Lecture Notes: 10/14

Soft Bounds / CCured

1. Load Variables
   - Add local bound variables
2. Heap Data
   - Shadow structure with bounds info.

Intuition
We cannot store the information of a FAT pointer like low, high and the pointer as it is too heavy and requires computation.

So we will store the information in hash table.

Example 1:

Struct list{
  Struct last **next;
  Int data;
};

void traverse(struct list *L
{
  while(L){
    f(L->data);
    L=L->next;
  }
}

Transformed Code:

void traverse(struct list *L, bounds L_b)
{
  while(L){
    bcheck(L,L_b);
    f(L->data);
    L = blockup(&L->next); // we are looking for the bounds of this pointer where it lives in the memory
    L=L->next;
  }
Complete picture how it works:

Struct list *prepad(struct list *L, int a)
{
    struct list *tmp = malloc(...); //we have to make bounds for tmp
    bounds tmp_b= {tmp,tmp+1}; // bounds for tmp

    // suppose the programmer does
    blookup(&tmp->next);
    *tmp->next; //tmp is not defined at this point, this is on the heap, we look up on the
    bounds on this table blookup

//Corrected program

Struct list *prepad(struct list *L, bounds L_b, int a)
{
    struct list *tmp = malloc(...);
    bounds tmp_b= {tmp,tmp+1};
    bcheck(tmp,tmp_b);
    tmp->data = a;

    Bnds[&tmp->next] = L_b;
    tmp->next = L; //this is pointing to the pointer which is on the heap

}

Note: Now what happens is that on the stack we have a pointer L.
L points to the starting address of the stack and also points to a location on the heap. So soft bounds handles pointers on the heap. tmp is on the stack and will remain on the stack, tmp->next will be on heap. Bounds of tmp goes with the pointer on the stack.

Q1. Why are we not keep track of every thing?
Ans: The disadvantage is performance. If the hash table is complicated u have to resolve collisions. We have to keep the data together as much as possible.

Example 2:
Now suppose we do this.

free(L->next);
Bnds[&L->next] = {0,0}
So the problem here is that we have to invalidate the pointer pointed to by L->next.

**Corrected Program**

Bnds[&L->next->next] = {0,0}; // it invalidates all the pointer that are pointed to by
// L->next...so it is based on the type of *L->next

free(L->next);
Bnds[&L->next] = {0,0};

**Explanation:**
Suppose there are three next in the diagram. What will happen is that the first next will become a dangling pointer; the second next cease to exist, the third one will remain in the bound table. Eventually the pointer of the first will be pointed to the third one but this is the step done by the compiler after executing the above statements.

**Suppose it is doubly link list:**

When the compiler sees: free(L->next)
Then it will see there are two pointers next and prev
So compiler adds this:

Bnds[&L->next->next] = {0,0}
Bnds[&L->next->prev] = {0,0}
free(L->next); // is done
Bnds[&L->next] = {0,0}
Explanation:
Suppose there are two pointers next and prev as shown in the diagram. What will happen is that the first prev will not be affected, the second will cease to exists and the third one will become dangling pointer.

Evaluation criteria for transformation

1. Effectiveness -> very good
2. Performance -> 67% is the performance if optimized version is used
3. Memory overhead -> is OK.
4. False positive -> Its OK as it check on dereference and not on arithmetic so false positive is very good in this case.
5. Separate Compilation -> yes it is separately compliable so this feature is supported
6. Linking -> it is moderately good.

Explanation of Linking:
If Soft bounds are used then the function name is changed to sb_prepad. Suppose there are two libraries one with all the ‘sb_‘ names and one without. So if the function is using softbound then it calls the first the one with ‘sb_‘ names in it otherwise it uses another.
Like shown in the diagram below

<table>
<thead>
<tr>
<th>SB program</th>
<th>Non SB program</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB Lib</td>
<td></td>
</tr>
</tbody>
</table>

Problem:
The problem here is that if we have SB program and Non SB library then that will create a problem. The problem here is if the pointer which is made in Non-SB lib and passed to SB program then there will be no pointer on heap to point to as it is made in Non-SB lib. So it will cause the problem.

<table>
<thead>
<tr>
<th>SB program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non – SB lib</td>
</tr>
</tbody>
</table>

7. No code change -> very good (No code change required)

External war in memory paper (discussion)

Defense mechanism discussed in the paper:
1. NX bit
2. Address Space randomization
3. Canaries
4. Memory safety transformation

Other things
1. Control flow integrity
2. Data flow integrity
3. Data space randomization

Address Space randomization (ASLR)
Because of low overhead it is successful on mac, android, windows, Linux and all other platform.

NX bit
Overhead is low, effectiveness is good.

Stack canaries
Requires compiler to change, effective, low overhead.

Problem is Softbound:
Overhead is killer and is bad.
No one will use the general compiler that inserts arbitrary code anywhere in the program.