

## Secure Software

---

*If a path to the better there be, it  
begins with a full look at the worst.*

-- Thomas Hardy

## References

---

- Secure Coding: Principles and Practices by Mark G. Graff and Kenneth R. van Wyk, O'Reilly, 2003.
  - First chapter is available at <http://www.oreilly.com/catalog/securecdng/chapter/index.html>

## References

---

- The CERT Coordination Center is a centre of Internet security expertise located at the Software Engineering Institute at Carnegie Mellon University.
- They handle computer security incidents and vulnerabilities, publishing security alerts, R&D into security, and education  
<http://www.cert.org>

## The Problem - Software Attacks

---

- CERT statistics predict that 5000 bugs will be discovered in 2003.

Year	2000	2001	2002	2003 1Q
Vulnerabilities	1,090	2,437	4,129	959

## Estimates of the Scope of the Problem

- There may be 1 security bug per 1000 LOC.
  - An OS can contain 100 million LOC.
- Even if bugs are detected, security patches may inject other bugs.
  - It is estimated that 10% to 15% of all fixes introduce new bugs.

## Vulnerability/Patch/Alarm Cycle



## What is an Attack?

---

- An attack is any malicious act against a system or systems.
- An attack has
  - Goals / subgoals
  - Activities and Events
  - Consequences and Impact

## Types of Attacks

---

- Architectural / Design
- Implementation
- Operations

## Architectural / Design

---

- The flaws that lead to these kinds of attacks are the hardest to fix because they occur at such an early stage of the life cycle.
- The design issue that can leave a system vulnerable are not necessarily mistakes.
  - The design may achieve the desired functionality but not address potential security issues, e.g. telnet.

## Architectural / Design - Attacks

---

- *Man-in-the-Middle or Eavesdropping*
  - Interception of a network transmission between two hosts.
  - Masquerade as one of the two and then insert additional items into the exchange.
- *Defense*
  - Encryption
  - Session checksums and shared secrets

## Architectural / Design Attacks

---

### ■ *Race Condition*

- Time step between parts of a process may be large enough to allow an attacker to compromise the system.
- The result often depends on the order in which parallel processes complete.

### ■ *Defense*

- Use atomic (indivisible) operations as opposed to non-atomic (divisible) ones.

## Architectural / Design Attacks

---

### ■ *Defense for Race Condition (cont'd)*

- Be aware of processes with security implications, e.g.
  - Opening a file
  - Invoking a subroutine
  - Checking a password
  - Verifying a username

## Architectural / Design Attacks

---

- *Replay*
  - The attacker captures an entire transaction between two parties and then replays part of it (impersonation).
- *Defense*
  - Encryption
  - Introduce elements into the transmission that will differ from session to session such as a session number or date/time.

## Architectural / Design Attacks

---

- *Sniffer*
  - A sniffer is a program that silently records all traffic sent over a LAN.
  - It can record sensitive plaintext.
- *Defense*
  - Encryption
  - Network level solutions

## Sniffer Software

---

- Free packet sniffers are listed at <http://www.netsecurity.about.com/library/blfreepacsniiff.htm>
- Ethereal is a free network sniffer.
- From their website: [www.ethereal.com](http://www.ethereal.com)

Ethereal is a free network protocol analyzer for Unix and Windows. It allows you to examine data from a live network or from a capture file on disk. You can interactively browse the capture data, viewing summary and detail information for each packet.

## Implementation Attacks

---

- *Buffer Overflow*
  - This can occur in program that allocate fixed length buffers (prevalent in the programming language C).
  - Example: character string for user input.
  - If the application does not perform adequate bounds checking then it may accept more characters than it can safely store.

## Implementation Attacks

---

- *Buffer Overflow (cont'd)*
  - An attacker can cause the buffer to overflow on purpose and insert unauthorized executable commands into the code.
- *Defense*
  - Do not use languages that allow this feature.
  - Avoid reading text strings of indeterminate length into fixed size buffers.

## Buffer Overflow

---

- CERT Advisory CA-2003-12
- There is a remotely exploitable vulnerability in sendmail that could allow an attacker to gain control of a vulnerable sendmail server. Due to a variable type conversion problem (char to signed int), sendmail may not adequately check the length of address tokens. A specially crafted email message could trigger a stack overflow.

## Implementation Attacks

---

- *Back Door*
  - A back door is code inserted by a programmer into a program to allow access control to be bypassed.
- *Defense*
  - Check all code for back doors - SQA.

## Implementation Attacks

---

- *Parsing Error*
  - A user may input malicious data that will be accepted by the program causing undesirable behaviour.
  - Occurs when a program does not check the input data for safety, e.g. web servers not checking for “..” in URLs.
- *Defense*
  - Use data checking tools.

## Operations Attacks

---

- *Denial-of-Service*
  - This is caused by a high volume of service requests or input to a system.
  - The system cannot handle this volume and consequently service is denied to legitimate users.
- *Defense*
  - Make modest demands on system resources

## Operations Attacks

---

- *Defense (Denial-of-Service)*
  - Make modest demands on system resources, such as disk space, number of open files, number of connections, etc.
  - Monitor resource utilization and plan for excessive load.

## Operations Attacks

---

- *Default Accounts*
  - Many systems are installed with easy to guess default user account names and passwords, e.g. guest/guest, field/service.
- *Defense*
  - Remove all default accounts after installation.
  - Check for default after any installation of new software or upgrades.

