CSE 509 : Feb 15th (Thursday)
Static Analysis: Type Qualifier Inference

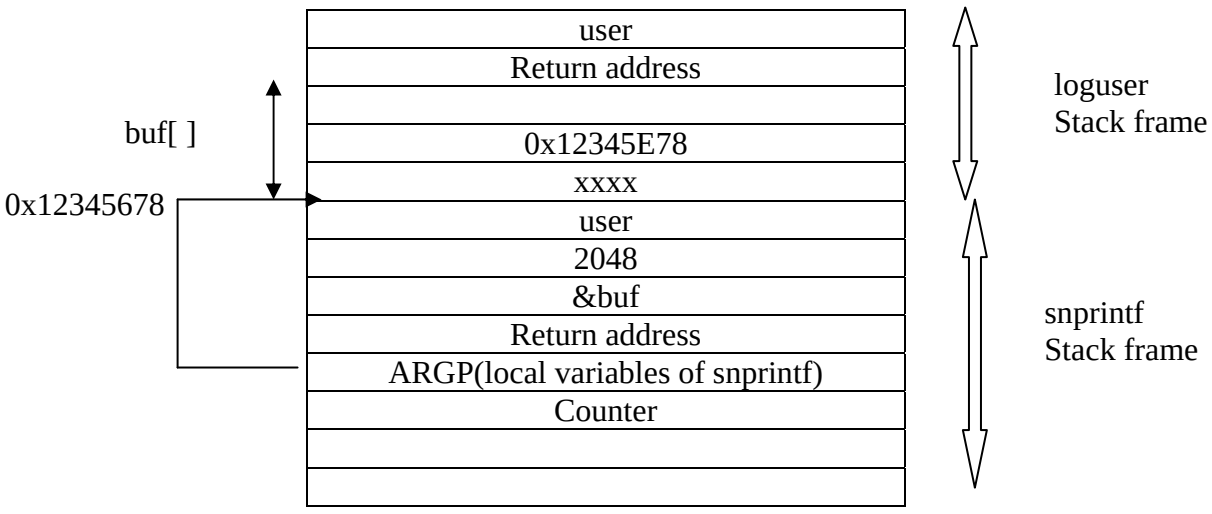
Detecting Format String Bugs

Consider the following source code :

```
void loguser(char * user)
{
        char  buf[2048];
        snprintf(buf, sizeof(buf), user);
        …..
        …..

}
```

To understand the format string bug in the above code, consider the stack diagram .



Note:

ARGP : initially points to the third argument in snprintf. This is the place where there can be additional optional arguments.

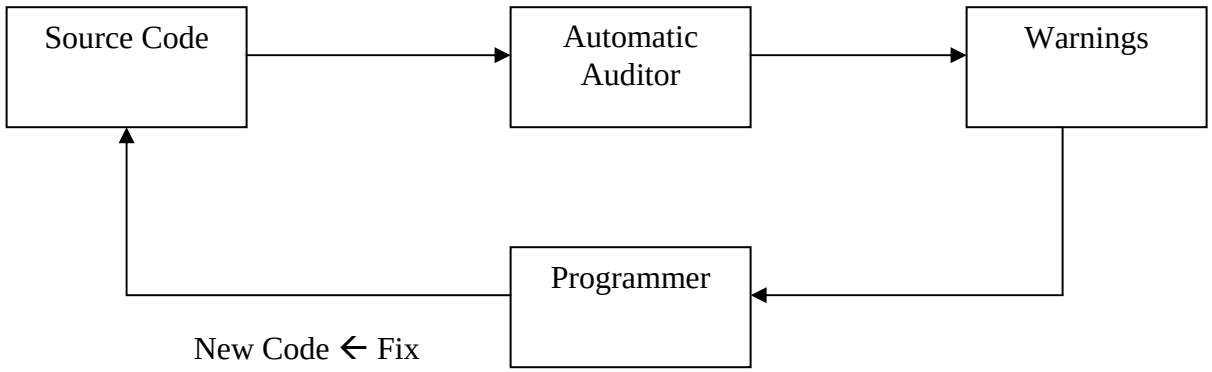
Counter: Keeps track of how many bytes had been printed out so far. This is also a local variable of snprintf function.

