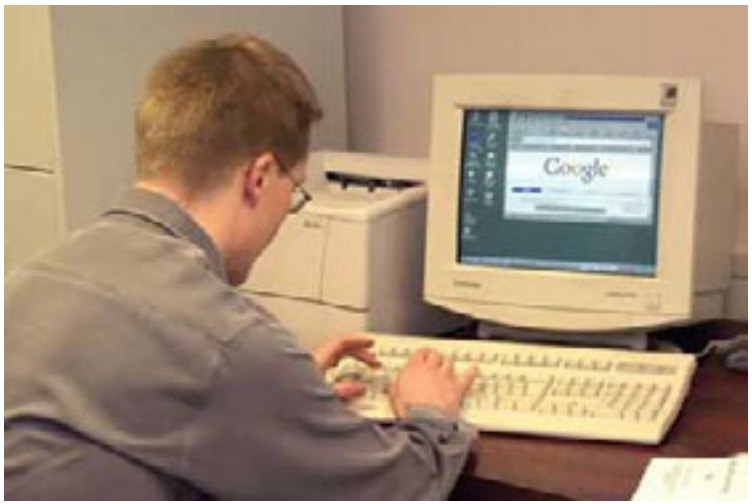
How dev sees the CSIRT

- Defend the “perimeter” with a firewall and IDS/IPS
  - “Only ports 80 & 443 are allowed through my firewall”
- Over reliance on crypto
  - “You MUST use SSL”
- “Review” products when they’re done
  - “We use the latest pen testing tools on all production apps”
- Disallow that which they don’t understand
  - “Extensible systems (Java and .NET) are dangerous”
- All they do is tell us “no, you may not do that”

The “security ops guy” does not really understand software development.

How the CSIRT sees dev

- Narrow minded focus on functional spec
  - "If the customer didn't ask for it, it's not our job"
- Doesn't study attack methods and tools
  - "My boss doesn't require me to"
- Can't protect apps from common attacks
  - "What's the big deal about cross-site scripting?"
- Won't stop making the same coding mistakes
  - "But I always use strcpy()"



Dev often doesn't appreciate how dangerous the net is

What's missing in the CSIRT perspective?

- Security must be built into the software to be effective
  - Plugging it in later is futile
- A perimeter security view of the world is antiquated and unrealistic
  - …and has been for some time

An entire room full of firewalls, IDSs, IPSs, fingerprint scanners, and surveillance cameras will not protect our information from ***bad software***

What's missing in the dev perspective?

