POS36-C. Observe correct revocation order while relinquishing privileges

In case of set-user-ID and set-group-ID programs, when the effective user ID and group ID are different from those of the real user, it is important to drop not only the user-level privileges but also the group privileges. While doing so, the order of revocation must be correct.

POSIX defines `setgid()` to have the following behavior [Open Group 2004]:

> *If the process has appropriate privileges, `setgid()` shall set the real group ID, effective group ID, and the saved set-group-ID of the calling process to `gid`.*
>
> *If the process does not have appropriate privileges, but `gid` is equal to the real group ID or the saved set-group-ID, `setgid()` shall set the effective group ID to `gid`; the real group ID and saved set-group-ID shall remain unchanged.*

Noncompliant Code Example

This noncompliant code example drops privileges to those of the real user and similarly drops the group privileges. However, the order is incorrect because the `setgid()` function must be run with superuser privileges, but the call to `setuid()` leaves the effective user ID as nonzero. As a result, if a vulnerability is discovered in the program that allows for the execution of arbitrary code, an attacker can regain the original group privileges.

```
/* Drop superuser privileges in incorrect order */

if (setuid(getuid()) == -1) {
  /* handle error condition */
}
if (setgid(getgid()) == -1) {
  /* handle error condition */
}

/* It is still possible to regain group privileges due to
 * incorrect relinquishment order */
```

Compliant Solution

This compliant solution relinquishes group privileges before taking away the user-level privileges so that both operations execute as intended.

```
/*  Drop superuser privileges in correct order */

if (setgid(getgid()) == -1) {
  /* handle error condition */
}
if (setuid(getuid()) == -1) {
  /* handle error condition */
}

/*
 * Not possible to regain group privileges due to correct relinquishment order
 */
```

Supplementary Group IDs

A process may have a number of supplementary group IDs in addition to its effective group ID, and the supplementary groups can allow privileged access to files. The `getgroups()` function returns an array that contains the supplementary group IDs and may also contain the effective group ID. The `setgroups()` function can set the supplementary group IDs and may also set the effective group ID on some systems. Using `setgroups()` usually requires privileges. Although POSIX defines the `getgroups()` function, it does not define `setgroups()`.

Under normal circumstances, `setuid()` and related calls do not alter the supplementary group IDs. However, a setuid-root program can alter its supplementary group IDs and then relinquish root privileges, in which case it maintains the supplementary group IDs but lacks the privilege necessary to relinquish them. Consequently, it is recommended that a program relinquish supplementary group IDs immediately before relinquishing root privileges. The following code defines a `set_sups()` function that will set the supplementary group IDs to a specific array on systems that support the `setgroups()` function.

```

```

```
/* Returns nonzero if the two group lists are equivalent (taking into
   account that the lists may differ wrt the egid */
int eql_sups(const int cursups_size, const gid_t* const cursups_list,
             const int targetsups_size, const gid_t* const targetsups_list) {
  int i;
  int j;
  const int n = targetsups_size;
  const int diff = cursups_size - targetsups_size;
  const gid_t egid = getegid();
  if (diff > 1 || diff < 0 ) {
    return 0;
  }
  for (i=0, j=0; i < n; i++, j++) {
    if (cursups_list[j] != targetsups_list[i]) {
      if (cursups_list[j] == egid) {
        i--; /* skipping j */
      } else {
        return 0;
      }
    }
  }
  /* If reached here, we're sure i==targetsups_size. Now, either
     j==cursups_size (skipped the egid or it wasn't there), or we didn't
     get to the egid yet because it's the last entry in cursups */
  return j == cursups_size ||
    (j+1 == cursups_size && cursups_list[j] == egid);
}


/* Sets the suplimentary group list, returns 0 if successful  */
int set_sups(const int target_sups_size,const gid_t* const target_sups_list) {
#ifdef __FreeBSD__
  const int targetsups_size = target_sups_size + 1;
  gid_t* const targetsups_list = (gid_t* const) malloc(sizeof(gid_t) * targetsups_size);
  if (targetsups_list == NULL) {
    /* handle error */
  }
  memcpy(targetsups_list+1, target_sups_list, target_sups_size * sizeof(gid_t) );
  targetsups_list[0] = getegid();
#else
  const int targetsups_size = target_sups_size;
  const gid_t* const targetsups_list = target_sups_list;
#endif
  if (geteuid() == 0) { /* allowed to setgroups, let's not take any chances */
    if (-1 == setgroups(targetsups_size, targetsups_list)) {
      /* handle error */
    }
  } else {
    int cursups_size = getgroups( 0, NULL);
    gid_t* cursups_list = (gid_t*) malloc( sizeof(gid_t) * cursups_size);
    if (cursups_list == NULL) {
      /* handle error */
    }
    if (-1 == getgroups( cursups_size, cursups_list)) {
      /* handle error */
    }
    if (!eql_sups(cursups_size, cursups_list, targetsups_size, targetsups_list)) {
      if (-1 == setgroups(targetsups_size, targetsups_list)) { /* will probably fail... :( */
        /* handle error */
      }
    }
    free( cursups_list);
  }

#ifdef __FreeBSD__
  free( targetsups_list);
#endif
  return 0;
}
```

Risk Assessment

Failing to observe the correct revocation order while relinquishing privileges allows an attacker to regain elevated privileges.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
POS36-C	high	probable	medium	P12	L1

Automated Detection

Tool	Version	Checker	Description
Axivion Bauhaus Suite	6.9.0	**CertC-POS36**	
Compass/ROSE			Can detect some violations of this rule. In particular, it warns when calls to `setgid()` are immediately preceded by a call to `setuid()`
Klocwork	2018	**SV.FIU.PROCESS_VARIANTS SV.USAGERULES. PERMISSIONS SV.USAGERULES. PROCESS_VARIANTS**	
Parasoft C /C++test	10.4.2	**CERT_C-POS36-a**	Observe correct revocation order while relinquishing privileges
Polyspace Bug Finder	R2019b	CERT C: Rule POS36-C	Checks for bad order of dropping privileges (rule fully covered)

Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the CERT website.

Related Guidelines

Key here (explains table format and definitions)

Taxonomy	Taxonomy item	Relationship
ISO/IEC TR 24772	Privilege Sandbox Issues [XYO]	Prior to 2018-01-12: CERT: Unspecified Relationship
CWE 2.11	CWE-696, Incorrect behavior order	2017-07-07: CERT: Rule subset of CWE

CERT-CWE Mapping Notes

Key here for mapping notes

CWE-696 and POS36-C

CWE-696 = Union(POS36-C, list) where list =

- Misordered executions besides dropping group privileges before dropping user privileges

Bibliography

[Chen 2002]	"Setuid Demystified"
[Dowd 2006]	Chapter 9, "UNIX I: Privileges and Files"
[Open Group 2004]	`setuid()` `setgid()`
[Tsafrir 2008]	"The Murky Issue of Changing Process Identity: Revising 'Setuid Demystified'"