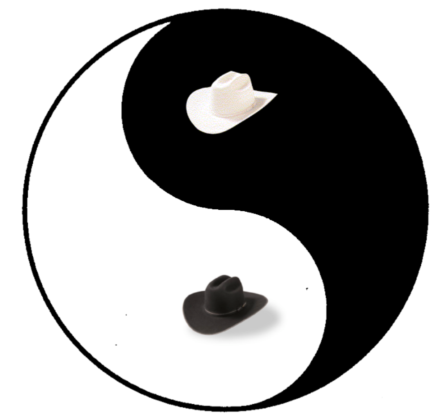
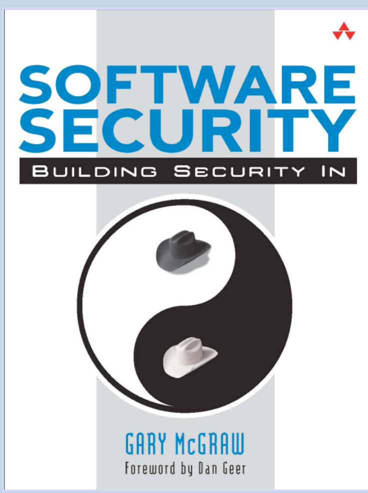
cigital

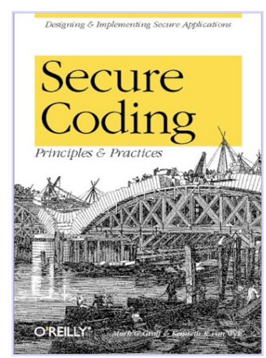
KRvW Associates

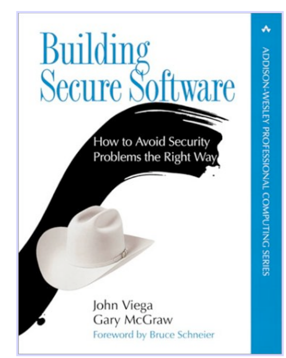
Software security

Setting the stage









Agenda

9:00-10:00	Software [in]security
10:15-12:00	Exploiting Software and exercise
1:00-2:30	Software security touchpoints
2:45-4:30	Seven pernicious kingdoms
4:30-5:00	Code review and next steps

KRvW Associates

cigital

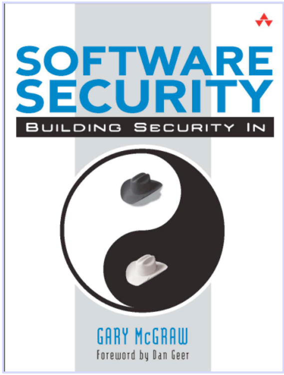
Pop quiz

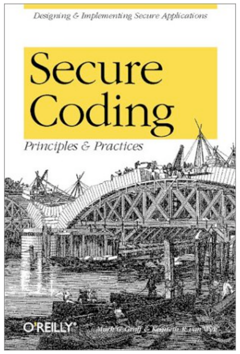
- What do wireless devices, cell phones, PDAs, browsers, operating systems, servers, personal computers, routers, public key infrastructure systems, and firewalls have in common?

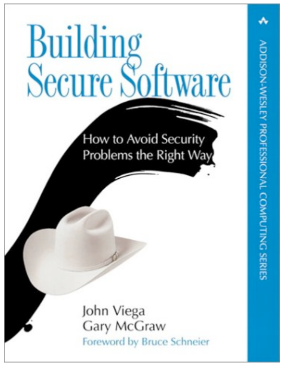
Questions for you

- Who is from dev? How about testing? Anyone here from product management?
- What languages do you use? C? C++? Java?
- How do you describe and capture software architecture and design?
- Do you follow a particular software process in your group?

Software [in]security

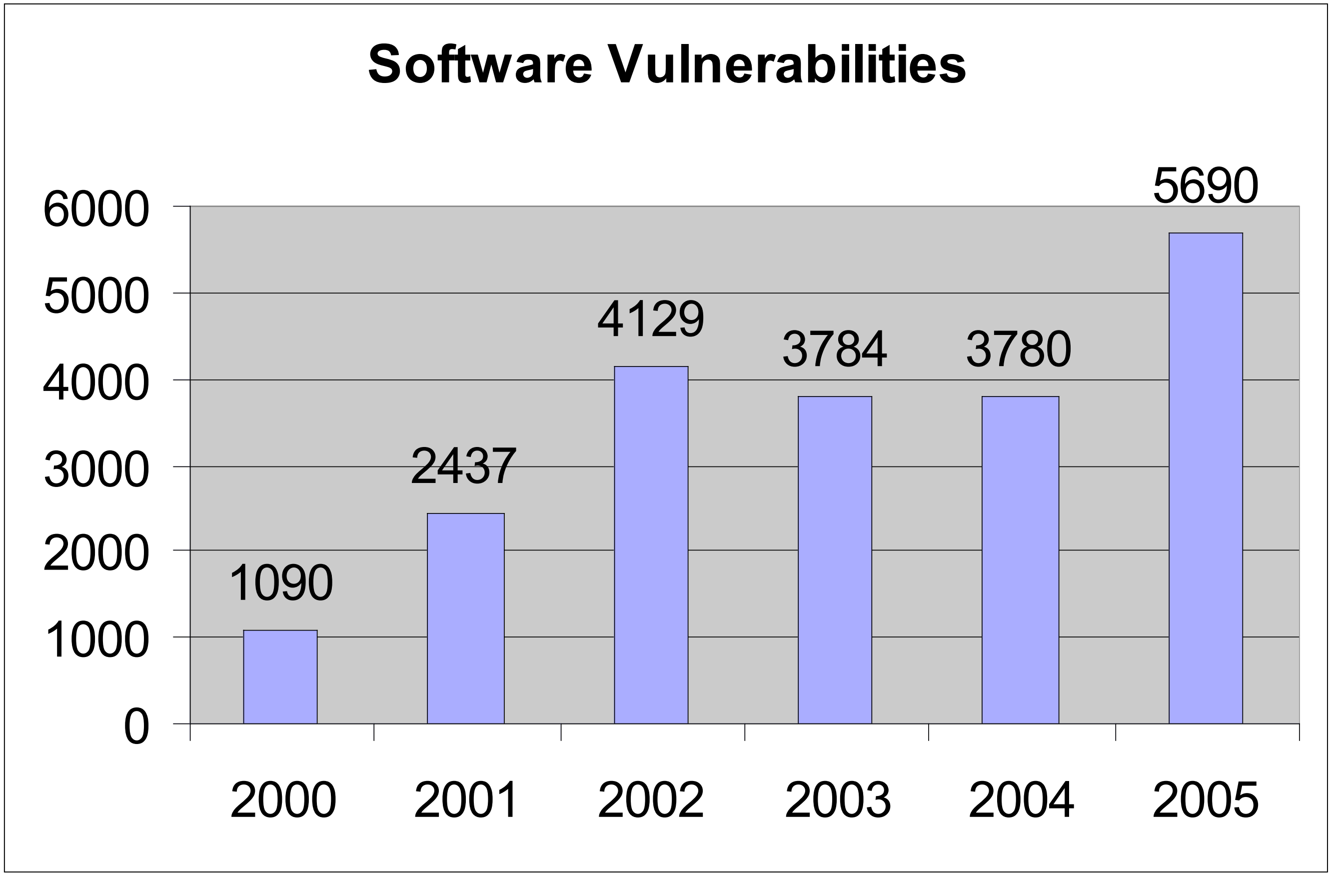






The Problem

Software vulnerability growth



The Trinity Of Trouble: Connectivity

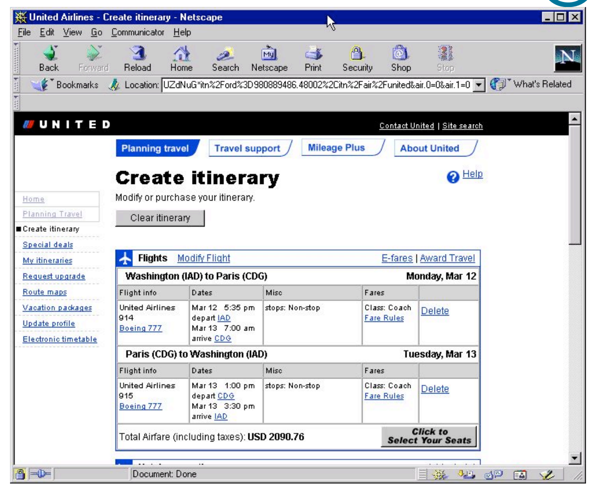
- The Internet is everywhere and most of our software is on it
- When was the last time that you did business with a major vendor who had no Internet connectivity?
- Tried VoIP on your mobile phone in a coffee shop WiFi hotspot yet?

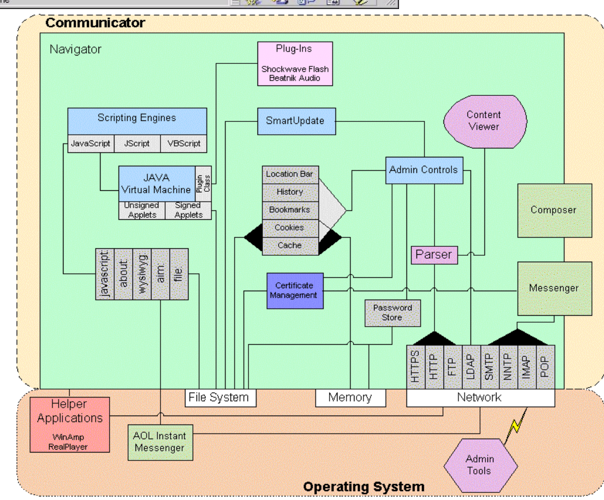
The network is the computer.

Sun microsystems

The Trinity Of Trouble: Complexity

- A simple user interface can be enormously complex "under the hood"
- Consider what happens behind the scenes in one of today's AJAX web applications
- But it sure does make for a compelling "user experience"



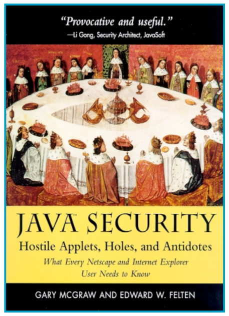


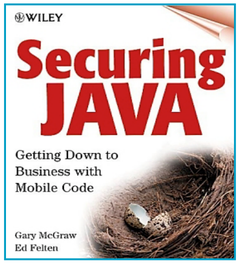
KRvW Associates

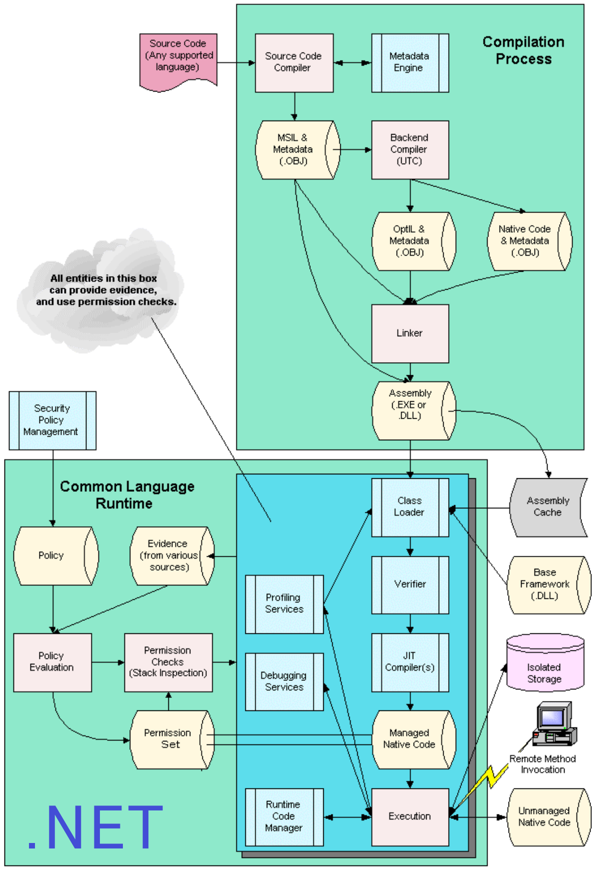
cigital

The Trinity Of Trouble: Extensibility

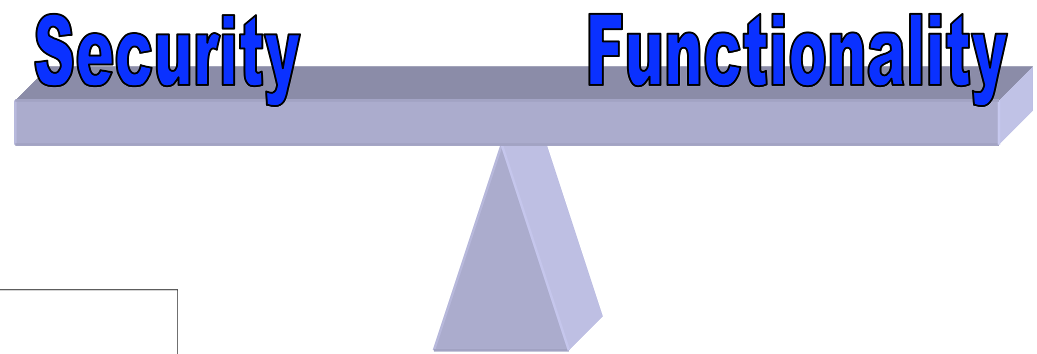
- Systems evolve in unexpected ways and are changed on the fly
- After all, who would want a computing device that can't be functionally extended?
- From J2ME to desktop PC users (running with administrative privileges)

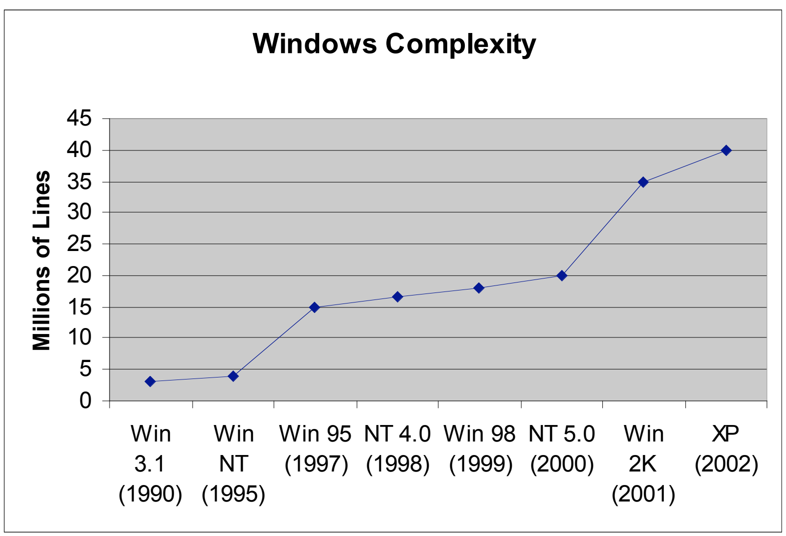






The classic security tradeoff





So what's the problem?

- Well, for starters
  - Consumers don't demand more
  - Software developers tend to lack knowledge of vulnerabilities, attacks, and threats
  - IT security tends to not understand software development
- But that's not all!

KRvW Associates

Additional problems - 1

- We don’t pay enough attention to our failures
- Consider other engineering disciplines

KRvW Associates

Additional problems - 2

- We fail to consider business risks first and foremost
- Business must drive technology
- Consider Wi-Fi, Word macros, USB drives, etc.

KRvW Associates

Additional problems - 3

- Old school information security solutions don't adequately protect the software
- Consider IM, Skype, Wi-Fi, VPNs

KRvW
Associates

Additional problems - 4

- Software testing does not adequately address security
- Penetration testing is not sufficient

Additional problems - 5

- Too much attention is paid to functional spec
- Consider what can go wrong as well

Additional problems - 6

- IT security is viewed as an impediment to business
- Don’t just be the person that says no

Security problems are complicated

IMPLEMENTATION BUGS

- Buffer overflow
  - String format
  - One-stage attacks
- Race conditions
  - TOCTOU (time of check to time of use)
- Unsafe environment variables
- Unsafe system calls
  - System()
- Untrusted input problems

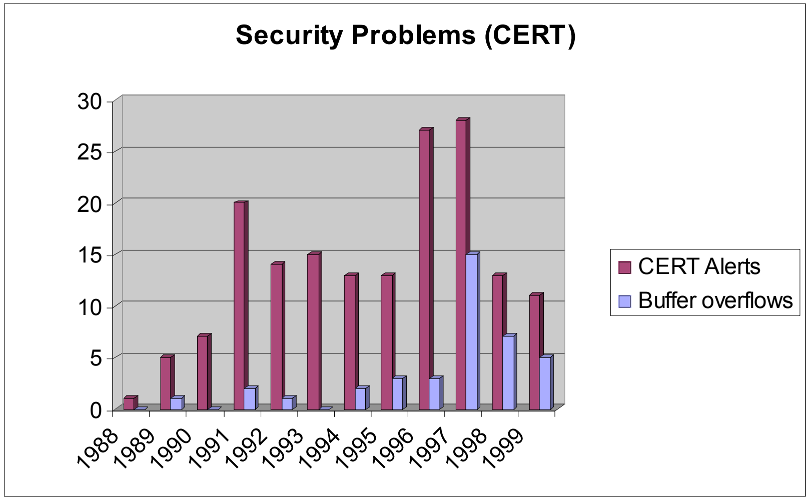
ARCHITECTURAL FLAWS

- Misuse of cryptography
- Compartmentalization problems in design
- Privileged block protection failure (DoPrivilege())
- Catastrophic security failure (fragility)
- Type safety confusion error
- Insecure auditing
- Broken or illogical access control (RBAC over tiers)
- Method over-riding problems (subclass issues)
- Signing too much code

BUG: The dreaded buffer overflow

- Overwriting the bounds of data objects
- Allocate some bytes, but the language doesn't care if you try to use more
  - `char x[12];`
    `x[12] = '\0';`
- Why was this done? Efficiency!
- Two main flavors of buffers
  - Heap allocated buffers
  - Stack allocated buffers
  - Smashing the stack is the most common attack

- The most pervasive security problem today in terms of reported bugs



Pervasive C problems lead to BUGS

```
void main() {
   char buf[1024];
   gets(buf);
}
```

- How not to get input
  - Attacker can send an infinite string!
  - Chapter 7 of K&R (page 164)

- Calls to watch out for

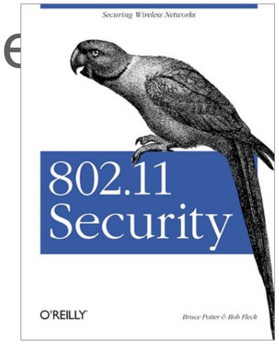
Instead of:	Use:
gets(buf)	fgets(buf, size, stdin)
strcpy(dst, src)	strncpy(dst, src, n)
strcat(dst, src)	strncat(dst, src, n)
sprintf(buf, fmt, a1,…)	snprintf(buf, fmt, a1, n1,…) (where available)
*scanf(…)	Your own parsing

- Hundreds of such calls
- Use static analysis to find these problems
  - ITS4, Fortify
- Careful code review is necessary

KRvW
Associates

FLAW: 802.11b WEP crypto

- Well-documented flaws in the design of the WEP protocol
- Even if implemented 100% perfectly, the design is flawed and the encryption easily circumvented
- 802.11b is widely deployed and wildly popular
- It was designed by experts
- Would you entrust a mission-critical enterprise to run over it?