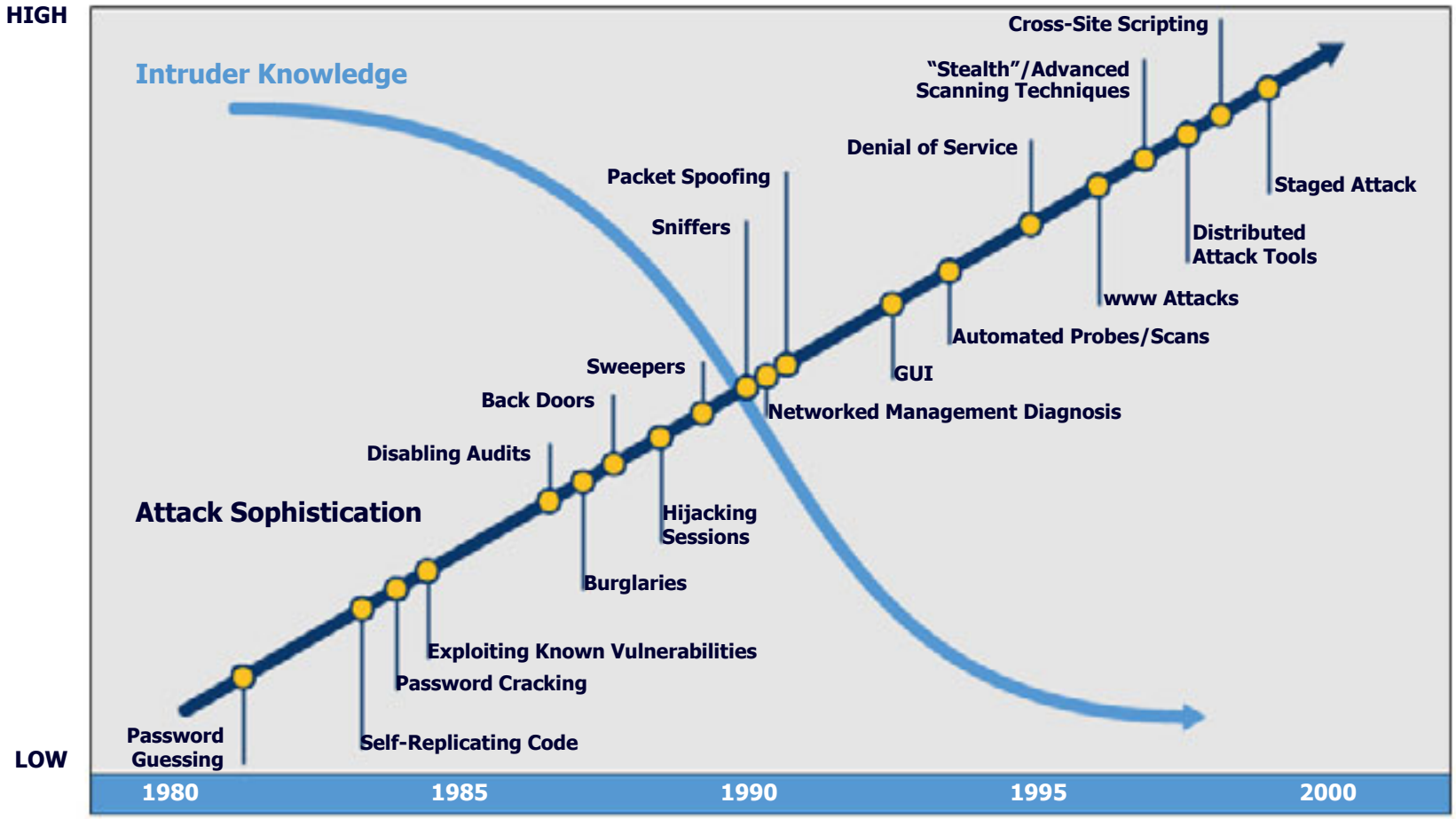
- Software developers tend to focus on functional spec
  - Very good at building things that perform to customer needs
  - Not often as good at developing code that resists attack
- Software developers often underestimate the threats
- Thinking about building things vs. thinking about breaking thing
  - What's the difference between a civil and a mechanical engineer?

