Exploiting Software: How to Break Code

Gary McGraw, Ph.D.
CTO, Cigital
http://www.cigital.com
Pop quiz

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

Software

badness-ometer
- Founded in 1992 to provide software security and software quality professional services
- Recognized experts in software security and software quality
  - Widely published in books, white papers, and articles
  - Industry thought leaders
So what’s the problem?
Patches are attack maps
Builders versus operators

- Most security people are operations people
  - Network administrators
  - Firewall rules manipulators
  - COTS products glommers
  - These people need training

Security means different things to different people

- Most builders are not security people
  - Software development remains a black art
  - How well are we doing teaching students to engineer code?
  - Emergent properties like security are hard for builders to grok
  - These people need academic education
Attaining software security gets harder

The Trinity of Trouble

- **Connectivity**
  - The Internet is everywhere and most software is on it

- **Complexity**
  - Networked, distributed, mobile code is hard

- **Extensibility**
  - Systems evolve in unexpected ways and are changed on the fly
Who is the bad guy?

- Hackers
  - “Full disclosure” zealots
  - “Script kiddies”
- Cyber criminals
  - Lone guns or organized
- Malicious insiders
  - Compiler wielders
- Business competition
- Police, press, terrorists, intelligence agencies
1995
- Dan Farmer fired from Silicon Graphics for releasing SATAN with Wietse Venema
- FUD: possible attack tool!

2009
- Any system administrator not using a port scanner to check security posture runs the risk of being fired

Fall 2004
- John Aycock at University of Calgary publicly criticized for malware course
- FUD: possible bad guy factory

Should we talk about attacking systems?
The good news and the bad news

**Good news**
- The world loves to talk about how stuff breaks
- This kind of work sparks lots of interest in computer security

**Bad news**
- The world would rather not focus on how to build stuff that does not break
- It’s harder to build good stuff than to break junky stuff
Know your enemy: How stuff breaks
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
Attackers do not distinguish bugs and flaws

- Both bugs and flaws lead to vulnerabilities that can be exploited
- Attackers write code to break code
- Defenders are network operations people
  - Code?! What code?
The attacker’s toolkit

- The standard attacker’s toolkit has lots of (software analysis) stuff
  - Disassemblers and decompilers
  - Control flow and coverage tools
  - APISPY32
  - Breakpoint setters and monitors
  - Buffer overflow
  - Shell code
  - Rootkits
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: buffer overflow foo

- Find targets with static analysis
- Change program control flow
  - Heap attacks
  - Stack smashing
  - Trampolining
  - Arc injection
- 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

<table>
<thead>
<tr>
<th>Function arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Address</td>
</tr>
<tr>
<td>Canary Value</td>
</tr>
<tr>
<td>Frame Pointer</td>
</tr>
<tr>
<td>Local Variable: Buffer A</td>
</tr>
<tr>
<td>Local Variable: Pointer A</td>
</tr>
<tr>
<td>Local Variable: Buffer B</td>
</tr>
</tbody>
</table>
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 exist for
  - X86 (win32)
  - RISC (MIPS and sparc)
  - Multiplatform payloads
Attacker’s toolkit: rootkits

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

- See *Hacking World of Warcraft: An exercise in advanced rootkit development*
  - Greg Hoglund’s presentation from Black Hat 2006
State of the art

- Combine injected payload with cloaking and thread hijacking to FORCE in-game events
  - Spell casting
  - Movement
  - Chat
  - Acquire and clear targets
  - Loot inventory
Attacker’s toolkit: other miscellaneous tools

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

The standard process
- Scan network
- Build a network map
- Pick target system
- Identify OS stack
- Port scan
- Determine target components
- Choose attack patterns
- Break software
- Plant backdoor

Attacking a software 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
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()
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 (opera cookie foo)
Breaking stuff is important

- Learning how to think like an attacker is essential
- Do not shy away from discussing attacks
  - Engineers learn from stories of failure
- Attacking projects is useful
Great, now what do we do about this?
Three pillars of software security
1. Risk management framework
2. Touchpoints
3. Knowledge
Software security touchpoints
What works: BSIMM

- Building Security In Maturity Model
- Real data from real initiatives
A Software Security Framework

<table>
<thead>
<tr>
<th>The Software Security Framework (SSF)</th>
<th>Governance</th>
<th>Intelligence</th>
<th>SSDL Touchpoints</th>
<th>Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy and Metrics</td>
<td>Attack Models</td>
<td>Architecture Analysis</td>
<td>Penetration Testing</td>
<td></td>
</tr>
<tr>
<td>Compliance and Policy</td>
<td>Security Features and Design</td>
<td>Code Review</td>
<td>Software Environment</td>
<td></td>
</tr>
<tr>
<td>Training</td>
<td>Standards and Requirements</td>
<td>Security Testing</td>
<td>Configuration Management and Vulnerability Management</td>
<td></td>
</tr>
</tbody>
</table>

- Twelve practices
Ten surprising things

1. Bad metrics hurt
2. Secure-by default frameworks
3. Nobody uses WAFs
4. QA can’t do software security
5. Evangelize over audit
6. ARA is hard
7. Practitioners don’t talk attacks
8. Training is advanced
9. Pen testing is diminishing
10. Fuzz testing

informIT & Justice League

- **www.informIT.com**
- No-nonsense monthly security column by Gary McGraw

- **www.cigital.com/justiceleague**
- In-depth thought leadership blog from the Cigital Principals
  - Scott Matsumoto
  - Gary McGraw
  - Sammy Migues
  - Craig Miller
  - John Steven
IEEE Security & Privacy Magazine + 2 Podcasts

- Building Security In
- Software Security Best Practices column edited by John Steven
- www.computer.org/security/bsisub/

- www.cigital.com/silverbullet
- www.cigital.com/realitycheck
Software Security: the book

- How to DO software security
  - Best practices
  - Tools
  - Knowledge
- Cornerstone of the Addison-Wesley Software Security Series
- www.swsec.com
Cigital’s Software Security Group invents and delivers Software Quality Management

See the Addison-Wesley Software Security series

WE NEED GREAT PEOPLE

Send e-mail: gem@cigital.com

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

-Bill Gates