# Software Security Testing

George Yee
Aptusinnova Inc. and Carleton University
Ottawa, Canada

SECURWARE 2018 Tutorial, September 16, 2018

# 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[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
- …

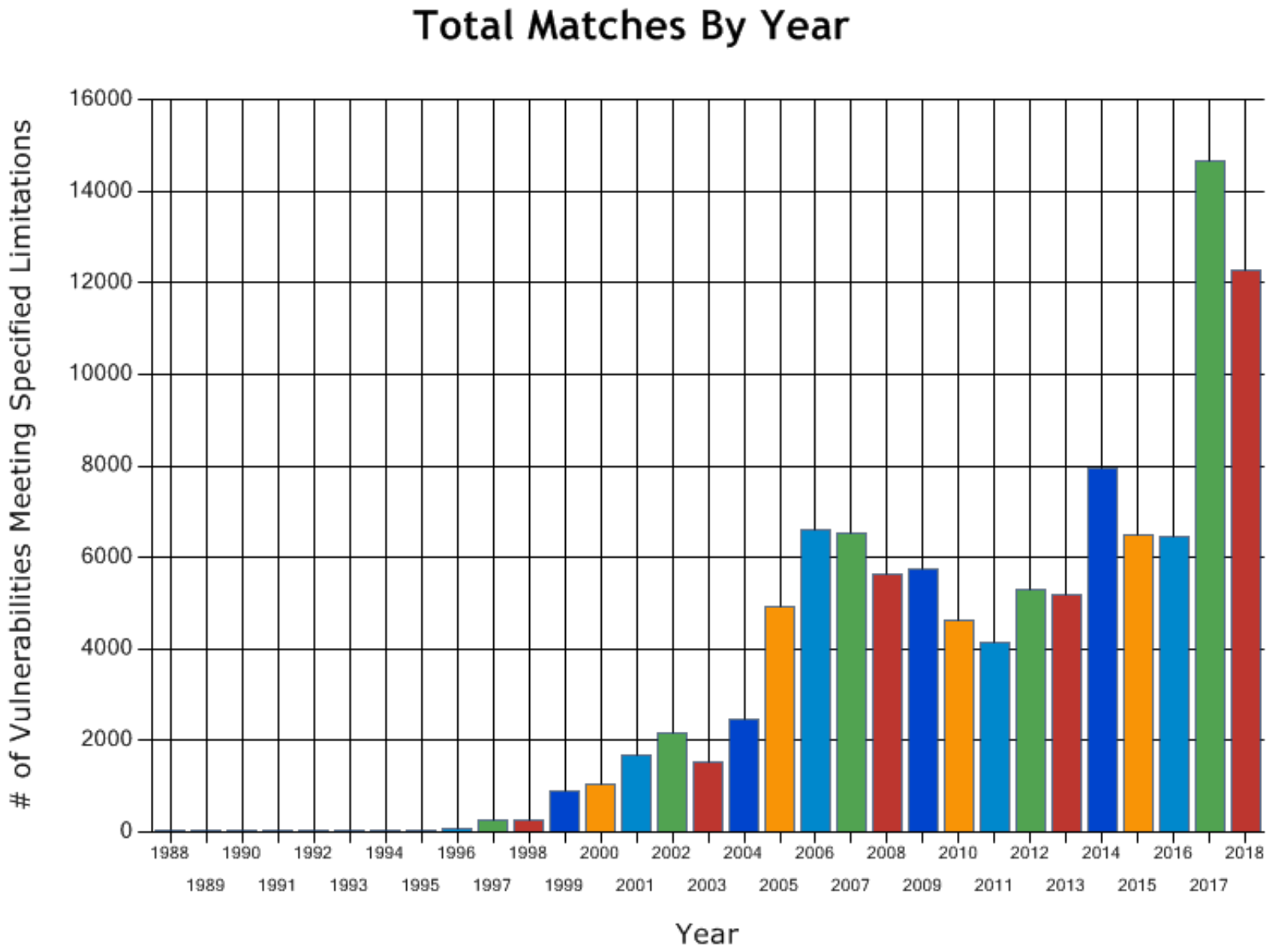
## Architectural flaws (50%)

- Misuse of cryptography
- Compartmentalization errors
- Privileged block protection failure
- Catastrophic security failure
- Broken access control
- …

