Lecture 15 - 11/08/2019

Buffer Overflow

Shell Code

- Note attackers code doesn't need to be in the separate area of memory.
- Test on a similar machine to find an approximate location of where to insert code into the victim's machine.

Preventing

- Look or all externally sourcing data. Might be standard libraries, 3rd party libraries, or system code.

Defenses

Compile-time defenses

- Useful for new code you will write.

1. Choose a good language, unless you must don't write in vulnerable languages.
    a. If you're not writing device drivers or embedded code, write in languages like C# and Java
2. Safe coding techniques
    a. When youre getting unknown input, sanitize and code for the unexpected
    b. Graceful failure.
3. Improved programming environments
    a. OpenBSD project
        i. Free multi platform BSD based unix like OS.
        ii. Has gone through a lot of scutenization
        iii. Carefully written and constructed
    b. Windows
    c. Augmented compilers
    d. Safer standard libraries (libsafe, fgets)
4. Stack protection
    a. E.g. Stack guard
        i. Uses a canary, places a random value on the stack and compares it after code executes to ensure that the value hasn't changed
    b. E.g. stack shield
        i. Similar to stack guard but just safes the return address, doesn't add any extra values

Run-time defenses

- Useful for legacy code that can not be changed easily.
- Would have string performance implications
- You have to balance risk to reward in terms of performance

1. Executable address space protection.
    a. Don't allow executable code on the stack.
    b. However, some programs need to put executable code in the stack.
2. Address space randomization
    a. Randomize the locations of the stack and heap
3. Guard pages
    a. Limits buffer to a single 'page' of the stack.
    b. An attack can only hit the same page that the input is writing to
4. Deep packet inspection
    a. Examine packets that are coming in and see if they look like a buffer overflow attack.

Software security

- Related to software reliability, but different.
- The ability to handle any small variation from expected input
- Also the ability to handle large variations from expected input
- You must take a look at every aspect of your code

Handling program input

Related to the sense of buffer overflow. Input must be handled, not only user input; but files, network connections, os/config files, data from sensors.

- Validate input size
- Interpretation of program input, what are we getting and is it what we expected
    - Injection attacks
        - Command injection
        - SQL injection
        - Code injection
    - XSS attacks
        - Embedded web material that can
- Validate syntax
    - There can be many different encodings for the same value.
    - Locale differences, language differences.
    - Canonicalization, (C14N)

Input Testing

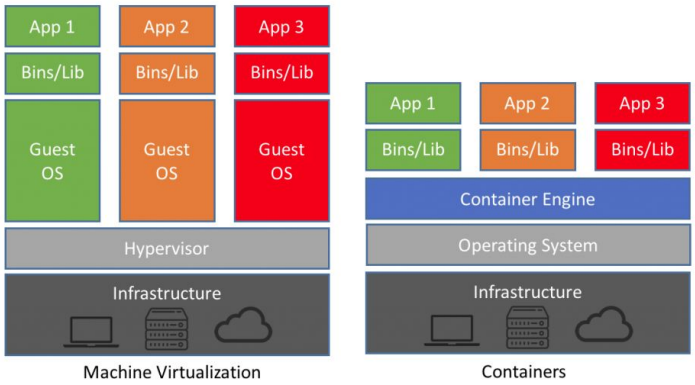
- Input fuzzing
    - Try random inputs at a high cardinality
    - Checks that a library or piece of code fails nicely on bad inputs
    - Does not guarantee that it is safe, only give a probabilistic model that it is safe.
    - Fuzzing is what attackers will use

Writing safe code

- Ensure correct algorithm implementation
    - E.g. poorly seeded RNG.
        - Using a low entropy seed can lead to predictable keys.
    - E.g. predictable sequence numbers
        - Session hijacking could be predictable if attackers can predict the session code.
    - E.g. debug code
        - TAKE IT OUT BEFORE GOING TO *P R O D U C T I O N*
- Ensure machine language corresponds to source code
- Ensure correct interpretation and use of data values.
    - C is typically the culprit language as pointers and integers can be easily interchanged
- Ensure correct use of dynamic memory
    - Allocate and DEALLOCATE your memory for your heap
- Race conditions or deadlocks
    - Use semaphores and write properly protected code
- Ensure that interactions are protected
    - Ensure $PATH is correct
    - Use least privilege
    - Use safe temporary file use

Containers

- Advantages of containers
    - Lightweight
    - Reliable across environments
- Container image
    - The thing that gets shipped or distributed
- Container
    - The actual running version of an image



Advantages

- Containers are conceptually similar to VMs
- Containers are MUCH smaller due to the lack of needing a guestOS.
- They can be installed and ran much quicker

Security

Containers can make achieving security (much) more complex

1. Developments cycles have shrunk from months to days/hours
    a. This means that the software may not be as vetted as in the past.
2. Interactions between containers
    a. Microservices architecture
3. Containers share resources
    a. Binaries, libraries, OS
    b. If the integrity of one container is compromised, then it can possibly compromise other containers running near itself.
4.