NO SIGNAL

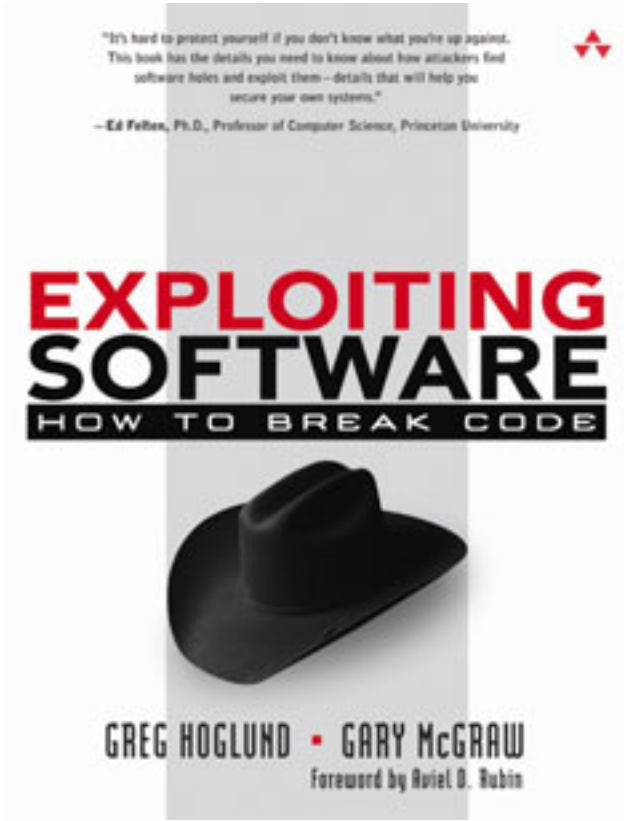
Software security lessons

- Understanding how attackers break software tends to be knowledge and experience intensive
  - Reading stories is fine, but there's no substitute for *time in cockpit*
- But the dev guys don't know what attacks look like in a real world context
  - We do…
- Yet, when the CSIRT participates at all in the dev process it is in the last phase to do the dreaded application penetration test
  - What's wrong with penetration testing?

Attacks are evolving



Breaking stuff is important



- Learning how to think like an attacker is essential
- Do not shy away from carrying out attacks on your own stuff
  - Engineers learn from stories of failure
- Attacking is fun! Fun is good!

Further reading list

- Security Tracker – http://www.SecurityTracker.com
- Risks Digest – http://www.risks.org
- Phrack – http://www.phrack.org
- Full Disclosure – http://archives.neohapsis.com/archives/fulldisclosure/
- Rootkits – http://www.rootkit.com
- US CERT – http://www.us-cert.gov
- OWASP – http://www.owasp.org

- Secure Coding List – http://www.securecoding.org/list/
- Build Security In – http://BuildSecurityIn.us-cert.gov

Incident Handling functions

- Unlike software developers, Incident Handlers have spent years doing
  - Protecting networks and systems from attack
  - Detecting attacks when they occur
  - Responding to detected attacks to protect business interests
- Resulting knowledge base
  - Attack tools
  - Attack techniques
  - Defense tools
- We have an arguably healthy level of mistrust

How about a hybrid solution?

- We should be able to find a way to help the dev team benefit from the knowledge that we have built up, right?
- How about integrating ourselves in the dev process?
- Dev does the software, but we contribute attack knowledge and experience
- Best of both worlds? *(Maybe, maybe not)*
- Let's explore some ideas, but first…

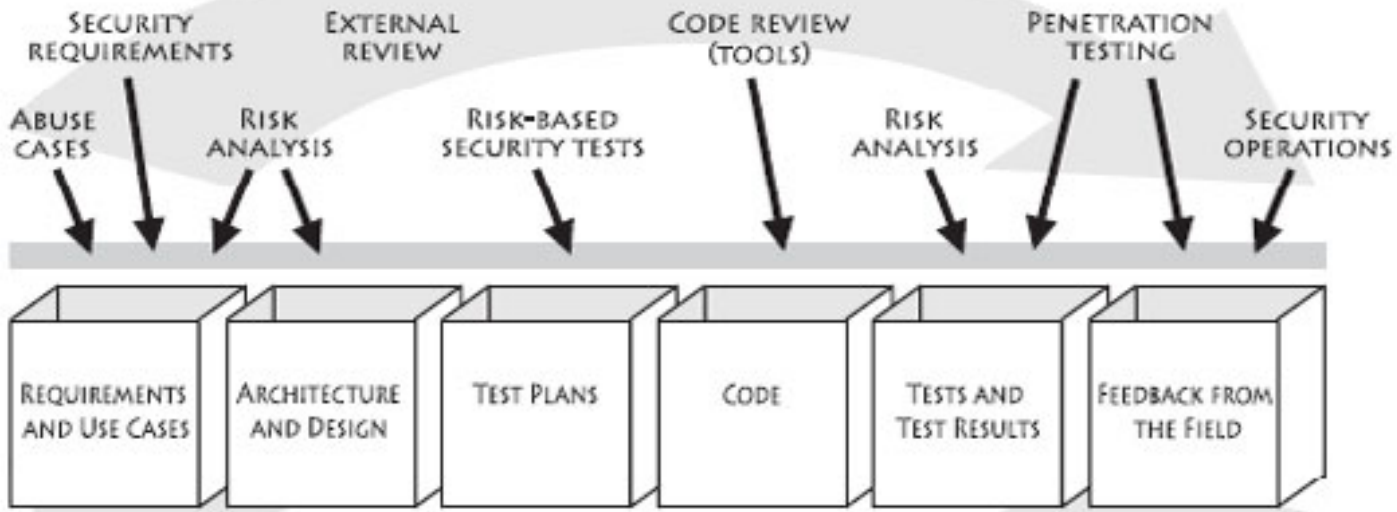
Setting the stage

- It is vital to facilitate the collaboration carefully
  - Cooperative, not adversarial
  - Constructive, not destructive
- All participants must perceive a common goal
  - Protect the business
- It helps to have an assertive but non-threatening moderator