# Security Related Bugs Differ from Traditional Bugs

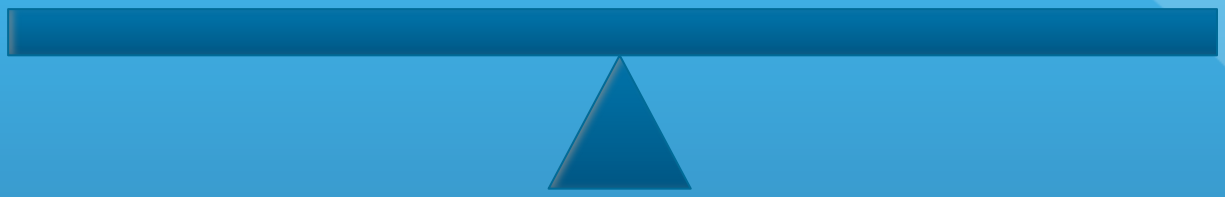
- Users do not normally try to intelligently search out software bugs but malicious attackers intelligently search for vulnerabilities
- Developers can (and do) learn to avoid poor programming practices that can lead to buggy code, but the list of insecure coding practices is long and grows longer every year

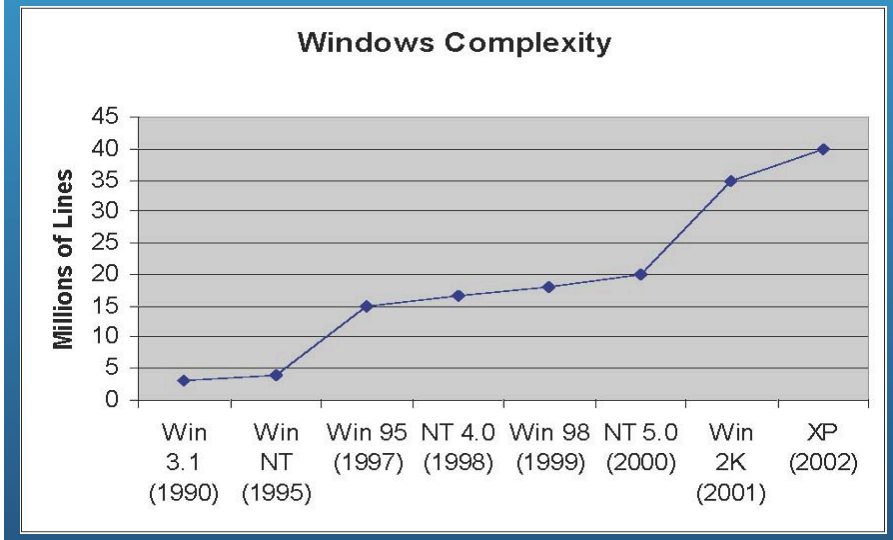
# Statistics on software vulnerabilities reported since 1988 – Source: NIST: US Dept. of Commerce

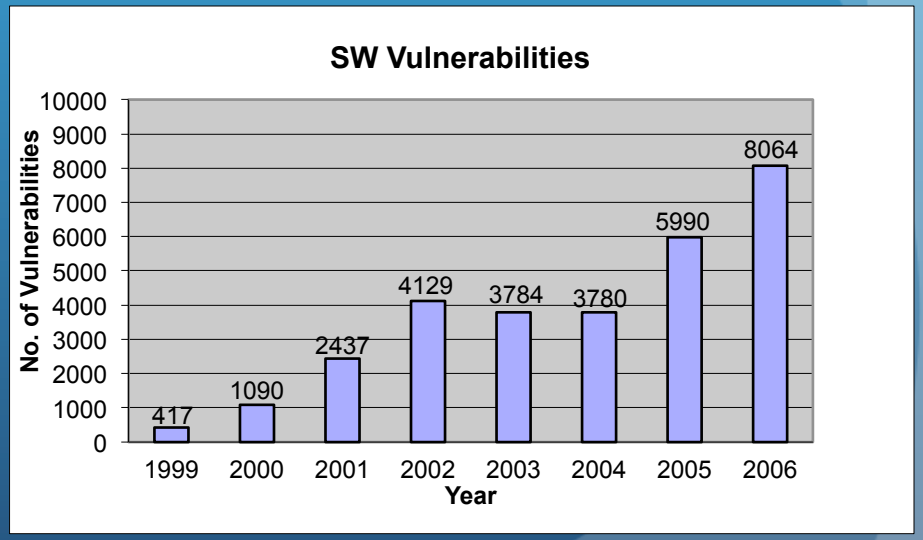


# Classic Security Tradeoff

**Functionality, Complexity** — **Security**





# 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 worst.
  - 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

[+]C.E. Landwehr, "Formal Models of Computer Security", ACM Computing Surveys, Vol. 13, No. 3, September 1981.

