Building Secure Software With The Team Software Process

Noopur Davis
Shawn Hernan

Naval-Industry R&D
Partnership Conference

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Sponsored by the U.S. Department of Defense
© 2003 by Carnegie Mellon University

Trademarks and Service Marks

The following are service marks of Carnegie Mellon University.

- Capability Maturity Model Integration℠
- CMMI℠
- Team Software Process℠
- TSP℠
- Personal Software Process℠
- PSP℠

The following are registered trademarks of Carnegie Mellon University.

- Capability Maturity Model®
- CMM®
- CERT®

Overview

- Defective software is not secure
- The response strategy
- The TSP-Secure strategy

Some Security Issues

Storage security
Transmission security
Personnel security
Physical security
Communication protocols
Encryption
- primitives
- protocols
- key management

Covert channels and information leakage

Vulnerability remediation
- vendors
- users

Incident response
Intrusion detection
Deployment and configuration
Application quality
Assurance and testing
Malicious software
Trust

© 2003 by Carnegie Mellon University

The Security of Applications

"An external attacker's first measure of success is to gain the credentials of internal, legitimate users."

"Today, the path of least resistance to this goal is not the network gear, not crypto-mathematics, nor bribery; it is application software."

– -- Germanow et all

Defective Software is not Secure

Common software defects are a principal cause of software security incidents.

- Over 90% of software security incidents are due to attackers exploiting known software defect types.[1]
- Analysis of 45 e-business applications showed that 70% of the security defects were design defects.[2]

Conclusion: There is no such thing as a poor-quality secure system.

1. CERT/CC
2. "The Security of Applications: Not All Are Created Equal" by Andrew Jacquith

© 2003 by Carnegie Mellon University

Security Design Defects

Examples
- failure to authorize and authenticate users
- failure to validate user input
- failure to encrypt and/or protect sensitive data

Everyday software “bugs” are also a major risk.

For example, a buffer overflow can cause system failure or allow a hacker to take control of a system.

Many common defect types can produce a buffer overflow.
- declaration error
- logic errors in loop control or conditional expression
- failure to validate input
- interface specification error

© 2003 by Carnegie Mellon University

Never Make the Same Mistake Twice

Most vulnerabilities arise from similar causes.

- Top 10 causes account for about 75% of all vulnerabilities.
- Of all the public security bulletins issued by Microsoft from January 2002 through August 15, 2002, 50% involved buffer overflows.

Buffer Overflow - 1

A program is just a series of linear instructions.

```
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
```

Buffer Overflow -2

But how do you get to a different section of code?

```
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
```

```
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
jmp $sub
```

Jump

```
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
ret RA
```

Return

```
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 3
```

Buffer Overflow -3

Writing to the return address

- The goal is to gain control of the program.
- When a function exits, the address of the next instruction to be executed is loaded from the return address on the stack frame.

```
void myFunc(char **strNames)
{
  char   *tmpBuf[MAXNAMES];

   for (int i=0; i <= MAXNAMES; i++)
  {
   tmpBuf[i] = (char *) malloc(MAXNAMELEN);
   tmpBuf[i] = strNames[i];
  }
}
```

Buffer Overflow -4

Overwriting the return address lets you take control. A simple error can cause this!

```
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
```

```
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
jmp $sub
```

```
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 3
```

This code never gets executed

```
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
ret RA
```

```
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 2
add %1 $2
xor #4, 7
addl &4, &4
bizt 6,6
rst 0
mov %e, 1
sub %a, 3
```

After overwriting the return address, the program will jump to a location chosen by the intruder!

Some History

November 1988: Morris Worm
Major component of the attack is a buffer overflow vulnerability in *fingerd*
Incident affected between 2000 and 6,000 computers -- 5 to 10% of the *whole* Internet

Are We Having Fun Yet?

08/97: Widespread attacks against buffer overflows in UW IMAP
02/98: Widespread attacks against buffer overflows in rpc.statd
05/98: Widespread attacks against buffer overflows in BIND
02/99: Widespread attacks against buffer overflows in FTP servers
08/99: Widespread attacks against buffer overflows in rpc.cmsd
11/00: Widespread attacks against buffer overflows in rpc.statd
03/00: Widespread attacks against buffer overflows in snmpXdmid
06/01: *Code Red*
09/02: *Slapper*

What is significant about this?

Emulation Infidelity

One program often has to emulate the behavior of another

- Firewalls
- Virus-scanning programs

Vulnerabilities may occur because of emulation infidelity

One Recent Example

Firewalls have to examine packets
Hosts also have to examine packets
Flags determine the semantics of the packet

- SYN
- SYN&ACK
- FIN
- URG

The Meaning of Packets

What do the following packets mean

- SYN
- SYN&ACK
- SYN&FIN
- SYN&ACK&FIN

It depends on who you ask

The Semantics of “Connection Attempt”

One definition: SYN & !ACK

Under this definition, any of the following packets would be “connection attempts”

- SYN&FIN
- SYN&URG
- SYN&URG&FIN

Another definition: SYN and *nothing else*

The Semantics of Connection Teardown

One definition: FIN and anything else

- SYN&FIN
- URG&FIN

Another definition

- FIN and nothing else

What, exactly, does SYN&FIN mean? Connection attempt or connection teardown?