Format String Bug Detection and Prevention

Solutions:

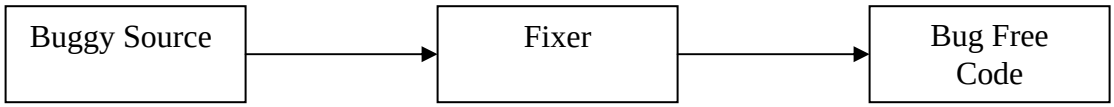
1. Educate programmers
   - Mistakes can still occur

2. Automatic Code Auditing
   Source code is fed into the automatic auditor.



The problem with this approach is that there can be to many bugs and we will not be able to manually fix all the bugs.

3. Automatic Bug Fixing



From the paper : " Detecting Format String Vulnerabilities With Type Qualifiers "
By : Shankar, Talwar, Foster, Wagner.

Idea : The paper presents a system to automatically detect format string bugs at run time .

1. This system uses static, type theoretic analytic techniques .
2. Data flow analysis

Consider the following code segment :

Code 1

```
int       printf(char * fmt, …);
char *  getenv(char *);

int  main()
{
     char *s, *t;
     s = getenv("Foo");
     t = s;
     printf(t);   // safe way to do this : printf("%s", t);
}
```

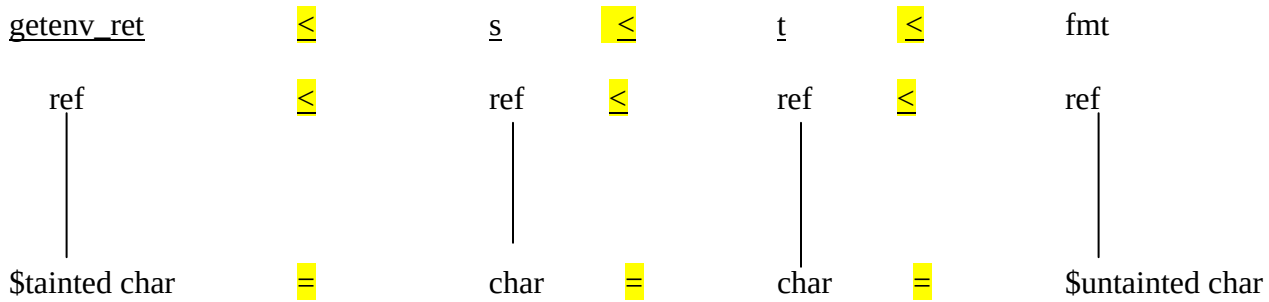
Code 2:

```
     char *s = "Hey";
     printf(s);
```

The above Code 2 is considered safe because no dangerous input flows to the format arg of printf.

Bug Finding Tools

1. Requires annotations (Consider Code 1)



So clearly there is a contradiction at the lowest level, since a char can't be both tainted and untainted.

Contradiction => Potential Error

Vector	<	Object
$untainted	<	$tainted
$untainted char	<	$ tainted char

- False Positive – Warnings but no bugs
- False Negative – Bugs, but no warnings.
- Complete - No false positive
- Precise - Less false positive
- Sound - No false negative

Cqual results

Cqual in theory is sound.
Tools used usually have any two of the three properties:

1. Sound.
2. Complete.
3. Terminate.

Cqual is both sound and terminates.

	Warnings(Cqual)	Warnings 2(Diff Tool)	Bugs
1. muh			1
2. cfengine			1
3. bftpd			1
Total	19	5	3

Current state of the tool : 20 % false rate.

Points to ponder :

- Attacker needs to find 1 bug to win.
- Defender needs to find all the bugs to win.