# Software Security Testing

# 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

# 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

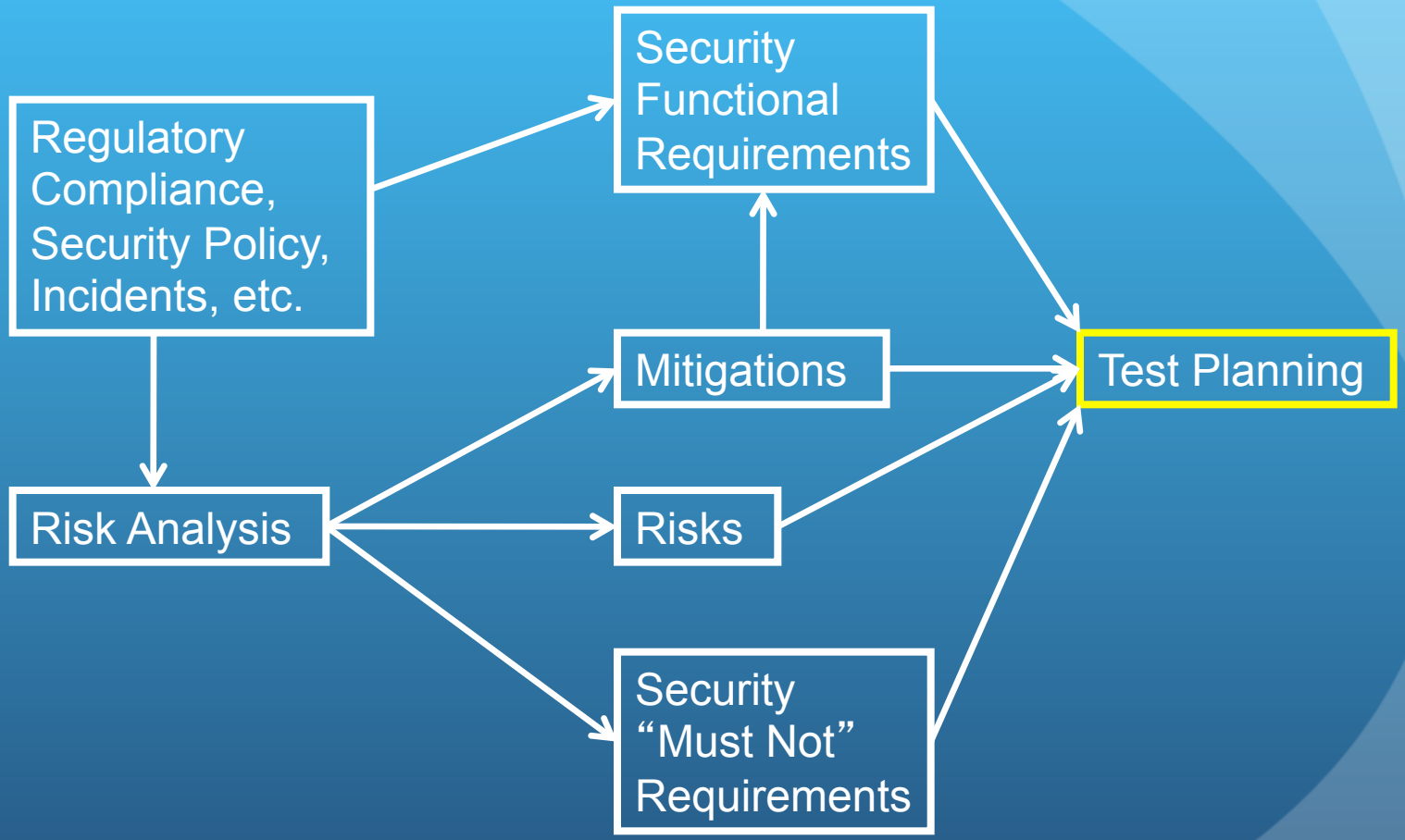
# Test Planning I

- 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



# 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<br>Buffer Overflows<br>Symbolic Links<br>File Descriptor Attacks | Race Conditions<br>File and Dir. Permissions<br>Trojans<br>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.

