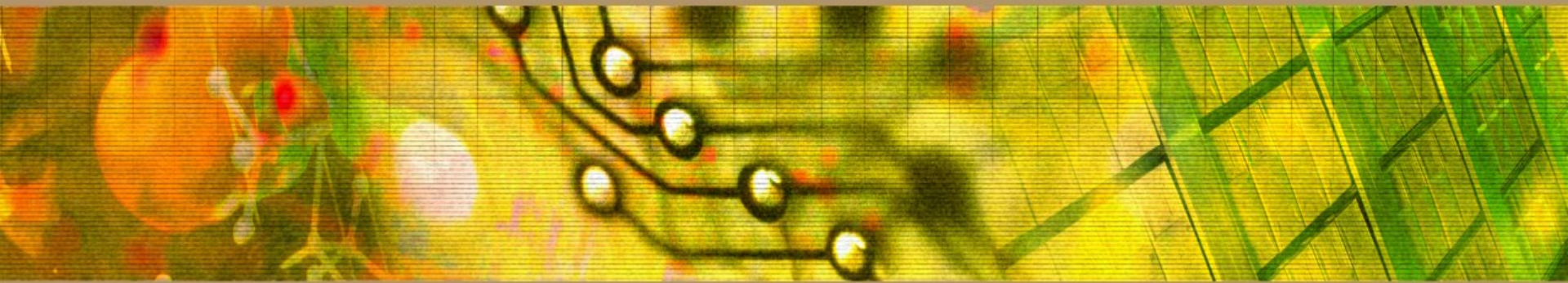


CISSP: Application Security

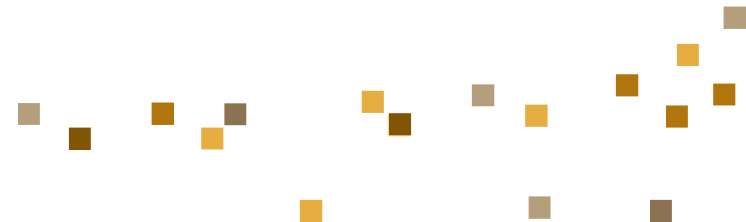


Presented by Pascal Meunier, M.Sc., Ph.D., CISSP
Purdue University CERIAS
August 21, 2007

- This presentation:
 - won't teach you everything you need to know
 - will present an overview of the subject area
 - » this will help you prepare your plan of study and identify areas you need to strengthen

- From the (ISC)² Store:
 - Applications Development & Programming Concepts
 - Audit and Assurance Mechanisms
 - Malware
 - Database and Data Warehousing Environments
 - Web Application Environments

- Software Engineering
- Threats and Countermeasures
- Database Systems security
- Web Application Security



- Software controls
 - vs environment
- Software lifecycle
 - Change control
- Object-oriented components
- Some technologies
 - Java, etc...



- Appliances (e.g., firewalls) do not provide complete security
 - Insider attacks
 - Bypassed by proxy servers, VPNs
- Same software can be used by different users with different privileges
- Many security layers help

- Reactive (expensive and slow)
 - Vulnerability scanning and patching
 - Intrusion detection
 - Incident response
- Proactive
 - More secure software and technologies
 - » Designed
 - » Configured

- **Black Lists**
 - Forbid known bad things (or look for them)
 - Always one step behind
 - Prone to failure as attackers figure out ways around the blocks
- **White Lists**
 - Allow only what's known to be safe
 - Make it provably safe

- Requirements
- Design
- Construction
- Testing
- Maintenance
- See: “Software Engineering Body of Knowledge”
a.k.a. “SWEBOK”

- Configuration Management
- Engineering Management
- Engineering Process
- Engineering Tools and Methods
- Software Quality
 - Audit and Assurance
 - CMMI (Capability Maturity Model Integration)
- Knowledge Areas of Related Disciplines

- Security functional
 - Logging
 - Password quality checks
- Security Assurance
 - Testing
 - Code audits
 - » Code checking tools
- See lists in Common Criteria (NIST)

- Threats vs countermeasures (controls)
 - Threat modeling
 - Threat assessment
 - Risk analysis
 - Risk assessment

- Read as “things that improve security”
 - Prevent
 - Detect
 - Correct

} policy violations

- Using means
 - Physical
 - Administrative
 - Technical (a.k.a. logical)

- Policies
- Procedures
- Standards
- Guidelines
- e.g., must get approval to make a code change

- Software development and production environments should be secured
- Physical access
 - Servers, workstations
 - Network plugs
- Temperature, power
- Activity monitoring
 - Cameras

- Protect data (confidentiality, integrity) and resources (availability)
- Database controls
 - Accounts
 - Roles
 - Transactions
- File system controls
 - Permissions (someone name another?)

