



Concepts Systems

The Center of Excellence

Project Title	Static Detection of Buffer-Overruns
Registration Number	OS05199
Engineering College	Pune Institute Of Computer Technology (PICT) Pune - 411043, Maharashtra.
Project Members	SHARAD B. DARADE BABASAHEB A. NAIKWADI MAHENDRA V. CHAVAN
Project Guide	Prashant D.
Academic Year	B.E. (2004-2005)
Synopsis	

IDEA

The size and complexity of software systems is growing, increasing the number of bugs. Many of these bugs constitute security vulnerabilities. Most common of these bugs is the buffer overrun vulnerability. This is an old, well-known problem, yet it continues to resurface.

What is buffer?

A buffer is any data structure which is used for storage of stream of data. For example, an array or a dynamically allocated block. Mainly they are used to hold strings which are generally input from user. They can be local variables, global variables, function arguments etc.

What is buffer overrun and why it occurs?

Every buffer has its own size. If data more than actual size of buffer is copied into it, it is called buffer overrun.

The reasons that buffer overruns are a problem to this day, are poor coding practices, the fact that both C and C++ give programmers many ways to shoot themselves in the foot, a lack of safe and easy-to-use string-handling functions, and ignorance about the real consequences of mistakes.

Purpose:

Anyone who reads the BugTraq mailing list at <http://www.securityfocus.com/> can see reports almost daily of buffer overrun issues in a large variety of applications running on many different operating systems. 50% of the attacks reported at CERT last year point to buffer overrun. Being such an important security issue it needs to be addressed in detail.

One disadvantage of dynamic approaches is that they increase performance overhead. More importantly, such approaches do not eliminate the flaw but simply replace it with a denial-of-service vulnerability. Recovering from a detected problem typically requires terminating the program. Hence, although security-sensitive applications should use damage-limitation techniques, the approach should not supplant techniques for eliminating flaws.

The static analysis is extremely complex due to the unknown string lengths, complete pointer analysis, complete flow and context sensitive analysis. The correct estimation of buffer overrun in a statement is possible only if we know how many times that statement gets executed and in what context. For example to detect the vulnerability of an 'strcat' statement in a for loop we need to consider how many times the loop is executed. Also we must consider the values of global variables at each function call.

Most static tools available currently are not fully flow sensitive and context sensitive which our software tries to be. The problem with static analysis is that both precision and scalability are hard to achieve at the same time so some trade-off must be chosen.

HOW THE SOFTWARE WORKS?

The source code is analyzed to find all the buffers along with the two ranges- used range and allocated range for each buffer. Used range is used to indicate the end of string currently in buffer whereas allocated range is the allocated space for the buffer. Used range is initialized to allocated range as there may not be end of string character ('\0') at the time of allocation. Ranges can be approximate in case of dynamic allocation depending on user input.

Every access to the buffer is examined to see whether it is beyond the allocated range. If it is, then it's certainly a buffer overrun. If the access is beyond the used range but within the allocated range, modify the used range according to the data being copied.

e.g. If '\0' of a string is overwritten by any other character, then the used range is modified depending on '\0' in the data being copied, otherwise it is set to allocated range just to be conservative

ALGORITHM

1. Start scanning statements one by one starting from first statement in the main.
2. If the statement is declaration of a buffer,
make an entry in the Buffer Range Information Table(BRIT.. structure given in fig no.3) containing name, used range and allocated range.
3. If the statement is a buffer access
determine how many times¹ statement is executed, which may be an approximate or exact range.
if the access is beyond the allocated range, then it's certainly a buffer overrun. Generate counter example for the same.
if the access is beyond the used range but within the allocated range, modify the used range according to the data being copied.
4. If the statement is a function call,
if entry for this function is present in the Function Constraints Table(FCT as in Fig. 4)
check the actual arguments for these constraints for this function call ,
else,
analyze code and determine all the overruns which are independent of parameters, global variables and determine the constraints for parameters and global variables.
Make an entry of these constraints in the FCT.
Check the actual arguments for these constraints for this function call.

-
1. To determine how many times a statement is executed we need to find all basic blocks of code and find (approximate) how many times they are executed .

PROBLEM DOMAIN:

We concentrate mainly on string manipulation which may include pointers.

We are going to take care of stack buffer overrun and heap buffer overrun.

The standard for source code language will be ANSI C.

