

[UNIX] PPPd DoS

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2005-09/0054.html>

From: SecuriTeam (support_at_securiteam.com)

Date: 09/13/05

To: list@securiteam.com

Date: 13 Sep 2005 17:07:54 +0200

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

PPPd DoS

SUMMARY

<<http://www.samba.org/ppp/features.html>> ppp is "an implementation of (PPP) Point-to-Point Protocol for Unix systems".

Improper verification of header fields allows an attacker cause the pppd server access memory it isn't allowed to, which in turn causes the server to crash. There is no possibility of code execution, as there is no data being copied, just a pointer dereferenced.

DETAILS

Vulnerable Systems:

* ppp Version 2.4.1

The actual vulnerable code appears in the file `/pppd/cbcp.c`, line 334. A brief walk through of how it is reached: Starting in the `/pppd` directory, in `main.c` we have the function `get_input()`, which is called when there is data ready on the network. It reads in the packet at line 932, at most 1500 + PPP header sized bytes into a static packet buffer called `inpacket_buf`.

Depending on the protocol, a handler is picked out of an array of handlers

by matching the protocol field of the PPP header. We are interested in when the protocol is CBCP, Callback Control Protocol. A snip from that function is shown here:

```

/* process an incoming packet */
static void
cbcp_input(unit, inpacket, pktlen)
    int unit;
    u_char *inpacket;
    int pktlen;
{
    u_char *inp;
    u_char code, id;
    u_short len;

    cbcp_state *us = &cbcp[unit];

    inp = inpacket;

    if (pktlen < CBCP_MINLEN) {
        error("CBCP packet is too small");
        return;
    }

    GETCHAR(code, inp);
    GETCHAR(id, inp);
    GETSHORT(len, inp);

#ifdef 0
    if (len > pktlen) {
        error("CBCP packet: invalid length");
        return;
    }
#endif

    1] len -= CBCP_MINLEN/*4*/; /* HOLE */

    switch(code) {
    case CBCP_REQ:
        us->us_id = id;
    2] cbcp_rcvreq(us, inp, len);
    break;

```

- 1) len has not been validated yet, if it is < 4, the subtraction will wrap around to a large 2 byte unsigned number.
- 2) len is passed to the request processing function, which now thinks that packet is longer than it really is.

We then move onto the `cbcp_rcvreq()` function to process the request, this function is in `/pppd/cbcp.c` :

```

/* received CBCP request */
static void
cbcp_rcvreq(us, pckt, pcktlen)
    cbc_p_state *us;
    char *pckt;
    int pcktlen;
{
    u_char type, opt_len, delay, addr_type;
    char address[256];
    int len = pcktlen;

    address[0] = 0;

1] while (len) {
    dbglog("length: %d", len);

    GETCHAR(type, pckt);
2] GETCHAR(opt_len, pckt);

    if (opt_len > 2)
        GETCHAR(delay, pckt);

    us->us_allowed |= (1 << type);

    switch(type) {
    case CB_CONF_NO:
        dbglog("no callback allowed");
        break;

    case CB_CONF_USER:
        dbglog("user callback allowed");
        if (opt_len > 4) {
            GETCHAR(addr_type, pckt);
            memcpy(address, pckt, opt_len - 4);
            address[opt_len - 4] = 0;
            if (address[0])
                dbglog("address: %s", address);
        }
        break;

    case CB_CONF_ADMIN:
        dbglog("user admin defined allowed");
        break;

    case CB_CONF_LIST:
        break;
    }
3] len -= opt_len; /* HOLE */
    }

```

Securiteam: [UNIX] PPPd DoS

```
    cbc_presp(us);  
}
```

1) The loop continues processing the packet as long as len is != 0. Each iteration the packet pointer is moved forward in the GET_ macros.

2) The option length is retrieved from the packet, and is not validated in any way.

3) The option length is subtracted from the len variable, which controls the loop. There are a number of ways to exploit this calculation. Actually, `_any_` malformed packet will screw up that loop. It relies on the `opt_len` values in the packet all summing to len, if they don't, the loop won't stop, unless by pure luck of encountering the right value somewhere in the `.data` section (the packet buffer is global). Net result, is that eventually in the GET_ macros, the packet pointer will be advanced to far, and hit unmapped memory and crash the server.

ADDITIONAL INFORMATION

The information has been provided by <<mailto:infamous41md@hotpop.com>>
sean.

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

list-unsubscribe@securiteam.com

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====

=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.