# Anti-UPX Unpacking Technique - JPCERT/CC Eyes

By 朝長 秀誠 (Shusei Tomonaga)

Published: 2022-03-14 · Archived: 2026-04-05 17:56:08 UTC

- IoT

Malware targeting Windows OS (PE format) has a variety of obfuscation and packing techniques in place so that they complicate the code analysis processes. On the other hand, there are only a few types of packing techniques for Linux-targeting malware (ELF format), and it is mainly UPX-based. This blog article explains the details of Anti-UPX Unpacking technique, which is often applied to Linux-targeting malware.

## Malware with Anti-UPX Unpacking Technique

The most well-known malware using Anti-UPX Unpacking technique is Mirai and its variants, which target IoT devices. Figure 1 shows the headers of UPX-packed binary and Mirai. The normal UPX packing uses “UPX!” as a magic number, while Mirai assigns a different value to each sample.

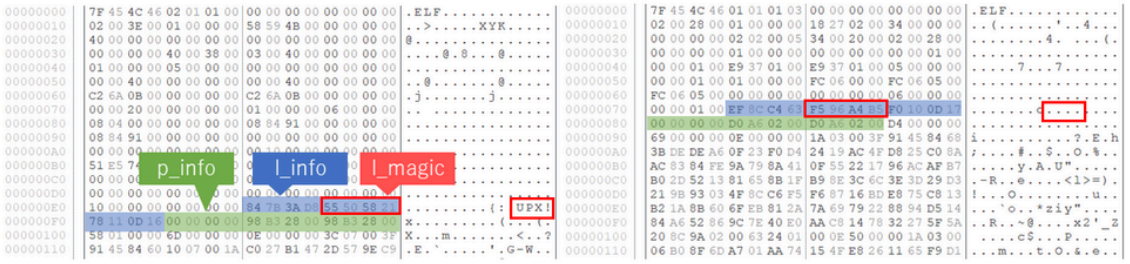



Figure 1: UPX-packed binary (left) and Mirai header (right)

UPX-packed binary contains the following information in the header. Normally, only “l_magic” is altered, but “p_filesize” and “p_blocksize” are also zero-padded in some samples.

```
struct l_info       // 12-byte trailer in header for loader (offset 116)
{
    uint32_t l_checksum;
    uint32_t l_magic;    // magic number = "UPX!"
    uint16_t l_lsize;
    uint8_t  l_version;
    uint8_t  l_format;
};

struct p_info       // 12-byte packed program header follows stub loader
{
    uint32_t p_progid;
    uint32_t p_filesize;
```

```
    uint32_t p_blocksize;
};
```

Besides Mirai, there are many other types of malware using this technique, including BoSSaBot (seen around 2014), as well as some coin miners and SBIDIOT malware, more recently. This is also applied to some types of malware which were used by Lazarus group. Figure 2 shows a part of ELF-VSingle's code, which is associated with the group. The magic number is replaced with "MEMS".