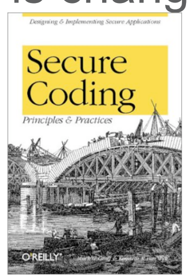


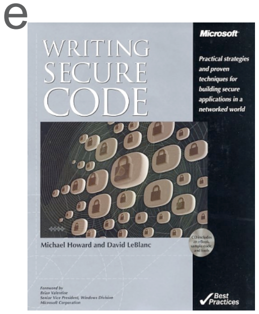
KRvW Associates

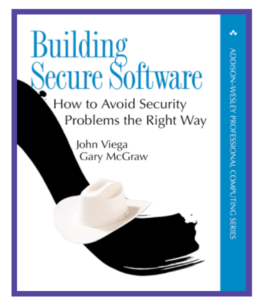
Software security: state of the practice

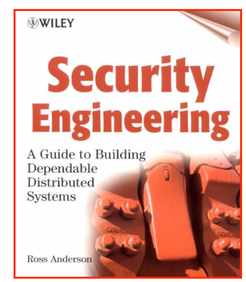
- Programming is hard
- Popular languages are really awful (C/C++)
- Many subtleties to learn
- Lots to know
- The only constant is change

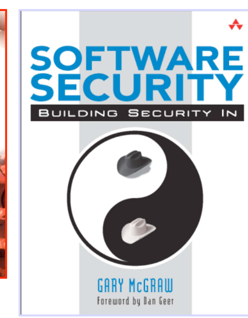
- Some good resources on software security
- Tools are getting better, but only cover BUGS





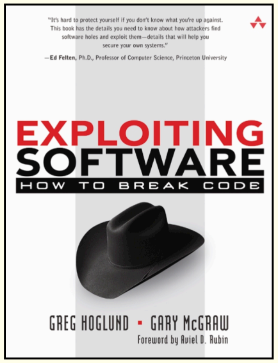






Software security is not security software!
Software security is about building things properly.

Exploiting software



Who is the bad guy?

- Hackers
  - "Full disclosure" zealots
- "Script kiddies"
- Criminals
  - Lone guns or organized
- Malicious insiders
  - Compiler wielders
- Business competition
- Police, press, terrorists, intelligence agencies

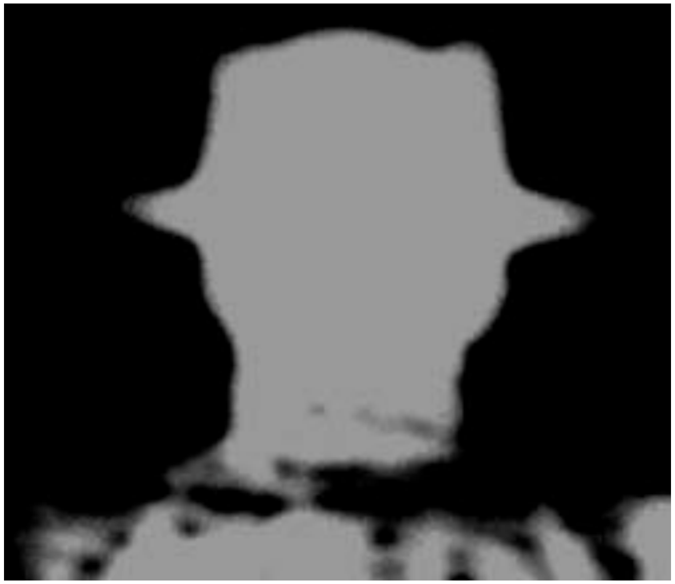
Above: Deputy Attorney General Christopher Bubb, center, N.J. Gov. Christie Whitman (left) and Attorney General Peter Verniero conferencing on alleged virus writer David L. Smith (right).

AP, ZDTV

KRvW Associates

Attackers do not distinguish bugs and flaws

- Both bugs and flaws lead to vulnerabilities that can be exploited
- Attackers are pragmatic in their approach
- Attackers write code to break your software's design and/or implementation



cigital KRvW Associates

How attacks unfold

- Attacking a system is a process of discovery and exploration
  - Qualify target (focus on input points)
  - Determine what transactions the input points allow
  - Apply relevant attack patterns
  - Cycle through observation loop
  - Find vulnerability
  - Build an exploit

<u>The standard process</u>

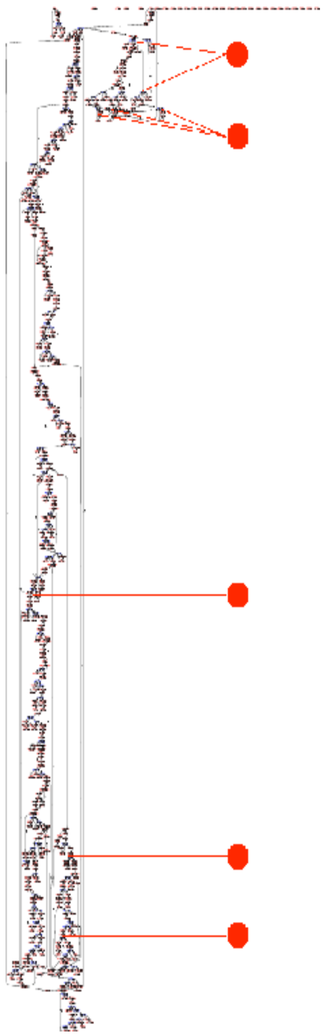
- Scan network
- Build a network map
- Pick target system
- Identify OS stack
- Port scan
- Determine target components
- Choose attack patterns
- Leverage environment faults
- Use indirection
- Plant backdoor

Attacker's toolkit: disassemblers and decompilers

- Source code is not a necessity for software exploit
- Binary is just as easy to understand as source code
- Disassemblers and decompilers are essential tools
- Reverse engineering is common and must be understood (not outlawed)
- IDA allows plugins to be created
- Use bulk auditing

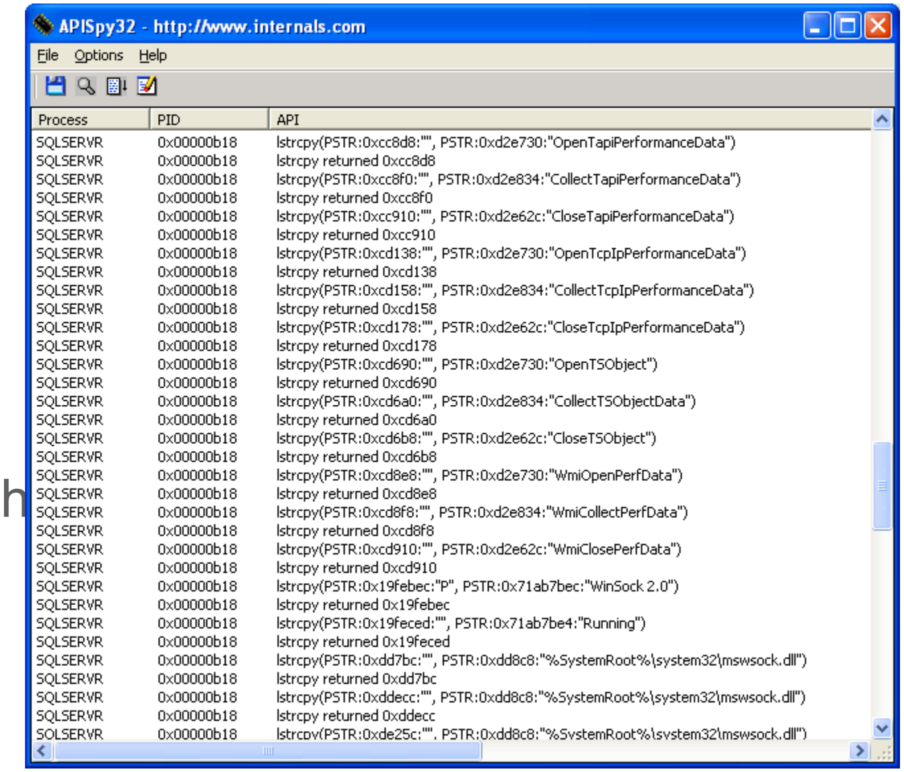
Attacker’s toolkit: control flow and coverage

- Tracing input as it flows through software is an excellent method
- Exploiting differences between versions is also common
- Code coverage tools help you know where you have gotten in a program
  - dyninstAPI (Maryland)
  - Figure out how to get to particular system calls
  - Look for data in shared buffers



Attacker's toolkit: APISPY32

- Look for broken system calls (at all levels in code)
- lstrcpy() makes a great example
- On win32 systems, use APISPY to determine which APIs are being used by a target program



Process	PID	API
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xcc8d8:"", PSTR:0xd2e730:"OpenTapiPerformanceData")
SQLSERVR	0x00000b18	lstrcpy returned 0xcc8d8
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xcc8f0:"", PSTR:0xd2e834:"CollectTapiPerformanceData")
SQLSERVR	0x00000b18	lstrcpy returned 0xcc8f0
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xcc910:"", PSTR:0xd2e62c:"CloseTapiPerformanceData")
SQLSERVR	0x00000b18	lstrcpy returned 0xcc910
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xcd138:"", PSTR:0xd2e730:"OpenTcpIpPerformanceData")
SQLSERVR	0x00000b18	lstrcpy returned 0xcd138
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xcd158:"", PSTR:0xd2e834:"CollectTcpIpPerformanceData")
SQLSERVR	0x00000b18	lstrcpy returned 0xcd158
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xcd178:"", PSTR:0xd2e62c:"CloseTcpIpPerformanceData")
SQLSERVR	0x00000b18	lstrcpy returned 0xcd178
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xcd690:"", PSTR:0xd2e730:"OpenTSObject")
SQLSERVR	0x00000b18	lstrcpy returned 0xcd690
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xcd6a0:"", PSTR:0xd2e834:"CollectTSObjectData")
SQLSERVR	0x00000b18	lstrcpy returned 0xcd6a0
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xcd6b8:"", PSTR:0xd2e62c:"CloseTSObject")
SQLSERVR	0x00000b18	lstrcpy returned 0xcd6b8
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xcd8e8:"", PSTR:0xd2e730:"WmiOpenPerfData")
SQLSERVR	0x00000b18	lstrcpy returned 0xcd8e8
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xcd8f8:"", PSTR:0xd2e834:"WmiCollectPerfData")
SQLSERVR	0x00000b18	lstrcpy returned 0xcd8f8
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xcd910:"", PSTR:0xd2e62c:"WmiClosePerfData")
SQLSERVR	0x00000b18	lstrcpy returned 0xcd910
SQLSERVR	0x00000b18	lstrcpy(PSTR:0x19febec:"P", PSTR:0x71ab7bec:"WinSock 2.0")
SQLSERVR	0x00000b18	lstrcpy returned 0x19febec
SQLSERVR	0x00000b18	lstrcpy(PSTR:0x19feced:"", PSTR:0x71ab7be4:"Running")
SQLSERVR	0x00000b18	lstrcpy returned 0x19feced
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xdd7bc:"", PSTR:0xdd8c8:"%SystemRoot%\system32\mswsock.dll")
SQLSERVR	0x00000b18	lstrcpy returned 0xdd7bc
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xddecc:"", PSTR:0xdd8c8:"%SystemRoot%\system32\mswsock.dll")
SQLSERVR	0x00000b18	lstrcpy returned 0xddecc
SQLSERVR	0x00000b18	lstrcpy(PSTR:0xde25c:"", PSTR:0xdd8c8:"%SystemRoot%\system32\mswsock.dll")

- Interposition attacks are a great thing to think about at this level

cigital KRvW Associates

Attacker's toolkit: breakpoints

- Breakpoints are central to debuggers
  - Use interrupt 3 on x86 architectures
- Mark entire blocks for access
- Single step at breakpoint (also as in debugging)

- Check out "The PIT" http://www.hbgary.com

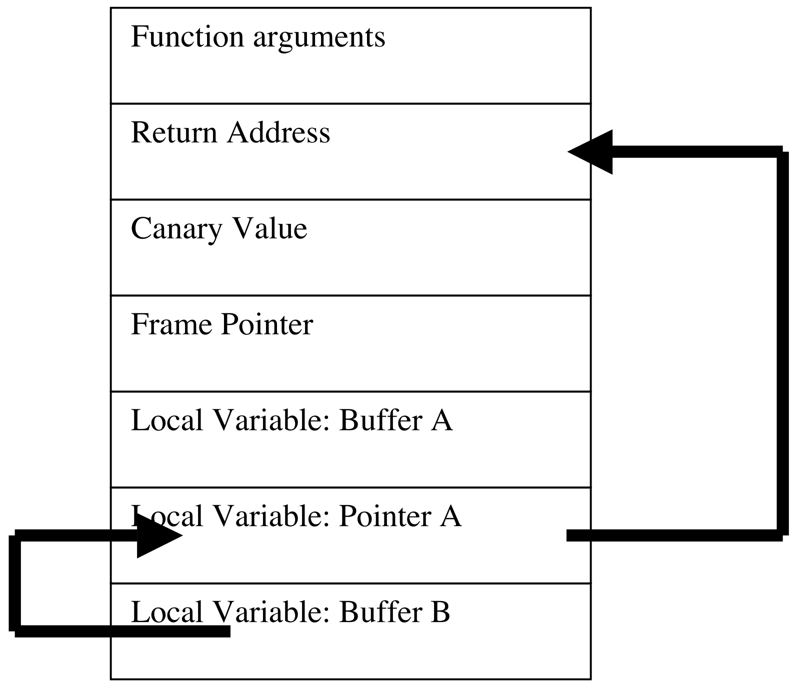
KRvW Associates

cigital

Attacker's toolkit: the buffer overflow

- Find targets with static analysis
- Change program control flow
  - Heap attacks
  - Stack smashing
  - Trampolining
- Particular examples
  - Overflow binary resource files (used against Netscape)
  - Overflow variables and tags (Yamaha MidiPlug)
  - MIME conversion fun (Sendmail)
  - HTTP cookies (apache)

- Trampolining past a canary



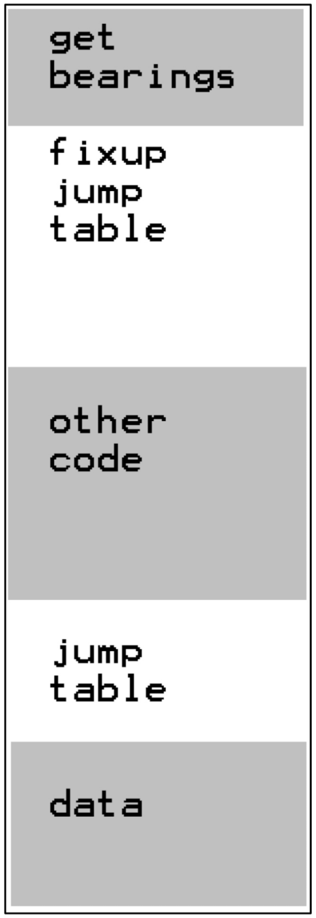
KRvW Associates

cigital

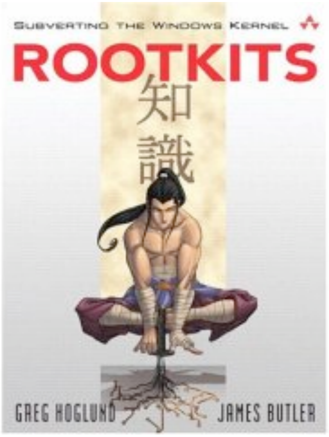
Attacker's toolkit: shell code and other payloads

- Common payloads in buffer overflow attacks
- Size matters (small is critical)
- Avoid zeros
- XOR protection (also simple crypto)

- Payloads for
  - X86 (win32)
  - RISC (MIPS and sparc)
  - Multiplatform payloads



Attacker's toolkit: rootkits



- The apex of software exploit…complete the machine
- Live in the kernel
  - XP kernel rootkit in the book
  - See http://www.rootkit.com
- Get into the microchips (hardware viruses)
- Hide files and directories by controlling access to process tables
- Provide control and access over the network

KRvW Associates

Attacker’s toolkit: other miscellaneous tools

- Debuggers (user-mode)
- Kernel debuggers
  - SoftIce
- Fault injection tools
  - Failure simulation tool
  - Hailstorm
  - Holodeck
- Boron tagging
- The “depends” tool
- Grammar rewriters

