CS 261 - Runtime Defenses
Scribe Notes – 09/11 (Ashima Atul)

Runtime Instrumentation Techniques

All runtime instrumentation techniques protect the integrity of the application. They can be exploited by Denial-of-Service attacks.

1. **StackGuard**
   StackGuard protects stack by inserting a canary between control data and buffer. Whenever there is a buffer overrun then the canary value gets corrupted first. A corrupted canary helps in finding out that there has been a buffer overrun.

Local variables	canary	…….	Return address

   Canary – random 32-bit value
   Canary is secret

   - Advantages - Performance hit is less

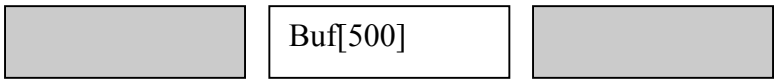
   - Attacks possible
     a. Can be exploited using format string attacks.
     b. Bad generation of canary could let the user statistically find the location of canary. The attackers can then use this information to draft exploits.
     c. Heap overflow attacks cannot be defended.
     d. Function pointers stored in virtual tables can be exploited.

2. **Non-Executable Stack**
   This technique makes the stack non-executable
   - Attacks possible
     a. Arc injection/return-to-libc attacks
     b. Inject code into heap and then make the program jump to that location by manipulating function pointers.

3. **W^X (Variant of Non-Executable Stack)**
   Arranges mapping of all addresses i.e. every page as either writable or executable. A page cannot be both writable and executable.

4. **Separate Stacks**
   Two separate stacks. One stack stores all fixed size objects like return address, etc. The other stack is stored in heap and stores variable length data like buffers.
   - Attacks possible
     a. Heap overflow.
     b. Function pointers stored in heap can be manipulated to draft attacks.

5. **Electric Fence**
   Electric fence is a debugging tool. Whenever memory is assigned using malloc electric fence uses the virtual page table and marks the pages before and after the page containing the allocated memory (using malloc) as unmapped. These unmapped pages act as guard pages. If the program tries to write to any of the guard pages then a fault is sent.



Guard page Guard page

Electric fence uses an instrumented implementation of malloc.

- Attacks possible
    a. Stack smashing.
    b. Exploit the page containing the buffer. E.g. if the buffer is of size 500 then the address after 500 till the page boundary can be exploited. But, this does not work for electric fence because it will not allow to write to memory after 500.

6. Valgrind Memcheck

Uses a bit vector T[ ].
Assume a 32-bit system having 4GB memory. Thus T[ ] will require 512 MB memory.

T = 1 addr 'i' is writeable
T = 0 addr 'i' is not writeable

For e.g. if an address stores a malloc header then its T bit will be set to 0 and for buffers the T bit will be set to 1.

On every load/store checks are performed to verify with the T bit whether the byte can be written or not.

- Attacks possible
    a. Only checks for heap exploits
    b. struct {
        char c[];
        int i;
    }s;
    In the above case the attacker can write past c and overwrite i. Such vulnerabilities can be exploited to overwrite function pointers stored inside a struct object.
    c. Format string vulnerabilities can be used to go past the current buffer and write malicious code to other buffers that have the T bit set to 1.

7. Address Randomization

This technique uses random addresses thus causing it difficult for the attacker to guess malicious code location.

8. CRED

- With Optimizations
    a. Optimized CRED tracks only chr buffers thus overwriting buffers other than char buffers can be used as an attack.

- Without Optimizations
    a. Non-buffer overflow attacks
    b. Overflow within struct object.
    ```
    struct {
        char c[];
        int i;
    }s;
    ```
    c. It is not clear what are they doing for stack allocated buffers.
    d. Attacks can happen if any un-instrumented library (library not compiled using CRED) is being used.
    e. Memcpy passing wrong string length

- Two ideas the apply generally
    a. Shadow data structures
        ⇒ Splay tree allowed retrieving the heap memory address the object maps to.
        ⇒ It is needed to have isolation between program data and control channel. Both data and control share the same address space and the operating system does not provide any kind of isolation between them. CRED protected this so that out-of-bound(OOB) addresses do not get written.
    b. Handle both instrumented and un-instrumented library code
        ⇒ Some techniques do bounds checking on pointers by replacing pointers in the program with a structure that stores information about the base, length of the buffer etc. This approach cannot work with applications using legacy code because the pointer representation has changed and would not match with the legacy code.
        ⇒ It is not clear from the paper whether format string vulnerabilities can be detected if printf is not compiled with CRED.

- CRED paper hasn't talked about space overhead.

- Out-of-bound address handling
    ```
    p = &buf[0];
    p += 1000;
    p -= 500;      This manipulation may point to the middle of another buffer
    *p;
    ```
    OOB object is used because it stores the original base address of the pointer.

9. **Taint Tracking**
   - Focused on data driven attacks

   - Perl was the first language to implement taint tracking

   - Applying taint techniques to C
     a. Byte level taint tracking.
     b. Performance optimization
        ⇒ Easier to implement in interpreted language like perl. Program will not tamper taint bit since there is inbuilt isolation of control channel and data in interpreted languages like perl.
     c. Control channel must not be tainted

   - Policy
     a. Control related data is never tainted. Eg. PC value, sql keywords, meta characters etc.
     b. Let us take a sample security policy e.g. $(\% [a\text{-}zA\text{-}Z])^T$
        ⇒ This does not cover all format string vulnerabilities.
        ⇒ To exploit this you can write %100d.
     c. If the black list approach taken misses some case of format string vulnerability in regex then an attack can be crafted.
     d. False positives could be present.
     e. Solution to this is to change the implementation of printf such that printf parses the input and then we check the parse string.

   - SQL injection policy
     Taking an example of a policy written for SQL injection
     $(‘ \mid ; \mid * \mid /* \mid \ldots.)^T$

     The above policy does not handle SQL keywords. Thus if somebody sends a query like

     “SELECT * from users WHERE name = “ + *input*

     input can be entered as *abc OR 1 = 1*. This input would cause an exploit.

     The Taint tracking paper addresses this problem by constructing a parse tree for the SQL query. All the sub-trees which have their root tainted are tainted as well.

   - Cross-site scripting attacks

     Example security policy - $(<\text{script})^T$

     Exploit – s c r i p t (one white space between each letter)
     S c ript

     Solution - Parse the html and generate a DOM tree. Mark all the leaves as tainted.