If the internal hosts and the firewall disagree, packets can get through that will cause a connection to be established

© 2003 by Carnegie Mellon University

Concerns

- Emulation infidelity shows up in lots of places
- Difficult to detect because IDS systems themselves are forms of emulators

Overview

Defective software is not secure

The response strategy

The TSP-Secure strategy

What is a Vulnerability?

Different people have different definitions.

The CERT/CC has a shared understanding:

- Violates an explicit or implicit security policy.
- Usually caused by a software defect.
- Similar defects are the same vulnerability (e.g. SNMP was 2 vulnerabilities).
- Often causes unexpected behavior.

We specifically excludes from “vulnerability”:

- Trojan horse programs (evil email attachments)
- Viruses and Worms (self propagating code)
- Intruder tools (scanners, rootkits, etc.)

Vulnerabilities are the defects that permit these things to exist.

© 2003 by Carnegie Mellon University

The Response Strategy

The current software security approach could be called a response strategy.

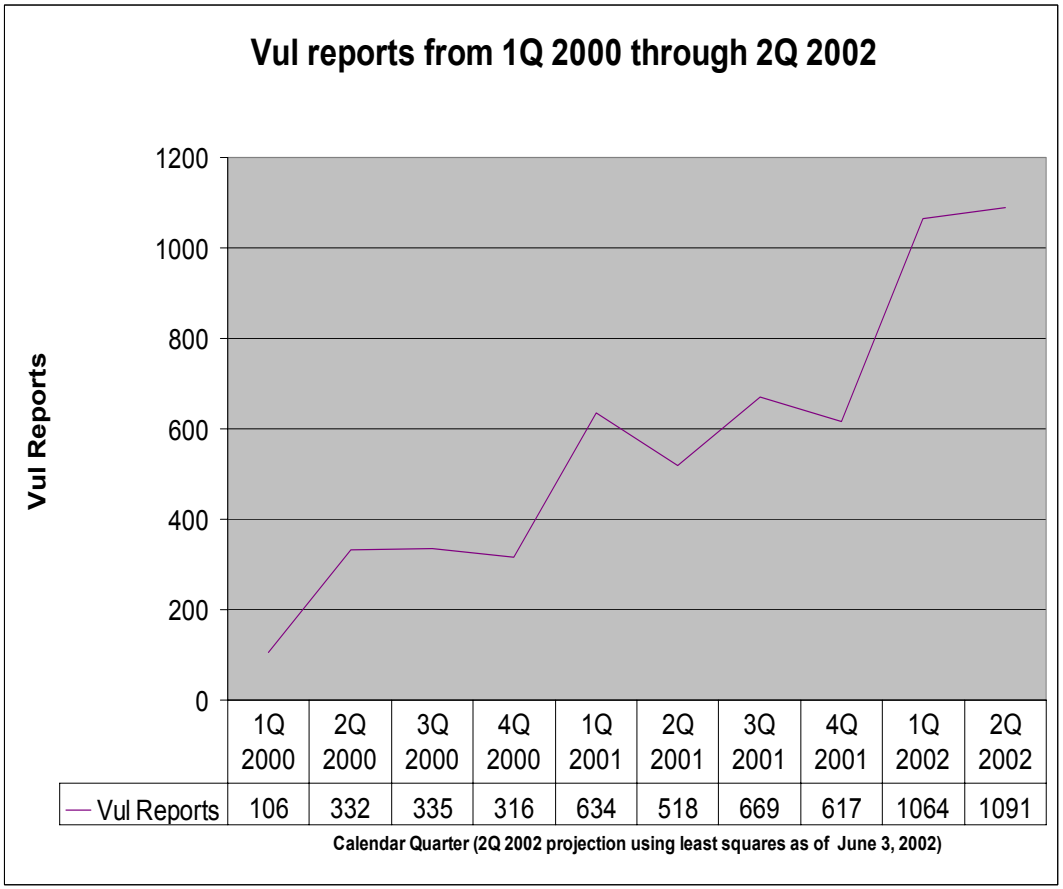
The development of software for secure applications is handled the same way as other software.

This typically results in many delivered defects.

The manufacturer then waits for attackers to find vulnerabilities (vuls) before developing fixes.

The system's users then apply these fixes to prevent further similar attacks.

