

M1

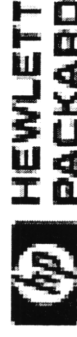
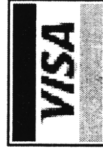
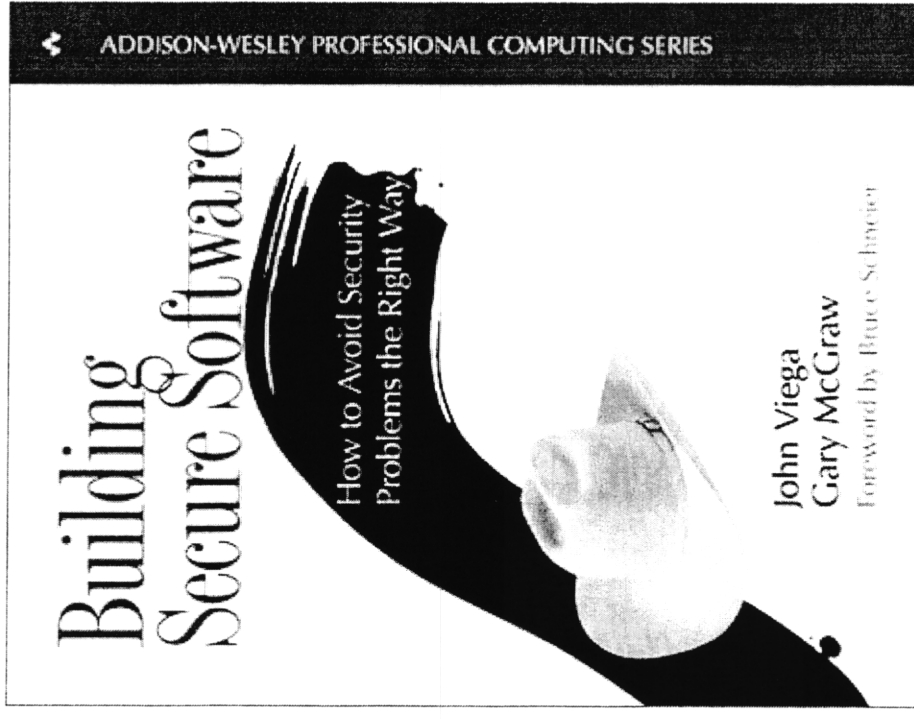
Building Secure Software
Why the standard approach to security
doesn't work.

II

Gary McGraw

This talk sponsored by Cigital

- Software Risk Management Specialists
 - Safety
 - Security
 - Reliability
- Fortune 1000 clients
- Cigital Labs
- A.k.a. Reliable Software Technologies



cigital



Pop quiz: question one

- What do wireless devices, cell phones, PDAs, browsers, operating systems, servers, PCs, PKI, and firewalls have in common?

Software

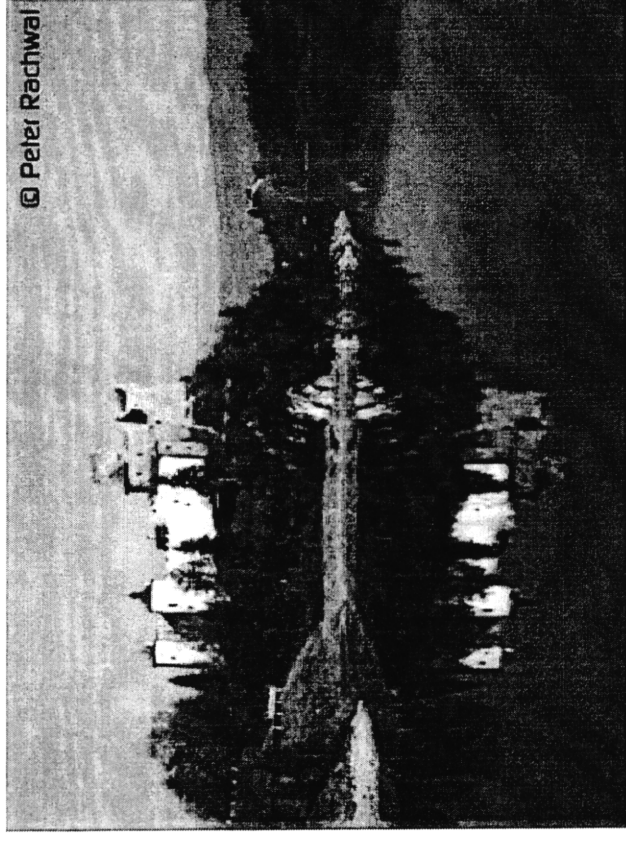
Pop quiz: question two

- Terrorist attack the weakest link in a system, in novel ways. What is the weakest link in our critical infrastructure?

Software

Defending castles is passé

Today's computer and network security mechanisms are like the walls, moats, and drawbridges of medieval times. At one point, effective for defending against isolated attacks, mounted on horseback. Unfortunately, today's attackers have access to airplanes and laser-guided bombs!



© Peter Rachwal

It's all about the software

- Securing a platform is well understood
 - dynamic nature (penetrate and patch) is a challenge
 - many people do this for a living
- Securing a network is also a solved problem
 - this can even be outsourced!
 - Counterpane Internet Security
 - commercial firewall solutions
- We find the biggest risks in application code hosted on the net
 - often developed in-house
 - weakest link in the chain