[2] *Swiderski and W. Snyder, "Threat Modeling", Microsoft Press, 2004.*
[3] *C. Salter, O. Sami Saydjari, B. Schneier, J. Wallner, "Towards a Secure System Engineering Methodology", Proceedings of New Security Paradigms Workshop, Sept. 1998.*

# Risk Analysis Example: Threat Modeling II

Method for system threat modeling:

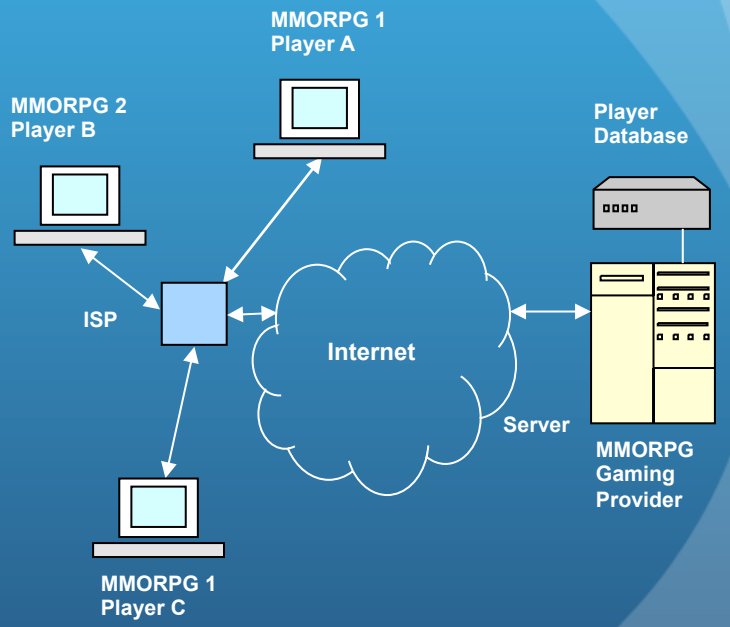
- **Identify threats**: examine all available details of the system and enumerate possible threats
- **Create attack trees for the system**: for each threat, take the attacker's view and find the weak points in the system and the paths which will lead to realizing the threat
- **Apply weights to the 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 counter-measures for the remaining (most exploitable) attack paths

