Software Security Testing:
Seeking security in an insecure world

Gary McGraw, Ph.D.
CTO, Cigital

http://www.cigital.com
Software security is getting 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
Old school security is reactive

- Defend the “perimeter” with a firewall
  - To keep stuff out
- Promulgate “penetrate and patch”
- “Review” products when they’re complete
  - Throw it over the wall testing
  - Too much weight on penetration testing
- Over-rely on security functions
  - “We use SSL”

The “network guy with keys” does not really understand software testing. Builders are only recently getting involved in security.
Making software behave is hard

- Can you test in quality?
- How do you find (adaptive) defects in code?
- What about bad guys doing evil on purpose?

- What’s the difference between security testing and functional testing?
- How can you analyze security design?
- How can you codify non-functional, emergent requirements like security?
- Can you measure security?
Software vulnerability growth

Software Vulnerabilities

- 2000: 1090
- 2001: 2437
- 2002: 4129
- 2003: 3784
- 2004: 3780
- 2005: 5690
- 2006: 8064
- 2007: 7236

Number of Software Vulnerabilities
The classic security tradeoff

Security vs. Functionality

Windows Complexity

Software Vulnerabilities

© 2009 Cigital
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
Security = breaking stuff + building stuff

- Security requires two hats
  - Offense and defense
  - Building and breaking
- Security design based on software engineering
- Security analysis based on attack

- Testing has two flavors
  - Functional security testing (constructive)
  - Risk-based security testing (destructive)
# A Software Security Framework

<table>
<thead>
<tr>
<th>Governance</th>
<th>Intelligence</th>
<th>SDL 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>
</tr>
<tr>
<td>Compliance and Policy</td>
<td>Security Features and Design</td>
<td>Code Review</td>
<td>Software Environment</td>
</tr>
<tr>
<td>Training</td>
<td>Standards and Requirements</td>
<td>Security Testing</td>
<td>Configuration Management and Vulnerability Management</td>
</tr>
</tbody>
</table>

- Four domains
- Twelve practices
- See informIT article
Security as Knowledge Intensive
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)
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)

<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>
Warning! Knowledge can be easily misused

Software security
- Requires input into design and implementation
- High expertise
- Design software to be secure
- Build secure code
- Security analysis
- Security testing
- Inside → Out

“Application security”
- Works for COTS software
- Low expertise
- Protect installed software from harm
- Protection against malicious code
- Policy issues
- Outside → In

badness-ometer
Top 11 reasons why top 10 lists don’t work

1. Executives don’t care about technical bugs
2. Too much focus on bugs
3. Vulnerability lists help auditors more than developers
4. One person’s bug is another person’s yawner
5. Using bug parade lists for training leads to awareness but does not educate.
6. Bug lists change with the prevailing technology winds
7. Top ten lists mix levels
8. Automated tools can find bugs---let them
9. Metrics built on top ten lists are misleading
10. When it comes to testing, security requirements are more important than vulnerability lists.
11. Ten is not enough.

BSIMM-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

Attackers are Software People
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
  - Binary scanners
  - Control flow, data flow, and coverage tools
  - APISPY32
  - Breakpoint setters and monitors
  - Buffer overflow kits
  - Shell code, payloads (multi-platform)
  - Rootkits (kernel, hardware)
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
Is FUZZ good?

- Wisconsin academics invent the notion of sending random noise to UNIX utilities
  - Fuzz I: 1990
  - Fuzz II: ten years later
- Fifteen years later, security people hit on the same idea

- FUZZ can be useful
  - SPIKE
  - Peachfuzz
  - Mangle (HTML)
  - FileFuzz
  - beStrorm
  - Codenomicon

- White box testing is better
Breaking stuff is important

- Learning how to think like an attacker is essential (especially for good testing)
- Think hard about the “can’ts” and “won’ts”
- Do not shy away from teaching attacks
  - Engineers learn from stories of failure
  - Testers must deeply understand how things break
Resources on security testing

- Building Secure Software (Viega/McGraw)
- Writing Secure Code (Howard/LeBlanc)
- How to Break Software Security (Whittaker/Thompson)
- Web Security Testing Cookbook (Hope/Walther)
Web Security Testing Cookbook
What it ain’t
What it IS

- Reference and job aid for web testers
  - Exploratory testing
  - Regression testing
  - Automated testing
  - Unit testing
- Starts *really* basic
- Ends rather complicated
Table of Contents

1. Introduction
2. Installing Free Tools
3. Basic Observation
4. Web-Oriented Data Encoding
5. Tampering with Input
6. Automated Bulk Scanning
7. Automating Tasks with cURL
8. Automating Tasks with LibWWWPerl
9. Seeking Design Flaws
10. Attacking AJAX
11. Manipulating Sessions
12. Multifaceted Tests
Example: Login to eBay

- Series of commands
- Build up state/cookies
- Check for username in output

```
${CURL} -s -L -A "${UA}" -c "${JAR}" \
  -b "${JAR}" -e ";auto" \
  -d MfcISAPICommand=SignInWelcome \
  -d siteid=0 -d co_partnerId=2 -d UsingSSL=1 \ 
  -d ru= -d pp= -d pa1= -d pa2= -d pa3= \ 
  -d il=-1 -d pageType=-1 -d rtmData= \ 
  -d userid="${USER}" \ 
  -d pass="${PASS}" \ 
  -o "step- ${step}.html" \ 
  "https://signin.ebay.com/ws/..."

if [ $? = 0 ]; then
  step=$step+1
  echo -n "OK] [${step} "
else
  echo "FAIL]"
  exit 1
fi
```
Other topics

- LDAP injection
- Zip of death
- Billion laughs
- Pathological XML
- Malicious cookies

- All of these are scripted
- All are repeatable
- All can be part of a routine QA process
Stuff that Works for Cigital
Three pillars of software security

- Risk management framework
- Touchpoints
- Knowledge
Software security touchpoints
Touchpoint: 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

- Abuse cases are great for test planning
Touchpoint: 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
Touchpoint: 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
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!
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
Where to Learn More
informIT & Justice League

- [www.cigital.com/justiceleague](http://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

WE NEED GREAT PEOPLE

See the Addison-Wesley Software Security series

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