There Are Too Many Vuls to Patch



The Administrative Workload

5500 vulnerabilities reported in 2002

- 229 days to read vul descriptions (5500 vuls * 20 minutes per vul)
- 69 days to install 10% of patches (550 vuls * 60 minutes per vul)

298 days just to read security news and patch a single system!

The Response Strategy Is Failing

The response strategy accepts the costs of initial attacks.

It is impractical for system administrators.

It is expensive for suppliers.

- excessive development and repair costs
- unknown and possibly unlimited litigation exposures

It only addresses the vuls that are reported. Most security incidents are not reported.

Net: The response strategy cannot consistently or economically produce secure software.

The TSP Security Strategy

TSP-Secure follows a prevention strategy.

- It is an extension of the TSP.
- TSP reduces delivered defects by orders of magnitude.

With proper development and testing, the security defects are similarly reduced.

A continuing response capability is needed to address new types of attack.

- TSP-Secure will be augmented to address new types of attacks.

Overview

Defective software is not secure

The response strategy

➡ TSP-Secure

The Software Quality Problem

Software quality is highly variable and generally poor in non-mission critical systems.

Widely-used operating system and application software typically have over 2000 defects per MLOC.

If only 5% of these defects were potential security concerns, there would be 100 vuls per MLOC.

Software Practice and Quality

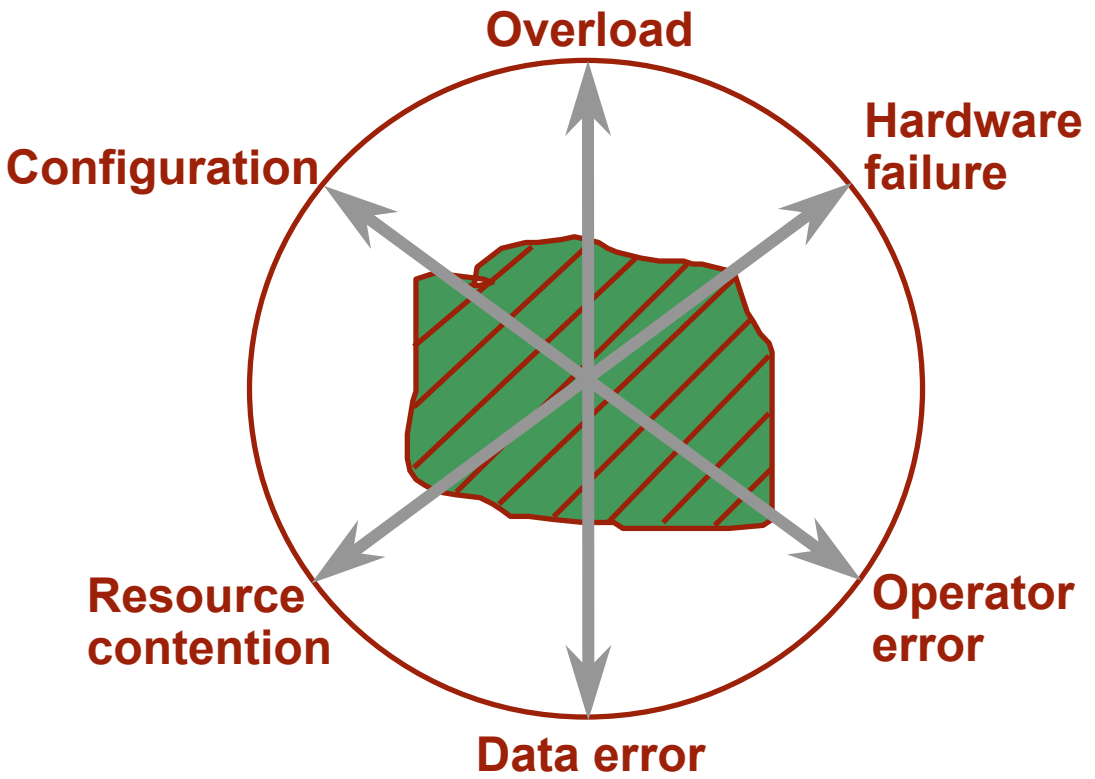
Software is the only modern technology that ignores quality until test. Typically, software engineers

- do not plan their own work
- race through requirements and design
- do the design while coding

These practices introduce volumes of defects.

- Experienced engineers inject a defect every 7 to 10 lines of code.
- For even moderate-sized systems, there are thousands of defects.
- Most of them must be found in test.
- This usually takes about half of the development schedule.

The Problem with Testing



Safe and secure region = tested (shaded)

Unsafe and insecure region = untested (unshaded)

TSP and Secure Software

The TSP provides a framework, a set of processes, and disciplined methods for producing quality software.

Software produced with TSP has 1 to 2 orders of magnitude fewer defects than current practice.

- 0.02 defects/KSLOC vs. 2 defects/KSLOC
- 20 defects per MSLOC vs. 2000 defects per MSLOC

If 5% of the defects are potential security holes, with TSP there would be 1 vulnerability per million SLOC.