Attack Patterns

KRvW Associates

Knowledge: 48 Attack Patterns

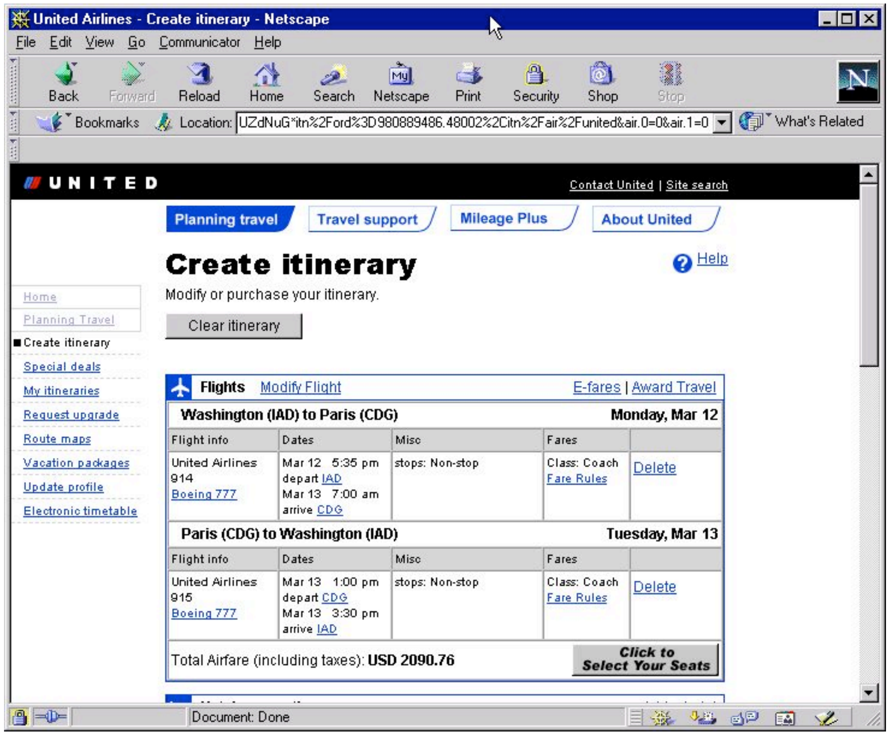
- Make the Client Invisible
- Target Programs That Write to Privileged OS Resources
- Use a User-Supplied Configuration File to Run Commands That Elevate Privilege
- Make Use of Configuration File Search Paths
- Direct Access to Executable Files
- Embedding Scripts within Scripts
- Leverage Executable Code in Nonexecutable Files
- Argument Injection
- Command Delimiters
- Multiple Parsers and Double Escapes
- User-Supplied Variable Passed to File System Calls
- Postfix NULL Terminator
- Postfix, Null Terminate, and Backslash
- Relative Path Traversal
- Client-Controlled Environment Variables
- User-Supplied Global Variables (DEBUG=1, PHP Globals, and So Forth)
- Session ID, Resource ID, and Blind Trust
- Analog In-Band Switching Signals (aka “Blue Boxing”)
- Attack Pattern Fragment: Manipulating Terminal Devices
- Simple Script Injection
- Embedding Script in Nonscript Elements
- XSS in HTTP Headers
- HTTP Query Strings
- User-Controlled Filename
- Passing Local Filenames to Functions That Expect a URL
- Meta-characters in E-mail Header
- File System Function Injection, Content Based
- Client-side Injection, Buffer Overflow
- Cause Web Server Misclassification
- Alternate Encoding the Leading Ghost Characters
- Using Slashes in Alternate Encoding
- Using Escaped Slashes in Alternate Encoding
- Unicode Encoding
- UTF-8 Encoding
- URL Encoding
- Alternative IP Addresses
- Slashes and URL Encoding Combined
- Web Logs
- Overflow Binary Resource File
- Overflow Variables and Tags
- Overflow Symbolic Links
- MIME Conversion
- HTTP Cookies
- Filter Failure through Buffer Overflow
- Buffer Overflow with Environment Variables
- Buffer Overflow in an API Call
- Buffer Overflow in Local Command-Line Utilities
- Parameter Expansion
- String Format Overflow in syslog()

cigital

KRvW Associates

Attack pattern 1: Make the client invisible

- Remove the client from the communications loop and talk directly to the server
- Leverage incorrect trust model (never trust the client)
- Example: hacking browsers that lie



Attack pattern 2: Command delimiters

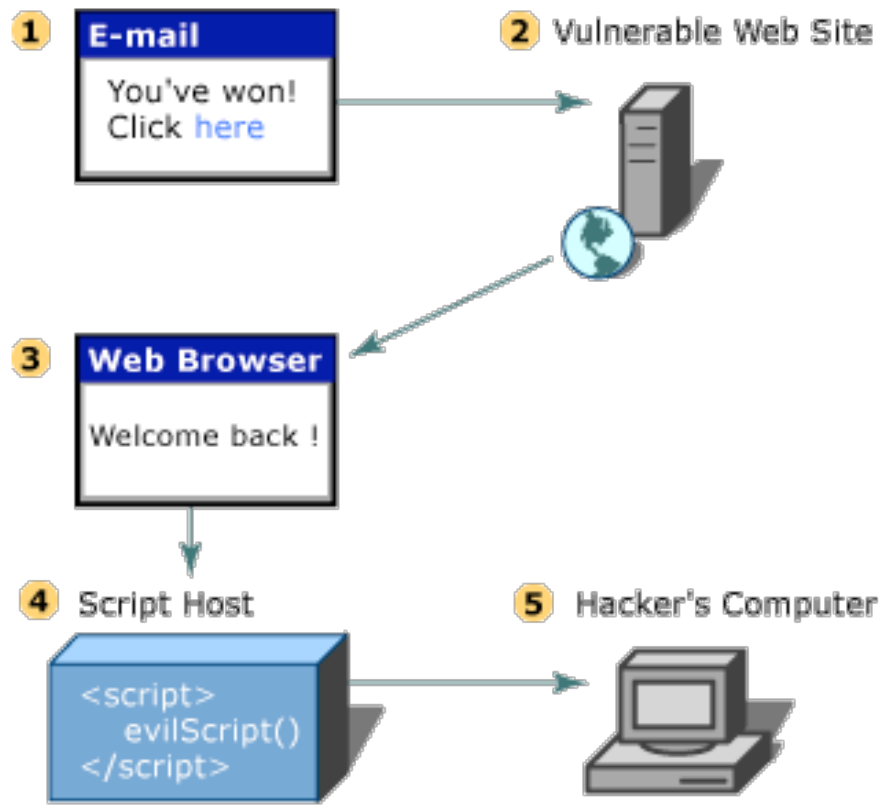
- Use off-nominal characters to string together multiple commands
- Example: shell command injection with delimiters

<input type=hidden name=filebase value="bleh; [command]”>

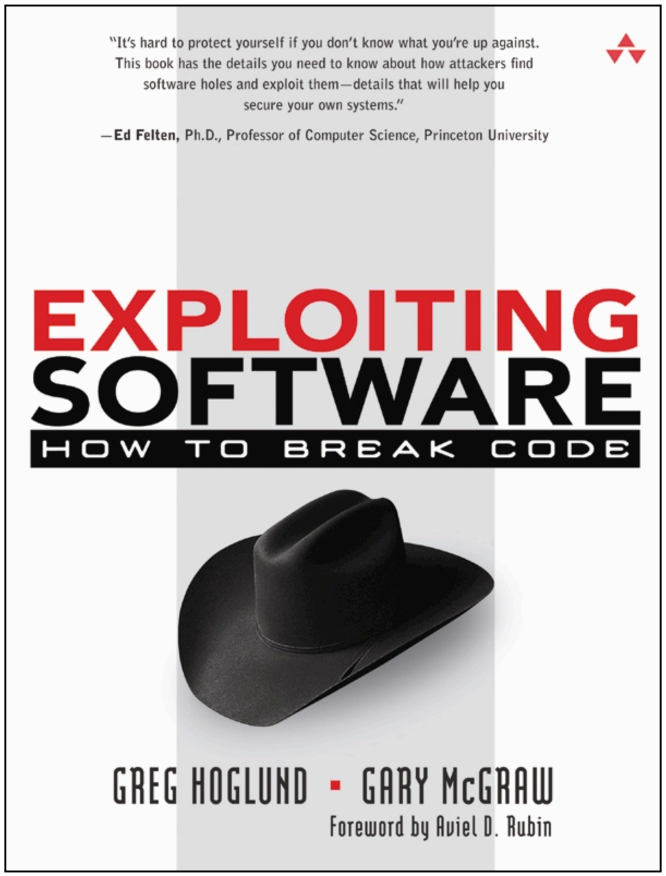
cat data_log_; rm -rf /; cat temp.dat

Attack pattern 3: Cross site scripting

- XSS
  - Attacker sends active content to a victim
  - Content invokes a script on the vulnerable website
  - Later invoked by a web browser hitting the website
  - The script runs
  - Attacker allowed access
- Examples
  - Javascript injection
  - Inject in non-script elements
  - HTTP headers
  - Query strings

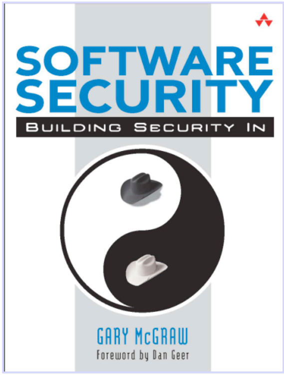


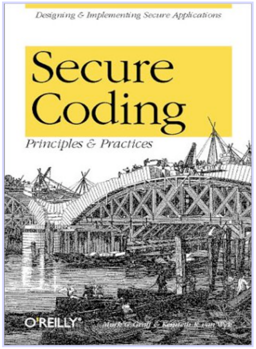
Breaking stuff is important

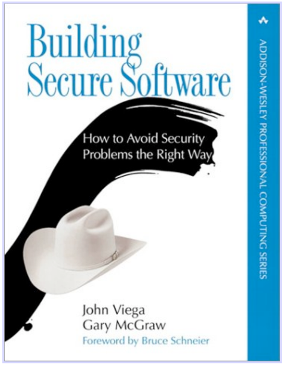


- Learning how to think like an attacker is essential
- Do not shy away from carrying out attacks on your own stuff
  - Engineers learn from stories of failure
- Attacking is fun! Fun is good!

Software security touchpoints



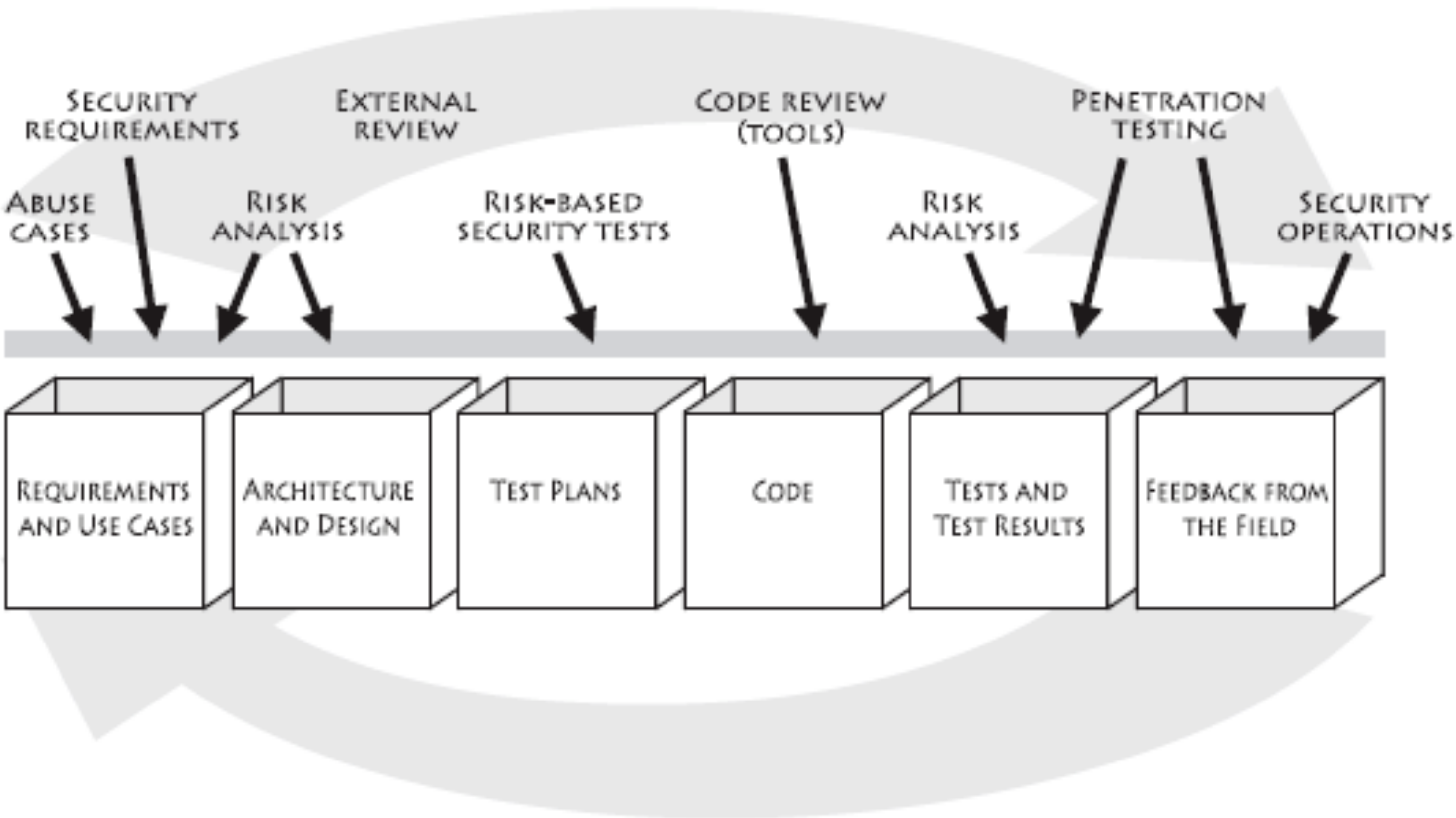




Software security touchpoints

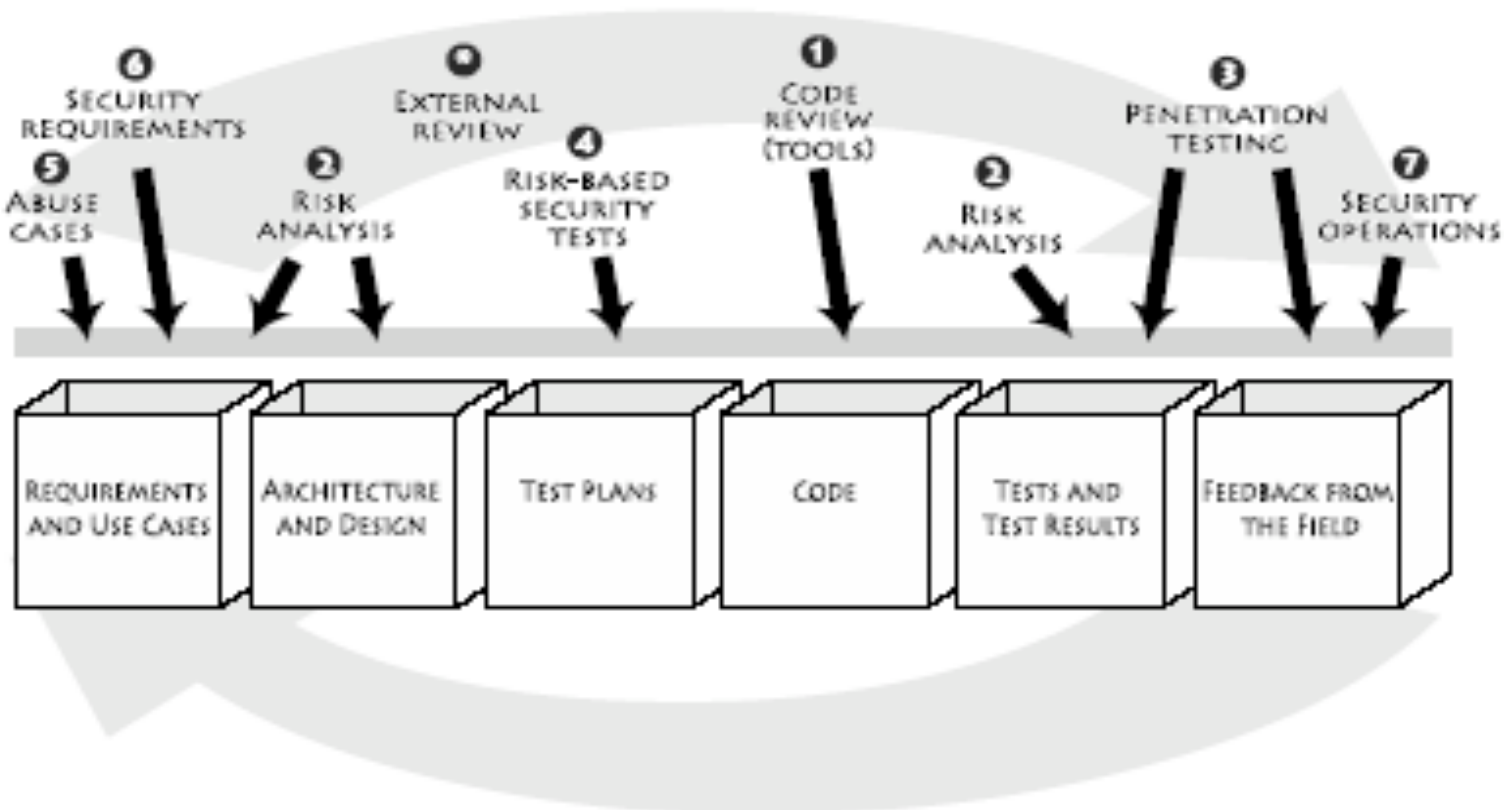
KRvW Associates

Software security touchpoints



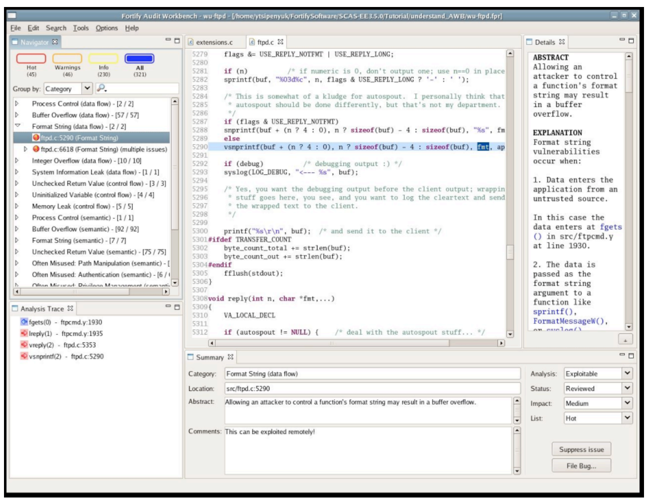
KRvW Associates

Adopting the touchpoints



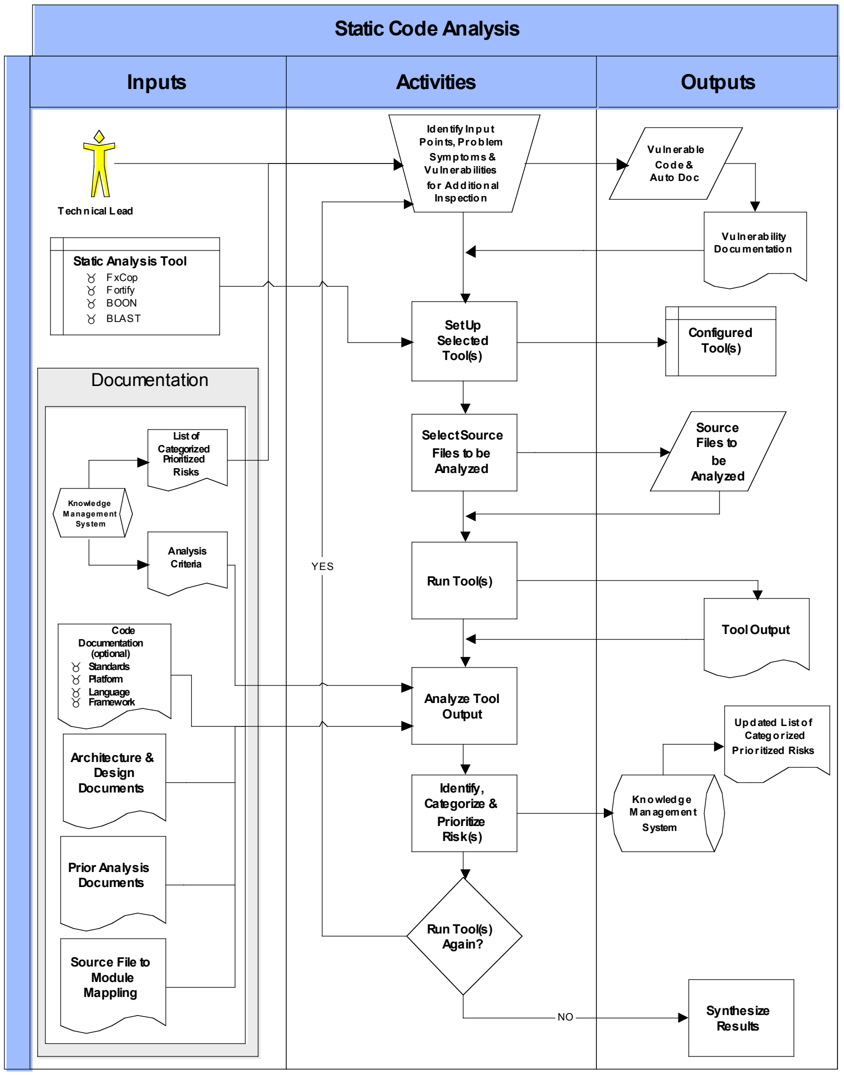
Touchpoint 1: code review (with a tool)