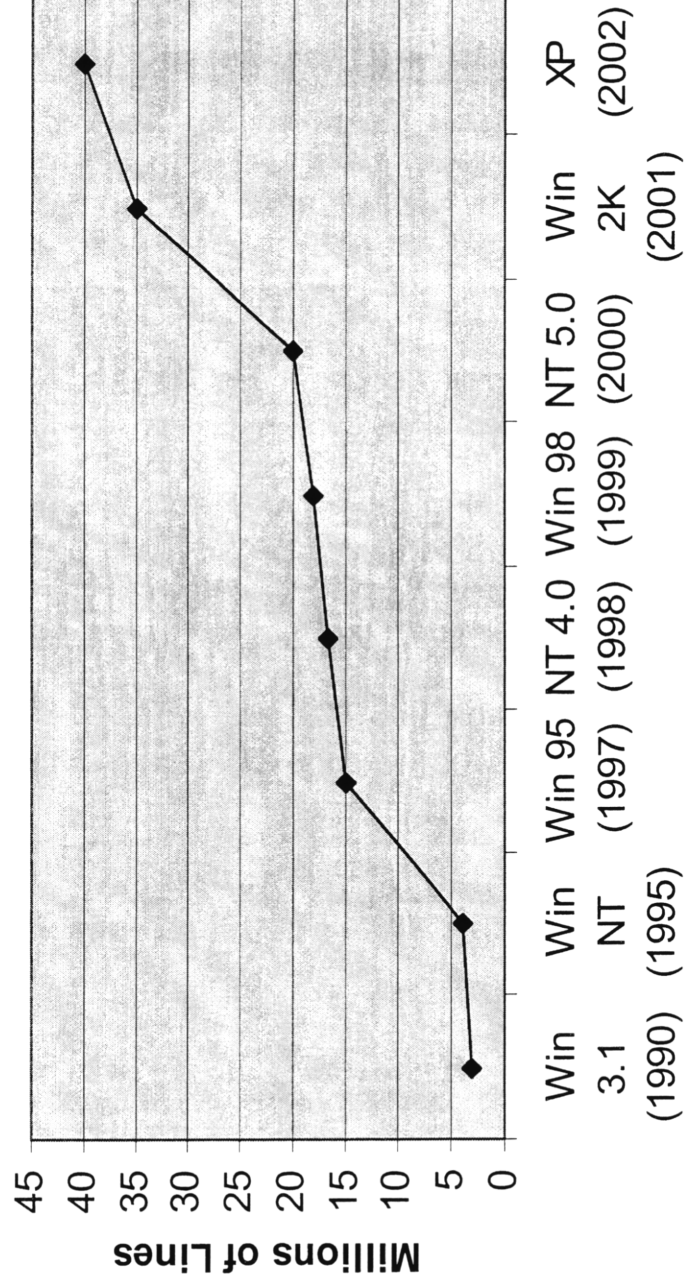


Counterpane
Internet Security, Inc.



Software complexity growth

Windows Complexity





Seemingly simple things...

United Airlines - Create itinerary - Netscape

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop

Bookmarks Location: UZdNuG*in%2Fond%3D980869486.48002%2Cin%2Fai%2Funited&ar.0=0&ar.1=0 What's Related

UNITED

Contact United | Site search

Planning travel | Travel support | Mileage Plus | About United

Home

Planning Travel

Create itinerary

Special deals

My itineraries

Request upgrade

Route maps

Vacation packages

Update profile

Electronic timetable

Flights

Modify Flight

E-fares | Award Travel

Create itinerary

Modify or purchase your itinerary.

Clear itinerary

Help

Washington (IAD) to Paris (CDG)

Monday, Mar 12

Flight info	Dates	Miso	Fares
United Airlines 914	Mar 12 5:35 pm depart IAD	stops: Non-stop	Class: Coach Fare Rules
Boeing 777	Mar 13 7:00 am arrive CDG		Delete

Paris (CDG) to Washington (IAD)

Tuesday, Mar 13

Flight info	Dates	Miso	Fares
United Airlines 915	Mar 13 1:00 pm depart CDG	stops: Non-stop	Class: Coach Fare Rules
Boeing 777	Mar 13 3:30 pm arrive IAD		Delete

Total Airfare (including taxes): USD 2090.76

Click to
Select Your Seats

Document: Done

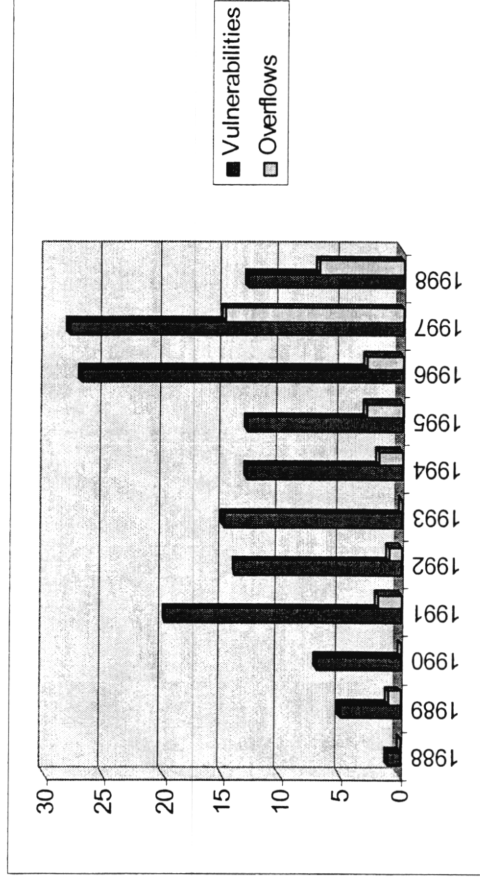


© 2001 Gary McGraw

Software security

Software security analysis

- Focus on applications inside an existing system
- Not penetration testing
- Not firewall analysis
- Not crypto cracking
- Independent, objective, expert analysis



- Software is now a security Achilles's Heel
- Examples
 - buffer overflows
 - I/O and in between
 - randomness

Common software problems

- In the Unix world...
 - sendmail, fingerd, ftpd, login, lpr, ...
- In the Windows world
 - IIS
 - Windows password security
 - VB scripting
- The key is designing things to be secure from the start

Unfortunately, this is hard



Software attacks

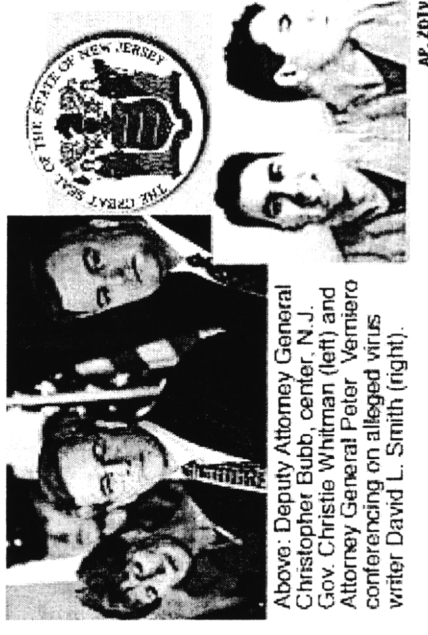
- Mirror real world attacks
- Theft, destruction, spying, forgery, etc.