- Do the defenses match the risks?
- Cost vs risk
 - Typically more secure systems cost more to build
 - » Not always true, c.f. Spark EAL 5 study
 - But the CISSP test doesn't want to argue fine points
- Security vs functionality

- Vulnerabilities and the Software Development Life Cycle
 - Object-Oriented Programming
 - Capability-Maturity Model
- Malware definitions
- Detecting attacks
 - Expert systems and artificial intelligence

- Feasibility study
- Requirements definition
- Design
- Implementation
- Integration and testing
- Operations and maintenance

- Design (first 3 phases)
- Implementation (phases 4 and 5)
- Operation and maintenance
- *People often like to discuss vulnerabilities according to the SDLC phase in which they were introduced*
 - *sometimes ambiguous*

- “Construction” in SWEBOK corresponds to which SDLC phase?

- Vulnerability: “An instance of an error in the specification, development, or configuration of software such that its execution can violate the security policy” [Krsul 98].
- Flaw: a flaw defines or implies what should have been done to prevent policy violations; it is a problem at a higher level of abstraction that may potentially enable several different attacks and create various vulnerabilities.
 - example flaw: missing input validation

PURDUE **Vulnerability Types**

UNIVERSITY

- It is common to refer to the set of vulnerabilities that enable attack scenario X as “X vulnerabilities”, e.g., “cross-site scripting vulnerabilities”
- Sometimes the name of a technology is used instead of the name of an attack, e.g., “format string vulnerabilities”.
 - Question: name another popularly named vulnerability type.

- A set of processes, techniques and a way of thinking about software engineering to minimize the occurrence of flaws and vulnerabilities
 - In general, fewer bugs mean fewer vulnerabilities
 - » Software engineering methods that prevent bugs are good for security

- Input validation
 - Boundary identification and formation
 - Input awareness
 - Taint tracking
- Secure programming principles
- Code checkers
- Code integrity with formal methods

- Least privilege
- Economy of mechanism
- Complete mediation
- Open design
- Separation of privilege
- Least common mechanism
- Psychological acceptability
- Fail-safe defaults (deny by default; white list)

- Fundamental idea: encapsulate data with the procedures that manipulate it into a logical entity (“object”)
 - Forbid direct access to the data, so that
 - » Invariants can be enforced
 - Data integrity is stronger
 - » Only pre-defined actions can be taken
 - Decreases the possibility of malicious or erroneous actions

- Instance
- Inheritance
- Polymorphism
 - Common methods defined by a common parent class (superclass)
- Polyinstantiation
- Object Request Brokers (ORB)

- Software development processes
 - Level 1: Ad Hoc (heroics)
 - Level 2: Repeatable
 - Level 3: Defined
 - Level 4: Quantitatively Managed
 - Level 5: Optimizing
- From an artisan approach (1) to an engineering science (5)

■ Attack code

- Attack code aims at exploiting vulnerabilities, and is commonly found in the form of attack scripts or proof-of-concept exploits. Worms are another example of attack code. Malicious code isn't necessarily attack code, but its mere presence may imply that the system was compromised by a prior attack. Malicious code resident on a victim computer and performing an undesirable function, such as spyware, rootkits or backdoors, is to be differentiated from attack code that exploits vulnerabilities.

■ Parasitic code

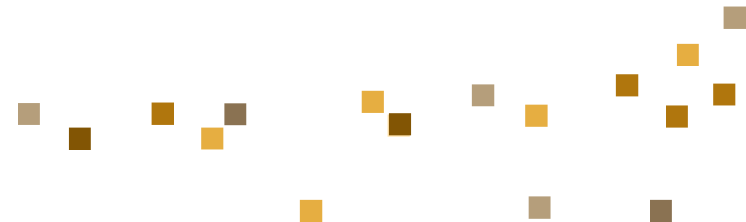
- Parasitic code is code that is attached or included in another document or executable and violates its integrity. Intended or original properties of the document or executable must be identifiable in order to determine the presence, nature and extent of the parasite. Parasitic code is not necessarily attack code.