OUR ORIGINAL CONTRIBUTION:

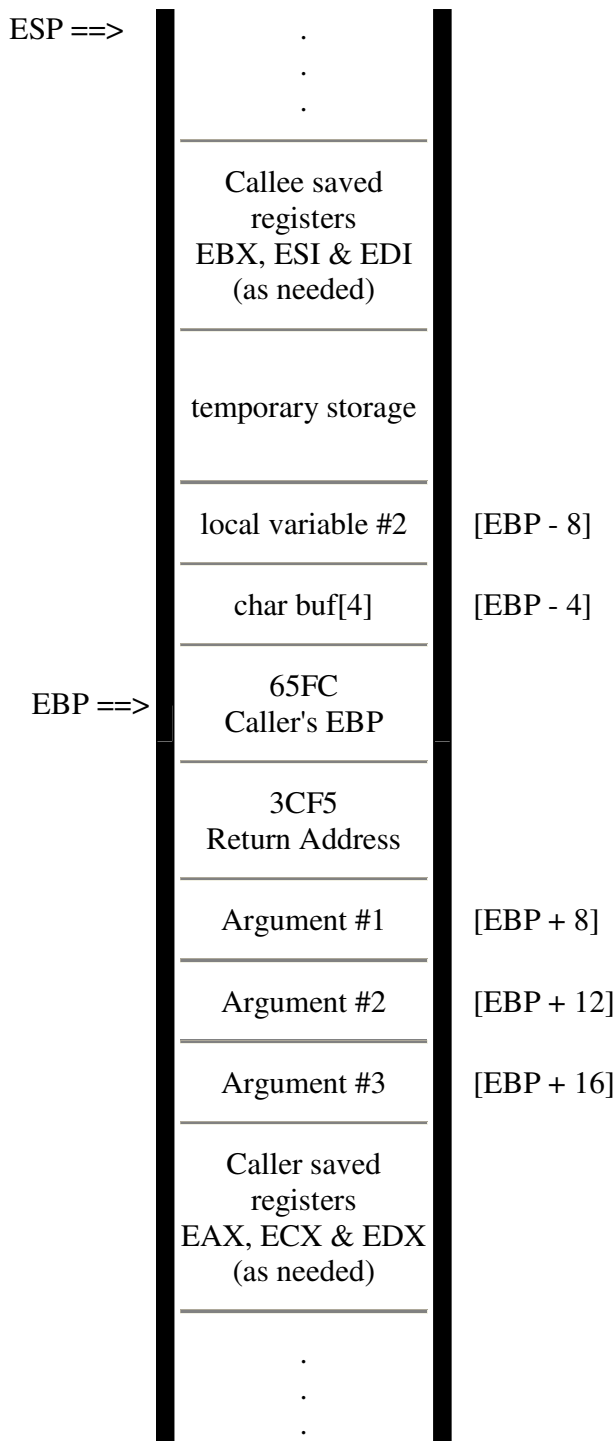
To solve the problem of static detection of buffer overrun vulnerability, we have devised new algorithm that does basic block analysis and integer range analysis .