With the added bonus of:

- Automation
- Remote action
- Instant technique propagation

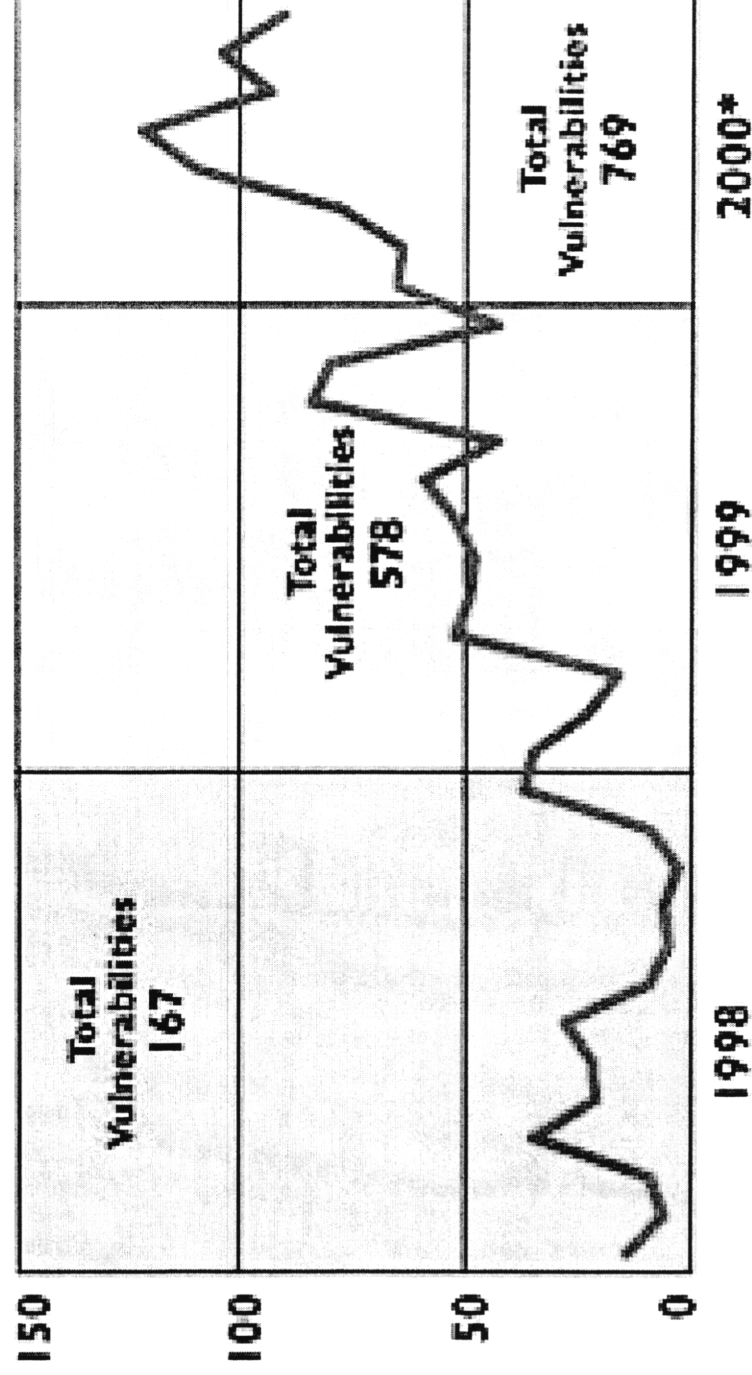
Who is the bad guy?

- Hackers
 - “Full disclosure” zealots
- “Script kiddies”
- Criminals
 - Lone guns or organized
- Malicious insiders
 - Not just employees
- Business competition
- Police, press, terrorists, intelligence agencies





