Security Testing Fundamentals

SecAppDev 2014
Ken van Wyk, @KRvW

Leuven, Belgium
10-14 February 2014

KRvW Associates, LLC

Copyright© 2014 KRvW Associates, LLC
Confessions of a pen tester

Typical scenario looks like this
Customer calls and asks for a test
2-3 weeks prior to product going “live”
Security test required by auditors
Want to ensure “hackers can’t get in”
How secure are we?

What problems do you see here?
The problem

Too many organizations have either:
  Neglected security testing entirely
  Assumed (incorrectly) their QA testing will catch security issues
  Adopted a late-cycle penetration test process as their sole security test

*When you ask the wrong questions, you won’t get the answers you need!*
Security testing is different

Security focus should primarily be on non-functional aspects of the software
  Not just focused on what the software can or should do
  Active deception of software intent
  Need to test every aspect of app

*QA team often has a tough time “thinking like an attacker”*
Uninformed “black box” testing

Advantages
- Unencumbered by prejudices of how things “should” behave
- Accurately emulates what an outsider might find
- Can be inexpensive and quick

Disadvantages
- Coverage is abysmal (10-20% LOC not abnormal)
- No notion of risk prioritization
Informed testing

Advantages
- Effort can be allocated by risk priority
- Can ensure high coverage through careful test design
- Emulate an insider attack

Disadvantages
- Functional “blinders” might miss things
Testing methods

Common practices include
  Fuzzing
  Penetration testing
  Dynamic validation
  Risk-based testing
Fuzzing

Basic principle
Hit software with random/garbage
Look for unanticipated failure states
Observe and record

Any good?
MS estimates 20-25% of bugs found this way
Watch for adequate coverage
Fuzzing techniques

Smart fuzzing and dumb fuzzing

“Dumb” refers to using random, unchosen data

“Smart” implies using chosen garbage

Example - fuzzing a graphic renderer

• Dumb approach is to throw it randomness

• Smart approach is to study its expected file formats and to construct garbage that “looks” like what it expects, but isn’t quite right
What to fuzz

Fuzz targets
  File fuzzing
  Network fuzzing
  Other I/O interfaces

Constructing “dumb” scenarios for each is easy, so let’s look at some smart approaches
File fuzzing

Smart scenarios

- Really study the expected file format(s)
- Look for things like parameters in data
- Construct nonsensical input data parameters
  - Negative or huge bitrate values for audio/video
  - Graphic dimensions
Network fuzzing

Smart scenarios

Really study the software-level network interfaces
  • Coverage here must include state

Look for things like flags, ignoring state

Also, HTTP/HTTPS interfaces
  • GET/POST
  • SOAP and RESTful interfaces too
    Don’t stop with the functions specified in the WSDL

Construct nonsensical input data parameters
  • “Insane” packet sizes
  • Data overflows and underflows
Interface fuzzing

Smart scenarios for all other “surfaces”

Really study the data interfaces

- APIs, registry, environment, user inputs, etc.

Construct nonsensical input data parameters

- Overflows and underflows
- Device names when file names are expected
Automation is your friend...

…and your foe

Lots of fuzz products are appearing

How can one size possibly fit all?

How would you fuzz a browser Javascript (AJAX) function?

Best fuzzing tools are in fact frameworks

Examples

OWASP’s JBroFuzz, PEACH, SPI Fuzzer
Finding value in pen testing

Enough with what’s wrong
Consider informed testing
Quick form of attack resistance analysis
Risk-based prioritization
Nightmare scenarios from architectural risk analysis
Abuse case stories
Start with vendor tools, but then roll your sleeves up and do it yourself
  • Scripting tools can help tremendously
Pen testing strategies

Inside => out approach is most likely to yield meaningful results

It doesn’t hurt to also do an outside => in test

One very small part of overall testing

Adversarial approach

Surprises happen
Basic pen testing methods

Target scan
   Take inventory of target space

Vulnerability scan
   What potential preliminary weaknesses are present?

Vulnerability exploit
   Attempt entry

Host-based discovery
   What interesting “stuff” is on each breached system?

Recursive branching
   Repeat until finished
Pen test results

Results need to be actions for dev team
Traditional pen test teams report to IT
Need to adapt to different audience
Map findings to modules and code
Automation is really your friend

Pen test tool market is (arguably) one of the strongest in the security business
  Choices abound in commercial and open source
  Many are quite mature
  Almost a commodity market
Examples include
  Nmap, nessus, Metasploit, ISS, Core Impact, Retina
Dynamic validation

Time to verify all those security requirements and functional specs

QA will have easiest time building test cases with these

Fault injection often used

Helps if requirements verbiage is actionable

Can also be driven by design

Look for key assumptions

- E.g., “session token is always HTTPS”
Automation, what’s that?

