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 overflowed 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.

```
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
68    /* Initialize.  */
69    insize = 0;
70    strsize = s->ds_length;
71
72    /* Read the input string.  */
73    next_ch = getc (f);
74    while (next_ch != eos && next_ch != EOF)
75      {
76        if (insize >= strsize - 1)
77      {
78        ds_resize (s, strsize * 2 + 2);
79        strsize = s->ds_length;
80      }
81        s->ds_string[insize++] = next_ch;
82        next_ch = getc (f);
83      }
84    s->ds_string[insize++] = '\0';
85
86    if (insize == 1 && next_ch == EOF)
87      return NULL;
88    else
89      return s->ds_string;
90  }
91
```

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.

```
781  static void
782  read_pattern_file ()
783  {
784    int max_new_patterns;
785    char **new_save_patterns;
786    int new_num_patterns;
787    int i;
788    dynamic_string pattern_name;
789    FILE *pattern_fp;
790
791    if (num_patterns < 0)
792      num_patterns = 0;
793    max_new_patterns = 1 + num_patterns;
794    new_save_patterns = (char **) xmalloc (max_new_patterns * sizeof (char *));
795    new_num_patterns = num_patterns;
796    ds_init (&pattern_name, 128);
797
798    pattern_fp = fopen (pattern_file_name, "r");
799    if (pattern_fp == NULL)
800      open_fatal (pattern_file_name);
801    while (ds_fgetstr (pattern_fp, &pattern_name, '\n') != NULL)
802      {
803        if (new_num_patterns >= max_new_patterns)
804      {
805        max_new_patterns += 1;
806        new_save_patterns = (char **)
807          xrealloc ((char *) new_save_patterns,
808                    max_new_patterns * sizeof (char *));
809      }
810        new_save_patterns[new_num_patterns] = xstrdup (pattern_name.ds_string);
811        ++new_num_patterns;
812      }
813    if (ferror (pattern_fp) || fclose (pattern_fp) == EOF)
814      close_error (pattern_file_name);
815
816    for (i = 0; i < num_patterns; ++i)
817      new_save_patterns[i] = save_patterns[i];
818
819    save_patterns = new_save_patterns;
820    num_patterns = new_num_patterns;
821  }
```

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.