The problem is growing



*January to September. Source: SecurityFocus.com, October 2000

Penetrate and patch is bad

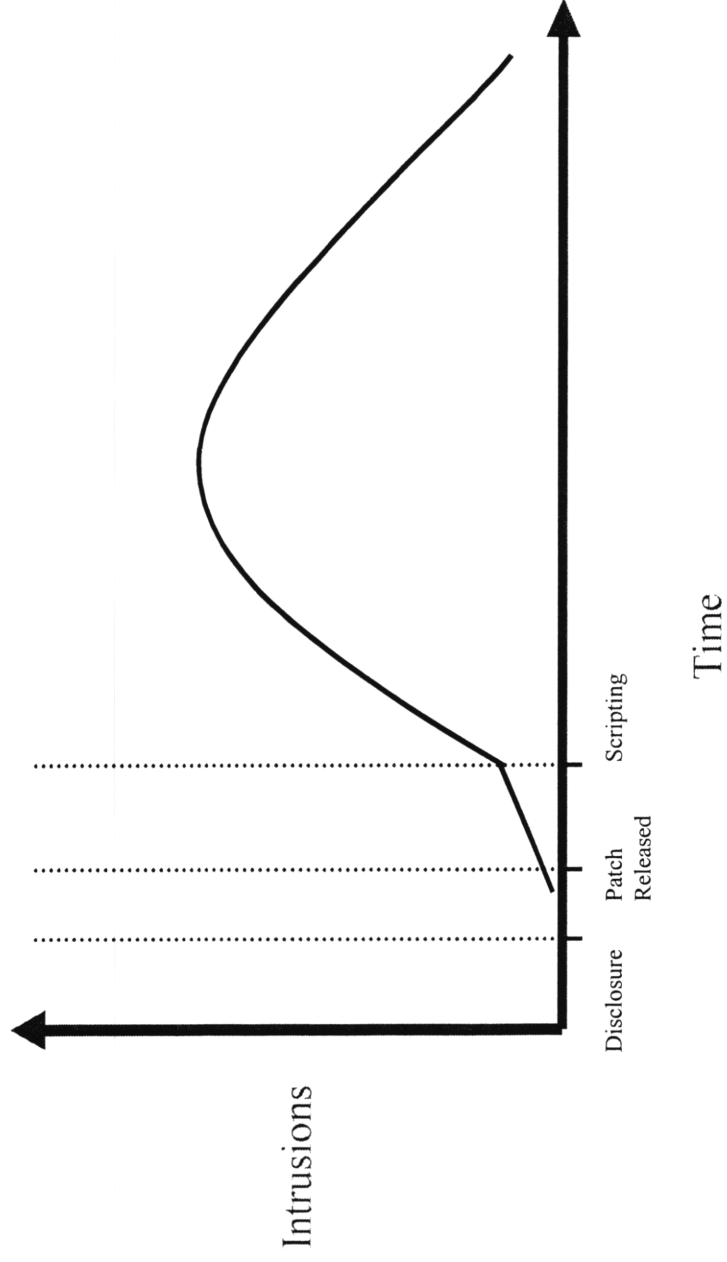
- Avoid in-the-wild analysis (by bad guys)
 - too late
 - patches ignored
 - patches have holes
- Create architectures with security in mind
 - external assessment
 - sound engineering
- Identify and manage security risks

"...any program, no matter how innocuous it seems, can harbor security holes. ... I thus have a firm belief that everything is guilty until proven innocent."

-Steve Bellovin



...and does not work well



Environment is important

- C/C++ versus Java (not to mention scheme/ML)
 - Java was designed with security in mind
 - C was designed with efficiency in mind
- Wireless technology
 - No worthwhile security yet (802.11 war driving)
 - ARP cache poisoning attack
 - Bluetooth “instant registration”
- Memory protection model of OS



Why SRM is Hard

- Many subtleties
- Too much to know
- No good resource on SRM methods
- Tools are bad
- Programming is hard
- Popular languages are really awful
- The only constant is change

The trinity of trouble

- **CONNECTIVITY**
 - The Internet is everywhere and everything is on it
- **EXTENSIBILITY**
 - Systems evolve in unexpected ways and are changed on the fly
- **COMPLEXITY**
 - Networked, distributed, mobile code is hard



Attacking Malicious Code: A Report to the Infosec Research Council

Gary McGraw, Editor
Greg Mortenson, Executive Editor

In October of 1999, the Infosteel Research Council created a Science and Technology Study Group focused on malicious code. The Malicious Code ISIRG is charged with developing a national research agenda to address the accelerating threat from malicious code. The

This paper is organized as follows. Section 2 presents the problem of malicious code. In this section, we discuss important problems posed by malicious code. In this report, we discuss important trends that are making malicious code an increasingly serious problem. We then survey existing techniques for preventing, detecting, and removing malicious code, pointing out their limitations and suggesting new approaches. Section 3 presents a new approach to detecting malicious code. This approach is based on the idea of detecting anomalous behavior in the system's intended function. Although the problem of malicious code has a long history, a number of recent, widely-published attacks and certain economic problems suggest that malicious code is likely to become an important security issue in the near future. We conclude with a discussion of some open research issues, and suggest some possible future work.

This report is a by-product of two meetings of Study Group members and their invited guests. Although this report was written by two of the study group members, we believe it represents an accurate distillation of the ideas and insights of all the participants.

What is Malicious Code?

- Malicious code is any code added, changed, or removed from a software system to intentionally cause harm or solvent
- Viruses are pieces of malicious code that attach to host programs and propagate when an infected program executes.
- Worms are particularly not networked

Copyright © 2007 John Wiley & Sons, Inc.

- Gary McGraw & Greg Morrisett (2000) Attacking Malicious Code: A Report to the Infosec Research Council. *IEEE Software*, Volume 17, Number 5, pages 33-41, September/October 2000.
- Bill Arbaugh, Bill Fithen, and John McHugh. Windows of Vulnerability: A Case Study Analysis. *IEEE Computer*, December 2000.



Software Security RISKS

Know your enemy

Software Risks

- The dreaded buffer overflow
 - The largest SW security problem today
- Race conditions
 - The danger lurking in distributed computing
- Random number generation
 - Predictability is a burden



