GitHub - karcherm/xz-malware: Stuff discovered while analyzing the malware hidden in xz-utils 5.6.0 and 5.6.1

By karcherm

Archived: 2026-04-06 00:26:23 UTC

Information about the liblzma (xz-utils) backdoor

Decoder for the string recognition automaton

The backdoor code, in the version extracted by Florian Weimer, contains no readable ASCII strings. It also contains no obfuscated ASCII strings. Instead, it has a single state automaton to recognize the required strings. Searching for a string is performed by inputting all candidate start addresses into the string detection automaton, and checking whether the intended string is recognized. The string detection automaton returns a string ID.

The string detection automaton is implemented in the function `_Lsimple_coder_update_0`, which has this signature:

```
int detect_string(const void* startptr, const void* optional_endptr);
```

It returns 0 if no known string is detected at startptr (the search is aborted if endptr is encountered before a match is detected), otherwise it returns a "string ID".

This is the table that is generated by running the script in this repository:

```
 810: ' from '
678: ' ssh2'
 d8: '%.48s:%.48s():%d (pid=%ld)\x00'
708: '%s'
108: '/usr/sbin/sshd\x00'
870: 'Accepted password for '
1a0: 'Accepted publickey for '
c40: 'BN_bin2bn\x00'
6d0: 'BN_bn2bin\x00'
958: 'BN_dup\x00'
418: 'BN_free\x00'
4e0: 'BN_num_bits\x00'
790: 'Connection closed by '
 18: 'Could not chdir to home directory %s: %s\n\x00'
 b0: 'Could not get agent socket\x00'
960: 'DISPLAY='
9d0: 'DSA_get0_pqg\x00'
```

```
468: 'DSA_get0_pub_key\x00'
7e8: 'EC_KEY_get0_group\x00'
268: 'EC_KEY_get0_public_key\x00'
6e0: 'EC_POINT_point2oct\x00'
b28: 'EVP_CIPHER_CTX_free\x00'
838: 'EVP_CIPHER_CTX_new\x00'
2a8: 'EVP_DecryptFinal_ex\x00'
c08: 'EVP_DecryptInit_ex\x00'
3f0: 'EVP_DecryptUpdate\x00'
 f8: 'EVP_Digest\x00'
408: 'EVP_DigestVerify\x00'
118: 'EVP_DigestVerifyInit\x00'
d10: 'EVP_MD_CTX_free\x00'
af8: 'EVP_MD_CTX_new\x00'
6f8: 'EVP_PKEY_free\x00'
758: 'EVP_PKEY_new_raw_public_key\x00'
510: 'EVP_PKEY_set1_RSA\x00'
c28: 'EVP_chacha20\x00'
c60: 'EVP_sha256\x00'
188: 'EVP_sm'
8c0: 'GLIBC_2.2.5\x00'
6a8: 'GLRO(dl_naudit) <= naudit\x00'
1e0: 'KRB5CCNAME\x00'
cf0: 'LD_AUDIT='
bc0: 'LD_BIND_NOT='
a90: 'LD_DEBUG='
b98: 'LD_PROFILE='
3e0: 'LD_USE_LOAD_BIAS='
a88: 'LINES='
ac0: 'RSA_free\x00'
798: 'RSA_get0_key\x00'
918: 'RSA_new\x00'
1d0: 'RSA_public_decrypt\x00'
540: 'RSA_set0_key\x00'
8f8: 'RSA_sign\x00'
990: 'SSH-2.0'
4a8: 'TERM='
 e0: 'Unrecognized internal syslog level code %d\n\x00'
158: 'WAYLAND_DISPLAY='
878: '__errno_location\x00'
2b0: '__libc_stack_end\x00'
228: '__libc_start_main\x00'
a60: '_dl_audit_preinit\x00'
9c8: '_dl_audit_symbind_alt\x00'
8a8: '_exit\x00'
5b0: '_r_debug\x00'
5b8: '_rtld_global\x00'
```

```
a98: '_rtld_global_ro\x00'
 b8: 'auth_root_allowed\x00'
1d8: 'authenticating'
 28: 'demote_sensitive_data\x00'
348: 'getuid\x00'
a48: 'ld-linux-x86-64.so'
7d0: 'libc.so'
7c0: 'libcrypto.so'
590: 'liblzma.so'
938: 'libsystemd.so'
 20: 'list_hostkey_types\x00'
440: 'malloc_usable_size\x00'
 c0: 'mm_answer_authpassword\x00'
 c8: 'mm_answer_keyallowed\x00'
 d0: 'mm_answer_keyverify\x00'
948: 'mm_answer_pam_start\x00'
 78: 'mm_choose_dh\x00'
 40: 'mm_do_pam_account\x00'
 50: 'mm_getpwnamallow\x00'
 a8: 'mm_log_handler\x00'
 38: 'mm_pty_allocate\x00'
 a0: 'mm_request_send\x00'
 48: 'mm_session_pty_cleanup2\x00'
 70: 'mm_sshpam_free_ctx\x00'
 58: 'mm_sshpam_init_ctx\x00'
 60: 'mm_sshpam_query\x00'
 68: 'mm_sshpam_respond\x00'
 30: 'mm_terminate\x00'
c58: 'parse PAM\x00'
400: 'password\x00'
4f0: 'preauth'
690: 'pselect\x00'
7b8: 'publickey\x00'
308: 'read\x00'
710: 'rsa-sha2-256\x00'
428: 'setlogmask\x00'
5f0: 'setresgid\x00'
ab8: 'setresuid\x00'
760: 'shutdown\x00'
d08: 'ssh-2.0'
2c8: 'ssh-rsa-cert-v01@openssh.com\x00'
 88: 'sshpam_auth_passwd\x00'
 90: 'sshpam_query\x00'
 80: 'sshpam_respond\x00'
 98: 'start_pam\x00'
9f8: 'system\x00'
198: 'unknown\x00'
```

```
b10: 'user'
380: 'write\x00'
 10: 'xcalloc: zero size\x00'
b00: 'yolAbejyiejuvnup=Evjtgvsh5okmkAvj\x00'
300: '\x7fELF'
```

A fake allocator

liblzma has a memory allocation layer that just forwards allocation and freeing calls to dedicated allocators. Calling `lzma_alloc` or `lzma_free` with a given allocator object basically just calls a function pointer in that allocator object, which may or may not be related to actual memory allocation.

The backdoor contains a fake allocator object that looks up symbols instead of allocating and does nothing on freeing. The look-up function takes a string ID (see previous section) as size. As string IDs are divisible by 8 and between 10 and 0xd10, they look like plausible sizes at first.

This allocator object is returned by `.Lstream_decoder_memconfig.part.1`. The allocator structure contains a context pointer that is passed to the allocation and freeing functions. For this fake allocator, the opaque member points to an internal ELF module descriptor records.

The usage pattern (with a sane symbol name for the function that returns the fake allocator) thus is something like this:

```
    lzma_allocator* fake_alloc = GetFakeAllocator();
    fake_alloc->opaque = libc_elfmodule;
    void* symbol = lzma_alloc(0xAB8, fake_allocator);  // 0xAB8: string ID of "setresuid"
    // use symbol, maybe call it, maybe store it somewhere
    lzma_free(symbol, fake_allocator);                 // just decoy, does nothing
```

Note that `lzma_alloc` and `lzma_free` are not included in the backdoor object, but just the standard funktions provided by non-backdoor code in liblzma.

Source: https://github.com/karcherm/xz-malware