## Operations Attacks

---

- *Password Cracking*
  - There are cracking programs that use special algorithms and dictionaries of common words and phrases to generate guesses.
  - These guesses are checked against the password file or against login accounts.
- *Defense*
  - Require robust passwords.

## Factors that work against Secure Code

---

- Technical
- Psychological
- Real-world

## Technical

---

- Mistake by side-effect
  - Happens when the designer/programmer does not fully understand what a particular function does, e.g. allocating memory, reading/writing to disk.
  - Happens when components are combined and errors or misunderstands are exposed.
- Complexity!!!!!!

## Real-World

---

- Who is the source of the source code?
- Lack of software engineering education.
  - Standards and metrics are needed.
- Production pressures.
  - Testing time is limited.
- Security is not a priority.
  - Little impact on bottom line.
  - Less security = more features.

## Tragedy of the Commons

---

- Individual short-term gain =  
Group long-term loss.
- Tragedy of the Commons, Garrett Hardin,  
Science 162 (1968) 1243-1248.
- [http://www.garretthardinsociety.org/articles/art\\_tragedy\\_of\\_the\\_commons.html](http://www.garretthardinsociety.org/articles/art_tragedy_of_the_commons.html)

## A Case Study: rlogin

---

From

*rlogin(1): The Untold Story*

by Lawrence R. Rogers, Nov 1998,

Technical Report CMU/SEI-98-TR-017

## The Story of rlogin

---

- rlogin establishes a remote login session from its user's terminal to a remote host computer.
- It passes the terminal type description from the local host to the remote host.
  - rlogin passes the current terminal definition as identified by the TERM environment variable.

## The Coding Defect in rlogin

---

- The value of the TERM environment variable is copied without due care to an internal buffer.
  - Buffer overflow !!!!
  - The buffer is a variable local to the main subroutine; the local host's subroutine linkage information can be overwritten with data from the TERM environment variable.
  - This can transfer control to an arbitrary memory address

## More Bad News

---

- rlogin requires set-user-id root privileges so that it can obtain a port in the range 0-1023.
- When the “smashed stack” instructions are executed they run in full root mode since the setuid call to leave root mode is later in the code.

## The Bad Code

---

```
char term[1024];
```

```
(void) strcpy ( term, (p=getenv("TERM")) ? :  
    "network" );
```

```
Rem = rcmd ( &host, sp->s_port, pw->pw_name,  
    user, term, 0 );
```

```
(void) setuid ( uid );
```

## Exploitation

---

- A well crafted TERM environment variable is usually constructed to smash the stack and start a copy of /bin/sh.
  - The attacker can now run any command as root.

## Mitigation Strategies

---

- Non-executable Stack Regions
  - Remove execution permission from the stack segment of every process on a UNIX system.
  - But, some legitimate uses of an executable stack exist.
  - Not all processor MMUs can enforce this.
  - A corrupted stack could still transfer control to a non-stack-based address.

## Mitigation Strategies

---

- Truncating the Data
  - Some data may be lost and the reaction of the remote host to this is unpredictable.
  - Replace strcpy with strncpy and insert the NULL terminator at the end of the term string.
  - Replace rlogin with a wrapper program that truncates the TERM variable before passing it to rlogin.
    - Does not rely on changing the source code.

## Mitigation Strategies

---

- **Inspect Data for Length and Content**
  - Inspect both the TERM variable's length and its content.
  - Data that does not fit or is incorrect should be replaced by a meaningful default value to ensure predictable results.
  - This can be implemented via a wrapper or via changes to the source code.

## Avoidance Strategies

---

- *Defensive programming practices* strive to allow the programmer to implement the necessary functionality without injecting coding defects that will have to be repaired later in the life cycle.

## Defensive Programming

---

- Assume that bad input to a program is not just an error but can be malicious in intent.
  - Input includes data in files, arguments, environment variables, credentials, open file descriptors, data streams, and system resources.

## Practice 1: Trusting Untrustworthy Data

---

- When data crosses a boundary it must be considered untrustworthy and be made trustworthy before being used.
  - A boundary is an imaginary line separating two potentially competing domains.
  - One of the boundaries in the case of rlogin was between the user and the program.
  - This is the case when a user inputs data or passes arguments to a program.

## The Practice

---

- To make untrustworthy data trustworthy:
  - Identify the boundaries in a program.
  - Identify data that crosses boundaries.
  - Examine the data for correct form and substitute meaningful default values for nonconforming data.

## Practice 2: Shedding Privileges

---

- rlogin has to run as root to gain access to a reserved port as part of the authentication process.
- Actually this is a weak authentication scheme and its use of root privilege makes rlogin's security flaw extremely dangerous.

## The Practice

---

- Use privileges at the beginning of your program.
- Give up privileges immediately after you are done with them.
- Give up privileges in a manner so that they cannot be reclaimed.
  - Temporarily giving up privileges only slows down the attacker.

## The Practice

---

- Write concise privileged code so that it can be thoroughly inspected for defects.
- Isolate code segments that require privileges.
  - Change the programming interfaces to vendor-provided software.

## Other Sources of Information on Secure Code

---

- Fyodor's Good Reading List on insecure.org  
<http://www.insecure.org/reading.html>
- Must reads are
  - Mudge's tutorial on writing buffer overflows
  - Smashing the Stack for Fun and Profit
  - Murphy's Law and Computer Security
  - Insertion, Evasion and Denial of Service: Eluding Network Invasion Detection

## And for those teams that are using Tomcat...

---

- Have a look at the free chapter on Tomcat Security from the new book, Tomcat: The Definitive Guide at <http://www.oreilly.com/catalog/tomcat>