Contents

- Introduction
  - The security problem, why security is hard

- Security Testing
  - Description, Requirements, Test Planning, Risk Analysis, Penetration Testing, Vulnerabilities, Example Risk Analysis

- Recent Research
  - Examples of recent research in security testing

- Conclusions

- References
Introduction
My Testing Background

National Research Council of Canada

- Senior Research Officer: Information Security Group
  - Researched threat analysis, a basis for testing

Bell-Northern Research / Nortel Networks

- Advisor: Test Technology for Optical Transmission Verification and Optical Transmission Design
  - Software reliability, operations profiling, software design for testability, test tools
- Member of Scientific Staff: DMS Product Test Strategy
  - Formulate test strategy, together with automated testing
- Member of Scientific Staff: Network Planning Tools in Systems Engineering
  - Performed designer testing (unit testing)
Why Secure Software?

• Criticality
  ➢ Software controls and manages
    • manufacturing processes, water supplies, electric power generation and distribution, air traffic control systems, stock market trading systems, defense systems, etc.

• Necessity
  ➢ Internet indispensable for
    • governments, companies, universities, financial institutions

• Ubiquity
  ➢ Software is everywhere in our daily lives
    • Work, home, commute to work, leisure time
Recent Attacks\textsuperscript{5}

- November, 2017, Uber: Uber revealed that it became aware of a data breach in late 2016 that potentially exposed the personal information of 57 million Uber users and drivers.

- September, 2017, Equifax: This is one of the three largest credit agencies in the US. It announced a breach that may have affected 143 million customers, one of the worst breaches ever due to the sensitivity of the data stolen.

- March, 2017, Dun & Bradstreet: This business services company found its marketing database with over 33 million corporate contacts shared across the web.

- Many more!
Securing Software is Hard

- Can you test in quality?
- What about very smart malicious attackers? Attackers with lots of resources? Attackers sponsored by nation states?
- What’s the difference between security testing and functional testing?
- How can you analyze SW designs for security?
- Can you measure security?
Securing Software is Getting Harder

- **Triple Trouble**
  - **Connectivity**
    - The Internet is ubiquitous, and is the host for most software
  - **Complexity**
    - networked, distributed, interdependent
  - **Unpredictability**
    - Systems evolve unexpectedly and are changed without warning
Old Security Model is Reactive

- **React** to attacks by defending the “perimeter” with a firewall to keep bad things out
- **React** to security bugs with “patching”
- **React** to security issues by “reviewing” products only when they’re complete
  - Throw it over the wall testing (insufficient component testing)
  - Depending too much on penetration testing
- **React** to security problems by depending too much on security functions
  - “We employ SSL”
Security Model Must Become Proactive

- **Design for security** by building in security from the start of development
- **Identify vulnerabilities and secure them before they are exploited**
- **Minimize attack surface** as far as possible, e.g. minimize the quantity of sensitive data that is stored online
- **Make it expensive for an attacker to succeed** by using multiple layers of security, e.g. 2–factor authentication
Security Problems are Complex

Implementation bugs (50%)
- Buffer overflow
- Race conditions
- Unsafe environmental variables
- Unsafe system calls
- Untrustable input
- ...

Architectural flaws (50%)
- Misuse of cryptography
- Compartmentalization errors
- Privileged block protection failure
- Catastrophic security failure
- Broken access control
- ...

© 2018 Aptusinnova Inc.
Security Related Bugs Differ from Traditional Bugs

- Users do not normally try to intelligently search out software bugs but malicious attackers intelligently search for vulnerabilities

- Developers can (and do) learn to avoid poor programming practices that can lead to buggy code, but the list of insecure coding practices is long and grows longer every year
Statistics on software vulnerabilities reported since 1988
- Source: NIST: US Dept. of Commerce
Classic Security Tradeoff

Functionality, Complexity

Security

Windows Complexity

SW Vulnerabilities

© 2018 Aptusinnova Inc.
Challenge of Secure Software

- Building secure software systems is one of the greatest challenges of modern times.
  - Security problem has existed for > 40 years: “Efforts to build secure computer systems have now been underway for more than a decade.”
  - Recently the security problems have grown many times worse.
  - Researchers have proposed:
    - integrating security requirements with software functional requirements
    - Identifying specific dangers to watch for during design
    - automatic source code vulnerability checking tools
    - code obfuscation to resist reverse engineering
    - protecting critical memory locations at run time
    - many others

Security = Building + Breaking

- Security requires 2 hats
  - One to build - building secure software based on software engineering
  - One to break - determining how software can be broken based on vulnerabilities and threats

- Security testing has 2 sides
  - Functional security testing (constructive)
  - Risk/threat based security testing (destructive)
(Software) Security Testing

• Use of testing techniques specifically to probe security
  ➢ Goal: Reduce vulnerabilities within a software system
  ➢ Business case: straightforward to justify