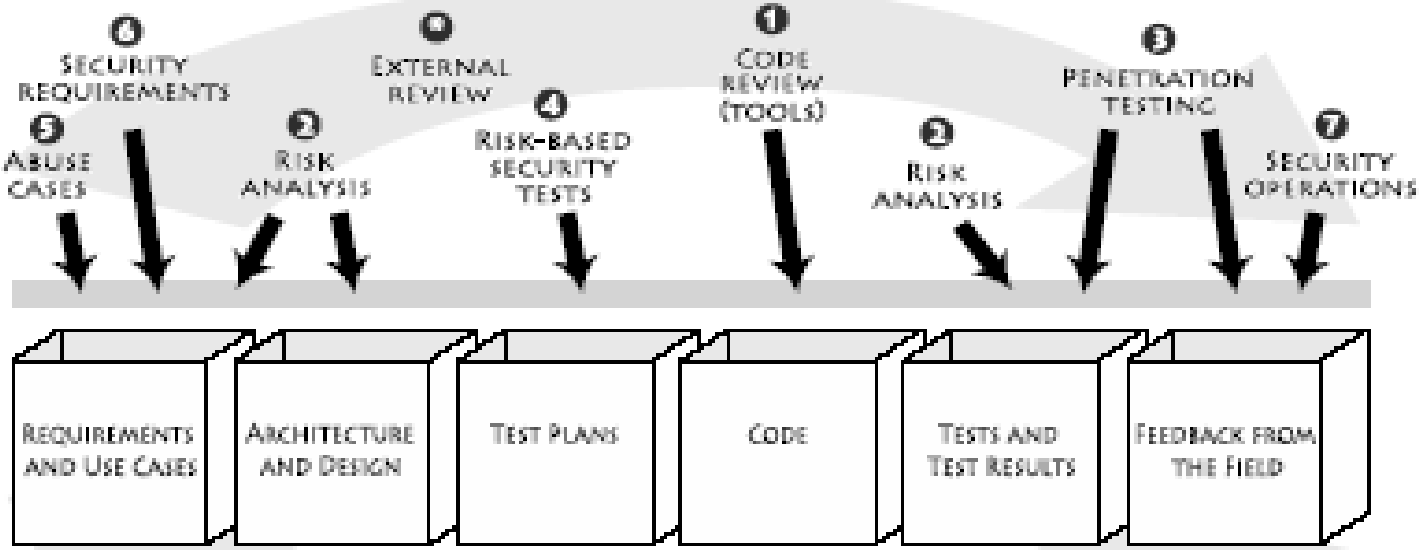
- Now, let's consider how this might work

