Detection of s/w Attacks via Information Flow Tracking

Vijay Nagarajan

Software Security - CIA

- Confidentiality
  - Ensure secrecy of program/data
- Integrity
  - Ensure the validity of program/data
- Availability
  - Ensure software/data is available
Software Attacks

- Uses vulnerabilities in the program to prevent CIA
- Vulnerabilities in program
  - Buffer overflow
  - Format string
  - Integer overflow
- Worms

Importance of s/w attacks

- Security of programs against software attacks have become increasingly important
  - Frequency of attacks are increasing (source: CERT)
  - Number of vulnerabilities identified increasing.

- Dealing with worms and viruses cost US businesses 67 billion per year. (source: The 2005 FBI computer crime survey)
Buffer Overflow

```c
int single_source(char *fname)
{
    char buf[256];
    FILE *src;
    src = fopen(fname, "rt");
    while(fgets(buf, 1044, src)) {
        ...
    }
    return 0;
}
```
Buffer Overflow

- Code Injection
  - Execute user specified arbitrary code
  - Disrupt CIA
  - Non executable stack can thwart
- Return into libc
  - No new code executed
  - Redirect into loaded library
    - The “system” function
  - Again disrupts CIA

Format String

```c
int main(int argc, char **argv)
{
    char buf[100];
    
    if (argc != 2) exit(1);
    
    snprintf(buf, 100, argv[1]);
    buf[sizeof buf - 1] = 0;
    printf("buffer: %s\n", buf);
    
    return 0;
}
```

Equally as powerful as buffer overflows!
Solutions

- Write bug free code
  - Impossible!!
- Don’t use C
  - Use type safe languages
- Numerous other solutions
  - Need changes to compiler / source
    - Not transparent
    - Too slow

Existing Techniques

- Static Checks
  - Lack of runtime info
  - False positives and/or negatives
- Dynamic Checks
  - Prevent malicious writes
    - Stackguard, Bounds checking
    - Attack specific
  - Prevent malicious use of inputs
    - Point guard
Observation

- Attacker feeds malicious input
- Corrupts control data
  - Data that potentially controls PC
- Diverts control to execute required code
  - Injected
  - Library

Information flow tracking

- Generally user does not provide code pointers, instructions
- OS marks data from input channels spurious
  - File input
  - Network input
- Flow of tainted data is tracked
- Usage of tainted data in “malicious” fashion detected
  - Values dependent on inputs should not alter the PC
Information Flow tracking

I/O, other processes → Operating System → Vulnerability → Detect → Unintended Uses

- Step 1. OS tags potentially malicious inputs as *spurious*
- Step 2. Processors track the flow of the spurious values
- Step 3. Detect attacks - Check and restrict the use of spurious values

Security Tags; 0 – authentic, 1 – spurious

Dynamic Information Flow Tracking

Processors checks + trap handler

Observations

- Need to associate tags with registers and memory
  - Lets assume tag registers, tag memory
- For almost all instruction, we need to do tracking – propagation of tags
- For some instructions we need to do checking – detection of attacks
Initial Tagging

- OS tags data from certain input channels as spurious
- Alter read system call
  - Sys_read transfers i/p data into Mem
  - Write ‘1’ (spurious) into corresponding tag memory Mem’

Tracking

- Propagate the tags for each instruction
- Several policies
  - Copy dependency
    - Ld Reg, Mem
    - Ld tReg, Mem’
  - Computation dependency
    - Add Reg1,Reg2,Reg3
    - Or tReg1,tReg2,tReg3
  - Load/Store Address
Checking

- Actual detection of attack
- Several Policies
  - Jump targets
    - Jmp Reg
    - If tReg == 1 exception()
  - Instructions themselves
  - Load /Store Addresses
    - Ld Reg, Addr
    - If Addr == tainted exception()

Example

```
void fun() {
    char buf[10];
    choice = get();
    //bounds checking
    if(choice > 3) exit(0);
    ...
    //vulnerability
    gets(buf);
    ...
    switch(choice):
    Case 1: ...
    Case 2: ...
    Case 3: ...
}
```

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Call get</td>
<td>1. Call get</td>
<td></td>
</tr>
<tr>
<td>2. Cmp eax,3</td>
<td>2. Mov eax, 0</td>
<td></td>
</tr>
<tr>
<td>3. Jz &lt;exit&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Call &lt;gets&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Pop eax</td>
<td>6. Mov eax, esp’</td>
<td></td>
</tr>
<tr>
<td>7. Add eax, ebp</td>
<td>7. or eax, ebp</td>
<td></td>
</tr>
<tr>
<td>8. Jmp eax</td>
<td>8. Cmp eax,1 jz &lt;exception&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Implementation h/w or s/w

- **s/w**
  - Need to allocate registers for tags
  - Execute additional instructions for tracking
  - Need a DBT
  - Performance may take a hit

- **h/w**
  - Processor pipeline changed to perform tracking and checking

---

**h/w Implementation**

- Tag bits
  - Register
  - Memory
    - Cache
- ALU
- External bus
Tag Management

- Memory 1 bit per byte of memory
  - 12.5% overhead
- Reducing this
  - Observations
    - Most of memory is made up of untainted data
    - Some pages full of tainted data
  - Have page wise tags
  - Reduces overhead to 4.8%

Evaluation

- No false positives on spec benchmarks
- No false negatives on attack benchmarks
- Performance overhead is low (< 1%)
- Memory overhead low (< 5%)
Discussion - h/w

- Non Trivial changes to h/w
  - Cannot work in current machines!!
  - Changes solely for security
  - Is it practical?

Software implementation

- Source code, compiler, binary instrumentation
  - Not practical: legacy code
  - Not secure!!
- Require DBT (dynamic binary translator)
  - Need registers to store tags
  - Extra instructions to perform tracking
  - Prior implementations
    - Causes 40x slowdown!!!
    - Another implementation reduces it to 4x
Tracking in s/w using CMP

- Use additional core to perform tracking
- Extra registers and other resources available
- Tracking can be performed concurrently

Discussion – Control flow

- int critical_val = 0;
- If (critical_val)
  - system("\bin\sh"); // control dependent on c.v
- Data dependency via cd
  - For i = 0..n
    - If i == j break;
- To track cd each PC must have a notion of taintedness
  - RIFLE tracks cds
- Lots of false positives possible
Summary

- Information flow tracking a general technique to counter s/w attacks
- Not really a light-weight technique
  - Current s/w implementations are slow
  - Requires h/w support for efficient implementation
  - Multicore implementation.