  ➢ Testing security functionality

  ➢ Testing vulnerability to malicious attacks
  ➢ Driven by probing undocumented assumptions and areas of particular complexity to determine how software can be broken

  ➢ Testing vulnerability emphasizes what an application must not do rather than what it should do - impacts testing

  ➢ Present throughout the SDLC in various stages
  ➢ Well known form is Penetration Testing (more later)

  ➢ Main activities: risk analysis, test planning, actual testing

© 2018 Aptusinnova Inc.
Indirect Benefits of Security Testing

- May be the only *dynamic analysis* (executing) that the software is ever subjected to, for problems that are better found dynamically
- Help confirm that the developers did not overlook some insecure programming practices
- Help identify and mitigate risks from third-party components, where development artifacts like source code and architecture diagrams are unavailable
- Provide metrics of software insecurity and help raise the alarm when software is seriously flawed from the security standpoint
Bases for Security Testing

- Security requirements consist of
  - Functional security requirements → testing security functionality
  - “Must not” requirements → testing for malicious attacks
    - Harder to obtain and test than functional requirements

- Risk analysis
  - Different forms - threat modeling recommended (more later)
  - Use list of common vulnerabilities as starting point
Security Requirements I

• Sources

• Functional security requirements (aka “positive” requirements)
  • “When a specific thing happens, the software should respond in a certain way”
  • Defined at the start of the SDLC (e.g. from regulatory compliance, security policy, etc.)
  • Defined from mitigations due to risk analysis (e.g. mitigate privacy risk with encryption)
  • Straight forward to test, especially if mapped to software artifact responsible
Security Requirements II

• Sources (cont’d)
  
  • “Must not” requirements (aka “negative” requirements)
    
    • “A specific thing must not happen”
    
    • Defined from risk analysis
    
    • May be difficult to test (e.g. “no module may be susceptible to buffer overflow” - not implemented in a specific place)
  
  • Some security requirements may not be testable, but can neither be refined nor dropped, e.g. “an attacker should never be able to take control of the application”
  
  • Most developers are not security experts, and may not understand how to implement some security requirements
Tests from Negative Requirements

- Test templates describe tests for specific risks and requirements in specific types of modules
  - Captures past experience - use if available

- Incident reports may contain descriptions of successful exploits → tests

- Threat modeling - identified threats → tests, e.g. “script kiddies”

- Requires a deep knowledge of the software and its environment
A Software Security Tester Needs to Understand

- A software component and its environment - how the component can corrupt the environment and vice versa
- The assumptions of the developers (attackers attack the assumptions of developers)
- Different abstraction levels of software, e.g. code level abstraction may show vulnerabilities not visible at the architectural abstraction level
- The mindset of the attacker and be prepared to devise tests that may fall outside the range of normal testing
Test Planning I

• Test Plan Purpose: organize security testing process

• Incorporate both a high-level outline of which artifacts are to be tested and what test methodologies to use

• Include a general description of the tests themselves, including prerequisites, setup, execution, and a description of what to look for in the test result

• Holistic, takes place throughout development process

• Fractal, similar planning activities occur at different abstraction levels
Test Planning II

• Works with risk analysis, which also takes place throughout development
  • Need to devise tests for mitigations identified in risk analysis

• Benefits
  • Provides written record of what needs to be done
  • Allow project stakeholders to sign off on the intended testing effort - helps to obtain stakeholder support
  • Provides a way to measure progress (e.g. report to stakeholders)
  • Records test priorities
Test Planning III

- Regulatory Compliance, Security Policy, Incidents, etc.
- Risk Analysis
- Risks
- Mitigations
- Security 'Must Not' Requirements
- Security Functional Requirements

Test Planning
Security Needs to Permeate SDLC

• **Initiation Phase**
  - Preliminary risk analysis using past experience with similar systems → early focus for test planning
  - Environment?, security needs?, impact of security breach?

• **Requirements and Design Phases**
  - Test Plan: Outline how security requirements will be tested, possibly revise requirements that are not testable
  - Add details to preliminary risk analysis → possible new mitigation features and security requirements

• **Coding Phase**
  - Software available for testing → begin security testing
Security Testing Within Typical Types of Testing I

- **Unit Testing (Developers)**
  - Testing positive security functional requirements - ensure test plan includes these requirements
  - Do not under estimate the security threats to units
  - What assumptions does an unit make about its interactions? Are those assumptions being checked?