Software security touchpoints

Software security touchpoints

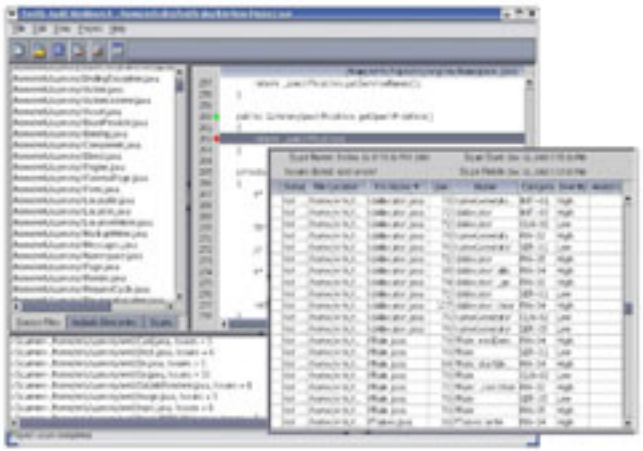


Adopting the touchpoints

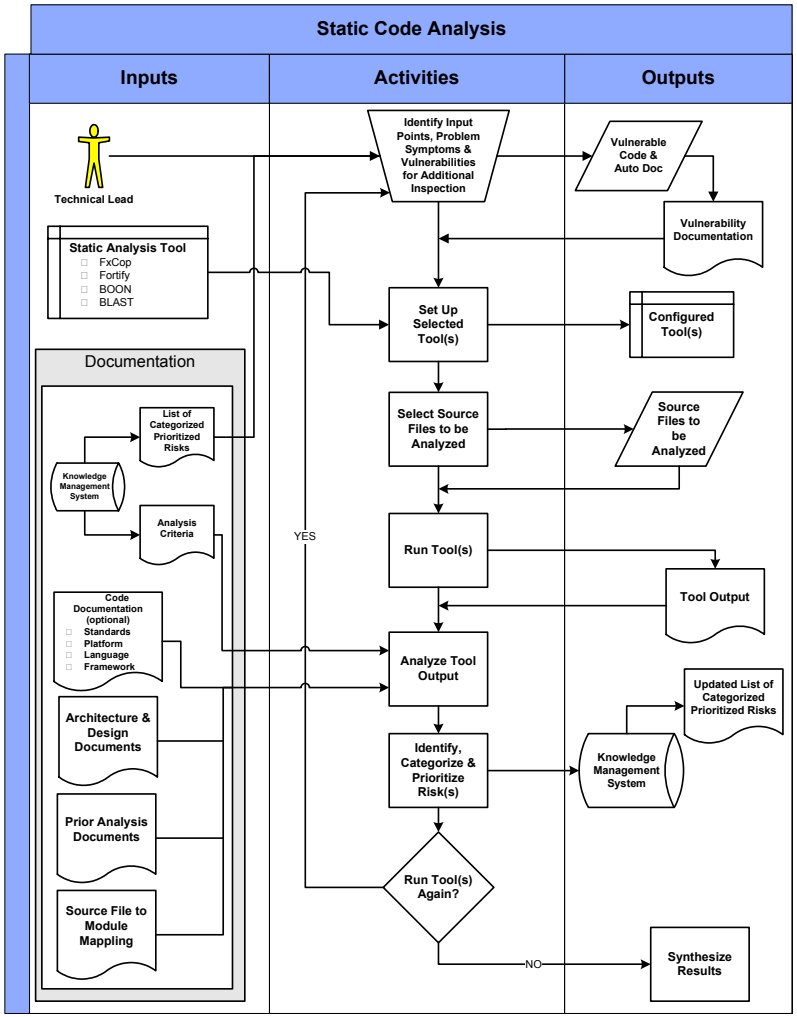


Touchpoint 1: Code review

- Code review is a necessary evil
- Better coding practices make the job easier
- Automated tools help catch silly errors
  - Fortify/dev (Cigital rules)



- Implementation errors do matter
  - Buffer overflows can be uncovered with static analysis
  - Fortify SCA
    - Over 500 C/C++ rules
    - Over 100 Java rules
- Tracing back from vulnerable location to input is critical
  - Software exploits
  - Attacking code



TP1: Code review

- There are many ways to apply code review technology
- Use a tool
- Integrate into the build

TP1: How can the CSIRT help?