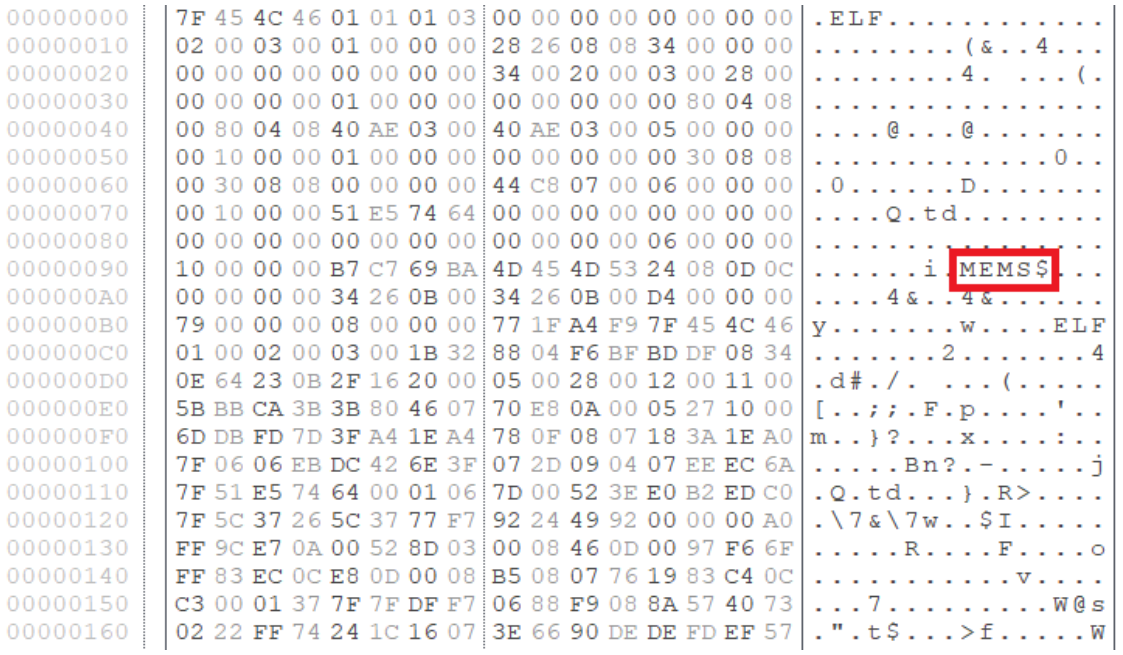



Figure 2: ELF_VSingle header

## Unpacking Anti-UPX Unpacking binary

Binary based on Anti-UPX Unpacking technique cannot be unpacked using the normal upx command. However, it is actually easy to unpack it. In most cases, the only change made to such binary is its magic number "UPX!". You can unpack it with upx command by changing this value back to "UPX!". Figure 3 shows the process of changing the magic number in order to unpack it using upx command.