- **Integration Testing**
  - Rich in component interactions → security bugs
  - Determine what data can and cannot be influenced by an attacker, e.g. input values, check values where possible
  - Don’t forget error handlers
Security Testing Within Typical Types of Testing II

- System Testing
  - The complete system is attacked
  - Stress testing
    - Software performs differently under stress → security problems, e.g. component disabled due to lack of resources
- Penetration testing
  - Tests the actual artifact that will be deployed
  - Real vulnerabilities uncovered
Penetration Testing I

- Attempt to circumvent the security features of a system based on an understanding of the system design and implementation

- Purpose: identify methods of gaining access to a system by using common tools and techniques used by attackers

- Very labor-intensive and requires expertise to minimize the risk to targeted systems

- Requires rules of engagement, e.g. IP addresses to be tested, identification of restricted hosts, times, etc.

- Can be overt or covert
Penetration Testing II

- Can simulate an inside and/or outside attack
- Incorporate results of risk analysis
- Consists of 4 phases:
  - Planning: rules, management approval, goals
  - Discovery: starts with actual testing, includes vulnerability analysis
Penetration Testing III

- Typical vulnerabilities exploited

- Kernel Flaws
- Buffer Overflows
- Symbolic Links
- File Descriptor Attacks

- Race Conditions
- File and Dir. Permissions
- Trojans
- Social Engineering
Risk Analysis I

- Main ingredient of a secure software development process

- Two main purposes:
  - Forms the basis for risk-based testing
  - Forms the basis for risk prioritization

- Identify threats and vulnerabilities for
  - Development of overall test strategy
  - Particular tests based on the threats, vulnerabilities, and assumptions
  - Increasing test coverage and focus in risky areas
  - Selecting test data inputs based on threats and usage profiling

- Carry out at different abstraction levels (initial concepts, high level design, architecture, code)
Risk Analysis II

- Multiple methods from researchers and vendors, but prototypical approach is:
  - Learn as much as possible about the analysis target (e.g. from specifications, discussions, code)
  - Discuss security issues surrounding the system (e.g. identifying vulnerabilities using tools or lists of common vulnerabilities, mapping out exploits)
  - Determine the probability of compromise (e.g. determine likelihood by comparing attacks against controls or defenses)
  - Perform impact analysis (e.g. determine impact on assets and business goals)
  - Rank risks
  - Develop a mitigation strategy (e.g. recommend countermeasures to mitigate risks)
  - Report findings (describe risks, impacts, where to spend resources)
Vulnerabilities for Risk Analysis I

- From those identified for penetration testing earlier

   - Kernel Flaws
   - Buffer Overflows
   - Symbolic Links
   - File Descriptor Attacks
   - Race Conditions
   - File and Dir. Permissions
   - Trojans
   - Social Engineering
Vulnerabilities for Risk Analysis II

- From 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 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()
Risk Analysis Example: Threat Modeling I

- Threat modeling (aka threat analysis): a method for systematically assessing and documenting the security risks associated with a system\(^2\)

- Some terminology: asset, attack path, threat, threat model, threat profile, threat tree or attack tree, vulnerability, vulnerability landscape

- Method for system threat modeling (based on Salter et al.\(^3\)):
  - Identify threats.
  - Create attack trees for the system.
  - Apply weights to the leaves.
  - Prune the tree so that only exploitable leaves remain.
  - Generate corresponding countermeasures.

---


Method for system threat modeling:

- **Identify threats**: examine all available details of the system and enumerate possible threats
- **Create attack trees for the system**: for each threat, take the attacker’s view and find the weak points in the system and the paths which will lead to realizing the threat
- **Apply weights to the leaves**: for each leaf, assign qualitative values for risk, access, and cost to the attacker
- **Prune the tree so that only exploitable leaves remain**: prune leaves that represent objectives that are beyond the attacker’s capabilities or that offer an inadequate return
- **Generate corresponding countermeasures**: identify countermeasures for the remaining (most exploitable) attack paths
MMORPGs (Massively Multiplayer Online Role-Playing Games)

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Connection</td>
<td>Connected to host server through Internet</td>
</tr>
<tr>
<td>Player Authentication</td>
<td>UserID and Password</td>
</tr>
<tr>
<td>Game Objectives</td>
<td>Accumulate virtual property through skillful game play to reach game objectives.</td>
</tr>
<tr>
<td>Number of Players</td>
<td>A large number of players can all compete with one another playing the same instance of the game.</td>
</tr>
<tr>
<td>Payment for Use</td>
<td>Pay for network connection time by buying a card associated with a certain amount of connection time via a serial number on the card.</td>
</tr>
</tbody>
</table>

¹e.g. World of Warcraft

Risk Analysis Example: Threat Modeling III
Identify threats:

By considering the characteristics of a MMORPG system, obtained the following list of potential threats from an attacker:

- Gain illegal access to play the game
- Cheat at game play
- Disrupt game play
- Cheat at paying for game play
- Steal proprietary parts of the software

These threats lead to 5 attack trees. We will consider the attack tree for “steal proprietary parts of the software”
**Attack tree for “steal proprietary parts of the software”**