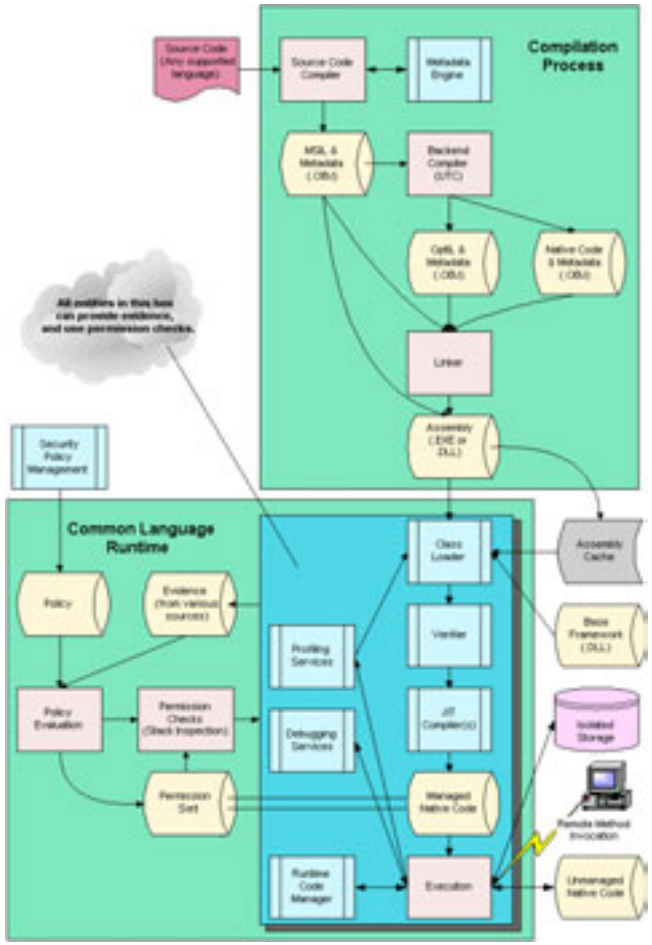
- Not many infosec engineers are proficient at today's high level languages
- How about helping evaluate a finding presented by a scanning tool?
  - "Have attacks against *this* coding issue been seen elsewhere?"
- Useful?
  - Maybe, maybe not…
  - Depends on the people

Touchpoint 2: Architectural risk analysis

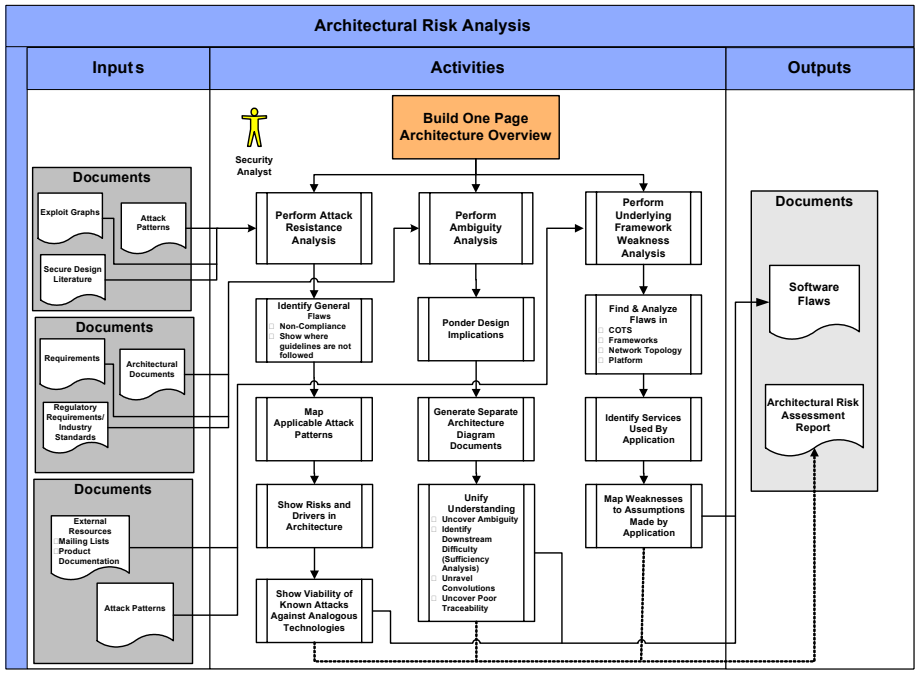
- To assess and understand the risks, ask questions:
  - What is the likelihood of an attack?
  - What does the software do to support your organization's mission?
  - Is there a disaster recovery plan?
  - What would the impact be if the software were unavailable?
  - What is a tolerable down time?
- Whom should you ask?
  - Software owner
  - IT manager
  - Key users

TP2: Architectural risk analysis

- Designers should not do this
- Build a one page white board design model (like that →)
- Use hypothesis testing to categorize risks
  - Threat modeling/Attack patterns
- Rank risks
- Tie to business context
- Suggest fixes
- Repeat



TP2: Risk analysis



- Start by building a one page overview of your system
- The apply the three step process we will describe more fully later
  - Attack resistance
  - Ambiguity analysis
  - Weakness analysis

TP2: How can the CSIRT help?