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



- Implementation errors do matter
    - Buffer overflows can be uncovered with static analysis
    - Static analysis
        - C/C++
        - Java
        - .NET
        - PSQL
- Tracing back from vulnerable location to input is critical

TP1: Code review



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

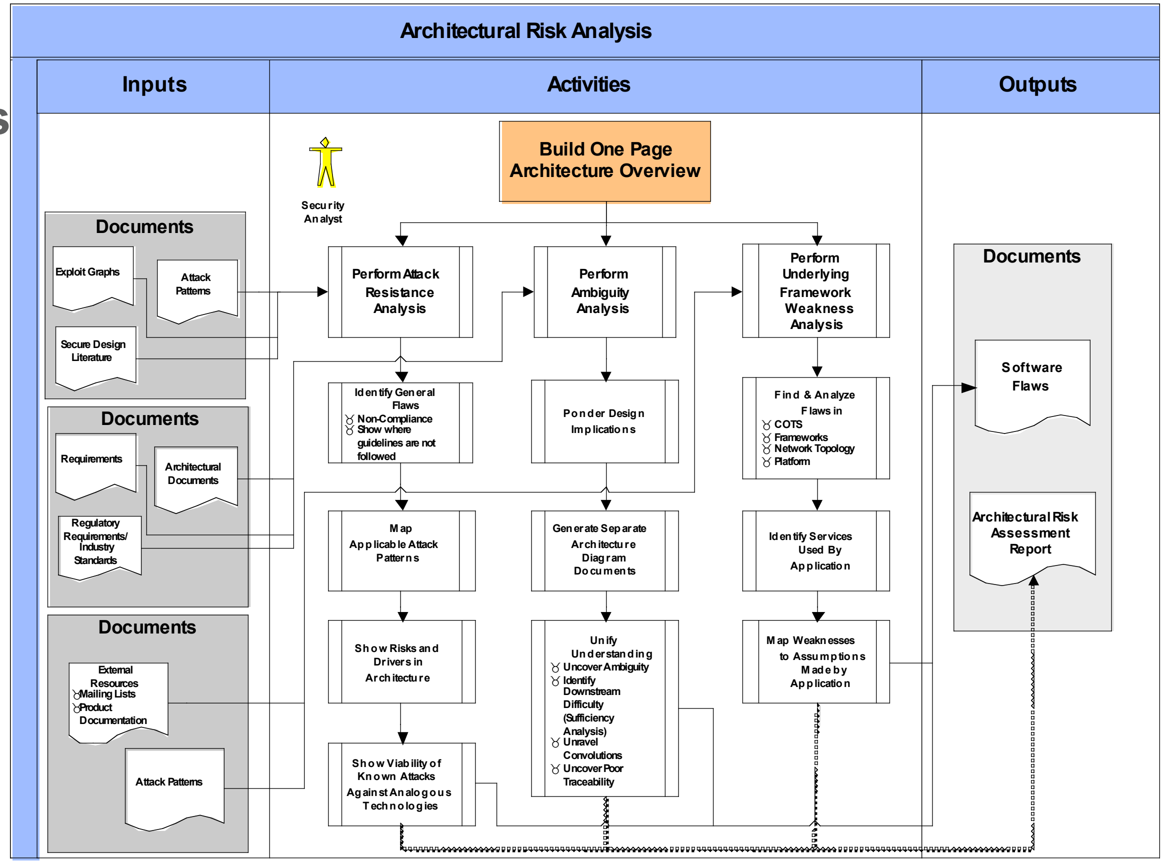
Touchpoint 2: Architectural risk analysis

- 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

KRvW Associates

TP2: Architectural risk analysis

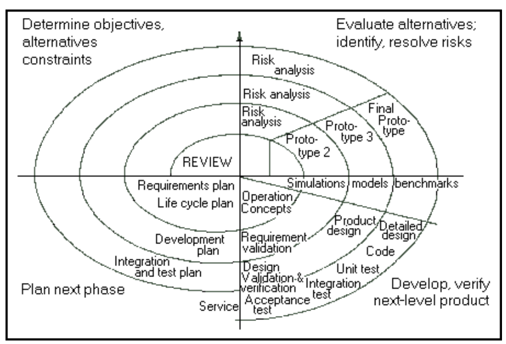
- **Follow a process**
- Build an overview (one page)
- Three steps
  - Attack resistance analysis
  - Ambiguity analysis
  - Weakness analysis
- Rank risks
- Build mitigations

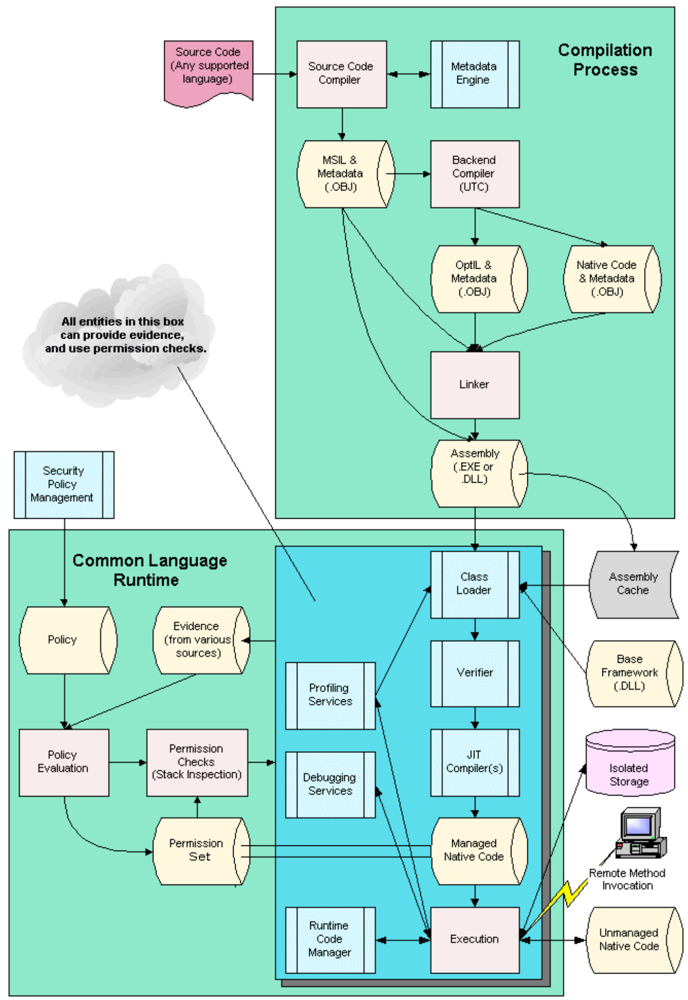


KRvW Associates

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 step: Attack resistance

- Identify general flaws
  - Non-compliance
  - Where guidelines are not followed
- Map applicable attack patterns
- Identify risks in architecture
- Consider known attacks against similar technologies

- Attack Patterns
  - Pattern language
  - Database of patterns
  - Actual flaws from clients
- Exploit Graphs
  - Ease mitigation
  - Demonstrate attack paths
- Secure design

Example flaws from experience…

- Transparent authentication token generation/management
- Misuse of cryptographic primitives
- Easily subverted guard components, broken encapsulation
- Cross-language trust/privilege issues

TP2 step: Ambiguity analysis

- Consider implications of design
- Generate separate arch. diagrams
- Unify understanding
  - Uncover ambiguity
  - Identify downstream difficulty (traceability)
  - Unravel convolution
- Apprenticeship model
- Use system, technology experts
  - Win32 knowledge
  - JVM/managed code
  - Language/compiler knowledge
- Previous experience

Example flaws from experience…

- Protocol, authentication problems
- Javacard applet firewall, inner class issues, instantiation in C#
- Type safety and type confusion
- Password retrieval, fitness and strength

TP2 step: Weakness analysis

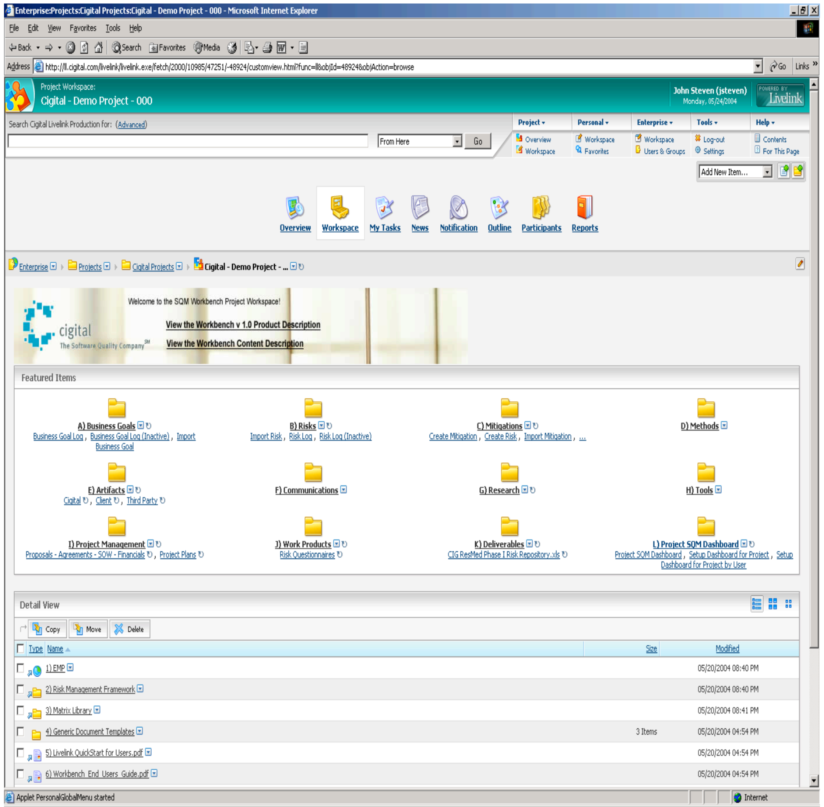
- Consider systemic flaws
  - COTS
  - Frameworks
  - Network topology
  - Platform
- Identify services
- Map weaknesses to assumptions
- Experience base
  - Assessments of COTS and platforms
- Attack patterns
- Other resources
  - Mailing lists
  - Product documentation

Example flaws from experience…

- Browser and other VM sandboxing failures
- Insecure service provision: RMI, COM, etc.
- Debug (or other operational) interfaces
- Unused (but privileged) product "features"
- Interposition attacks: DLLs, library paths, client spoofing

TP2: Keep track of risks

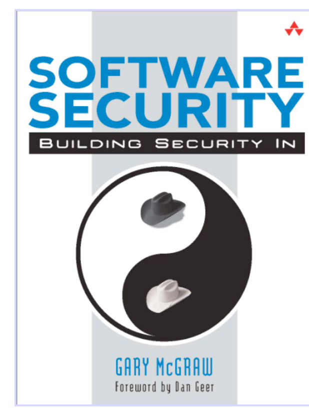
- The key to making a process like the one we described work is to KEEP TRACK of what you've found
- Use excel if you have nothing better
- Cigital uses the Cigital workbench
- Remember the RMF? Use it!



cigital KRvW Associates

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
- Not a silver bullet!



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

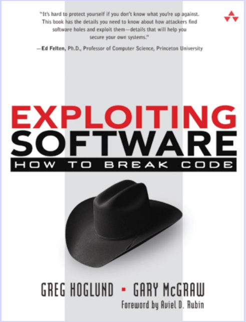
- Identify areas of potential risk in the system
  - Requirements
  - Design
  - Architecture
- Use abuse cases to drive testing according to risk
- Build attack and exploit scenarios based on identified risks
- Test risk conditions explicitly

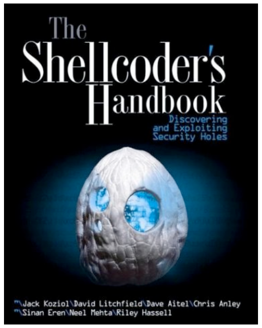
- Example: Overly complex object-sharing system in Java Card

KRvW Associates

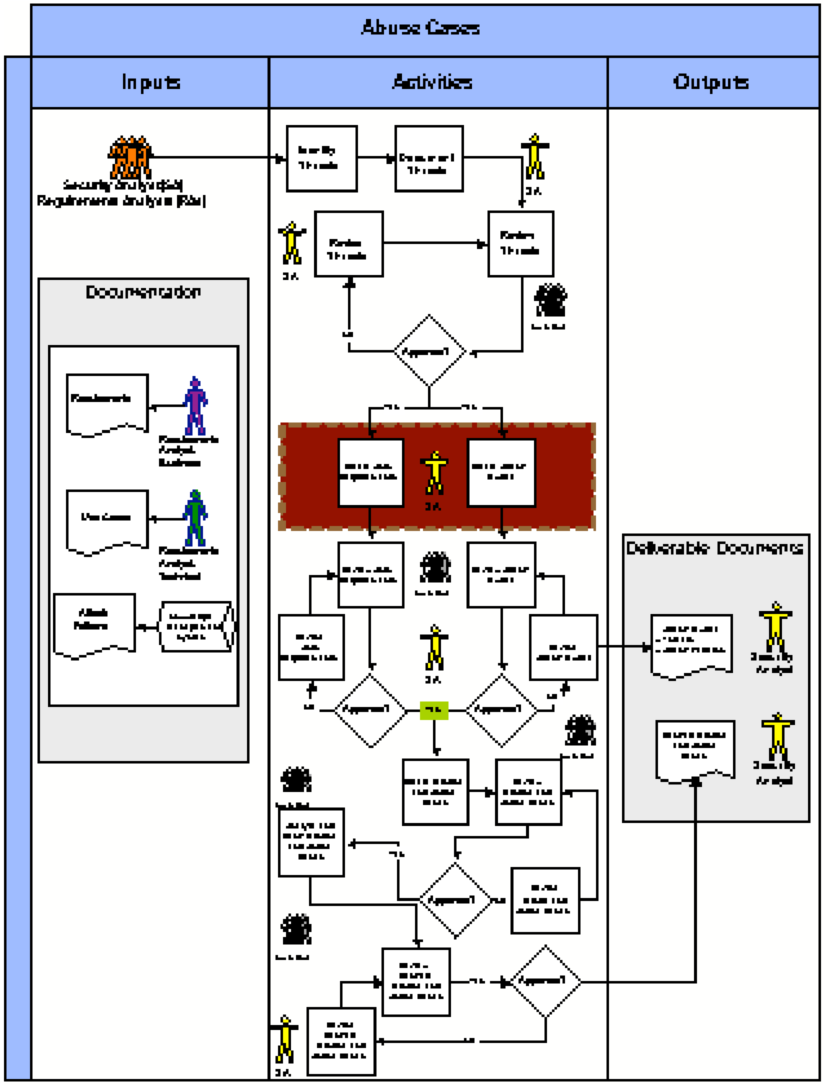
Touchpoint 5: Abuse cases

- 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



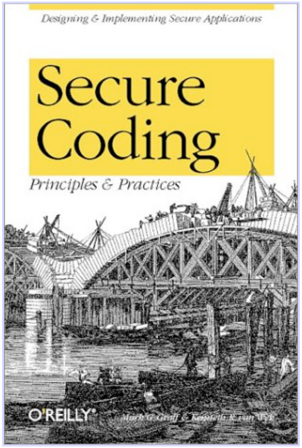
- Starting with attack patterns, requirements and use cases
- Identify anti-requirements
- Build an attack model
- Determine misuse and abuse cases

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

Touchpoint 7: Security operations

- Use your resources!
- Network security people know an awful lot about real attacks
- Involve knowledgeable security people in as many touchpoint activities as possible
- Fine tune the deployed environment to the specific needs of your application
  - "Standard OS build" process is not enough