1. **Threat:** Steal proprietary parts of the software.
2. **Insider arranges theft.**
3. **Force insider cooperation using personal threat.**
4. **Insider takes revenge on company for some perceived injustice.**
5. **Attack on server containing desired software.**
   - **Use Trojans to transmit desired code or design documents.**
   - **Use social engineering to acquire proprietary information.**
   - **Kidnap members of design team.**
7. **Illegally enter premise and copy code.**
8. **Hack into server and copy code.**
9. **Bribe unscrupulous insider.**
Risk Analysis Example: Threat Modeling VI

- **Apply weights and prune (combination of M’s and at least 1 H)**

![Threat Model Diagram]

- Insider arranges theft.
- Attack on server containing desired software.
- Use Trojans to transmit desired code or design documents. (L,L,L)
- Use social engineering to acquire proprietary information. (M,M,L)
- Kidnap members of design team. (H,M,M)
- Bribe unscrupulous insider. (M,M,H)
- Force insider cooperation using personal threat. (H,L,L)
- Insider takes revenge on company for some perceived injustice. (L,M,L)
- Illegally enter premise and copy code. (H,M,M)
- Hack into server and copy code. (L,M,L)
Risk Analysis Example: Threat Modeling VII

- **Identify countermeasures (in **yellow**)**

  (Threat) Steal proprietary parts of the software.

  - Insider arranges theft.
    - Force insider cooperation using personal threat. *(H,L,L)*
    - Increase physical security and penalties.
  - Attack on server containing desired software.
    - Insider takes revenge on company for some perceived injustice. *(L,M,L)*
    - Increase organizational sensitivity to employees. Improve org. management and communication with employees.
  - Hack into server and copy code. *(L,M,L)*
    - Use Trojans to transmit desired code or design documents. *(L,L,L)*
      - Scan and eliminate all malware.
    - Use social engineering to acquire proprietary information. *(M,M,L)*
      - Require strict procedures for information disclosure.
  - Use a combination of firewall and intrusion detection. As well, use obfuscation on the executables.

© 2018 Aptusinnova Inc.
Recent Research
Recent Research

- Recent published research articles in the ACM Digital Library deal with the following topics:
  - Testing methods
  - Testing of specific software
  - Test automation
  - Test tools
  - Combinations of the above

- **Testing methods**
    - Extends previous work in combining the attack pattern models with combinatorial testing in order to provide concrete test input.
Recent Research

- **Combination of testing methods and testing of specific software**
    - Proposes an adaptive model-based testing framework to perform automated, large-scale security assessments for OAuth 2.0 implementations in practice.
  - B. Garn et al. (2014), “On the applicability of combinatorial testing to web application security testing: a case study” [8]
    - Reports on a case study done for evaluating and revisiting a recently introduced combinatorial testing methodology used for web application security purposes.
Recent Research

- Combination of testing methods and testing of specific software (cont’d)
    - A planning-based approach is introduced for modeling and testing of web applications. The approach provides for specifying a specific problem and to generate plans, which in turn guide the execution of a program. In this way, new testing possibilities emerge that eventually lead to better vulnerability detection.
    - Presents a technique to automatically detect SQL injection vulnerabilities through targeted test generation; uses search-based testing to systematically evolve inputs to maximize their potential to expose vulnerabilities.
Recent Research

- **Combination of testing automation and testing of specific software**
  - Presents a taxonomy of XML injection attack types and uses it to derive 4 different ways to mutate XML messages, turning them into attacks (tests) automatically; further considers domain constraints and attack grammars, using a constraint solver to generate XML messages that are both malicious and valid, thus making it more difficult for any protection mechanism to recognize them, giving such messages a better chance at detecting vulnerabilities.
Recent Research

• Combination of testing automation and testing of specific software (cont’d)
    • Presents an automatic testing approach to detect a common type of Cross Site Scripting (XSS) vulnerability caused by improper encoding of untrusted data; authors automatically extract encoding functions used in a web application to sanitize untrusted inputs and then evaluate their effectiveness by automatically generating XSS attack strings.

• Combination of test tools and testing of specific software
    • Presents an Attack Traffic Generation (ATG) tool for security testing of in-vehicle CAN bus.
Recent Research

- **Combination of test tools and testing of specific software (cont’d)**
  - Proposes a test-bed for evaluating the security of industrial applications by providing different metrics for static testing, dynamic testing and network testing in industrial settings; uses these metrics and results of the three tests to compare industrial applications with one another from the security point of view.
Conclusions
Conclusions

- Security testing consists of testing security functionality and testing vulnerability to malicious attacks
- Security testing permeates the entire SDLC
- A risk analysis is essential for security testing
- Vulnerabilities to be targeted for testing should be identified in a risk analysis (list of common vulnerabilities used as input)
- Security testing requires the expertise of some one who understands security, development, and testing
References


References (cont’d)


Thank you!

Questions?