CVE (Common Vulnerabilities and Exposures)

- Lists known software vulnerabilities and security issues which is accessed through databases such as NVD (National Vulnerability Database) hosted by NIST(National Institute of Standards and Technology)
    - Better than nothing but
        - Classifications are not thought out very well
            - Too vague
            - Implementation vulnerabilities sought over others due to generalness of them
            - No standards: Bugs need to be found and reported, nothing is mandatory
                - This causes software companies to keep bugs under wraps, the idea being if people don't know about it, they can't exploit it! However, as history has shown people will find exploits if they are reported or not.
  - The chart on lecture slide 2 shows several different types of attacks
    - Created from CVE information
        - Example
            - SQL injection attack
                - String inserted into DB via a form or other device where it shouldn't be
                - String could be anything, a SQL Command for example could be inserted which allows the attacker to execute arbitrary code on the database.
  - As the chart shows lots of items fall into the “Others” category
    - Reason being we're simply not sure how to classify the attack

Attack types

Stack Smashing Attack

When a function g() is called by a function f(), first, the f’s BP is pointing at the beginning of the activation record of f(), when g() is called, first the return address is saved and then f’s BP must be saved in the stack, after that, g() gets executed.
An attacker can use a piece of malicious code to overflow the stack in order to overwrite the return address, so the return address will point to a piece of code that written by the attacker. When g() is finished, the program will continue to execute what the attacker has written.

- Slide 2 of lecture slides shows simple program with a buffer and how it is represented on the stack, it also shows the stack grown downwards
  - Address of buffer element calculated by : Starting address + 4 * X (e.g. the element is an integer)
    - X is our element index
        - Key point of attack is to cause a stack overflow which causes the return address to be overwritten and instead point to the buffer (See Diagram)
            - Allows arbitrary code to be executed
        - Other pointers can be attacked as well
            - If F calls G
                - We now have a base pointer we can overwrite which would work the same as overwriting the return address (See Diagram)
- Ways to Prevent
  - Don't allow code execution everywhere
    - Only allow at read areas or specify execute areas

- Microsoft and Intel have both added support for this approach to the problem
  - Good idea but unfortunately attackers are clever...
    - If I can't inject my code, why don't I just use yours?
      - Nx cannot prevent the kind of attack in which an attacker will use existing code. An attacker can change the return address to a specific function, e.g. system function
        - Example: Attacker wants a shell
          - Return to the area in memory that invokes a shell
          - Function parameters are pushed onto the stack above the return address, so by overwriting these values, the attacker can change the parameters going to the system function!
          - These are often called return-to-libc attacks
- In Summary
  - This causes attackers to be more constrained, but does not prevent all attacks
  - Because of virtual memory the attacker can always predict the location/addresses of items and have found ways to deal with minor uncertainties
    - For instance: Attacker knows address +/- 10 bytes
      - Insert nops to address this

<u>Guarding Techniques</u>

- Stack Guard
  - One of the first
  - Put a “canary” value immediately before the return address(See Diagram)
    - If value is overwritten we know an attack has happened then we simply abort the process
      - Different values can be used
        - Random value – otherwise an attacker can predict what the canary is and simply assign the same value when he/she overwrites the stack.
        - NULL - useful to prevent string copy attack.
        - Copy of Return Address— System will store a second copy of return address somewhere else, so if the return address in the stack is changed, we will discover that.
    - Major problem is this only protects the Return Address, other things such as a Base Pointer will be unprotected.
- Refinement of Stack Guard
  - ProPolice

    We can modify the canary value strategy such that we store simple local variables (rather than arrays) before return address and BP, then before simple local variables, we store canary value, so local variables can also be protected.
    - Change the stack around a little to confuse and mislead attackers

Reference Diagram