Determining the Fundamental Basis of Software Vulnerabilities

Larry Wagoner
NSA
Agenda

• Background
  – Analogous background
  – Matt Bishop work
  – CWEs
  – Tool reporting of CWEs
  – KDM Analytics

• Determining the fundamental basis
Elements of the world

• Classical elements in Babylonia (~17 B.C.): Sea, Earth, Fire, Sky and Wind

• Greek Classical Elements:
  – Four terrestrial elements: Earth, Water, Air and Fire
  – Sometimes a fifth element, Aether, was added
    • Aether is “pure, fresh air” or “clear sky”
  – Persisted throughout the middle ages

• Now viewed as a simplistic view of the world
More modern view

• Basic building blocks of matter
  – Atoms
    • Protons, neutrons, electrons
  – Elementary particles
    • Quarks, leptons, bosons
Periodic Table of Chemical Elements

Presented by the number of protons in the atom’s nucleus
WHAT ARE THE ELEMENTARY BUILDING BLOCKS OF SOFTWARE VULNERABILITIES?
“Vulnerabilities Analysis”


• Goal was to develop a classification scheme for vulnerabilities

• Scheme is deterministic
  – Each class has exactly one property
  – “yes” or “no” answer for membership

• Classification based on the code, environment or other technical details
  – Social cause is not a valid class

• Seeking consistent classification
Data and Stack Buffer Overflow Breakdown (Bishop)

- A buffer overflow attack can be decomposed into primitive conditions that must exist for the attack to succeed.
- Stop any of the primitive conditions and the attack cannot succeed.
- Four primitive conditions in *fingerd* attack on Unix system:
  - C1. Failure to check bounds when copying data into a buffer.
  - C2. Failure to prevent the user from altering the return address.
  - C3. Failure to check that the input data was of the correct form (user name or network address).
  - C4. Failure to check the type of the words being executed (data loaded, not instructions).
Invalidating these conditions prevents the exploitation (Bishop)

• C1’. If the attacker cannot overflow the bounds, the control flow will continue in the text (instruction) space and not shift to the loaded data.
• C2’. If the return address cannot be altered, then even if the input overflows the bounds, the control flow will resume at the correct place.
• C3’. As neither a user name nor a network address is a valid sequence of machine instructions on most UNIX systems, this would cause a program crash and not a security breach.
• C4’. If the system cannot execute data, the return into the stack will cause a fault. (Some vendors have implemented this negation, so data on the stack cannot be executed. However, data in the heap can be, leaving them vulnerable to attack.)
Common Weakness Enumeration (MITRE)

• Dictionary of software weaknesses
• Amalgamation of over a dozen taxonomies
  – CLASP, PLOVER, Pernicious Kingdoms, etc.
• Approximately 807 weaknesses or weakness categories described
• Example: CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
  – The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.
Reporting of CWEs

• Consider:

```c
/* Stack Overflow */
#define BUFSIZE 256
int main(int argc, char **argv) {
    char buf[BUFSIZE];
    strcpy(buf, argv[1]);
    }
```

• Is the vulnerability:
  – CWE-121 Stack based Buffer Overflow
  or
  – CWE-20 Improper Input Validation?
Software Fault Patterns

• Pilot by DoD, NIST and DHS
  – Develop a specification of software vulnerabilities that enables automation
  – Looked at subset of CWEs that could potentially be automated
    • Those that can be formalized
    • 302 CWEs
  – Clustered 302 CWEs into 50 software fault patterns
  – Developed whitebox definitions of a small number of CWEs
  – Formalization (machine readable) of 18 CWEs
  – Side effect of work identified a set of 81 Vulnerability Fundamentals
Fundamental Vulnerabilities

- Fundamental Vulnerability (FV) – A primitive condition in software that can serve as the basis for exploitation of the software
- FVs are defined in the format of a statement of fact
- An FV is the root cause of a software exploitation
- One or more FVs need to be exploited in order for an attack to occur
- An attack can be disrupted if one or more in a series of FVs is removed
Consider Buffer Overflow

• C1 (Bishop). Failure to check bounds when copying data into a buffer.
• FV: Check of array bounds before array access does not exist or is faulty
• C2 (Bishop). Failure to prevent the user from altering the return address.
• FV: Direct access to a memory address is permitted (can use/alter address of memory to access memory)
• C3 (Bishop). Failure to check that the input data was of the correct form (user name or network address).
• FV: Input checks do not exist or are faulty
• C4 (Bishop). Failure to check the type of the words being executed (data loaded, not instructions).
• FV: Code and data are indistinguishable in memory (commands are treated as data)
A Buffer Overflow Cannot Occur if...

• If a check of array bounds is performed correctly before the access
  – FV: Check of array bounds before array access does not exist or is faulty

• If memory addresses cannot be directly accessed or altered
  – FV: Direct access to a memory address is permitted (can use/alter address of memory to access memory)

• If input is validated correctly
  – FV: Input checks do not exist or are faulty

• If code and data is segregated in memory
  – FV: Code and data are indistinguishable in memory (commands are treated as data)
CWEs and FVs

- **CWE-369 Divide by Zero**
  - FV: Check that divisor is not 0 before division is performed does not exist or is faulty
- **CWE-732 Incorrect Permission Assignment for Critical Resource**
  - FV: Check of permissions do not exist or are faulty
- **CWE-561 Dead Code**
  - FV: Code exists in a program that is not on any execution path
FVs Rooted in Language Structure

• FV: There is a duality of a string and a null terminated array
• FV: There is syntactic ambiguity in the language
• FV: Signed and unsigned data types are converted from one the other
• FV: There is a disconnect between a pointer and the resource that it represents
FVs Across a Variety of Languages

- FV: Input checks do not exist or are faulty
- FV: Return value check does not exist or is faulty
- FV: Variable is used before it is initialized
- FV: Binary compilation is not functionally equivalent to its source
- FV: Hardware is not standardized
  - size of short, int, long differ between platforms
FVs Interaction with Environment

• FV: Security check is not performed local to the application
  – Security check is performed on client for a server application

• FV: Interface with another language is inconsistent
FVs Resource Interaction

• FV: Ownership of a resource expires
  – Memory containing sensitive information can then be read by some other program
• FV: History and provenance is not available for use at authentication points
  – No basis for determining the integrity of dynamically linked resource
• FV: Race condition for shared resource exists
Current Status

- About 70 FVs have been identified
- List has been mapped against the 2011 CWE/SANS Top 25 Most Dangerous Software Errors
- List is still being refined and expanded
Thank you.