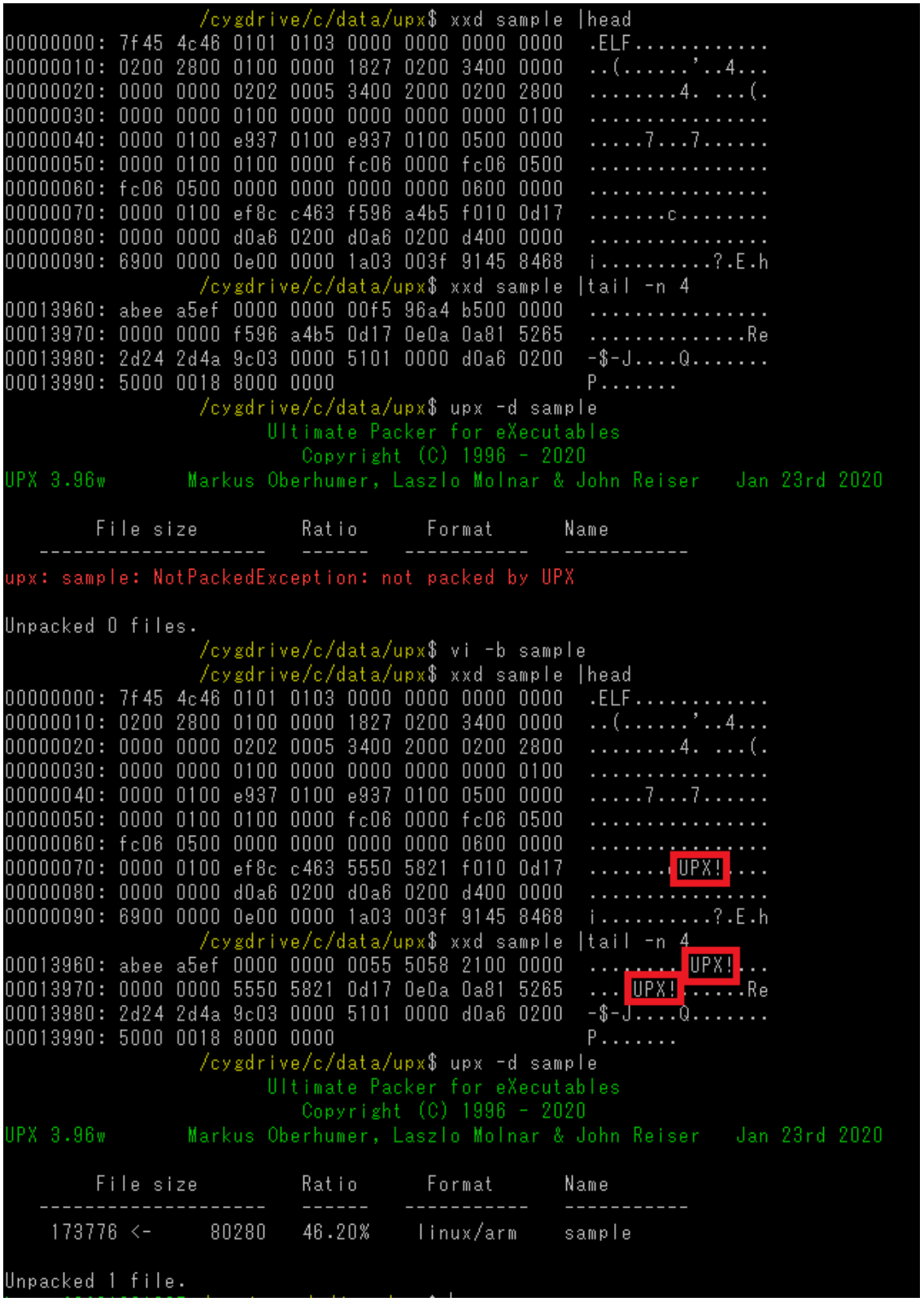



Figure 3: Example of unpacking binary

We have created a tool that enables unpacking binary with Anti-UPX Unpacking techniques. This tool is intended for this purpose only, and it may not work otherwise.