- Participate in architecture discussions to help question assumptions
- Attack resistance
  - Knowledge base of historical attacks
- Weakness analysis
  - Can help rate the severity and likelihood of architectural weaknesses
- Ambiguity analysis
  - Help identify design ambiguities

Touchpoint 3: Penetration testing

- A very good idea since software is bound in an environment
- How does the complete system work in practice?
  - Interaction with network security mechanisms
  - Firewalls
  - Applied cryptography
- Penetration testing should be driven by risks uncovered throughout the lifecycle
- Abuse cases also useful in defining scenarios
- Not a silver bullet!

TP3: How can the CSIRT help?

- "Pen testing" has been the purview of infosec in many organizations for years
- If team is sufficiently knowledgeable on attacks, they can ensure realism
  - Be wary of over reliance on tools
  - Best testers use tools as starting points only
- Use risk analyses to prioritize and optimize efforts
- Human judgment is important

Touchpoint 4: Security testing

- Test security functionality
  - Cover non-functional requirements
  - Security software probing
- Risk-based testing
  - Use architectural risk analysis results to drive scenario-based testing
  - Concentrate on what “you can’t do”
  - Think like an attacker
  - Informed red teaming

TP4: Risk-based testing

- Identify areas of potential risk in the system
  - Requirements
  - Design
  - Architecture
- Use abuse cases to drive testing according to risk
- Build attack and exploit scenarios based on identified risks
- Test risk conditions explicitly

- Example: Overly complex object-sharing system in Java Card

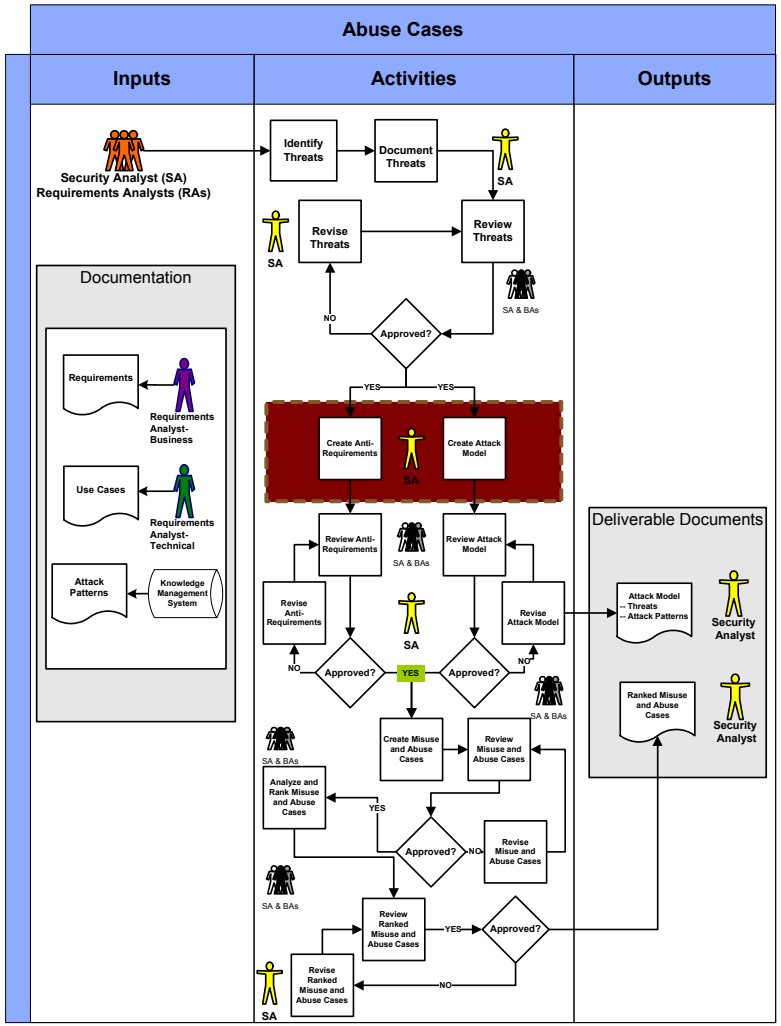
TP4: How can the CSIRT help?