■ Back-Door

- A back-door is code bypassing policy-approved user authentication mechanisms. Back-doors are usually hidden, hard to discover, and inserted and used for malicious purposes. For example, a remote user may issue commands as root through a previously installed back-door. Some back-doors are created by programmers for reasons of convenience (e.g., remote maintenance) and so the original intent may not be malicious. However, back-doors that violate security policies must be considered malicious, based on their behavior alone. Remote access mechanisms operating within policy are not to be confused with back-doors.

■ Trojan

- Code that gets executed by deceiving a user is a trojan (the deception aspect implies maliciousness, even if it is a mild prank). Trojans can carry and be the initial entry mechanism for malicious code of another nature (e.g., a back-door or keylogger).



- Self-Propagating code (two modes)
 - Viruses are parasitic and are spread by means of finding new host files (documents or executables) that will presumably also get read or run later. Macro viruses refer to viruses carried by documents which can carry “macros”, essentially a scripting capability which blurs the boundary between data and code.
 - Worms spread on their own, by duplicating their code to other systems and re-spawning their processes.

- Code Red
- Blaster
- SQL Slammer
- Nachi (Welchia)
- Be prepared to answer a few questions about viruses and worms

■ Spyware

- Spyware is code that reports user activities and system information to “unauthorized” parties (who is “unauthorized” may depend on perspective). An example is an “unauthorized” keylogger. Spyware could also take “interesting” forms such as being a virus, and reporting when a certain type of document is opened.

- Logic/Time triggered code
 - This is extraneous code that is not part of the expected function of a software artifact and remains dormant until very specific activation conditions are met. If it can then perform attacks, it is a “Bomb”. If not, and it simply exposes humorous (at least to some) content, it is considered an “Easter egg”. Some Digital Rights Management (“DRM”) code that violates a security policy, e.g., while trying to aggressively enforce a licensing agreement, could be considered triggered code.

■ Rootkit

- A rootkit is a set of software artifacts that attempts to conceal its existence and execution (and possibly that of other malicious software as well) from the rest of the operating system, other processes or security tools, and consequently from users and administrators. Typically a rootkit subverts or replaces the utilities included with an operating system for the purposes of hiding a compromise and a back-door. A rootkit may include attack code as one of its components and may resist removal.

■ Distributed code

- Distributed code has coordinated copies of itself on many hosts. By acting in a coordinated fashion, the distributed code attempts goals that would likely be unreachable for a single copy. The coordination mechanism may be the reception of commands, or interactions between copies or with a controller. Blindly following specially crafted rules of conduct may also result in the overall desired behavior. Worms have been known to include a time bomb for attacking a pre-determined target at a given time, resulting in a distributed attack.

- Botnets are organized “networks of robots” obeying commands from a particular source, unknown to the owners of the computers
 - Often ill-protected home computers
- Used for
 - Spamming
 - Distributed Denial-of-Service (DDoS) attacks

- Intrusion Detection Systems (IDS) typically use two types of technologies:
 - Expert systems
 - » rules defined by experts
 - » usually detects only known attack types
 - Artificial neural networks
 - » system is trained to detect attacks
 - » may detect new attacks

- 1999 DARPA IDS classification:
 - Probe (or surveillance, information gathering: activity)
 - Denial of service (consequence of an attack)
 - R2L (remote to local), i.e., unauthorized access from a remote machine
 - U2R (user to root), i.e., unauthorized transition to root for an unprivileged user
 - Data. This is meant to represent attacks whose goal is to obtain and extract (“exfiltrate”) confidential files from a system

- Detect abnormal patterns of behavior
- Neural network approach
 - Trained with normal behavior data
 - » False positives and false negatives are given a “cost”
 - » System tries to minimize overall cost
- Expert systems approach next slide

- Idea: observe what an application needs to do, and forbid everything else
 - Phase 1: make rules for what is allowed, based on observation
 - Phase 2: rules are frozen in a profile
- <http://en.opensuse.org/Apparmor>

- Look for known patterns of behavior that indicate attacks
- Static set of rules
 - e.g., Snort
- Clearly an expert system approach

- a.k.a. “target monitoring”
- Verify at regular intervals that what is not supposed to change hasn't
 - Compute hashes
 - Verify system invariants
 - » e.g. cash on hand = starting money + deposits – withdrawals
- e.g., Tripwire