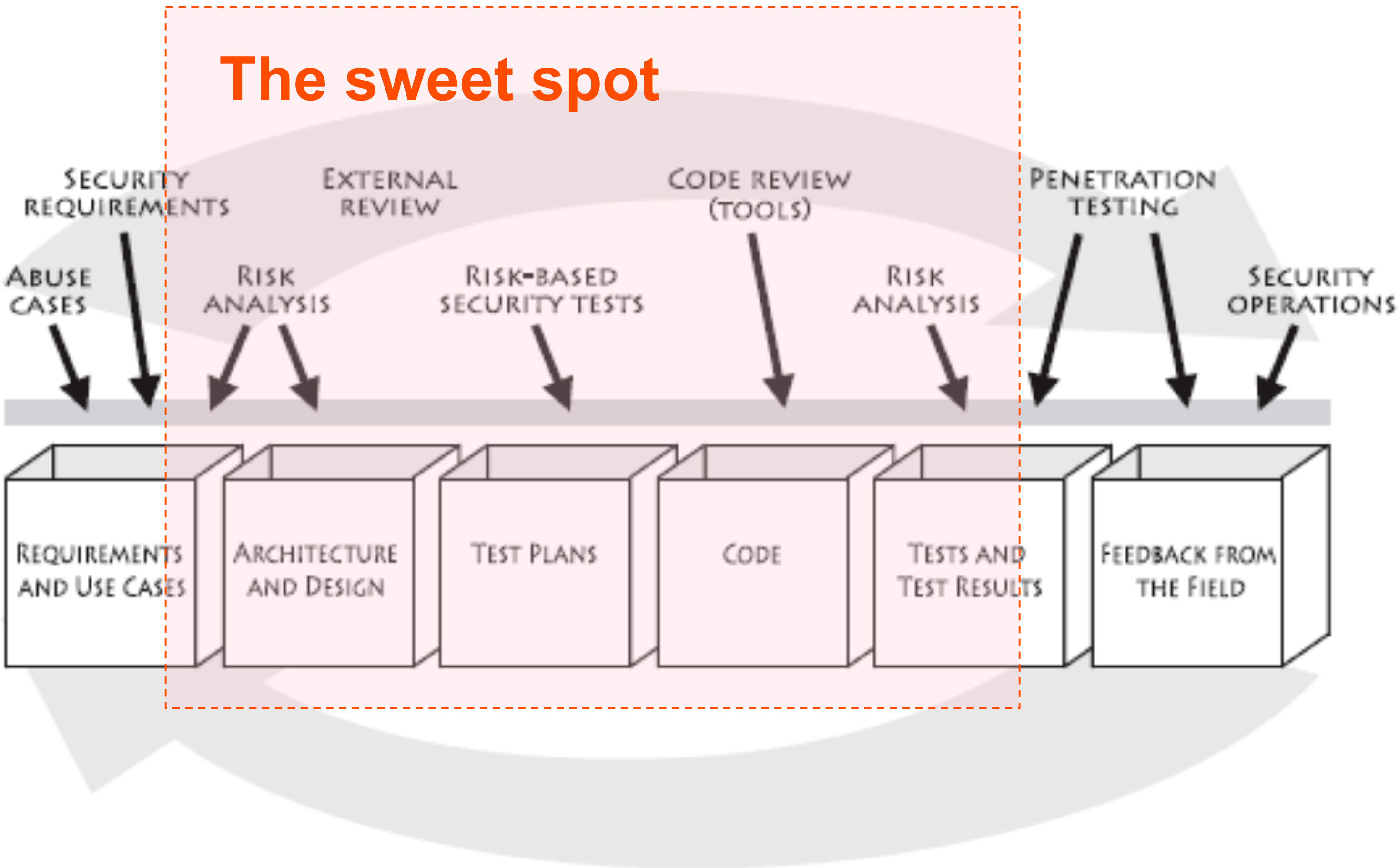
KRvW Associates

cigital

Always: External review

- Having outside eyes look at your system is essential
  - Designers and developers naturally get blinders on
  - External just means outside of the project
  - This is knowledge intensive
- Outside eyes make it easier to "assume nothing"
  - Find assumptions, make them go away
- Red teaming is a weak form of external review
  - Penetration testing is too often driven by outside→in perspective
  - External review must include architecture analysis
- Security expertise and experience really helps

Software security touchpoints



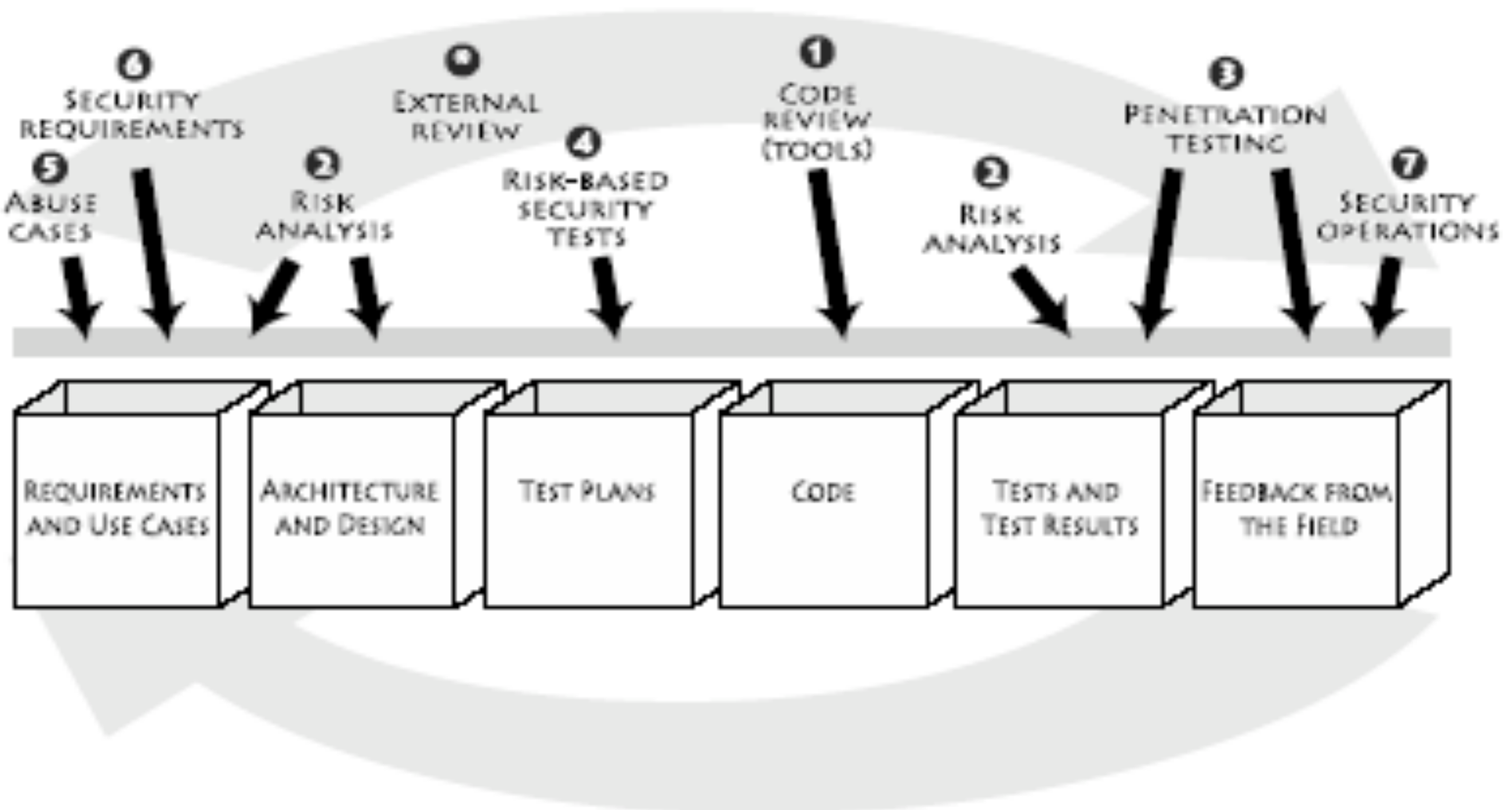
Reprise

Best practices reprise

- These best practices should be applied throughout the lifecycle
- Tendency is to "start right" (penetration testing) and declare victory
  - Not cost effective
  - Hard to fix problems
- Start as far to the left as possible

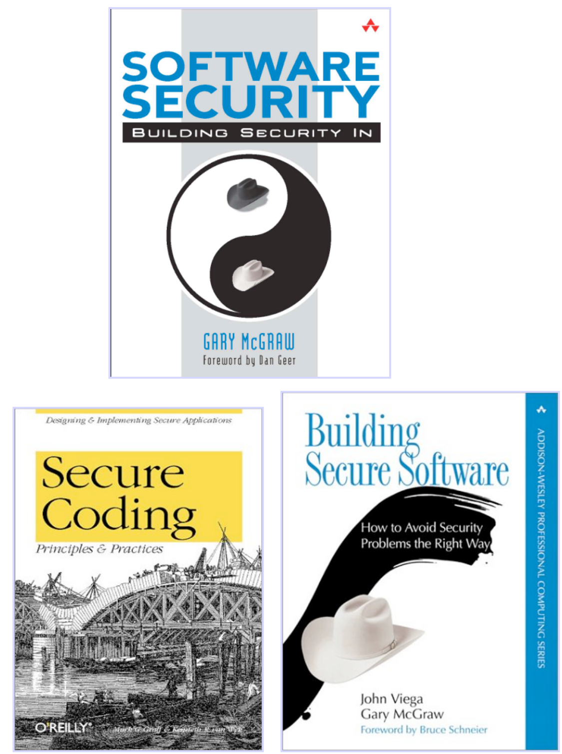
- Abuse cases
- Security requirements analysis
- Architectural risk analysis
- Risk analysis at design
- External review
- Test planning based on risks
- Security testing (malicious tests)
- Code review with static analysis tools

Adopting the touchpoints



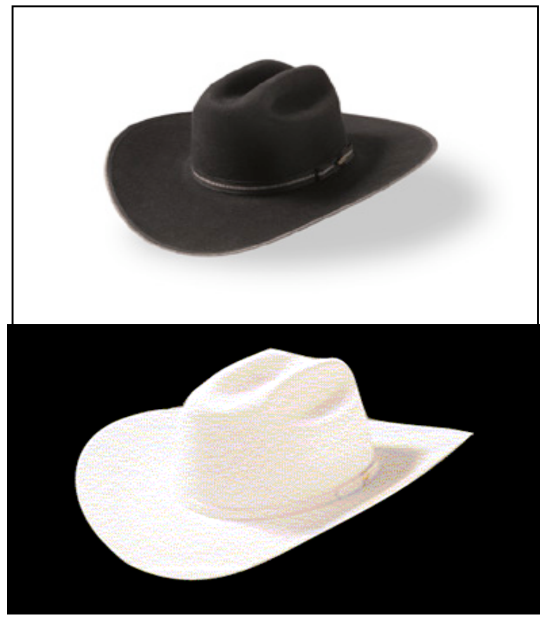
KRvW Associates

Seven pernicious kingdoms



Outline

- Classic Pitfalls
- Seven Kingdoms
- Static Analysis and Code Review



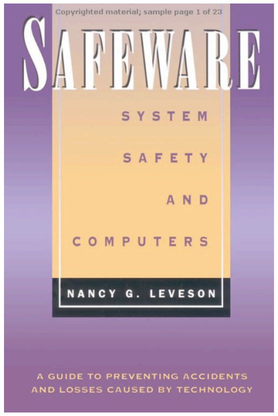
Classic Pitfalls

Learn from history

Those who cannot remember the past are condemned to repeat it.
-- Santayana

- Other engineering disciplines overcome failures by collecting failure data and analyzing failures for commonalty that could lead to avoidance of that kind of failure in the future

- Failure data in software is generally considered proprietary
  - Most failure data from product development is not available for open research



cigital KRvW Associates

Same old mistakes

- By understanding software security risks, developers can avoid them when writing their own code
- Learn by considering examples
  - Configuring applications
  - Scripts
  - Errors
  - Design flaws
- Many of the same problems crop up year after year
- Basic science to classify and categorize these problems has yet to be done
  - Bugs: implementation
  - Flaws: higher-level

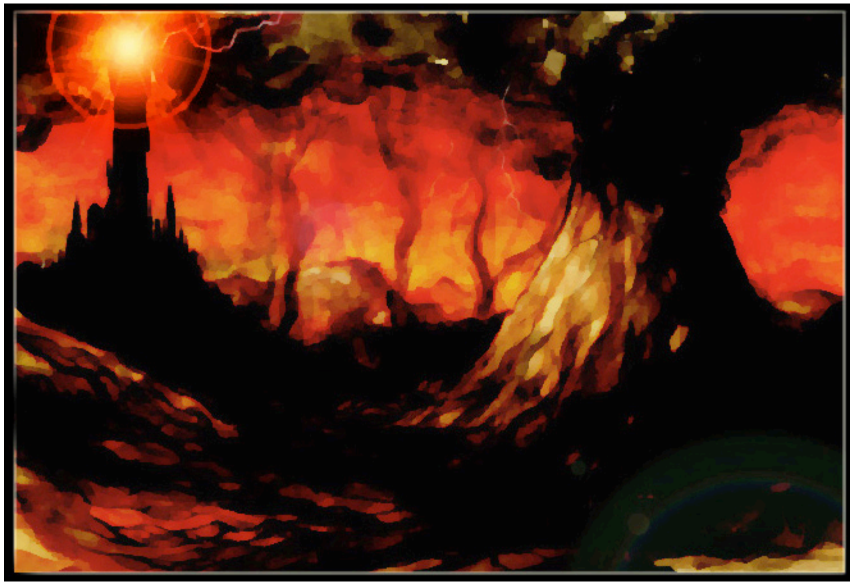
Seven Kingdoms

Seven pernicious kingdoms

- Input validation and representation
- API abuse
- Security features
- Time and state
- Error handling
- Code quality
- Encapsulation
- Environment

KRvW Associates

1. Input Validation and Representation



KRvW Associates

cigital

Pernicious kingdom one

- **Input Validation and Representation**
  - Problems due to metacharacters, alternate encodings, numeric representations, and trusting input
  - Example: **Buffer Overflow** phylum

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

The number one coding snafu

- “Scrubbing” user input pitfalls to avoid
  - SQL Insertion
  - Cross-Site Scripting (XSS)
  - Format string vulnerabilities
  - Integer overflows
  - Buffer overflows
    - Not a security problem per se in Java due to strict variable range enforcement
- Not a trivial issue, as complexity and subtlety abounds

KRvW Associates

cigital

Buffer overflows

- Pervasive problem, primarily in C and other non-type-safe (sometimes called “unmanaged”) code
- Responsible for huge percentage of reported vulnerabilities today
- Exploited by some of the most damaging worms
  - 1988: Morris worm
  - 2001: Code Red
  - Others: Slammer, Blaster, Sasser, Zotob

Buffer overflow causes

- String manipulation libraries
  - Flawed libc functions: strcpy, strcat, …
  - Multibyte characters
  - Null termination errors
- Off by one errors
- Array manipulation
- Pointer arithmetic
- Others
  - Format strings
  - Integer overflow
- These all relate to reliability as well as security

KRvW Associates

cigital

Historic example: the Morris worm of 1988

- Cornell grad student Robert Tappan Morris's "Internet worm" exploited a bug in the (then) popular BSD fingerd daemon
- The vulnerable fingerd contained the following code:

```
char line[512];
line[0] = "\0";
gets(line);
```

- 512 characters should be enough, shouldn't it?

Same issue in C++

- Although the gets() function was known to be horribly flawed for years, the same mistake was made in C++

```
char buf[BUFSIZE];
cin >> (buf);
```

- Those cows come home yet?

Problematic function: *strcpy()*

- Although not quite as bad as *gets(),* it's darn close

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

Problematic function: *sprintf()*

- As with the likes of *strcpy(),* you can use *sprintf()* safely, but it isn't easy
- Is the following good or bad? (we already know it's ugly)

```
char buf[42];
sprintf(buf, “Val1=%.8s Val2=%.8s Val3=%.8s”,
val1, val2, val3);
```

What's the deal with the *n* functions?

- Although the bounded versions of string functions, like strncpy(), are better, there's still room for silly mistakes
- Truncation can cause odd behavior
- Example: One simple mistake is to bound the data to the src buffer, as in this example from MSDN

```
int main(int argc, char *argv[]) {
…
char DirSpec[MAX_PATH + 1];
printf("Target dir is %s.\n", argv[1]);
strncpy(DirSpec, argv[1], strlen(argv[1])+1);
```

Problematic function: *strncat()*

- Example: The *strncat()* function is misleading because it doesn't accept a bound on the total size of the destination buffer, but rather the remaining space available in the destination buffer

```
char* buf[512];
strcpy(buf, “The argument is”);
strncat(buf, argv[1], 512);
```

KRvW Associates

Format string vulnerabilities

- Format string vulnerabilities occur when an attacker can control a format string
- Although not technically buffer overflows, they almost invariably lead to read/writes outside a buffer's bounds
  - Including execution of arbitrary code placed on stack by the attacker
- First seen around 1999, but in its first full year resulted in many root exploits
  - Wu-ftpd 2.*
  - Linux rpc.statd
  - Qualcomm qpopper 2.53
  - Apache + PHP3
  - BSD chpass
  - OpenBSD fstat

KRvW Associates

cigital

Format strings: root cause

- Misuse of formatting functions
  - A programmer wants to print a string
  - Which is correct?

```
printf("%s", string);
printf(str);
```

- If an attacker can control the format string, then %n can be used to write arbitrary values anywhere in memory
- Exploits then work the same way as traditional buffer overflows
  - Overwrite return address
  - Function pointer
  - Other important values

KRvW Associates

Example: *wuftpd 2.6.0*

- Widely publicized format string vulnerability occurs in the *vreply()* function, which looks much like this

```
while (fgets(buf, sizeof buf, f)){
  lreply(200, buf);
  …
}

void lreply(int n, char *fmt, …) {
  char buf[BUFSIZ];
  …
  vsnprintf(buf, sizeof buf, fmt, ap);
  …
}
```

KRvW Associates

SQL insertion

- Problem can exist when Java or middle-tier code interacts with back-end SQL-based database
- User inputs must be pedantically screened for SQL code
  - White space, quotes, etc., are indicators
- Regular Expression (regex) filtering is key

Problem: SQL insertion

- Can enable attacker to execute arbitrary SQL commands on back-end database
- PHP/SQL Example:
  - PHP code inputs USERNAME and PASSWORD and passes to SQL back-end
  - USERNAME is entered as bob
  - PASSWORD is entered as ’ or USERNAME=‘bob
  - Back-end executes Select ID from USERS where USERNAME=‘bob’ and PASSWORD=‘’ or USERNAME=‘bob’
  - Instead of Select ID from USERS where USERNAME=‘bob’ and PASSWORD=‘password’

KRvW Associates

cigital

SQL insertion - example

```
Pattern pattern;
Matcher matcher;
String regex = "\\W";

boolean userOkay = true;
boolean passOkay = true;

// check username field

pattern = Pattern.compile(regex);
matcher = pattern.matcher(userField.getText());

if (matcher.find()) {
   userOkay = false;
}
```

[1]

[2]

[1] Begin by defining the regular expression itself

[2] Compile the regex and apply it to the string in question

(Even better: use PreparedStatement)