**JPCERTCC/upx-mod - GitHub** https://github.com/JPCERTCC/upx-mod/releases/tag/v4.00-beta

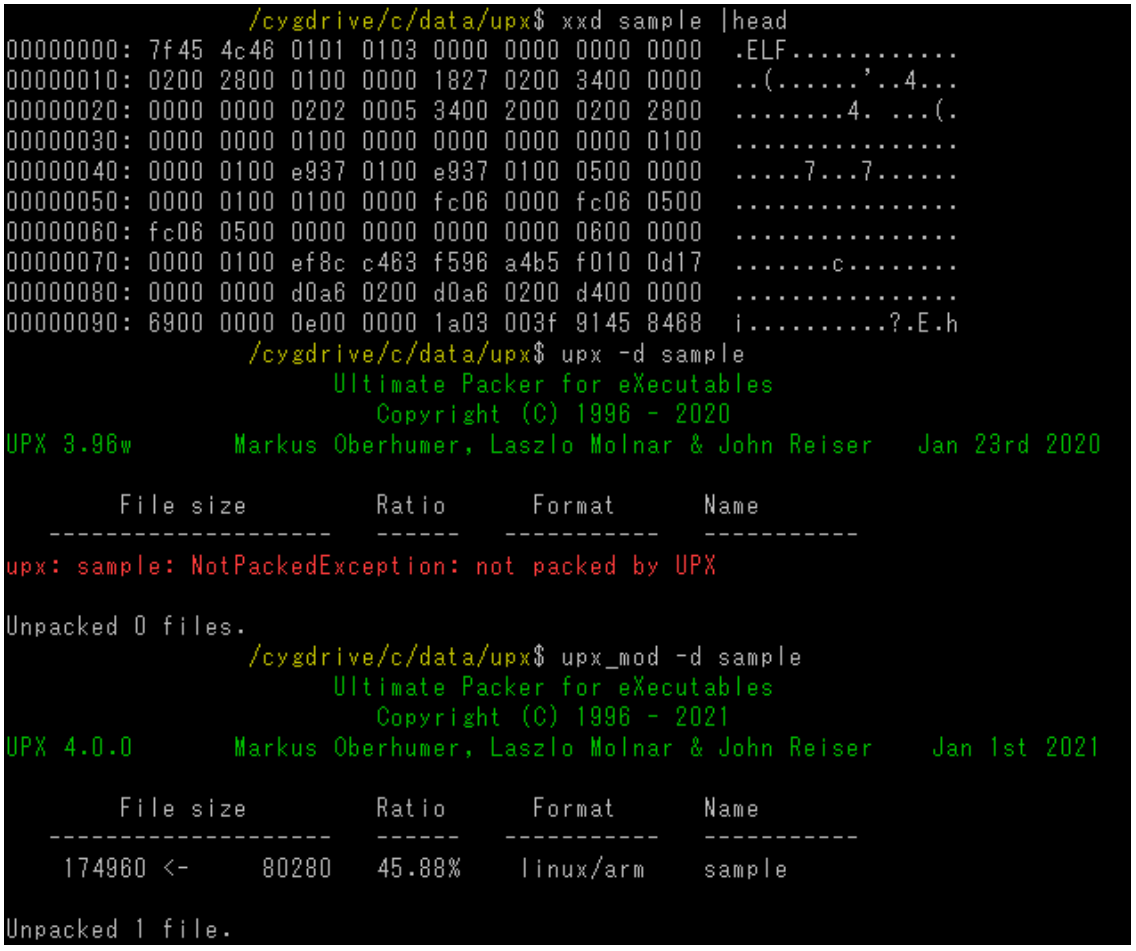



Figure 4: Sample use of upx_mod command

### Detect Anti-UPX Unpacking Technique

Binary packed with this technique can be identified manually, just by looking at the code. In order to avoid oversight, we recommend Yara-based automatic detection like below. This rule does not detect binary packed with normal UPX.

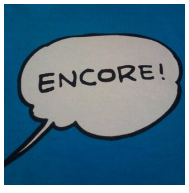
```
rule upx_antiunpack_elf32 {
     meta:
        description = "Anti-UPX Unpacking technique to magic renamed for ELF32"
        author = "JPCERT/CC Incident Response Group"

     condition:
       uint32(0) == 0x464C457F and
       uint8(4) == 1 and
       (
         (
           for any magic in (uint32(filesize - 0x24)) : (magic == uint32(uint16(0x2C) * uint16(0x2A)
           not for any magic in (0x21585055, 0) : (magic == uint32(uint16(0x2C) * uint16(0x2A) + uin
         )
         or
         (
```

```
            for any magic in (uint32(filesize - 0x24)) : (magic == uint32(uint16be(0x2C) * uint16be(0
            not for any magic in (0x21585055, 0) : (magic == uint32(uint16be(0x2C) * uint16be(0x2A) +
          )
        )
}

rule upx_antiunpack_elf64 {
     meta:
        description = "Anti-UPX Unpacking technique to magic renamed for ELF64"
        author = "JPCERT/CC Incident Response Group"

     condition:
       uint32(0) == 0x464C457F and
       uint8(4) == 2 and
       (
         (
           for any magic in (uint32(filesize - 0x24)) : (magic == uint32(uint16(0x36) * uint16(0x38)
           not for any magic in (0x21585055, 0) : (magic == uint32(uint16(0x36) * uint16(0x38) + uin
         )
         or
         (
           for any magic in (uint32(filesize - 0x24)) : (magic == uint32(uint16be(0x36) * uint16be(0
           not for any magic in (0x21585055, 0) : (magic == uint32(uint16be(0x36) * uint16be(0x38) +
         )
       )
}
```

## In closing

Many attack groups use malware based on Anti-UPX Unpacking technique. It is easy to unpack such malware, but you may waste your time in unpacking process unless you notice this feature beforehand. When you analyse packed ELF binary, we recommend checking first whether it uses Anti-UPX Unpacking technique.

Shusei Tomonaga
(Translated by Yukako Uchida)

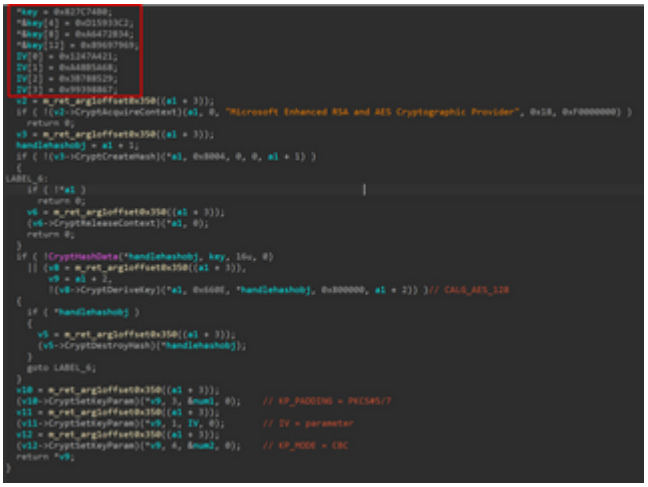



朝長 秀誠 (Shusei Tomonaga)