# Risk Analysis Example: Threat Modeling III

## MMORPGs (Massively Multiplayer Online Role-Playing Games[1])

**Table 1. Characterization of MMORPG**

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



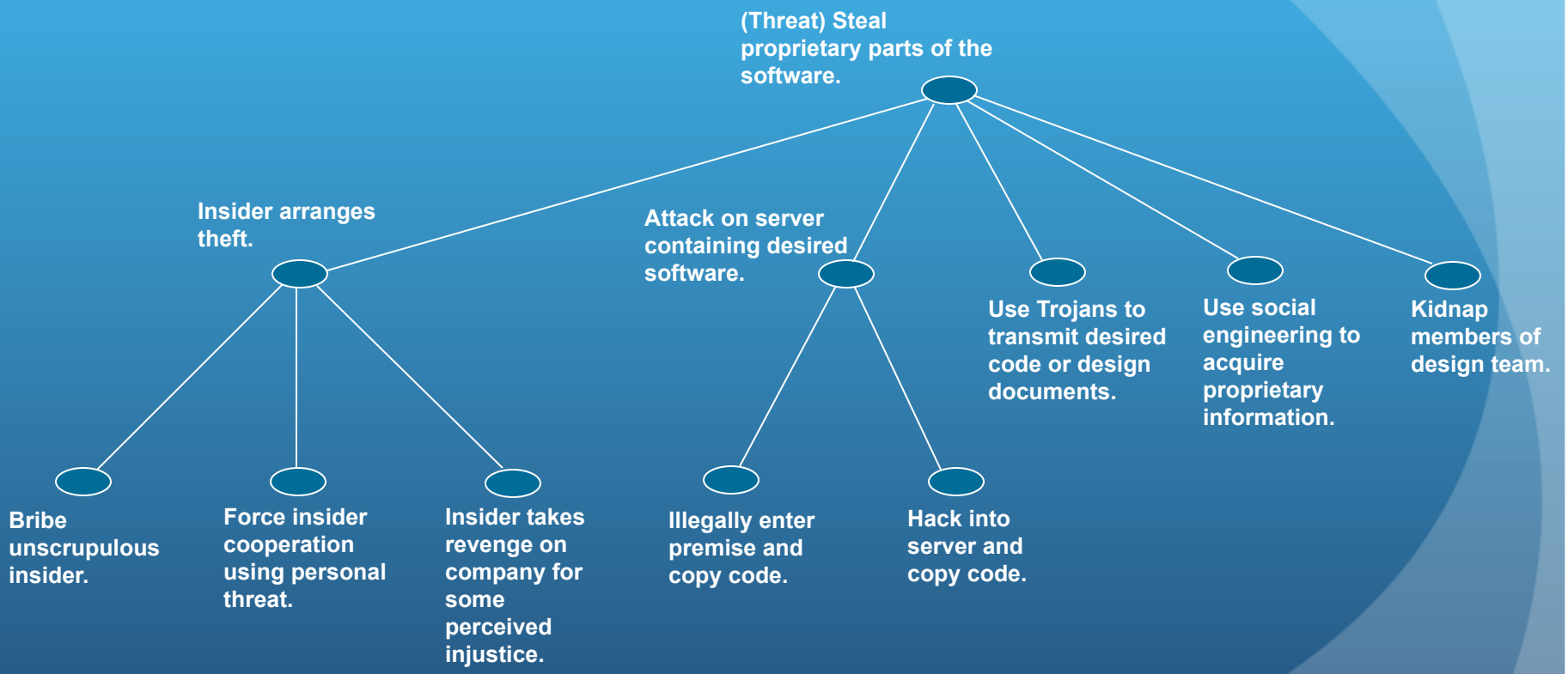

[1]e.g. World of Warcraft

# Risk Analysis Example: Threat Modeling IV

- 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"

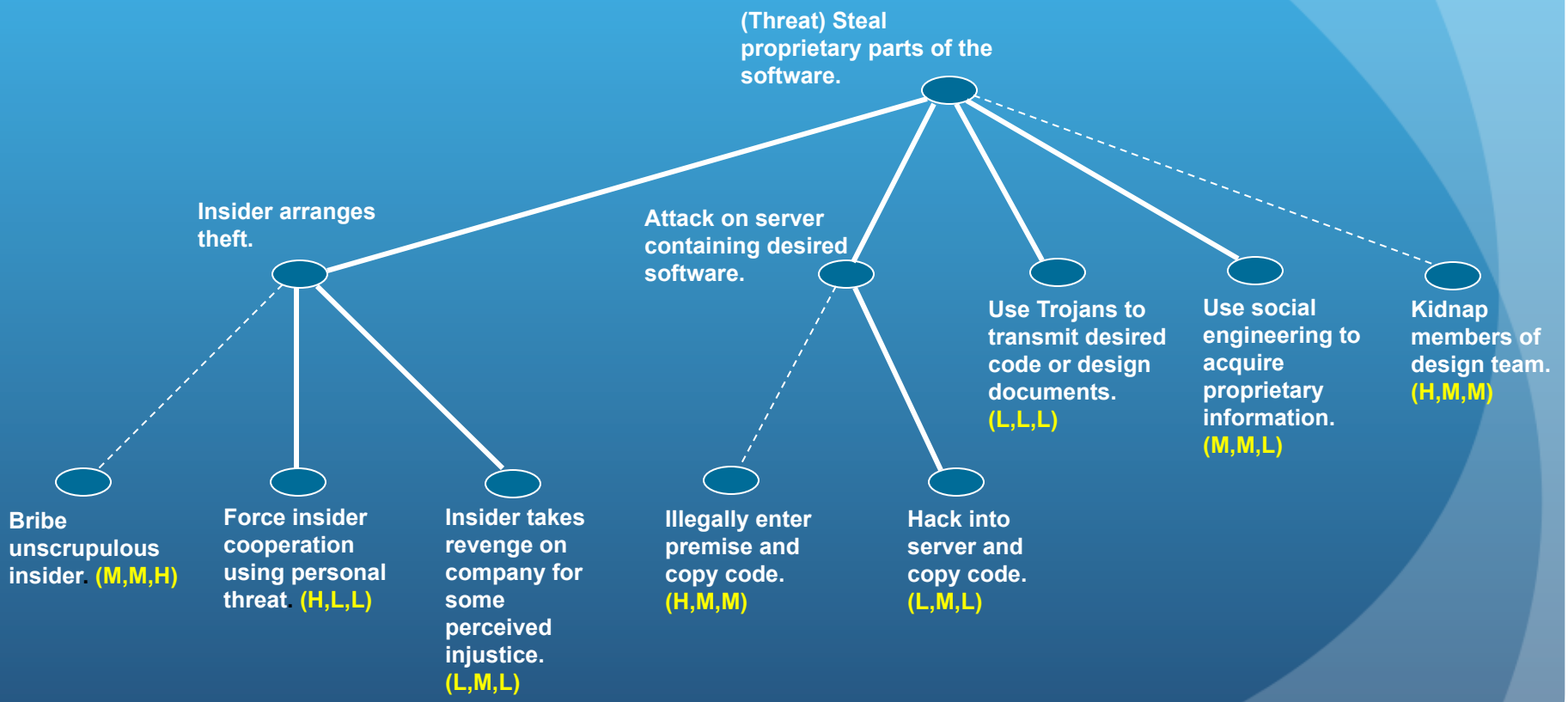
# Risk Analysis Example: Threat Modeling V