Complications in parsing input

- Lots of things can make parsing through input fields complex
- Whitelisting and blacklisting approaches
  - Assume input is dangerous until it is proven to be safe
- Internationalization
  - Unicode can be used to obfuscate SQL insertion, XSS, etc.
  - /etc/passwd—seems easy enough to parse, right?

cigital

KRvW Associates

Unicode - example

```
Pattern pattern;
Matcher matcher;
String regex = "[^(a-zA-Z0-9_\\-\\.)]";
String unicodeRegex = "[[u002f]u002F]";
String file = fileField.getText();

// check filename field

pattern = Pattern.compile(unicodeRegex);
matcher = pattern.matcher(file);

if (matcher.find()) {
   unicodeFound = true;
}

pattern = Pattern.compile(regex);
matcher = pattern.matcher(file);

if (matcher.find()) {
   filenameOkay = false;
}
```

[1]

[2]

[1] Define a regex to search for unicode characters (u002f = "\")

[2] Check for specified unicode characters in the file name

Good practice: take care with config files

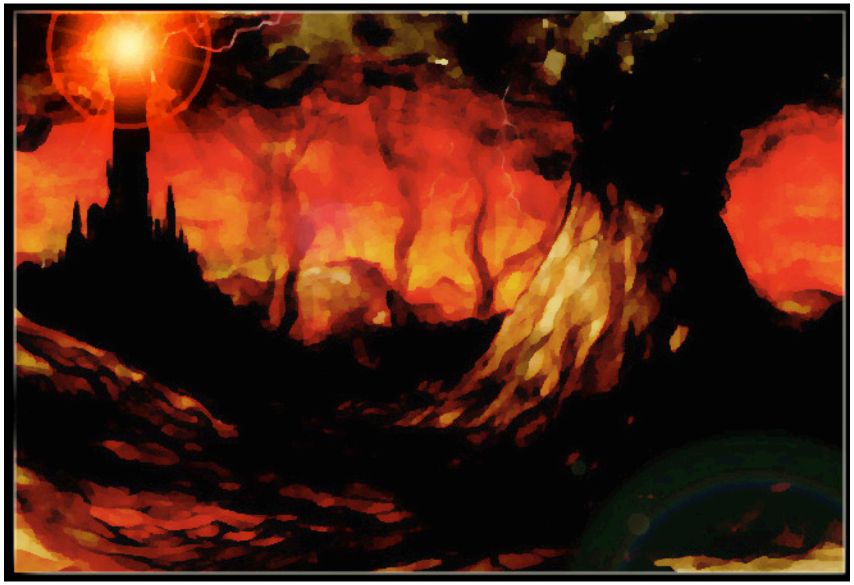
- Check configuration files
  - Can be ripe target for attackers
  - Verify read/write controls are safe
  - Verify data content before acting
- User inputs
  - Command line parameters and desktop icons
  - URLs
  - Assume it to be harmful until proven otherwise
- Consider also where other user inputs can come from
  - Signals, registry keys, mouse actions, and so on…

KRvW Associates

Phyla: Input validation and representation

- Buffer Overflow
- Command Injection
- Cross-Site Scripting
- Format String
- HTTP Response Splitting
- Illegal Pointer Value
- Integer Overflow
- Log Forging
- Path Traversal
- Process Control
- Resource Injection
- Setting Manipulation
- SQL Injection
- String Termination Error
- Struts: Duplicate Validation Forms
- Struts: Erroneous validate() Method
- Struts: Form Bean Does Not Extend Validation Class
- Struts: Form Field without Validator
- Struts: Plug-in Framework Not in Use
- Struts: Unused Validation Form
- Struts: Unvalidated Action Form
- Struts: Validator Turned Off
- Struts: Validator without Form Field
- Unsafe JNI
- Unsafe Reflection
- XML Validation

2. API Abuse



KRvW Associates

Pernicious kingdom two

- **API Abuse**
  - A caller fails to honor the contract between the caller and the callee
  - Dangerous function, Unchecked Return Value, and others
  - Example: **Often Misused: Authentication** phylum

```
String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getCanonicalHostName().endsWith(
            "trustme.com")) {
            trusted = true;
            // Relying on DNS lookup for recognizing trusted hosts
}
```

Comparing Java classes

- Never make a decision based on the name of a class
  - A program may treat two classes the same when they actually differ
  - Class names are trivial to forge or substitute
  - At the very least, verify that the name being checked is within the current classloader

cigital

KRvW Associates

Example: *readlink()*

- Abuse of *readlink(),* which although it fills a string buffer does not null terminate the buffer

```
readlink(path, buf, MAXPATH);
int length = strlen(buf);
```

- The value returned from *strlen()* is likely to be incorrect – perhaps wildly so – and may even result in a buffer overflow or other runtime erratic behavior

Example: SYN flood

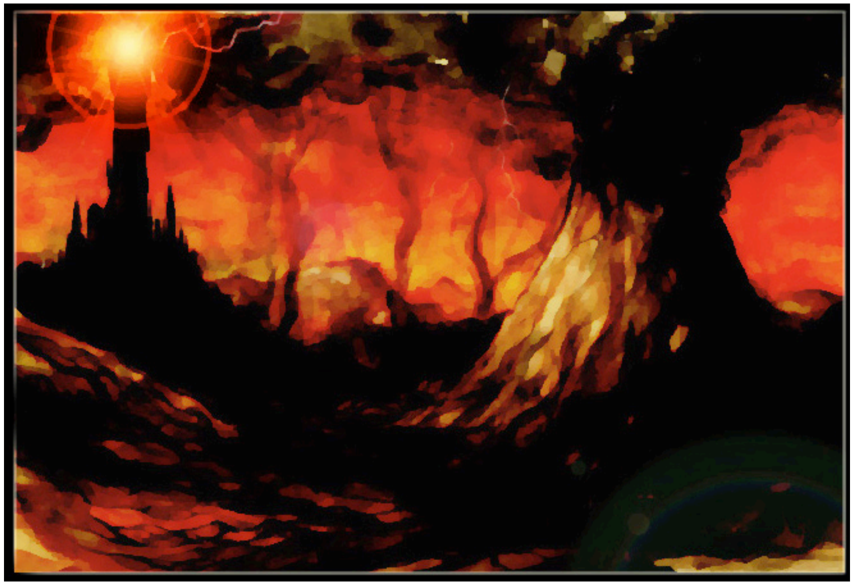
- Attacker initiates, but does not complete TCP session opening protocol
- Victim's TCP stack is left in a wait state
- Attacker repeats until victim's resource pool is saturated
- Victim is now effectively off the net – DoS
- Why would someone want to do this?

KRvW Associates

Phyla: API abuse

- Dangerous Function
- Directory Restriction
- Heap Inspection
- J2EE Bad Practices: getConnection()
- J2EE Bad Practices: Sockets
- Often Misused: Authentication
- Often Misused: Exception Handling
- Often Misused: Path Manipulation
- Often Misused: Privilege Management
- Often Misused: String Manipulation
- Unchecked Return Value

3. Security Features



KRvW Associates

cigital

Pernicious kingdom three

- **Security Features**
  - Poorly handled authentication, access control, confidentiality, cryptography, and privilege management
  - Insecure Randomness, Password Management, Privacy Violation, and others
  - Example: **Privacy Violation** phylum

```
id = getId();
pass = getPassword();
type = getType();
tstamp = getTimestamp();
...                                   // Private info leaking into a log file
dbmsLog.log(id+“:”+pass+“:”+type+“:”+tstamp);
```

Signing JAR files

- Signing JAR files can be dangerous
- A signed JAR might be trusted more than is warranted
  - But, signed JARs also can be useful
  - Authentication and integrity checking
- If you must sign, put your signed classes into one JAR file, all by themselves

KRvW Associates

cigital

Storing secrets

- "Hard coding" sensitive information in source code is dangerous
  - Class file can be viewed
  - Class de-compilers (e.g., jode) can expose

KRvW Associates

Storing secrets - example

```
public class SecretInfo {

   public static void main(String[] args) {

      String username = ”bassethoundsruletheworld”;
      String password = ”t0ps3cr3t!”;

      /* do something with username and password */

   }
}
```

[1]

[1] Two strings are defined that contain sensitive information

Storing secrets – example (cont'd)

```
/* SecretInfo - Decompiled by JODE
 * Visit http://jode.sourceforge.net/
 */

public class SecretInfo
{
    public static void main(String[] strings) {
        String string = "bassethoundsruletheworld";
        String string_0_ = "t0ps3cr3t!";
    }
}
```

[2]

[2] Using a decompiler, the values of both strings can be retrieved from a compiled .class file

KRvW Associates

cigital

Privilege handling

- Don’t forget the principle of least privilege
  - Avoid privileged code if at all possible
- Tips
  - Design things so that program does not need privileges
  - Develop code without privileges enabled
- Did you know?
  - 90% of Windows software can’t be installed without Administrator privileges
  - 70% can’t be run without Administrator privileges
  - 10,000 lemmings can’t be wrong!

KRvW Associates

Why privileges are needed

- Interact directly with hardware
- Other shared resources
  - Network ports, config, registry
- Alter OS behavior
- Override file system protections
  - Install new files
  - Update protected files
  - Access files that belong to other users

KRvW Associates

Case study: *lpr*

- Redhat lpr (Oct 1999)
- Setuid root in order to talk to printer device

```
int fd;
for (int i=1; i < argc; i++) {
   /* first make sure that the user can read the
   file, then open it */
   if (!access(argv[i], O_RDONLY)) {
        fd = open(argv[i], O_RDONLY);
   }
   print(fd);
}
```

KRvW Associates

cigital

Case study: *lpr*

- File access race condition! Fix:

```
int fd;
for (int i=1; i < argc; i++) {
  int uid = getuid(); int gid = getgid();
  int original_euid = geteuid();
  int original_egid = getegid();
  seteuid(uid); setegid(gid);
  fd = open(argv[i], O_RDONLY);
  seteuid(original_euid);
  setegid(original_egid);
}
print(fd);
```

Case study: lpr

- Do you think that it's fixed now?
- No! *seteuid()* return value ignored
- No one expects *seteuid()* to fail since we're root

- POSIX capabilities vulnerability (June 2000)
- Attackers can cause *seteuid()* call to fail

- Not so simple, is it?

KRvW Associates

cigital

When are random numbers needed?

- Some numbers need to be cryptographically secure
  - Crypto applications
  - Generated passwords
  - Port randomization
  - External unique identifiers such as session tokens
  - Discount codes
- Some do not
  - Monte Carlo simulation systems
  - Internal unique identifiers

KRvW Associates

Example: Security depends on unpredictability

- The following code generates "unique" identifiers for online users who make a purchase. Because *lrand48()* is a statistical PRNG, it is easy for an attacker to predict

```
char* CreateReceiptURL() {
    int num; time_t t1;
    char *URL = (char*) malloc(MAX_URL);
    if (URL) {
            (void) time(&t1);
            srand48((long) t1)
            sprintf(URL, "%s%d%s, http://test.com,
                    lrand48(), ".html");
    }
    return URL; }
```

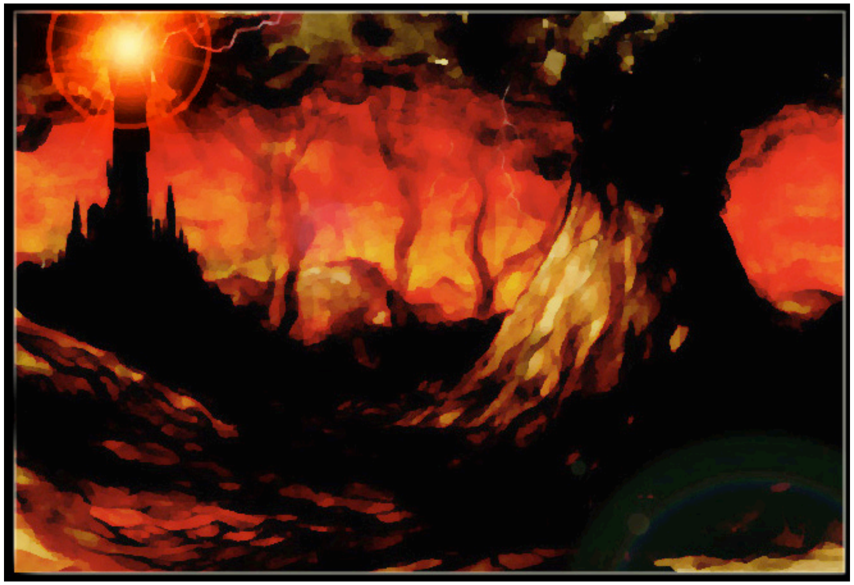
Choosing a PRNG

- Hardware can be good, if available
- OS may provide good random sources
  - /dev/urandom is almost always the right choice for user apps
  - /dev/random blocks and my be exhausted since shared
- Current state of the art
  - Fortuna (described in Schneier's Practical Cryptography)
  - Implementations
    - Win C++ (http://www.citadelsoftware.ca/fortuna/Fortuna.htm)
    - Linux /dev/urandom driver (http://jlcooke.ca/random)
- Freebie in Microsoft-friendly code
  - CryptoGenRandom()

Phyla: Security features

- Insecure Randomness
- Least Privilege Violation
- Missing Access Control
- Password Management
- Password Management: Empty Password in Configuration File
- Password Management: Hard-Coded Password
- Password Management: Password in Configuration File
- Password Management: Weak Cryptography
- Privacy Violation

4. Time and State



KRvW Associates

cigital

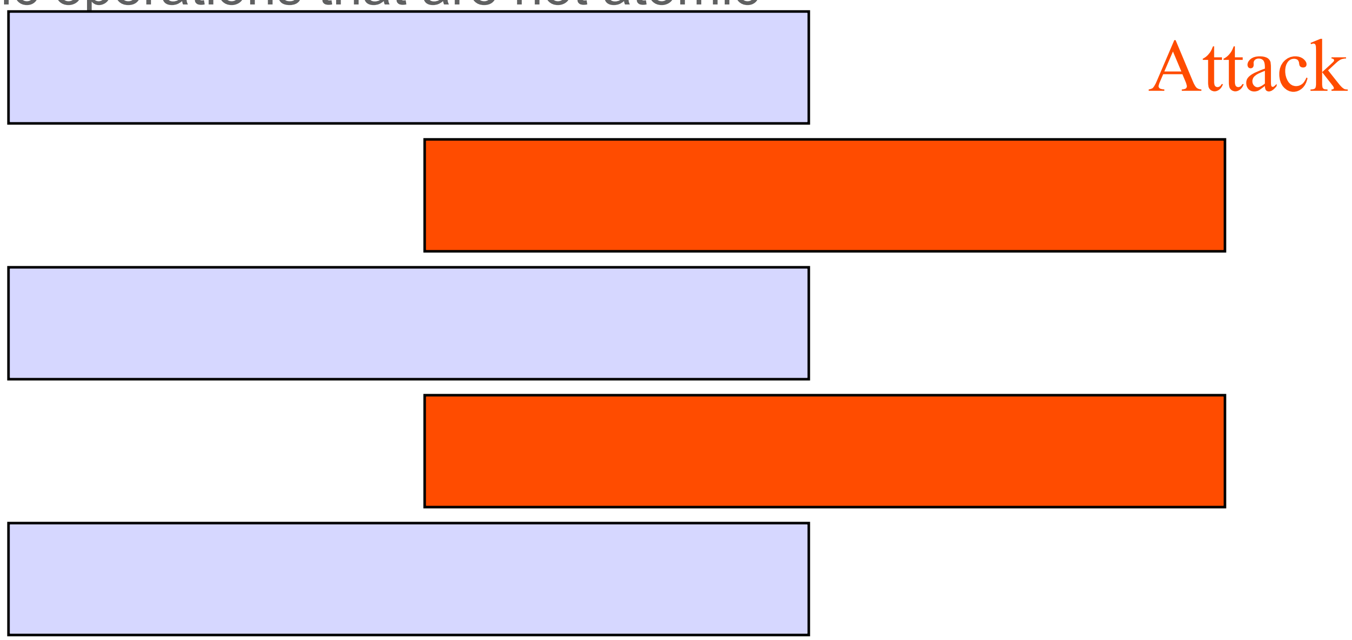
Pernicious kingdom four

- **Time and State**
  - Unexpected interactions between threads, processes, time, and information that happen through shared state: semaphores, variables, file system, etc.
  - File Access Race Condition TOCTOU, Deadlock, and others
  - Example: **Session Fixation** phylum

```
private void auth(LoginContext lc,
HttpSession session)
                    throws LoginException {
          …                          // No call to session.invalidate()
          lc.login();
          …
}
```

RISK: Race condition

- Time makes all the difference
- Atomic operations that are not atomic



A simple (broken) Java servlet

```
import java.io.*;
import java.servlet.*;
import java.servlet.http.*;
public class Counter extends HttpServlet{
  int count = 0;
  public void doGet(HttpServletRequest in, HttpServletResponse out)
                              throws ServletException, IOException {
     out.setContentType("text/plain");
     Printwriter p = out.getWriter();
     count++;
     p.println(count + " hits so far!");
  }
}
```

Race condition

KRvW Associates

cigital

A simple (fixed) Java servlet

```
import java.io.*;
import java.servlet.*;
import java.servlet.http.*;
public class Counter extends HttpServlet{
  int count = 0;
  public synchronized void
      doGet(HttpServletRequest in, HttpServletResponse out)
                      throws ServletException, IOException {
    out.setContentType("text/plain");
    Printwriter p = out.getWriter();
    count++;
    p.println(count + " hits so far!");
  }
}
```

TOCTOU

- Race conditions on Unix files are famous
- Passwd example
  - Step 1: open file and read it in
  - Step 2: create and open “ptmp” in same directory
  - Step 3: open password file again, copying unchanged contents into ptmp while updating
  - Step 4: Close both password file and ptmp, then name ptmp the password file
- If an attacker makes use of unix’s linking facility, an attack is possible
- Change the system state in a subtle way in order to cause the system to do something dangerous

Threads (J2EE)

- Thread management in a web application is prohibited by the J2EE standard
- Difficult and likely to produce unpredictable results such as deadlocks, race conditions and other synchronization errors
- Rather than managing threads directly, use standards such as message driven beans and EJB timer service provided by the container

KRvW Associates

cigital

Good practice: watch out for web content

- Web data
  - Watch out for data in hidden fields
  - Even though it is within page, user can still alter
- Web cookies
  - Can also be manipulated by user
  - Classic example: changing customer ID or shopping cart price totals
- State data must be protected
  - Encryption is commonly used
  - Verify that no data has been tampered with

Serialization

- Largely fixed in latest JDK versions
  - Previous default allowed serialization
  - New default requires class to implement Serializable interface
- When serialized, an object is written to disk directly, including internal memory
- If you must make something serializable, declare private data transient