Software Risks

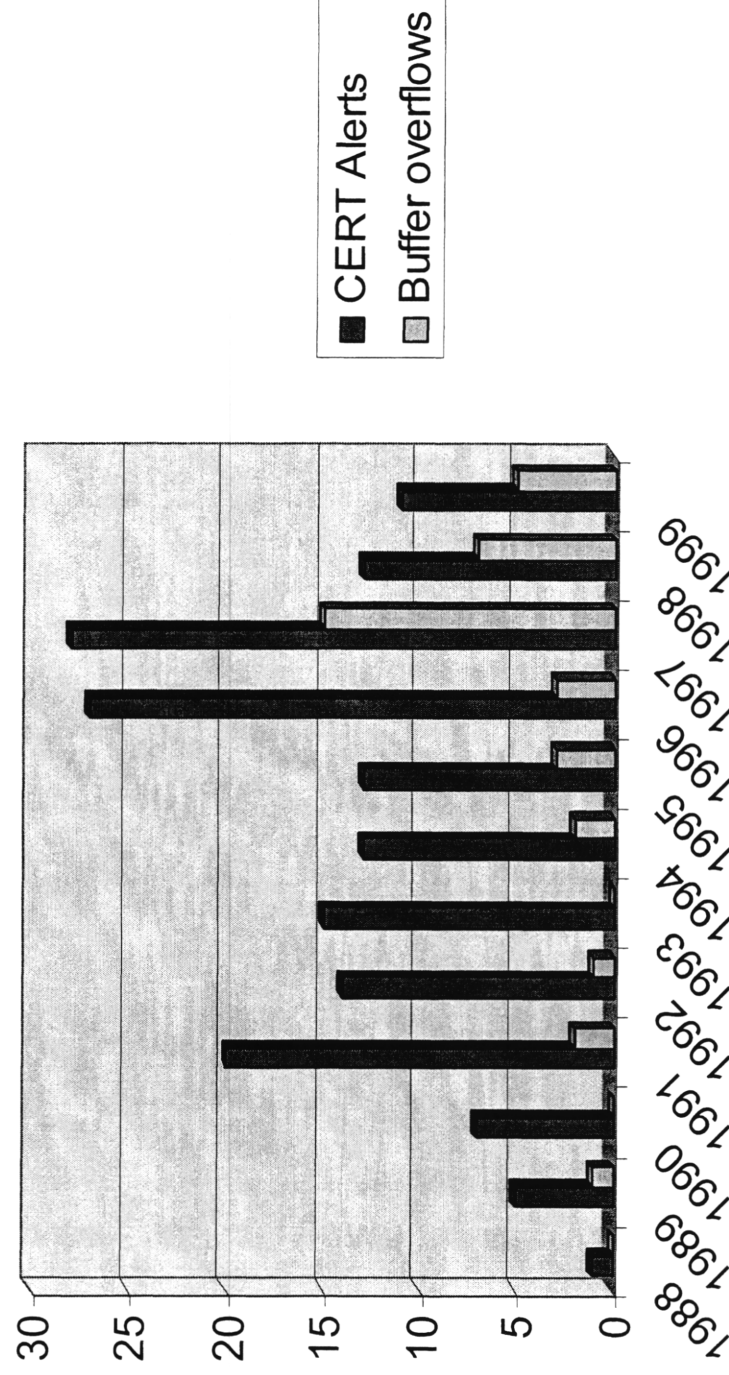
- Misuse of cryptography
 - Never roll your own algorithms
- Trust problems
 - Input validation should be standard practice
- Authentication problems
 - Software plays an important role

RISK: 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!
 - (remember in the 70's when code had to be tight?)
- The most pervasive security problem today in terms of reported bugs

Eleven years of CERT reports

Security Problems (CERT)



Buffer overflows: the target

- Two main flavors of buffers out there
 - Heap allocated buffers
 - malloc()'d and new'd data
 - Stack allocated buffers
 - Local variables (non-static)
 - Function parameters
- Stack overflows are easier (thus more common)

When a buffer overflows

- Programs act in strange ways
- Programs fail completely
- Programs proceed without any noticeable difference in execution
- This makes the problems hard to spot through standard testing processes

What are the implications?

- Overwrite important data
 - Privilege fields
 - Password fields
- Run arbitrary code
 - The dreaded stack overflows
 - How does that work?

