TLDR:
We noticed an integer overflow vulnerability in dstring.c that leads to out-of-bounds heap overwrite. We developed a poc exploit based on house-of-muney technique to demonstrate that this vulnerability can lead to arbitrary code execution.

Vulnerability:
The found integer overflow bug is at line 78 in dstring.c, where strsize*2+2 can be overflows to a negative value. When it is passed into ds_resize, no new memory reallocation would be triggered for the dynamic string and thus would cause out-of-bounds overwrite later at line 81.

```c
61 char *
62 ds_fgetstr (FILE *f, dynamic_string *s, char eos)
63 {  
64    int insize; /* Amount needed for Line. */
65    int strsize; /* Amount allocated for S. */
66    int next_ch;
67    /* Initialize. */
68    insize = 0;
69    strsize = s->ds_length;
70    /* Read the input string. */
71    next_ch = getc (f);
72    while (next_ch != eos && next_ch != EOF)
73       {  
74          if (insize >= strsize - 1)
75             {  
76                ds_resize (s, strsize * 2 + 2); |
77                strsize = s->ds_length;
78             }
79                s->ds_string[insize++] = next_ch;
80                next_ch = getc (f);
81            }  
82            s->ds_string[insize++] = '\0';
83            if (insize == 1 && next_ch == EOF)
84               return NULL;
85            else
86                return s->ds_string;
87 }  
```

The vulnerable function ds_fgetstr is called at multiple places, together with its wrapper ds_fgets.

The references to ds_fgetstr and ds_fgets:
- Line 68 in copyin.c (ds_fgets)
- Line 72 in copyin.c (ds_fgetstr)
- Line 801 in copyin.c (ds_fgetstr)
- Line 639 in copyout.c (ds_fgetstr)
- Line 92 in copypass.c (ds_fgetstr)
- Line 904 in util.c (ds_fgets)

Impacts:
Our POC exploited the ds_fgetstr call at line 801 in read_pattern_file function from copyin.c. The function is invoked from process_copy_in when -E pattern_filename is provided from command line.
Based on this **house-of-muney** technique, our exploit used the overwrite vulnerability in *ds_fgetstr* to corrupt the *size* and *prev_size* metadata in mmap chunk *new_save_patterns* so that when *xrealloc* at line 807 is called, freeing *new_save_patterns* could also override part of the memory mapping of LibC (*.gnu.hash*, *.dynsym*). The *ds_fgetstr* was then called to rewrite the *.gnu.hash* and *.dynsym* with fake symbol table entries that can achieve arbitrary code execution.

For demonstration, we have made a Kali Docker container in which we pop a shell on the latest *cpio* package, installed with `sudo apt install cpio`. We do this by overwriting the *chdir* symbol table entry with the information in *system*’s entry, and passing in the argument `-D /bin/bash` to the binary.

The reason that this is done in a Kali container and not an Ubuntu container is that Ubuntu’s *cpio* is full RELRO, while Kali’s is partial RELRO, but this vulnerability exists in both packages.
However, this exploit bypasses other binary protections, such as ASLR and stack canary, without any leaks. The exploit does require the attacker to know the correct libc version to produce the proper fake symbol table.

We also believe that other calls to `ds_fgetstr` in cpio could potentially also lead to arbitrary code execution, and the integer overflow bug is worthy of being patched. Additionally, even if the input is not malicious, just over a gigabyte of input into `ds_fgetstr` without proper memory layout feng shui will segfault the program.