KRvW Associates

cigital

Serialization - example

```
public class SerializableClass implements Serializable {

   private transient String secret = "Cats are smarter than dogs.";
   private String notSecret = "Dogs drool everywhere.";

   public SerializableClass() { }
```

[1]

```
notSecrett^@^RLjava/lang/String;xpt^@^VDogs drool everywhere.x
```

[2]

[1] A serializable class that defines two private strings

[2] Output of the serialized class when read by a simple text editor (note that the transient string is not displayed)

KRvW Associates

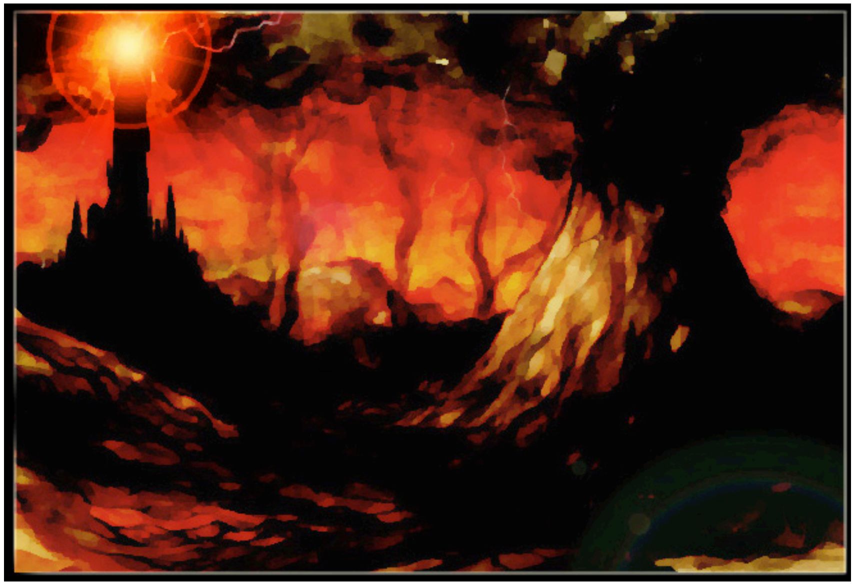
Phyla: Time and state

- Deadlock
- Failure to Begin a New Session upon Authentication
- File Access Race Condition: TOCTOU
- Insecure Temporary File
- J2EE Bad Practices: System.exit()
- J2EE Bad Practices: Threads
- Signal Handling Race Conditions

KRvW Associates

cigital

5. Error Handling



Pernicious kingdom five

- **Error handling**
  - Both poor error handling and generation of errors that either leak information or are difficult to handle
  - Empty Catch Block, Overly-Broad Catch Block, and others
  - Example: **Empty Catch Block** phylum

```
try {
        attempToDoSomethingImportant();
}
catch (ImportantException e) {
                        // How should this exception be handled?
}
```

KRvW Associates
cigital

Error handling: the problem

- Ignoring exceptional conditions and their ramifications
  - A symptom: failure to think about what could go wrong
  - An outcome: leads to inconsistent and unexpected program state
- Unchecked return values
- Exception handling
- Signal handling

Legacy problems

- Many standards to choose from
  - *fork()* – 0 == success
  - *strtol()* – 0 == failure
  - *strcmp()* – 0 == true
  - *issetugid()* – 0 == false
  - *fork() -- >0* == success
- And this doesn’t even address multithreaded apps
- Always check those reference manuals before assuming!

Allocation problems

- Failure to check for memory allocation failure

```
buf = (char*) malloc(req_size);
strncpy(buf, xfer, req_size);
```

- What could go wrong?
- Bad for at least three reasons
  - No opportunity to recover
  - Impossible to exit gracefully
  - No opportunity of collecting diagnostic information

Missing error handling (J2EE)

- Un-handled exceptions can provide an attacker with potentially dangerous information, such as an SQL query string, the type of database being used, or application version numbers
- Web applications should always specify default error pages and handle standard HTTP error codes

KRvW Associates

cigital

Missing error handling - example

Include the following entries in the *web.xml* file to specify default error pages...

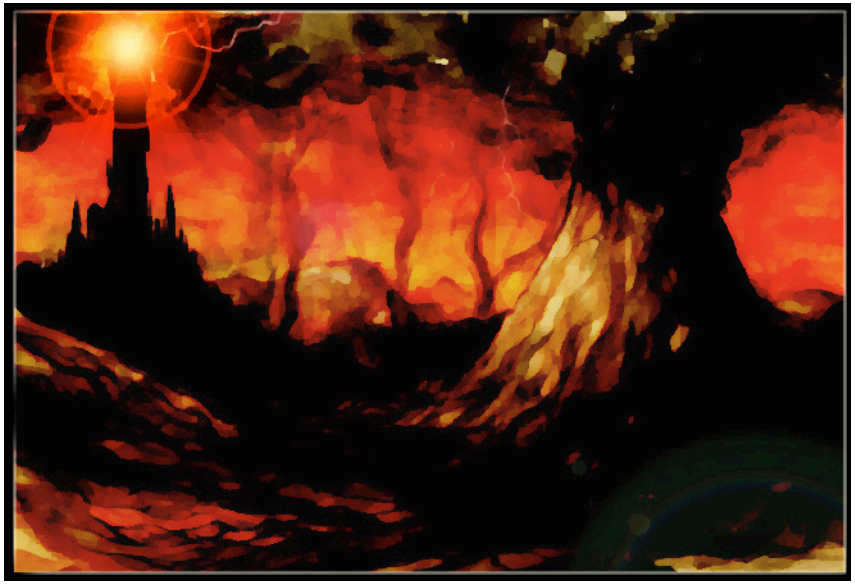
```
<error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/error.jsp</location>
</error-page>
<error-page>
    <error-code>404</error-code>
    <location>/error.jsp</location>
</error-page>
<error-page>
    <error-code>500</error-code>
    <location>/error.jsp</location>
</error-page>
```

Phyla: Error handling

- Catch NullPointerException
- Empty Catch Block
- Overly Broad Catch Block
- Overly Broad Throws Declaration
- Unchecked Return Value

KRvW Associates

6. Code Quality



KRvW Associates

cigital

Pernicious kingdom six

- **Code Quality**
  - Poor code quality indicates security problems likely
  - Memory Leak, Null Dereference, Uninitialized Variable, and others
  - Example: **Attribute Stored in HttpSession Might Not Be Serializable** phylum

```
public class MyAttribute {        // Not Serializable
    …
}

public void add (HttpSession s, MyAttribute a) {
    session.setAttribute("attribute", a);
}
```

Code quality issues

- All have the potential to allow denial of service attacks
- More often leads to unpredictable behavior
  - Exceedingly difficult to test for
  - Read “The Bug” by Ellen Ullman
- Unpredictable behavior is the friend of the attacker

KRvW Associates

cigital

Example: memory leak

- Find easy cases with tools like Purify
- Hard cases can be dynamic flow driven and really tough to find
- Common causes: error conditions, confusion over responsibility

```
char* getBlock(int fd) {
    char* buf = (char*( malloc(BLOCK_SIZE);
    if (!buf) {
        return NULL;
    }
    if (read(fd, buf, BLOCK_SIZE) != BLOCK_SIZE) {
        return NULL;
    }
}
return buf;
```

KRvW Associates

cigital

Example: use after free

```
char* ptr = (char*) malloc(SIZM);
…
if (err) {
   abrt = 1;
   free(ptr);
}
…
if (abrt) {
   logError("operation aborted before commit", ptr);
}
```

- And sometimes it works!
- Memory may be re-allocated by the time the error is logged

KRvW Associates

cigital

Example: double free

- Most often causes a crash, but can result in buffer overflow under rare circumstances

```
char* ptr = (char*) malloc(SIZM);
…
if (abrt) [
  free(ptr);
}
…
free(ptr);
```

Portability problems

- Internal buffer overflows in some implementations of *getopt()*
  - Avoid with good input validation
- In many cases, you cannot avoid problems
- Examples
  - *vfork()* behavior varies by platform
  - *strcmpi()* is not defined on many UNIX systems
  - *memmem()* problematic due to changes between versions whereby order of the arguments is reversed

KRvW
Associates

Returning mutable objects

- Mutable objects are references to specific locations in memory
  - The most common example is an array
- Returning a mutable object to malicious code enables an attacker to modify the contents of memory pointed to by the object

KRvW Associates

cigital

Returning mutable objects - example

```
public static void main(String[] args) {

   EmployeeInfo info = new EmployeeInfo("Steve", "Dallas", "Red");

   // modify the employee name (because we can)

   String[] localname = info.getName();         [1]

   // Here's where the danger is; the employee's name is being
   // stored into an array.  Since that is mutable, it can be changed
   // by an attacker, even if it is declared private.

   localname[0] = "L33t";
   localname[1] = "H4x0r";          [2]

   String[] name = info.getName();

   System.out.println("\nEmployee Name: " + name[0] + " " +
      name[1]);

   // will output "Employee Name: L33t H4x0r"
```

[1] Store a reference to a mutable array in a local context

[2] Modify the original array by changing the local array

KRvW Associates

cigital

Storing mutable objects

- In a similar way as returning mutable objects, storing mutable objects passed to your code can lead to problems
  - Especially if you act on the returned object(s)
- See example—MutableStorage

KRvW Associates

cigital

Public static final mutable objects

- Public static final mutable objects can still be modified, because only the reference to the object is constant

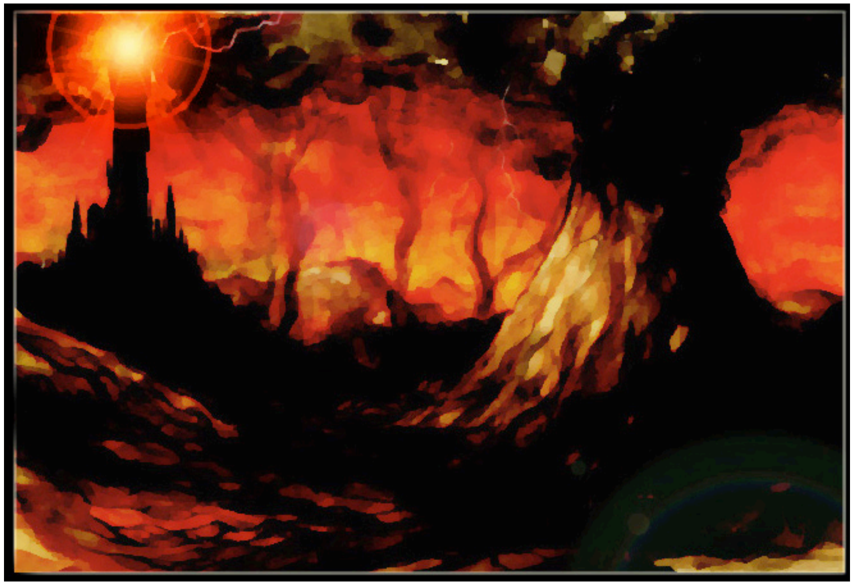
Java Initialization

- Java is supposed to initialize new variables cleanly, but it's still good practice to do so manually
  - Apart from anything else, this is just a good housekeeping

Phyla: Code quality

- Double Free
- Inconsistent Implementations
- Memory Leak
- Null Dereference
- Obsolete
- Undefined Behavior
- Uninitialized Variable
- Unreleased Resource
- Use After Free

7. Encapsulation



Pernicious kingdom seven

- Encapsulation
  - Violation of boundaries between software components with various trust level
  - System Information Leak, Trust Boundary Violation, Mobile Code: Non-Final Public Field, and others
  - Example: **Field Assignment in a Servlet** phylum

```
MyServlet extends HttpServlet {                          // Shared field
    private User user = new User();
    …
    void getInfo(HttpServletRequest req) {
        Session s = req.getSession();
        user.userId = s.getAttribute(“id”);
    }
}
```

Public fields

- Public fields can be accessed by all classes
- Declare private and provide get/set methods unless they must be public
- If you absolute have to use a public field, be sure to make it final

Public fields - example

Not a good idea...

```
public class BadUserInfo {

   public String username;
   public String favColor;

   public BadUserInfo(String username, String favColor) {
      this.username = username;
      this.favColor = favColor;
   }
}
```

A better idea...

```
public class GoodUserInfo {

   private String username;
   private String favColor;

   public GoodUserInfo(String username, String favColor) {
      this.username = username;
      this.favColor = favColor;
   }

   public String getUsername() {
      return username;
   }
```

KRvW Associates

Public methods

- Similarly, make sure that your methods are explicitly made private
- Prevents interface from being maliciously accessed
  - E.g., providing tainted data
- If a method must be made public, be sure to document the reason
- See example – MethodAccess

Public methods - example

```
private String username;
private String favColor;

public BadUserInfo(String username, String favColor) {
   setUsername(username);

   this.favColor = favColor;
}

public void setUsername(String user) {
   username = user;
}
```

[1]

[1] Be sure that methods are made private unless they must be public, otherwise they can be invoked by any class

KRvW
Associates

Public static modifier

- Public static fields and methods can be accessed by other classes even if they don't instantiate

Public static modifier - example

```
public class Widget {

   public static int height = 2;
   public int width = 10;

   public Widget() {}
}
```

[1]

```
public class StaticModifier {

   public static void main(String[] args) {

      System.out.println("\nWidget height: " + Widget.height + "\n");
   }
}
```

[2]

[1] The *Widget.height* field is defined as public static

[2] Any class is now able to access/modify the *height* field without instantiating the *Widget* class

Package scope

- Any class within a package can access the public and protected variables within other classes in the same package
- Thus, if you don't want to provide access to something, make it private explicitly

KRvW Associates

cigital

Package scope - example

```
package somepackage;

public class Widget {

   protected int height;
   int width;

   public Widget() {
      height = 2;
      width = 10;
   }
```

[1]

[1] The *height* field is accessible to any class that declares itself part of the *somepackage* package

Inner classes

- The manner in which JVMs compile inner classes opens up a loophole that enables an attacker to access private members of the outer class
- Entails making creative use of the Reflection API
- See example – InnerClasses

cigital KRvW Associates

Inner classes - example

```
public class InnerClassExample {
   private int outerValue = 2;     [1]     // note the "private" modifier

   // an inner class...

   private class Inner {
      private int innerValue;

      Inner() {

         // accessing a private field in the outer class

         innerValue = outerValue;     [2]
      }
   }
}
```

[1] A private integer field is defined in the outer class

[2] The inner class accesses the private field in the outer class (the Java compiler must create a loophole to allow this)

KRvW Associates

Inner classes – example (cont'd)

```
// call the access$000 method using reflection

Method access = insecure.getDeclaredMethod("access$000", new Class[]
   {insecure});
Object value = access.invoke(null, new Object[] {example});
```

[3]

[3] The Java compiler creates a method called *access$000* that can be called using Reflection to obtain the value of the private field

Finalization

- If methods and classes aren't made final, they can be extended in unforeseen ways and may enable an attacker to access or alter otherwise protected objects and information

KRvW Associates

Finalization - example

```
public final class Widget {     [1]

   private int height;
   private int width;

   public Widget(int height, int width) {
      this.height = height;
      this.width = width;
   }

   public final int getHeight() {
      return height;
   }
```

[1] Define classes to be final whenever possible to prevent them from being extended in unforseen ways

KRvW Associates

cigital

Cloning

- If an object can be cloned, an attacker may be able to bypass its constructor, which could lead to disclosing uninitialized memory space
- If an object must implement the Cloneable interface, make sure to provide an explicit final clone() method as early in the inheritance hierarchy as possible

Cloning - example

```
class Widget {

   private int height = 2;
   private int width = 10;

   public Widget() { }

   public int getWidth() {
      return width;
   }

   public final Object clone() throws java.lang.CloneNotSupportedException {
      throw new java.lang.CloneNotSupportedException();
   }
}
```

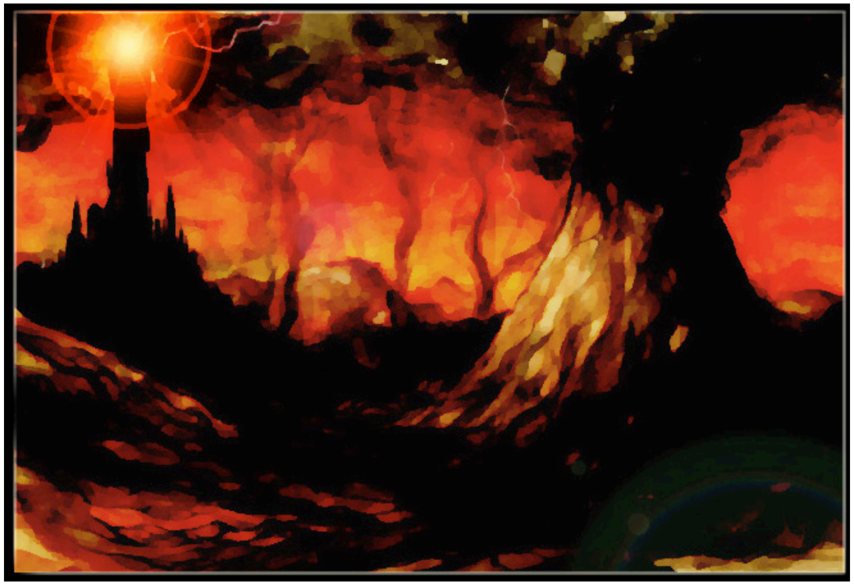
[1]

[1] To prevent cloning, override the *clone()* method and throw a *java.lang.CloneNotSupportedException*

Phyla: Encapsulation

- Comparing Classes by Name
- Data Leaking Between Users
- Leftover Debug Code
- Mobile Code: Object Hijack
- Mobile Code: Use of Inner Class
- Mobile Code: Non-Final Public Field
- Private Array-Typed Field Returned from a Public Method
- Public Data Assigned to Private Array-Typed Field
- System Information Leak
- Trust Boundary Violation

*. Environment



Bonus pernicious kingdom

- **Environment**
  - Everything that is outside of source code but is still critical to security
  - ASP .NET Misconfiguration: Password in Configuration File, Insecure Compiler Optimization, and others
  - Example: **ASP .NET Misconfiguration: Creating Debug Binary** phylum

```
<configuration>
        <compilation debug="true">          // Debug binary
                ...
        </compilation>
        ...
</configuration>
```

CLASSPATH

- Modifying the CLASSPATH environment variable is the equivalent of modifying a Windows/Unix PATH
  - An attacker can construct classes with “value added” features that perform malicious acts
  - Classic example is theft of username/password
  - Involves duping a user into running attacker's code

Weak access permissions (J2EE)