© 2003 by Carnegie Mellon University

TSP For Secure Software

TSP for Secure Software (TSP-Secure) is a joint effort of the TSP team and SEI's CERT group.

The work is based on proven TSP quality practices and CERT's extensive security skills and knowledge.

The goal of the project is to develop a TSP-based method that can predictably produce secure software.

TSP-Secure will

- support secure development practices
- predict the likelihood of latent security defects
- be dynamically tailored to respond to new threats

TSP-Secure is being pilot tested with selected TSP teams.

© 2003 by Carnegie Mellon University

First TSP-Secure Pilot Workshop

The purpose of the workshop was to

- convince the team that
  - software security is synonymous with software quality
  - the quality methods the team is already using with the TSP and the PSP can be easily extended to address security issues
- prove the feasibility of using the TSP to develop secure software
  - pilot and test initial ideas for TSP-Secure
  - establish a baseline from which to expand and refine TSP-Secure

Team Characteristics

Team characteristics

- all team members are PSP-trained
- team is using the TSP
- project has security requirements
- team members recently attended corporate security training
- work products produced by the team would be subject to one or more corporate security audits

Workshop –1

Lecture/ Exercise	Topics
Introduction	• Establish relationship between security and quality • Discuss current security issues for this project
Common Vulnerabilities	• Overview of common causes of vulnerabilities • Discuss common causes of vulnerabilities for this project
Secure Practices	• Lecture and discussion on best practices for secure software development.

Workshop –2

Lecture	Topics
Secure Implementation Practices Exercise	Hands-on exercise illustrating the application of secure practices using an Active Server Page (ASP) web application •personal secure code reviews •team secure code inspections
Secure Design Practices	Lecture and discussion on secure design practices
Secure Design Practices Exercise	• Hands-on exercise illustrating the importance of sound design practices when building secure software • Discussion of design practices affecting this project.
Wrap-up	

Results -1

Personal code reviews were conducted using
- a REVIEW script
- an ASP code review checklist

Team inspections were conducted using standard TSP
- inspection script
- form
- measures

Team Yield was 96%!

	Defects		
Engineer Name	Major	Minor	Estimated Yield
Engineer 1	13	3	50%
Engineer 2	15	2	58%
Engineer 3	22	2	85%
Engineer 4	15	1	58%
Engineer 5	13	2	50%
Engineer 6	15	5	58%
Engineer 7	13	2	50%
Engineer 8	16	1	62%
Totals			96%

Results -2

Capture/Recapture predicted 1 defect remaining.

Most surprising result: a working application had a defect density of 36.7 defects/KLOC.

Product Size	708	
Size Measure	LOC	
Total defects found for (A)	15	
Total defects found for all others (B)	24	
Number of Defects Found in common (C)	14	
Estimated Total Major Defects (AB/C)	26	
Number of Major Defects Found (A+B-C)	25	
Number of Major Defects Remaining	1	
Number of Minor Defects Found	5	
Meeting Time (minutes)	60	
Total Meeting Effort (hours)	8.00	
Total Inspection Hours	14.00	
Overall Review Rate	32.2	
Defects/Hour	1.1	
Estimated Defect Density	36.7	per KLOC

© 2003 by Carnegie Mellon University

Results -3

A TSP script for secure high-level design was used to re-design the sample web application.

The exercise illustrated

- the importance of state-machine design and analysis
- that most security issues are best addressed during design

Feedback and Project Results

The feedback from attendees of the workshop was very positive.

- security training is a necessary prerequisite
- need to develop specific scripts, checklists, and other practices
- proved the feasibility of TSP-Secure

The team completed the project on schedule with very few overall defects (.49 defects/KLOC).

Security audit performed by security experts found ZERO security code defects.

Next Steps

- conduct several more prototype workshops
- define security patterns
- develop the full workshop
- develop instructor training
- formulate introduction plans
- establish updating system to rapidly include any new security threats as they are discovered

Summary

TSP helps organizations to establish a mature and disciplined software engineering practice that

- improves cost and schedule predictability
- reduces time to market
- produces high-quality, reliable software, with fewer security-related defects

TSP for Secure Software

- builds on the TSP/CERT background
- creates a development process for secure software
- provides introduction and transition capabilities

For More Information

Visit the TSP web site

http://www.sei.cmu.edu/tsp/

Contact Presenters

Noopur Davis nd@sei.cmu.edu
Shawn Hernan svh@sei.cmu.edu

Contact SEI customer relations

Software Engineering Institute,
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Phone, voice mail, and on-demand FAX: 412/268-5800
E-mail: customer-relations@sei.cmu.edu

See the book

Winning With Software: an Executive Strategy, by Watts Humphrey, Addison-Wesley, 2002