- *Attack tree for "steal proprietary parts of the software"*



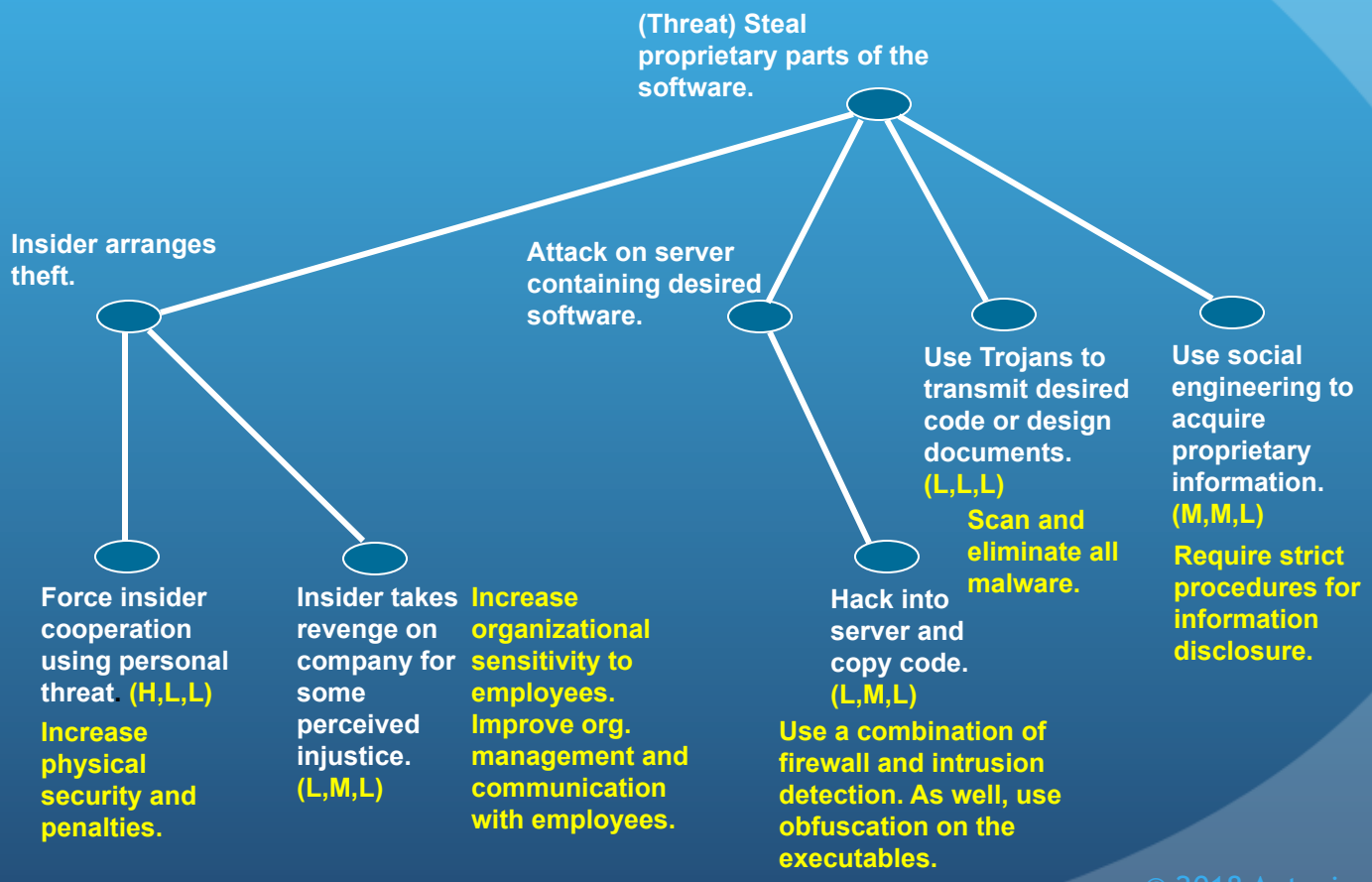
# Risk Analysis Example: Threat Modeling VI