- EJB method permissions should never grant access to the ANYONE role
- Indicates that access control for an application has not been carefully thought through
- Method permissions should always be restricted to the minimum set of roles that should be granted access

Weak access permissions - example

The following example illustrates the improper use of method access controls...

```
<ejb-jar>
    ...
    <assembly-descriptor>
        <method-permission>
            <role-name>ANYONE</role-name>
            <method>
                <ejb-name>SomeBean</ejb-name>
                <method-name>someMethod</method-name>
        </method-permission>
    </assembly-descriptor>
    ...
</ejb-jar>
```

cigital

KRvW Associates

Phyla: Environment

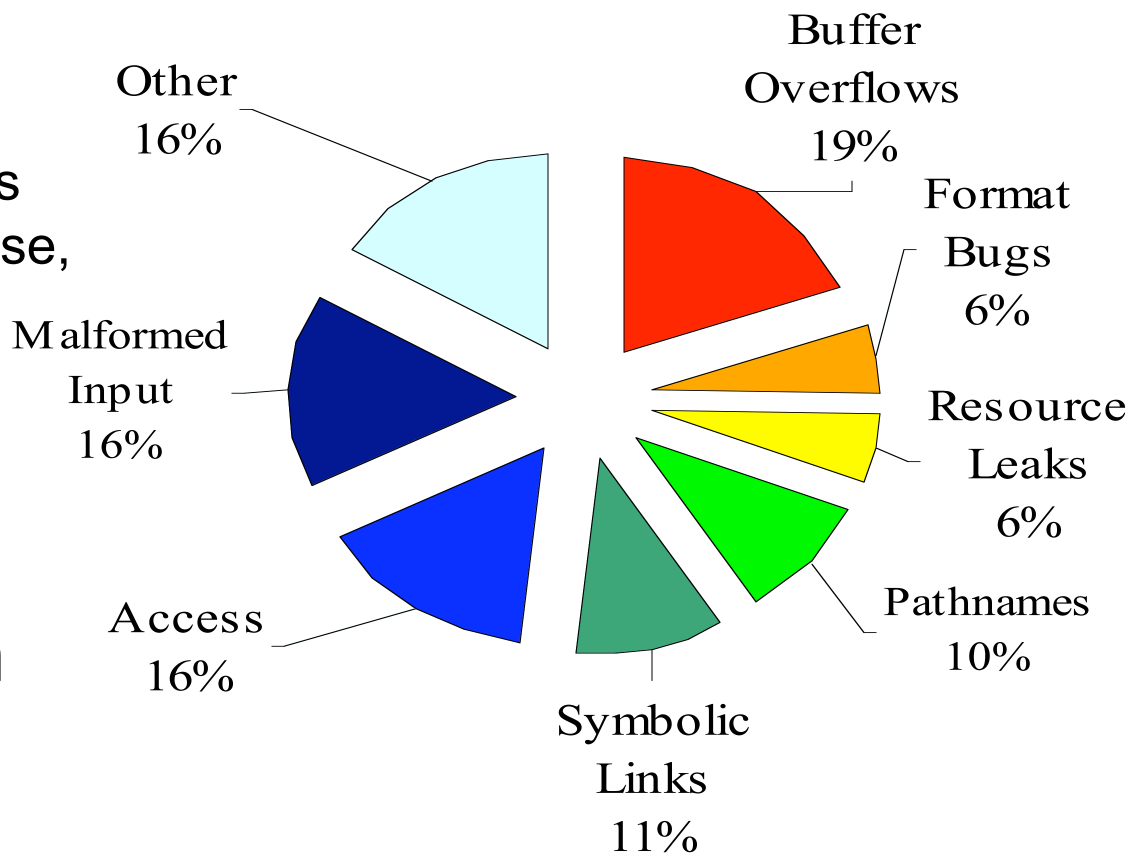
- ASP .NET Misconfiguration: Creating Debug Binary
- ASP .NET Misconfiguration: Missing Custom Error Handling
- ASP .NET Misconfiguration: Password in Configuration File
- Insecure Compiler Optimization
- J2EE Misconfiguration: Insecure Transport
- J2EE Misconfiguration: Insufficient Session-ID Length
- J2EE Misconfiguration: Missing Error Handling
- J2EE Misconfiguration: Unsafe Bean Declaration
- J2EE Misconfiguration: Weak Access Permissions

Static Analysis and Code Review

KRvW Associates

Reported flaws in Common Vulnerabilities and Exposures Database, Jan-Sep 2001.

56 % of CVE vulnerabilities could have been detected with straightforward static analyses!

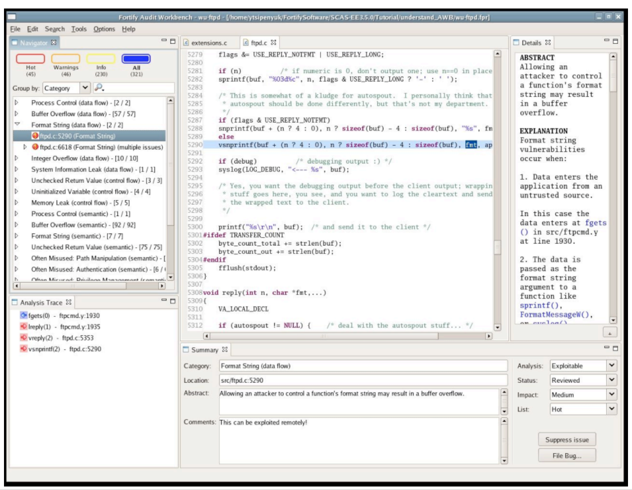


[Evans & Larochelle, IEEE Software, Jan 2002]

KRvW Associates

Touchpoint: code review (with a tool)

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



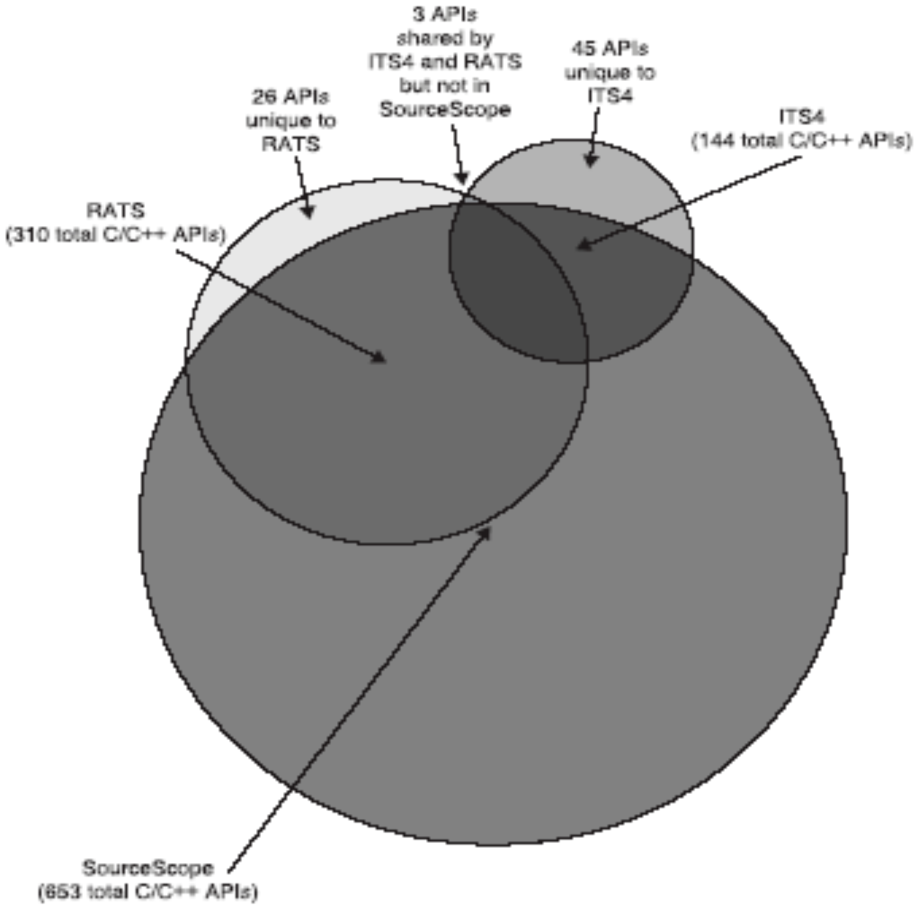
- Implementation errors do matter
  - Buffer overflows can be uncovered with static analysis
  - Static analysis
    - C/C++
    - Java
    - .NET
    - PSQL
- Tracing back from vulnerable location to input is critical

FORTIFY SOFTWARE

Code scanning tools

- Early static analysis tools (tokenizers)
  - ITS4
  - RATS
  - Flawfinder
- Modern tools (parsers)
  - Prefix
  - Fortify source code analysis suite
  - Ounce labs
  - Coverity
- The key is encapsulated know-how

Bug space coverage and early tools



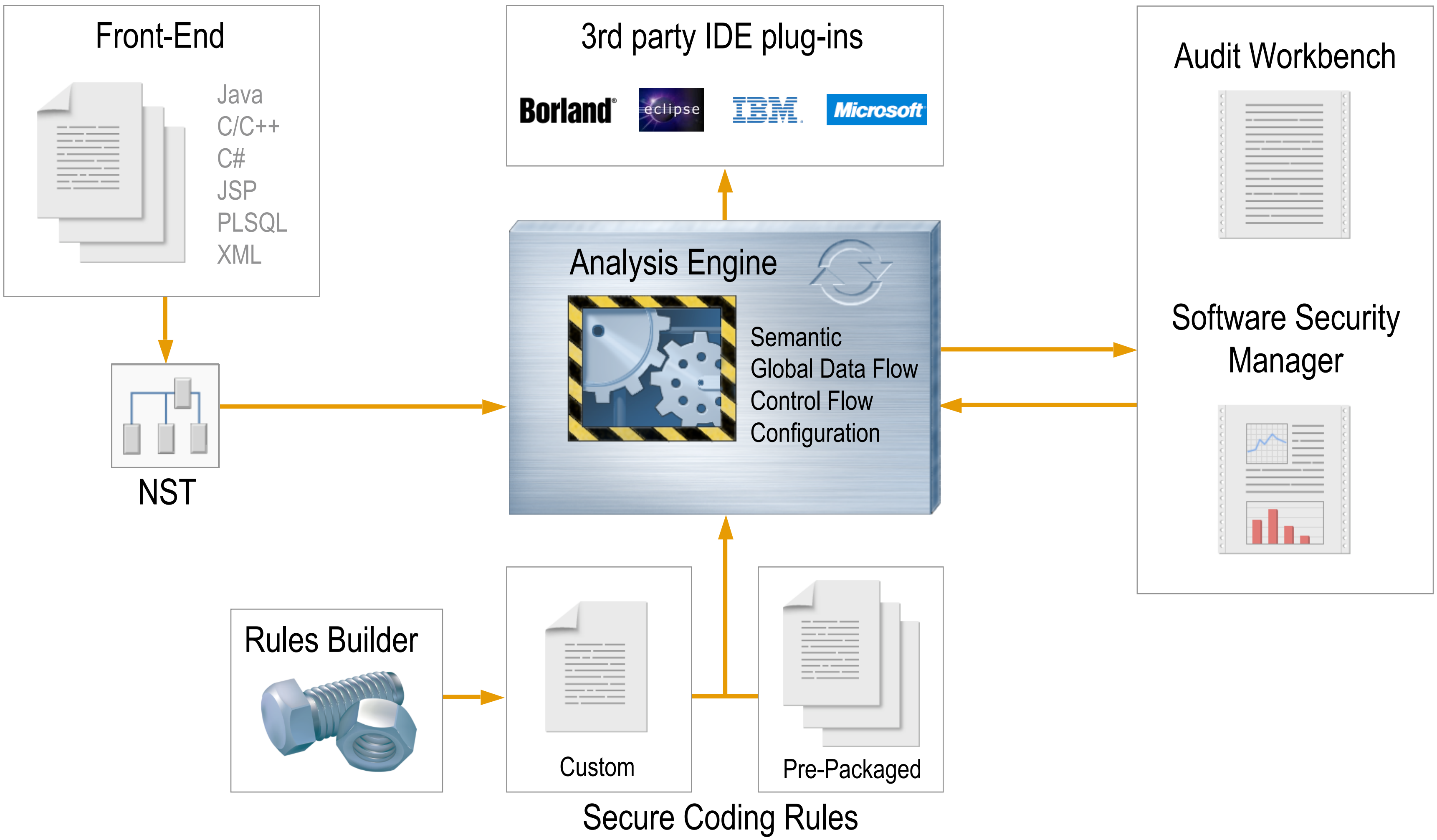
KRvW Associates

Fortify Source Code Analysis

- Integrated data flow analysis
- Broad platform support
- A comprehensive set of secure coding rules
  - Capability to add your own rules
- Proven large scale deployability

Commercially viable, accurate and effective analysis

Fortify architecture



Comprehensive secure coding rules

- Secure coding rulepacks based on the seven kingdoms
- Continuously updating and improving rulepacks
- Fortify Rules Builder allows you to further extend rulepacks to meet individualized needs
- Advanced context sensitive guidance inside in the IDE
- Intellectual property based on ten years of Cigital work

- see vulncat.fortifysoftware.com

The single largest compilation of secure coding techniques and guidance ever written

Next steps

Software security critical lessons

- Software security is more than a set of security functions
  - Not magic crypto fairy dust
  - Not silver-bullet security mechanisms
  - Not application of very simple tools
- Non-functional aspects of design are essential
- Security is an emergent property of the entire system (just like quality)
- Breaking stuff is important
- To end up with secure software, deep integration with the SDLC is necessary

Bottom up software security actions

- A few relatively simple things can make a tangible difference and can help you get started with software security
- Build checklists and use them
  - Sun's SAG checklist http://www.securecoding.org/companion/checklists/SAG/
- Begin to develop a resource set (e.g., portal)
- Start small with simple architectural risk analyses (think Smurfware)
- Don't forget to include business-case justifications
- Use code scanning tools

Top down software security actions

- Think of the problem as an evolutionary approach
- Chart out a strategic course of action to get where you want to be
  - Have a gap analysis performed
  - Make achievable, realistic milestones
  - Think about metrics for success
- Use outside help if you need it (Cigital)

cigital

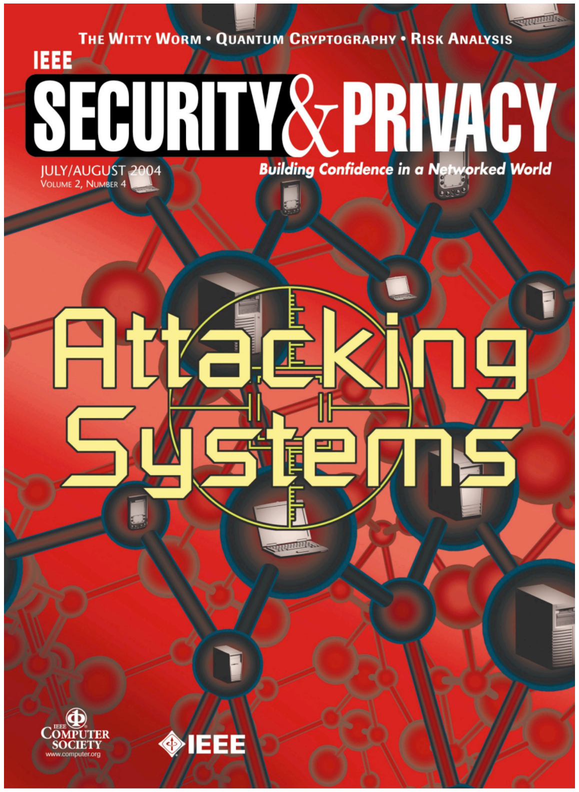
KRvW Associates

IEEE Security & Privacy Magazine

- Monthly Department on Software Security Best Practices called "Building Security In"

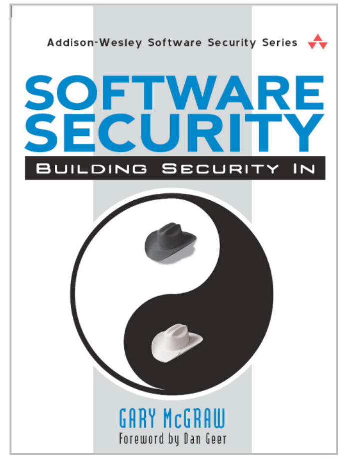
Table 1. Building Security In articles.

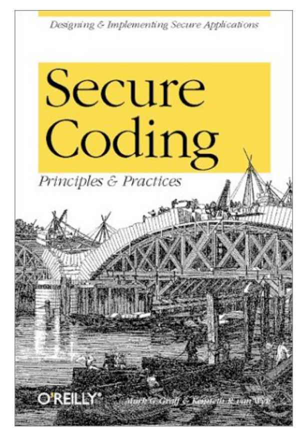
TITLE	KEY	AUTHOR	*IEEE SECURITY & PRIVACY* CITATION	URL: WWW.CIGITAL.COM PAPERS/DOWNLOAD/
Software Security	BSI1	Gary McGraw	2(2):80–83	bsi1-swsec.pdf
Misuse and Abuse Cases: Getting Past the Positive	BSI2	Paco Hope, Gary McGraw, Annie Anton	2(3):32–34	bsi2-misuse.pdf
Risk Analysis in Software Design	BSI3	Denis Verdon, Gary McGraw	2(4):79–84	bsi3-risk.pdf
Software Security Testing	BSI4	Bruce Potter, Gary McGraw	2(5):81–85	bsi4-testing.pdf
Static Analysis for Security	BSI5	Brian Chess, Gary McGraw	2(6):76–79	bsi5-static.pdf
Software Penetration Testing	BSI6	Brad Arkin, Scott Stender, Gary McGraw	3(1):84–87	bsi6-pentest.pdf
Knowledge for Software Security	BSI7	Sean Barnum, Gary McGraw	3(2):74–78	bsi7-knowledge.pdf
Adopting a Software Security Improvement Program	BSI8	Dan Taylor, Gary McGraw	3(3):88–91	bsi8-program.pdf
A Portal for Software Security	BSI9	Nancy R. Mead and Gary McGraw	3(4):75–79	bsi9-portal.pdf
Bridging the Gap between Software Development and Information Security	BSI10	Kenneth R. van Wyk and Gary McGraw	3(5):75–79	bsi10-bridge.pdf



For more

cigital





- See the Addison-Wesley Software Security series
- Send e-mail: gem@cigital.com Ken@KRvW.com
- http://www.cigital.com
- http://www.krvw.com

"So now, when we face a choice between adding features and resolving security issues, we need to choose security."

-Bill Gates