- Database types
- Database access control mechanisms
- Database integrity
- Database attacks and countermeasures

- Tables, rows, primary keys
 - » Foreign key constraints link a table to another
 - e.g., a table for people and another for possessions
 - Guarantees integrity by making sure that there aren't possessions without an owner
 - » 3rd normal form avoids duplicate and inconsistent data

- Views
 - Only provide the data needed
- Stored Procedures
 - Code executed in the database
- Roles
 - Avoid errors in the repetitive assignment of permissions to users
 - Reason about types of users

- Usually people think of the speed benefit
- Important security benefits as well
 - Add checks (controls) at choke point
 - » Protect invariants and data integrity
 - Guarantee that things are done the right way (processes are followed)
 - Define which actions are allowed

- Role “A” for creating tables
- Role “B” for creating stored procedures. Example: the procedures only allow reading (select) and inserts but not deleting or updating
- Role “C” that can only execute stored procedures. Role “C” can't delete or modify existing rows

■ Transactions

- Transactions are groups of database operations that either fail or succeed as a group
 - » e.g., read value A , calculate $A-X$, store new value
 - If A changes after reading and before storing the new value, bad things could happen
- Transactions preserve invariants

- Attack: SQL injection
 - Uses meta-characters such as ' " ; in data to change the meaning of commands sent to the database
- Countermeasure: Parameterized Queries
 - Send the commands and data separately so there can be no confusion

- Trust Analysis
- Polluted web sites
 - Cross-site scripting
 - Malicious ads
- Malicious web sites
- Client-side scripting as an attack vector



- Do you know where this data has been?
- Shopping web applications
 - Send list of items with price to client
 - Client returns to the server selected items for purchase, with the price
 - Server trusts the price returned by the client
 - » Attacker says the price is \$0.01 !

- Attackers adding malicious content to innocent web sites
 - Could be malicious ads
 - » Web site owners have little control over the ads provided to them for display
 - Could be in user-provided information
 - » XSS (cross-site scripting) attacks
 - » MySpace worm

- Attacks against your browser
- Attacks against other sites you visited before and authenticated to (cookie)
 - Site contains a reference to your webmail app, with the command to change your password or send an email
 - » CSRF attack
- Attacks against you (phishing)

- Most browser attacks use some form of client-side scripting
 - Compromised browser can mean compromised computer
 - Not required for CSRF attacks
- Great idea, execute foreign code from people I don't know or trust
 - Functionality over security
 - » Forced choice - enable it or go away

- Java has a strong security model
 - But slow and cumbersome so unpopular
- JavaScript is ubiquitous and powerful
- Flash can execute JavaScript
- Windows also has Active X, VBScript

- Internet Explorer Zones
- FireFox NoScript extension
- Phishing web site lists
 - Black list idea, likely to fail
- Server side
 - Strong input validation
 - Do not require client-side scripting

■ XSS

- Strong input validation
 - » White list approach: define what's allowed and reject everything else

■ CSRF

- Every form should contain a secret value set for each user at every session
 - » Receiving script should check that this secret is present

- Malory lays out a trap in the form of a URL or HTTP redirect
- The trap specifies a session ID; Alice clicks on it
- The web server accepts the session ID as valid
- Alice authenticates (gives user name and password)
- Session status is now "authenticated" and linked to Alice's user name
- Malory can now send commands as Alice because he knows the sessionID

- Why does the attack work?
 - Is it because the client chose the session-ID?
 - » – No: an attacker can first get a valid session-ID from the real server, and keep it alive as long as necessary (until an attack succeeds)
 - The server has no proof that Alice received the original session ID directly, and not through Malory
 - » This shared knowledge is the pitfall
 - » Resembles a partial Man-In-The-Middle attack

- Remove the possibility of decoupling the identification and authentication
 - Assign a proof of authentication nonce to Alice's browser only and directly in response to a successful authentication request
 - » Request valid nonce for every request afterwards

- GIAC white papers for the CISSP CBK
 - <http://www.giac.org/resources/whitepap>
- “All-in-one CISSP certification” (Harris S.)
- Rob Slade's list of reference books at <http://victoria.tc.ca/int-grps/books/techrev/mnbksccd.htm>

- Wikipedia
- CISSP prep guide (Krutz and Vines)
- My secure programming materials (CS390S)
 - www.cs.purdue.edu/homes/cs390s
- Matt Bishop's "Computer Security: Art and Science"