A classic error

```
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)

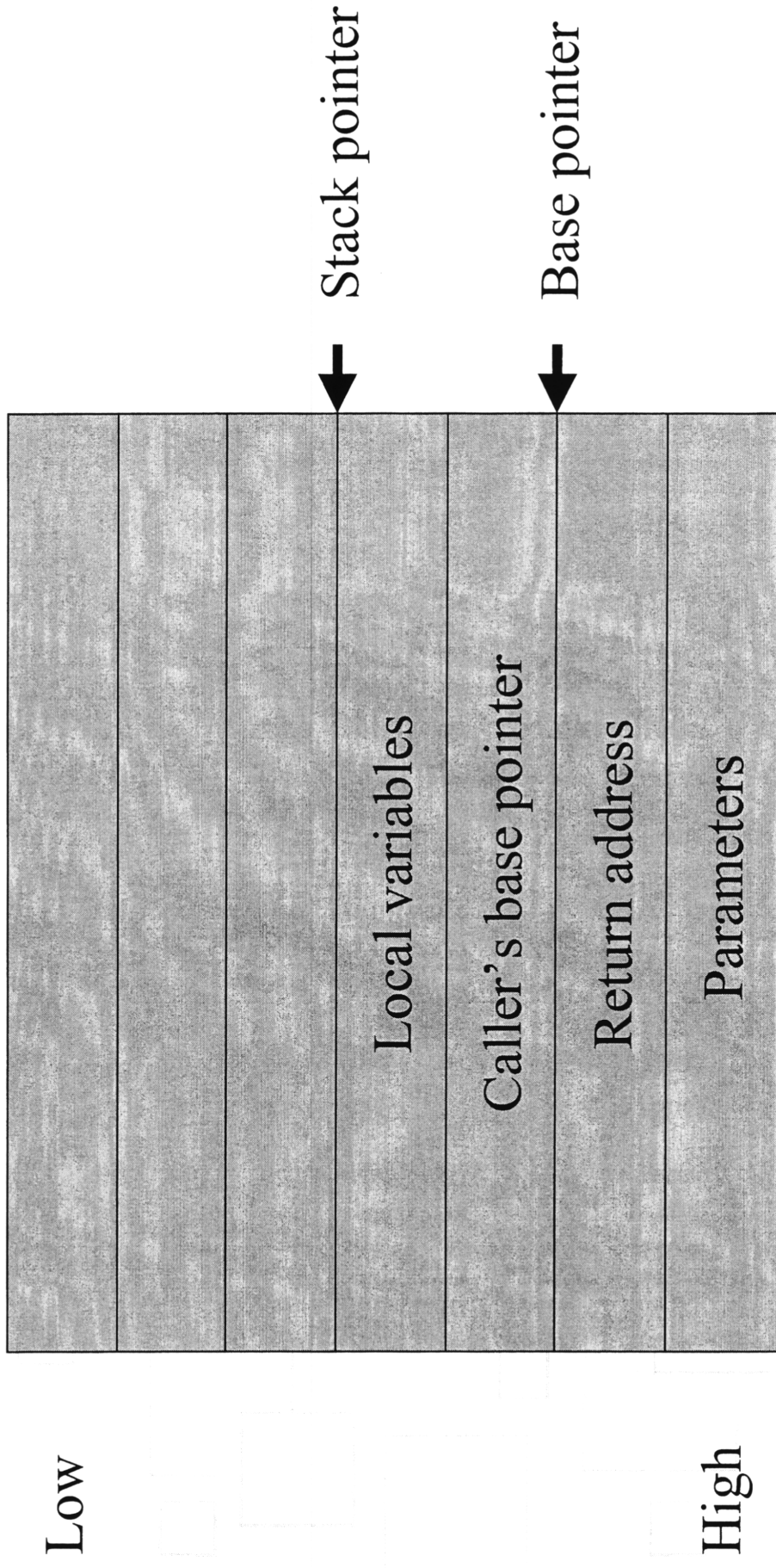




Why have a stack?

- Seamless handling of function specific data
- Supports recursion
- A common computational approach

The x86 program stack



Bad and Good system calls

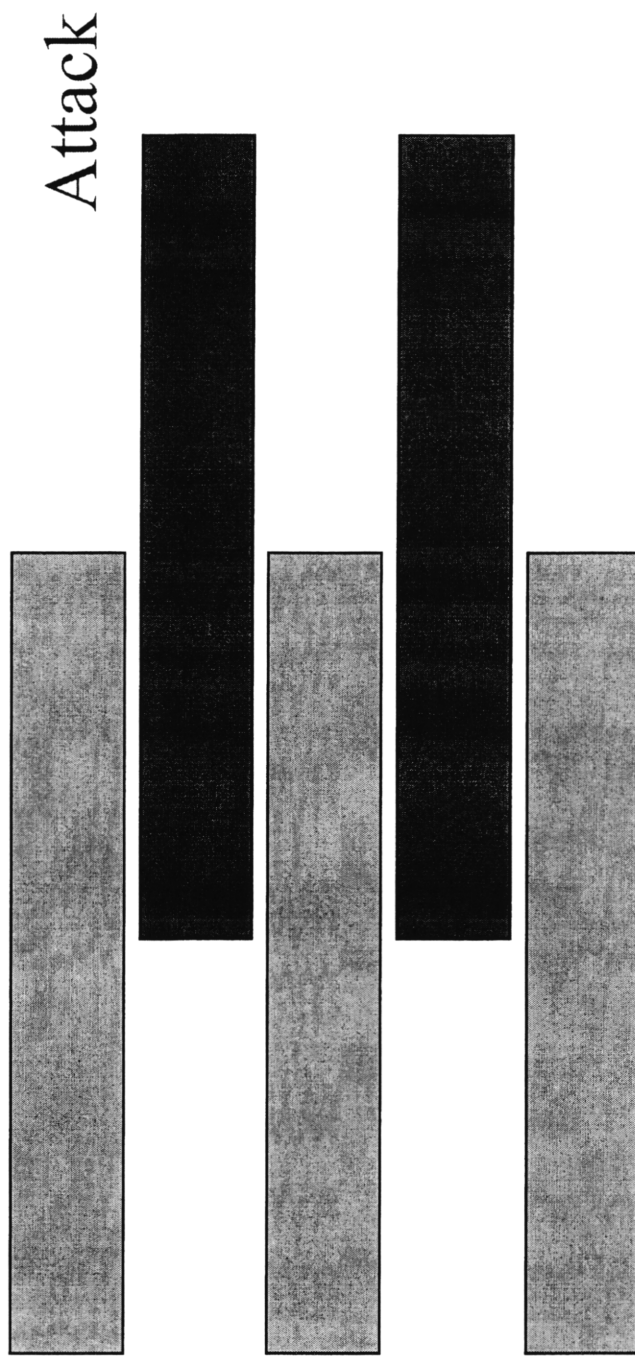
Instead of:	Use:
gets(buf)	fgets(buf, size, stdin)
strcpy(dst, src)	strncpy(dst, src, n)
strcat(dst, src)	strncat(dst, src, n)
sprintf(buf, fmt, a1,...)	snprintf(buf, fmt, a1, n1,...) (where available)
*scanf(...)	Your own parsing

References

- M. Weber, V. Shah, and C. Ren. A Case Study in Detecting Software Security Vulnerabilities using Constraint Optimization. In *International Workshop on Source Code Analysis and Manipulation (SCAM 2001)*, Florence, Italy, November 2001.
- D. Wagner, J. Foster, E. Brewer, and A. Aiken. A first step towards automated detection of buffer over-run vulnerabilities. In *Proceedings of the Year 2000 Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, 2000.

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

Two arrows originate from the text 'Race condition'. One arrow points to the 'count++;' line, and the other points to the 'p.println(count + " hits so far!");' line, illustrating that the state of 'count' is not thread-safe.



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

References

- Bishop, Matt and Mike Dilger (1996). Checking for Race Conditions in File Access, Computing Systems, 9(2):131-152.
- Basic operating systems research!



RISK: Random number generation