The algorithm incorporates

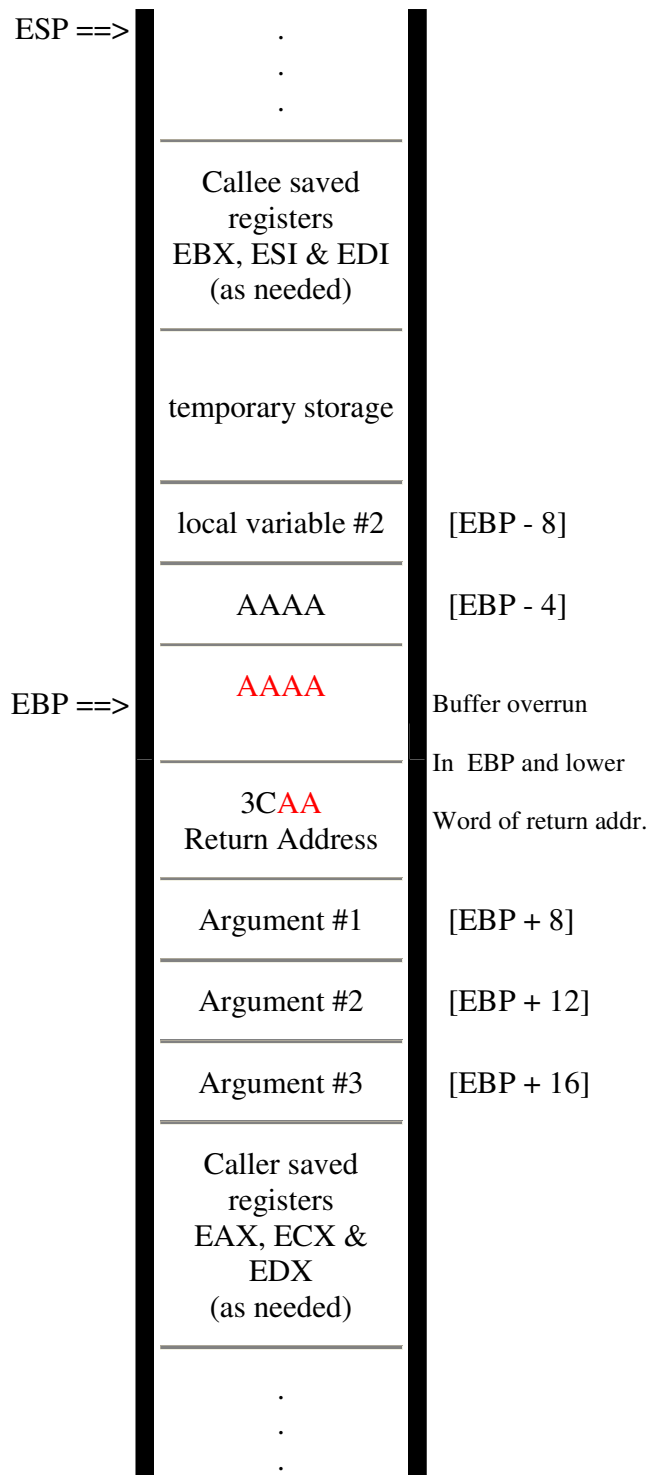
- **Completes pointer analysis.**
- **Complete Flow & context sensitive analysis**

MARKETABILITY OF THE PROJECT & COMMERCIAL APPLICATIONS:

We cannot underestimate the cost of a single buffer overrun attack. Some customers are going to get their systems compromised by attackers. The cost of a single compromise can be astronomical, depending on whether the attacker is able to further infiltrate a system and access valuable information such as credit card numbers. One sloppy mistake on your part can end up costing millions of dollars. Hence our software will be very useful to the software developer to check the source code for all buffer overrun vulnerabilities, before it gets deployed at customer site. Our tool can be integrated with existing compilers to generate warnings for buffer overrun.



Stack frame before
buffer overrun
Fig. 1



Stack frame after
buffer overrun
Fig. 2

Buffer name	Used range		Allocated range	
	Min	Max	Min	Max

Fig.3 : Buffer Range Information Table(BRIT)

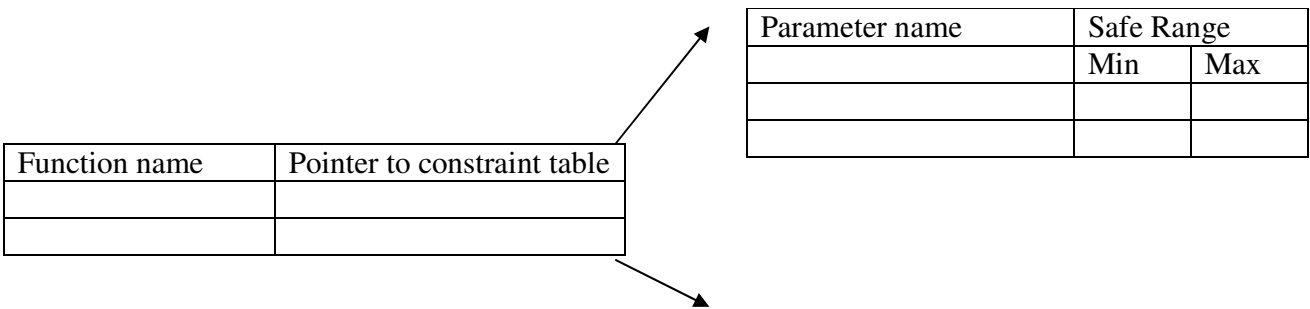


Fig. 4: Function Constraint Table(FCT)

Fig.: Constraint table