- Can help testers develop realistic test plans and scenarios
- Can share attack pattern knowledge base with testers and explain significance
- Provide attack examples, tools, exploits, etc., to testers

Touchpoint 5: Abuse cases

- Use cases formalize normative behavior (and assume correct usage)
- Describing non-normative behavior is a good idea
  - Prepare for abnormal behavior (attack)
  - Misuse or abuse cases do this
  - Uncover exceptional cases
- Leverage the fact that designers know more about their system than potential attackers do
- Document explicitly what the software will do in the face of illegitimate use

- Think like an attacker!



TP5: Abuse cases

- Starting with attack patterns, requirements and use cases
- Identify anti-requirements
- Build an attack model
- Determine misuse and abuse cases

TP5: How can the CSIRT help?

- Participate in brainstorming of abuse case scenarios
- Provide documentation to describe similar historical attacks

Touchpoint 6: Security requirements

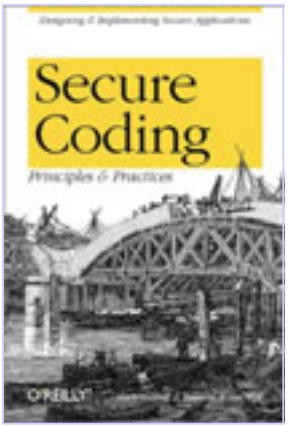
- Some security functionality maps naturally to clear requirements
  - Medical data should be cryptographically protected
  - Strongly authenticate users
  - Meet GLBA regulatory guidelines
- But do not forget that security is an emergent property of a complete system
  - An attacker needs to find only one hole
  - “Do not allow buffer overflows” is not much of a requirement!
  - “Make it secure” is vague

TP6: How can the CSIRT help?

- May be more familiar with regulatory issues than dev team
- Cite and research applicable regulations and laws

Touchpoint 7: Security operations

- Fine tune the deployed environment to the specific needs of your application
  - "Standard OS build" process is not enough
- Use white list methodologies to configure network, OS, and app environment
- Configure and execute event logging within the application
  - Application level audit trails
  - Watch over the app's "crown jewels"

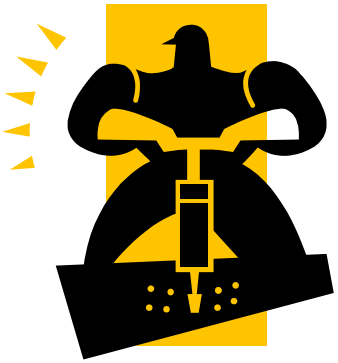


TP7: How can the CSIRT help?

- Can help provide bridge between dev and ops to help fine tune op environment to the specific needs of the app
- Can help ops monitors triage event log triggers 24/7

Will it work?

- What roadblocks do you see to including IT Security in your dev process?
  - “They don’t get it?”
  - “They’ll use the information against us?”
  - “Not enough time cycles?”
  - “Great, another thing to do.”



Discussion

Discussion

- Does your CSIRT participate in your dev process now? Other than just penetration testing?
  - If so, to what extent?
  - If not, what would prevent it from happening in your organization?
- Which of the described touchpoints are most likely to benefit from collaboration between dev and CSIRT?

Contact information

Gary McGraw
CTO
Cigital, Inc.
http://www.Cigital.com
gem@Cigital.com

Kenneth R. van Wyk
Principal Consultant

KRvW Associates, LLC
http://www.KRvW.com

Ken@KRvW.com