**Confidentiality** -> Revealing **Integrity** -> Modify **Availability** -> Access

**Completeness:** says only true things (no false positives)

**Soundness:** says all the true things (no false negatives)

**Metamorphic:** semantically different each generation

**Polymorphic:** different encryption but text stays the same

**Authentication:** Verifying that a user is who they claim to be

**Authorization:** Verifying that a user is allowed to perform some action

**Spatial Safety:** Only access memory that “belongs” to that pointer

**Temporal Safety:** Only access memory when it’s currently allocated, has not yet been freed, etc.

**Virus:** Runs when user initiates something, infects stored code.

**Worm:** Alters running code, propagates without user intervention.

**Prevention:** Stop a security breach from happening at all.

**Mitigation:** Limit amount of damage that a breach could cause.

**Whitelist:** Only allow specified inputs from a list.

**Blacklist:** Forbid specified inputs from list but allow any others.

**Propagation:** Spread of malware. **Payload:** What the malware does.

**Control-Flow Analysis:** Track where control is allowed to jump, abort if invalid jump location.

**Taint Analysis:** Prove that tainted sources can never reach sensitive/untainted sinks.

**Sanitization:** Altering input to make sure that it is safe to use.

**Checking:** Verify that input is safe to use, otherwise, reject it.

**Minimize Trust Computing Base:** Keep number of components you have to trust small. Keep simple.

**Least Privilege:** Give each component only as much authority as is strictly needed to do the job.

**Vulnerability –** a software defect that can be manipulated to yield an undesired behavior

**Capabilities:** Large random hidden fields that act as references to some saved state

**Cookies:** Session identifier, personalization, tracking users

**Session Fixation Attack:** Prevent by always using a new token when elevating a session.

**Referrer Validation:** only trust requests from pages that user could legitimately reach.

**Cross-Site Request Forgery (CSRF):** Craft malicious URL. Get user to click on it. Embed in <img/> to force user to send request. URL must be easy to predict.

Force login request with attacker info. User might input sensitive info into attackers account.

**Setting up the Stack:** Push args; Save eip (ret); save ebp; (canary); local vars

**%ebp:** Contains addr of old frame pointer.

**Halting Problem:**  Writing an analyzer that can prove for any program P and its inputs, will P terminate?

**Taint Analysis:**  Untainted < Tainted; Check feasible paths for conditionals.

**Implicit Flow:** One value implicitly influences another; Even if a value remains untainted, a tainted value could influence it. Lead to false alarms. Hurts performance. Tend to ignore them.

**Cross-Site Scripting Attacks: Stored XSS –** Inject script into target webpage, wait for user to visit

**Reflected XSS –** Attacker gets user to click link that has JS at the end, webserver echos input

**ASLR:** Randomly place elements in memory, make it harder to find libc functions/to guess were stack is

Only shifts offset of memory areas, May not apply to program code, Need sufficient randomness

**ROP –** string together pieces of existing code, **gadgets**, in order to run shellcode.

**Canaries:** Before return value. Check before return, if not expected value – ABORT!

**Types:** Terminator Canaries, Random Canaries, Random XOR Canaries

**GDB: i f =** info frame; **i r =** info registers; **x/<n> <addr> =** Examine n bytes at addr; **b <function> s =** break at func, step through execution

**Heap Overflow:** Overflow into the C++ object vtable; Overflow into adjacent objects; • Overflow heap metadata

**Spatial Safety: View pointers as capabilities: triples (p,b,e)**

• p is the actual pointer (current address)

• b is the base of the memory region it may access

• e is the extent (bounds) of that region (count)

• Access allowed iff b ≤ p ≤ (e-sizeof(typeof(p)))

• Stops buffer overflows and format string attacks

**Control Flow Integrity –** Define “Expected Behavior”: Control Flow Graph

Detect Deviations Efficiently: In-line Reference Monitor

Avoid Detector Compromise: Sufficient Randomness, Immutability

**Crypting Services:** Test malware until it doesn’t get detected by antivirus

**Rootkit:** Malicious code that takes steps to go undiscovered.

**Audit:** Retain enough information to determine the circumstances of a breach or misbehavior



