Instantly share code, notes, and snippets.

gquere / barebox_advisory.md

Last active last year

Star

Code Revisions 4

barebox_advisory.md

BareBox security advisory

Barebox security features

The security features of Barebox are:

- Signed images
- Signed "state variables" (shared with the kernel)
- Login password protection

The last two are vulnerable to side-channel attacks.

CVE-2021-37848: Password checking

The high-level algorithm used to check the bootloader's password and grant user access is as follows:

```
read password from stdin
hash read password
read actual hash from /env/etc/passwd
compare hashes and return appropriately
```

The problem is that both hashes are compared using strncmp() which is not time-constant:

```
int strncmp(const char * cs, const char * ct, size_t count)
{
    register signed char __res = 0;

    BUG_ON(!cs || !ct);

    while (count) {
        if ((__res = *cs - *ct++) != 0 || !*cs++)
            break;
        count--;
    }

    return __res;
}
```

The break condition is a fail-early that exists here for obvious performance reasons. But for cryptographic verifications all checks should be time-constant so that the attacker cannot get information about the secrets used by observing the system.

This would be very minor because all the attacker can recover here is the hash of the password, which is assumed uncrackable if appropriate measures are taken: using salted and scaling algorithms such as the available PBKDF2 and also having a robust password (e.g. >20 characters). Do note though that unsalted MD5 is the default value which is way more easily cracked offline.

Fixed by https://github.com/saschahauer/barebox/commit/a3337563c705bc8e0cf32f910b3e9e3c43d962ff

CVE-2021-37847: Signed state variables

```
This options enables HMAC based authentication support for
the state's header and data. This means the state framework
can verify both the data integrity and the authentication of
the state's header and data.
```

These variables are verified like this:

```
static int backend_format_raw_verify(struct state_backend_format *format,
        ret = digest_verify(backend_raw->digest, hmac);
```

All hashing algorithm have a `verify` function pointer that defaults to the following function:

```
int digest_generic_verify(struct digest *d, const unsigned char *md)
{
    int ret;
    int len = digest_length(d);
    unsigned char *tmp;

    tmp = xmalloc(len);
```

```
    ret = digest_final(d, tmp);
    if (ret)
        goto end;

    ret = memcmp(md, tmp, len);
    ret = ret ? -EINVAL : 0;
end:
    free(tmp);
    return ret;
}
```

The code for `memcmp` follows:

```
int memcmp(const void *cs, const void *ct, size_t count)
{
    const unsigned char *su1 = cs, *su2 = ct, *end = su1 + count;
    int res = 0;

    while (su1 < end) {
        res = *su1++ - *su2++;
        if (res)
            break; // returns on first non-matching byte
    }
    return res;
}
```

Here again a non constant-time function is used to compare cryptographic results. Measuring the time delta over and over could allow an attacker to forge a signature byte by byte until it passes the check, thus defeating the secure boot.

Fixed by https://github.com/saschahauer/barebox/commit/0a9f9a7410681e55362f8311537ebc7be9ad0fbe

Fix suggestions

All high-level cryptographic checks should rely on the digest's already existing dedicated `verify` function, insted of directly calling `memcmp` or `strcmp`. This would allow for centralization of crypto code. Of course the hash `verify` function should then run in constant-time.

Here's an example code that compares memory areas in constant-time:

```
int constant_time_memcmp(const void *v1, const void *v2, size_t n)
{
    uint8_t res = 0;
    const volatile uint8_t *s1 = v1;
    const volatile uint8_t *s2 = v2;

    for (size_t i = 0; i < n; i++) {
        res |= s1[i] ^ s2[i];
    }

    return res;
}
```