Dearth of available tools

Some process monitors are available and helpful
Test cases are easiest to define

Specific tool hints

Web app proxies (work great with mobile apps too)
Single stepping debuggers with key breakpoints
MITM tools between app and OS
Examples – HTTP 1

POST http://www.example.com/AuthenticationServlet HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20100404
Accept: text/xml,application/xml,application/xhtml+xml
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/index.jsp
Cookie: JSESSIONID=LVrRRQQXgwyWpW7QMnS49vtW1yBdqn98CGlkP4jTvVCGdyPkmn3S!
Content-Type: application/x-www-form-urlencoded
Content-length: 64

delegated_service=218&User=test&Pass=test&Submit=SUBMIT
Examples – HTTP 2

POST https://www.example.com:443/login.do HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20100404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: https://www.example.com/home.do
Cookie: language=English;
Content-Type: application/x-www-form-urlencoded
Content-length: 50

Command=Login&User=test&Pass=test
Examples – HTTP 3

POST https://www.example.com:443/login.do HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20100404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/homepage.do
Cookie: SERVTIMSESSIONID=s2JyLkvDJ9ZhX3yr5BJ3DFLkdphH0QNSJ3VQB6pLhjkW6F
Content-Type: application/x-www-form-urlencoded
Content-length: 45

User=test&Pass=test&portal=ExamplePortal
Examples – HTTP 4

GET https://www.example.com/success.html?user=test&pass=test HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20100404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: https://www.example.com/form.html
If-Modified-Since: Mon, 30 Jun 2010 07:55:11 GMT
If-None-Match: "43a01-5b-4868915f"
Risk-based testing

Time to animate those “nightmare scenarios” you uncovered in the architectural risk analysis

Start with abuse cases, weakness scenarios
Describe and script them
Try them one step at a time

Begin at the beginning and go on till you come to the end; then stop. – Lewis Carroll
Automation, what’s that?

Dearth of available tools

It’s rare that these scenarios lend themselves to general purpose automation

Test cases are really tough to define

But many of same tools used in dynamic validation can be useful
Additional considerations

There’s plenty other things to think about
Threat modeling
Results tracking
Five stages of grief
Knowledge sharing
Improvement and optimization
Threat modeling can help drive

Who would attack us?
What are their goals?
What resources do they have?
How will they apply technology?
How much time do they have?

*Answers can help in understanding feasibility of attacks*
Results tracking

Lots of good reasons to track results
  Use again during regression testing
  Ensure closure
  Knowledge transfer of lessons learned
  Justify time spent

Tools can help
Test Director
Five stages of grief

Security testers are often the bearers of bad news
Learn from the Kübler-Ross model

- Denial, anger, bargaining, depression, acceptance
- Watch out for denial and anger!

Understand and anticipate
Diplomacy and tact will optimize likelihood of acceptance
Knowledge sharing

Show the dev team how their code broke
Best way to learn
Learning from mistakes visually is hugely powerful

If a picture tells a thousand words, a live demonstration shows a thousand pictures
Improvement and optimization

Immediate goal is to find defects in today’s software, but preventing future defects is also a worthy goal

- Formalize lessons learned process
- Consider papers, blog entries, etc., to share new findings (once fixed) with others
- Learn from medical community model
Getting started

Some general tips and guidelines

Interface inventory
Let risk be your navigator
Get the right tools for the job
Scripting skills can be very valuable
Interface inventory

Start by enumerating every interface, API, input, output, etc.
This should be done per module as well as per application
List everything
Some call this the “attack surface”
This list should become a target list as you plan your tests
Flow/architecture charts are useful
Risk navigation

The target list is probably too big to do a thorough job
  Prioritize focus in descending risk order
  Follow the most sensitive data first
  Those flow charts will set you free

See now why rigorous testing should be informed?
Test scenario sources – 1

Develop test scenarios throughout SDLC

Start at requirements, such as

- US regs: GLBA, SOX, HIPPA
- ISO 17799 / BS 7799
- PCI
- OWASP’s WASS

Warning, they’re often fuzzy (no pun…)

- SOX says, “Various internal controls must be in place to curtail fraud and abuse.”
Test scenario sources – 2

Also look elsewhere in SDLC for test cases

Abuse cases
• Many cases translate directly to test cases

Architectural risk analysis
• Seek the doomsday scenarios

Code
• Compliance with coding standards
Deployment testing

Rigorous testing of environment

- Network services
- File access controls
- Secure build configurations
- Event logging
- Patch management

Test for all of this

- Not your job? Who is doing it? The pen testers?
References

Some useful additional reading


“The Security Development Lifecycle”, Michael Howard and Steve Lipner