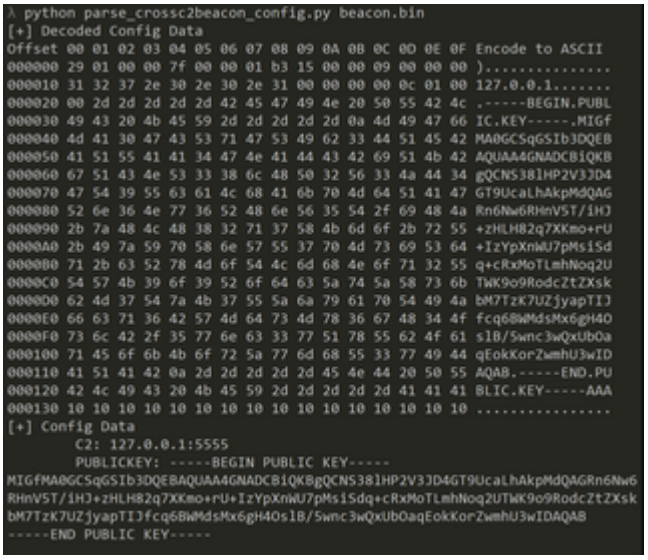
Since December 2012, he has been engaged in malware analysis and forensics investigation, and is especially involved in analyzing incidents of targeted attacks. Prior to joining JPCERT/CC, he was engaged in security

monitoring and analysis operations at a foreign-affiliated IT vendor. He presented at CODE BLUE, BsidesLV, BlackHat USA Arsenal, Botconf, PacSec and FIRST Conference. JSAC organizer.

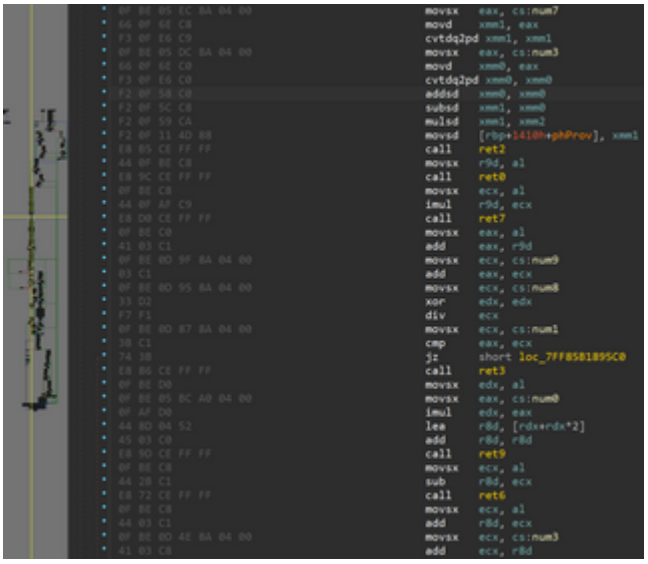
## Related articles

- 

  Update on Attacks by Threat Group APT-C-60

- 

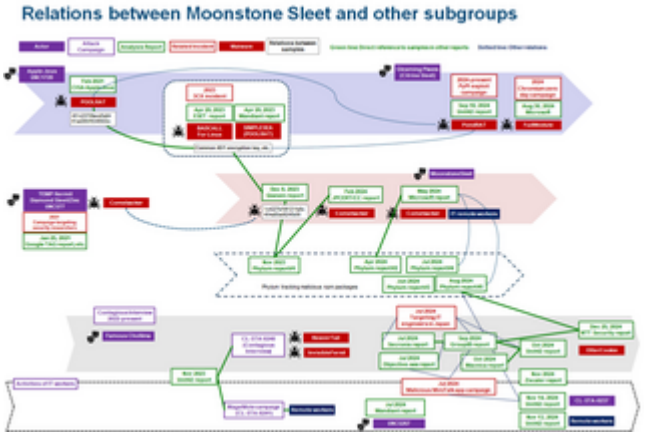

  CrossC2 Expanding Cobalt Strike Beacon to Cross-Platform Attacks

- 

  [Malware Identified in Attacks Exploiting Ivanti Connect Secure Vulnerabilities]

- 


  [DslogdRAT Malware Installed in Ivanti Connect Secure]

- 


  [Tempted to Classifying APT Actors: Practical Challenges of Attribution in the Case of Lazarus's Subgroup]

---

Source: https://blogs.jpcert.or.jp/en/2022/03/anti_upx_unpack.html