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



# Risk Analysis Example: Threat Modeling VII

- *Identify countermeasures (in yellow)*



# 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
  - J. Bozic et al. (2014), “Attack pattern-based combinatorial testing” [6]
    - 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
  - R. Yang et al. (2016), “Model-based security testing: an empirical study on OAuth 2.0 implementations” [7]
    - 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)
  - J. Bozic and F. Wotawa (2018), "Planning-based security testing of web applications" [9]
    - 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.
  - J. Thomé et al. (2014), "Search-based security testing of web applications" [10]
    - 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
  - S. Jan et al. (2016), “Automated and effective testing of web services for XML injection attacks” [11]
    - 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)
  - B. Chu et al. (2016), "Automatic web security unit testing: XSS vulnerability detection" [12]
    - 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
  - T. Huang et al. (2018), "ATG: An attack traffic generation tool for security testing of in-vehicle CAN bus" [13]
    - 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)
  - M. Azimi et al. (2014), "A security test-bed for industrial control systems" [14]
    - 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


1. C.C. Michael, K. van Wyk, and W. Radosevich, “Risk-Based and Functional Security Testing”, available Sept. 12, 2018 at: https://www.us-cert.gov/bsi/articles/best-practices/security-testing/risk-based-and-functional-security-testing

2. K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh, “Technical Guide to Information Security Testing and Assessment”, NIST Special Publication 800-115, available Sept. 12, 2018 at: https://csrc.nist.gov/publications/detail/sp/800-115/final

3. B. Potter, G. McGraw, “Software Security Testing”, IEEE Security and Privacy, Sept./Oct. 2004.

4. D. Verdon, G. McGraw, “Risk Analysis in Software Design”, IEEE Security and Privacy, July/Aug. 2004.

5. Identity Force, “2017 Data Breaches – The Worst So Far,” retrieved: February, 2018, https://www.identityforce.com/blog/2017-data-breaches

# References (cont'd)


6. J. Bozic, D. E. Simos, F. Wotawa, "Attack pattern-based combinatorial testing", Proceedings of the 9th International Workshop on Automation of Software Test (AST 2014), pp. 1-7, 2014.

7. R. Yang, G. Li, W. C. Lau, K. Zhang, and P. Hu, "Model-based security testing: an empirical study on OAuth 2.0 implementations", Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIA CCS '16), pp. 651-662, 2016.

8. B. Garn, I. Kapsalis, D. E. Simos, and S. WInkler, "On the applicability of combinatorial testing to web application security testing: a case study", Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing (JAMAICA 2014), pp. 16-21, 2014.

9. J. Bozic and F. Wotawa, "Planning-based security testing of web applications", Proceedings of the 13th International Workshop on Automation of Software Test (AST '18), pp. 20-26, 2018.

# References (cont'd)

10. J. Thomé, A. Goria, and A. Zeller, "Search-based security testing of web applications", Proceedings of the 7th International Workshop on Search-Based Software Testing (SBST 2014), pp. 5-14, 2014.

11. S. Jan, C. D. Nguyen, and L. C. Briand, "Automated and effective testing of web services for XML injection attacks", Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016), pp. 12-23, 2016.

12. B. Chu, H. R. Lipford, and E. Murphy-Hill, "Automatic web security unit testing: XSS vulnerability detection", Proceedings of the 11th International Workshop on Automation of Software Test (AST '16), pp. 78-84, 2016.

13. T. Huang, J. Zhou, and A. Bytes, "ATG: An attack traffic generation tool for security testing of in-vehicle CAN bus", Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES 2018), article no. 32, 2018.

# References (cont'd)

14. M. Azimi, A. Sami, and A. Khalili, "A security test-bed for industrial control systems", Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation (MoSEMInA 2014), pp. 26-31, 2014.

# Thank you!

# Questions?