```
void print3(){
    printf("%d, %d, %d\n", rand(), rand(), rand());
}

int main(){
    srand(12);  print3();
    return 0;
}
```

Prints: 423229905, 2125664504, 753467483

RISK: Pseudo-random numbers

- Computers are deterministic
- A "seed" (starting point) and some formula derive a sequence of random-looking numbers
- They're supposed to be reproducible!
- In security-critical uses, guessing the seed gives away the whole sequence
 - Brute force attacks work

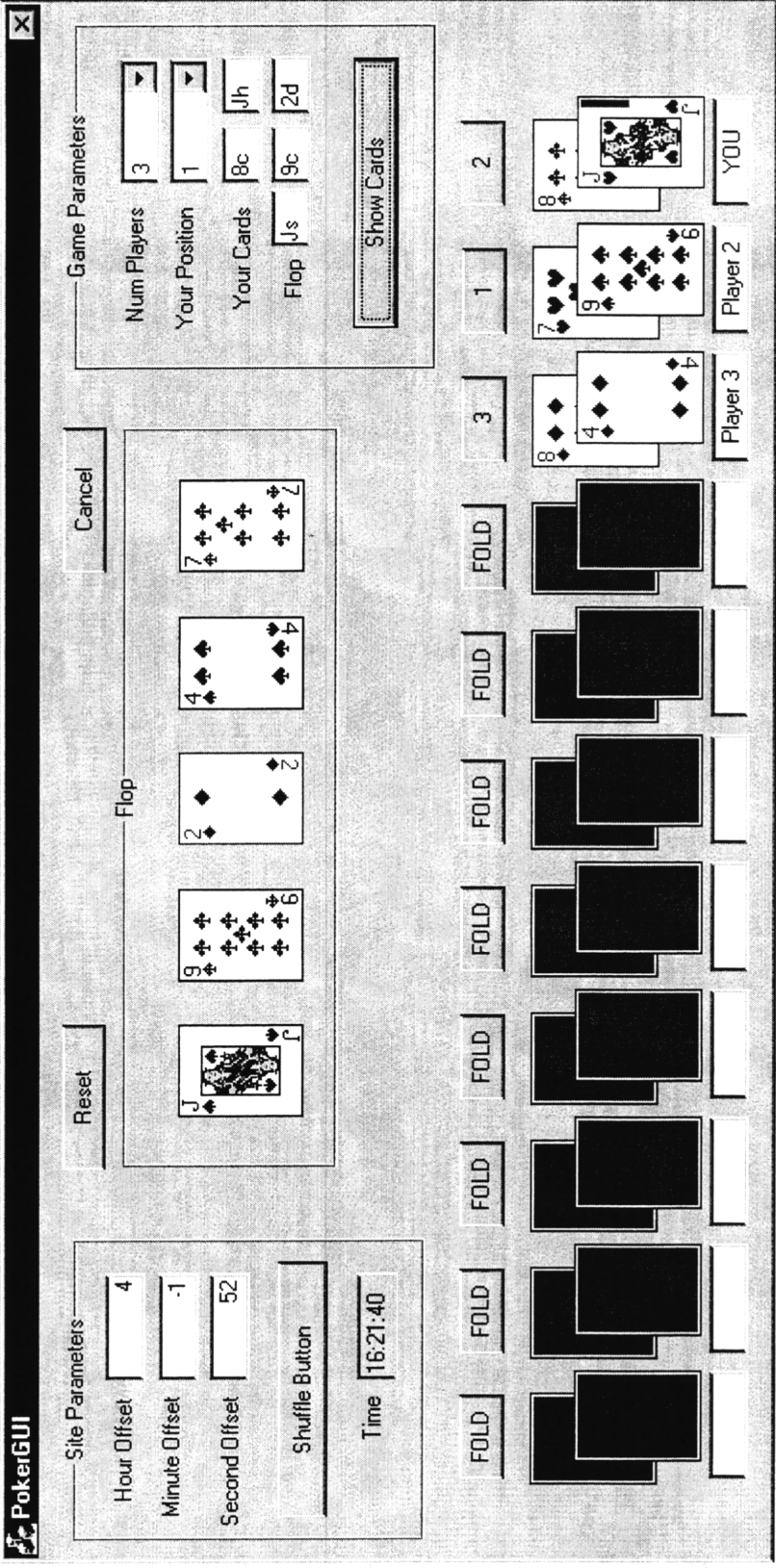
Guessing the seed

- 2^{32} possible seeds
- Look at your program's output, compare it to results expected from each seed, weed out seeds
 - 2^{32} isn't that big a number
- By default, most PRNG's are seeded with the current clock
- Informed guess dramatically reduces search

Not random: Online Poker

- Texas Hold 'Em
 - You get 2 cards face down
 - 5 shared cards in middle
 - Make the best poker hand
- Given the algorithm for shuffle and deal + seed
 - You know when to hold 'em and when to fold 'em
- If you sort of know the seed, you can figure it out after seeing a few cards!

Not random: Online Poker



Other common security risks

- Misuse of cryptography
- Poor authentication and input checking
- Long lived secrets
- Insecure "erase"
- Not checking the return value
- Believing people (social engineering)
- Physical security failure
- Covert channels



Guiding Principles for SW Security

Common sense for your software



Pithy nonsense

- The most secure computer in the world is one that has its disk sanded down and is buried in a ten foot hole filled with concrete.
- Security is all about making sure nobody gets anything done.

Reaching for the brass ring

- Design for security is critical
- Teaching people HOW to do this is very hard
 - Apprenticeship is the state of the art today
 - Guidelines can help (but tend to be unsatisfyingly high level)
- Our ten guidelines help, but are no magic bullet

General guidance

- History shows that this is very hard
 - Windows 95/98/ME problems
 - Unix problems
- Retrofitting security is extremely hard (and mostly impossible)
- Identify:
 - Data flow (between components)
 - Users, roles, rights (implicit and explicit)
 - Trust relationships
 - Solutions
- Security analyst should participate in design meetings
 - Do more analysis than design



Guiding principle #1

Identify and secure the weakest links

- Door locks $\neq$ home security
- Attackers simply
 - kick in the door
 - drill the lock
 - break a window
 - pick the lock
- NB: Cryptography + Firewalls $\neq$ Security
- Software is the weakest link

Guiding principle #2

Practice defense in depth

- Manage risk with multiple strategies
- Locks work better if you have alarms on them
- Adding motion sensors helps more
- Redundant subsystems can be as secure as the STRONGEST link
- EX: multiple forms of authentication



Guiding principle #3

Be reluctant to trust

- Is that server hacked?
- Why trust that COTS software?
- Bad guys are more resourceful than you think
- Don't mimic everyone else
 - common protocols are often fundamentally flawed
- Trust is *transitive*
 - Trusted programs should not invoke untrusted ones, especially indirectly! (e.g., an editor)
- EX: Don't trust security products

Guiding principle #4

Remember that hiding secrets is hard

- "Security by obscurity" doesn't work
- You often don't have much control over the final program environment
- Secrets can often be inferred from behavior
- Reverse engineering is not hard
- EX: consider your client software hacked



Guiding principle #5

Follow the principle of least privilege

- Give out no more privilege than necessary
- Extend privilege for shortest possible time
- Minimize windows of vulnerability
- Don't hand out the key to your office
- Provide data on a "need to know" basis
- EX: file access versus directory or disk access

Guiding principle #6

Fail and recover securely

- “Default to insecure” is bad but prevalent
 - Credit card verification
 - Backwards compatibility issues
- “Failsecure”, a la failsafe.
- EX: Java RMI crypto protocol issue

Guiding principle #7

Compartmentalize

- Limit attackers ability to do damage
- Mimic real examples:
 - chambers in submarines
 - jail cells
- Apply process protection, chroot()
- Use different keys for different devices
- EX: SUID in Unix (all or nothing is bad)

Guiding principle #8

KISS (Keep it simple, stupid!)

- Fewer links makes a system easier to secure
- Complex code tends to be more buggy
- Users need simple UIs
- Don't use technology for technology's sake
- Provide "choke points" without back doors
- Contradicts defense-in-depth guideline
 - "Don't put all your eggs in one basket"



Guiding principle #9

Keep trust to yourself

- Hackers launch attacks based on easily collected information
- Do not give out more information than you must
- Misinformation often doesn't hurt
- Proxy firewalls, honey pots, intrusion detection
- EX: Network daemons give away too much info

Guiding principle #10

Assume Nothing

- Don't trust security claims...
 - "snake oil" abounds.
 - See the "snake oil" FAQ
<http://www.interhack.net/people/cmcurtin/snake-oil-faq.html>
- Question all assumptions and choices
 - It's hard not to be short-sighted on your own code



Building Secure Software

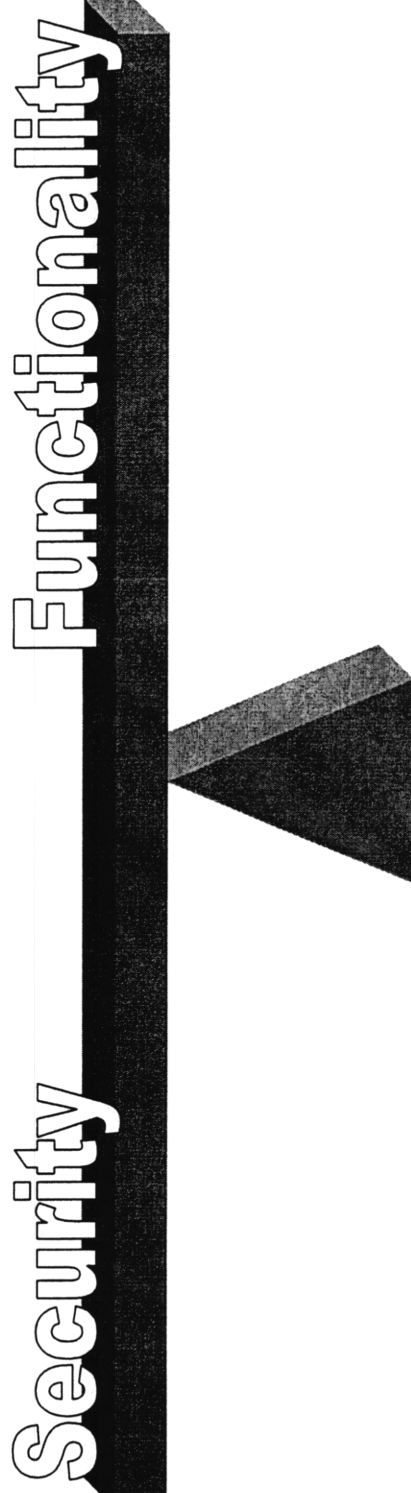
Software Risk Management for Security

Security is risk management

- There is no such thing as 100% secure
 - Must make tradeoffs
 - Should be business decisions
- The “spiral model” is effective
 - Security should NOT be an afterthought
 - Throw in a little bit the whole way through
 - A dedicated person on each project
 - Responsible for educating self on risks
 - Participates in all design and impl. Meetings
- All sound Software Engineering principles apply
 - Document! Assumptions, deployed solutions, etc.



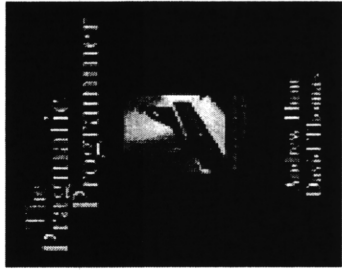
A classic tradeoff

Security  Functionality

Security Goals

- Prevention
 - Multi-level security
- Traceability and auditing
 - Anonymity
 - Authentication
 - Integrity
- Monitoring
- Privacy and confidentiality

Software project goals

- Project goals are covered in many texts. See:
 - 
 - 
- Functionality
 - Number one driver
- Usability
 - Security is a hassle
- Efficiency
 - Security is slow and expensive
- Time to market
 - Internet time
- Simplicity
 - Everybody wants this

• How do software project goals align with the security goals just discussed?!

Security risk analysis

- Start with a risk analysis of a high-level spec
 - Never let people involved with the system do this!
 - If at all possible, hire experts
 - Include all information about business requirements
- Don't wait until you start building
- Revise based on the results of analysis
- Iterate... analyze code too
- Implementation analysis
 - ITS4: www.digital.com/its4



Two critical stages

- Architectural analysis
 - Understand system
 - Identify and rank risks
 - Build attack trees
 - Think about system organization
- Implementation analysis
 - Dig into the code
 - See if the code implements the architecture
 - Focus in on areas of high risk
 - Should be guided by the architectural analysis



Pointers

- **Building Secure Software**
– Addison-Wesley 2001

- Cigital's Software Security Group and Cigital Labs invent and practice Software Risk Management

- By e-mail:
gem@cigital.com
- Sales:
Ed McComas
mccomas@cigital.com

