author	Ritvik Budhiraja <rbudhiraja@microsoft.com>	2024-11-11 11:43:51 +0000
committer	Steve French <stfrench@microsoft.com>	2024-11-21 10:44:03 -0600
commit	db363b0a1d9e6b9dc556296f1b1007aeb496a8cf (patch)	
tree	6b5079359537034d238d9520c1ab187f01fe743f /fs/smb	
parent	7a2158b73c36903e8822dae5442c27d6d0e1014b (diff)	
download	linux-db363b0a1d9e6b9dc556296f1b1007aeb496a8cf.tar.gz	

diff options

context:	3
space:	include
mode:	unified

CIFS: New mount option for cifs.upcall namespace resolution

```
In the current implementation, the SMB filesystem on a mount point can
trigger upcalls from the kernel to the userspace to enable certain
functionalities like spnego, dns_resolution, amongst others. These upcalls
usually either happen in the context of the mount or in the context of an
application/user. The upcall handler for cifs, cifs.upcall already has
existing code which switches the namespaces to the caller's namespace
before handling the upcall. This behaviour is expected for scenarios like
multiuser mounts, but might not cover all single user scenario with
services such as Kubernetes, where the mount can happen from different
locations such as on the host, from an app container, or a driver pod
which does the mount on behalf of a different pod.

This patch introduces a new mount option called upcall_target, to
customise the upcall behaviour. upcall_target can take 'mount' and 'app'
as possible values. This aids use cases like Kubernetes where the mount
happens on behalf of the application in another container altogether.
Having this new mount option allows the mount command to specify where the
upcall should happen: 'mount' for resolving the upcall to the host
namespace, and 'app' for resolving the upcall to the ns of the calling
thread. This will enable both the scenarios where the Kerberos credentials
can be found on the application namespace or the host namespace to which
just the mount operation is "delegated".

Reviewed-by: Shyam Prasad <shyam.prasad@microsoft.com>
Reviewed-by: Bharath S M <bharathsm@microsoft.com>
Reviewed-by: Ronnie Sahlberg <ronniesahlberg@gmail.com>
Signed-off-by: Ritvik Budhiraja <rbudhiraja@microsoft.com>
Signed-off-by: Steve French <stfrench@microsoft.com>
```

Diffstat (limited to 'fs/smb')

-rw-r--r--	fs/smb/client/cifs_spnego.c	16
-rw-r--r--	fs/smb/client/cifsfs.c	25
-rw-r--r--	fs/smb/client/cifsglob.h	7
-rw-r--r--	fs/smb/client/connect.c	20
-rw-r--r--	fs/smb/client/fs_context.c	39
-rw-r--r--	fs/smb/client/fs_context.h	10

6 files changed, 117 insertions, 0 deletions

```
diff --git a/fs/smb/client/cifs_spnego.c b/fs/smb/client/cifs_spnego.c
index af7849e5974ff3..28f568b5fc2771 100644
--- a/fs/smb/client/cifs_spnego.c
+++ b/fs/smb/client/cifs_spnego.c
@@ -82,6 +82,9 @@ struct key_type cifs_spnego_key_type = {
 /* strlen of ";pid=0x" */
 #define PID_KEY_LEN		7
```

```
+/* strlen of ";upcall_target=" */
+#define UPCALL_TARGET_KEY_LEN  15
+
 /* get a key struct with a SPNEGO security blob, suitable for session setup */
 struct key *
 cifs_get_spnego_key(struct cifs_ses *sesInfo,
@@ -108,6 +111,11 @@ cifs_get_spnego_key(struct cifs_ses *sesInfo,
        if (sesInfo->user_name)
                desc_len += USER_KEY_LEN + strlen(sesInfo->user_name);

+       if (sesInfo->upcall_target == UPTARGET_MOUNT)
+               desc_len += UPCALL_TARGET_KEY_LEN + 5; // strlen("mount")
+       else
+               desc_len += UPCALL_TARGET_KEY_LEN + 3; // strlen("app")
+
        spnego_key = ERR_PTR(-ENOMEM);
        description = kzalloc(desc_len, GFP_KERNEL);
        if (description == NULL)
@@ -156,6 +164,14 @@ cifs_get_spnego_key(struct cifs_ses *sesInfo,
        dp = description + strlen(description);
        sprintf(dp, ";pid=0x%x", current->pid);

+       if (sesInfo->upcall_target == UPTARGET_MOUNT) {
+               dp = description + strlen(description);
+               sprintf(dp, ";upcall_target=mount");
+       } else {
+               dp = description + strlen(description);
+               sprintf(dp, ";upcall_target=app");
+       }
+
        cifs_dbg(FYI, "key description = %s\n", description);
        saved_cred = override_creds(spnego_cred);
        spnego_key = request_key(&cifs_spnego_key_type, description, "");

diff --git a/fs/smb/client/cifsfs.c b/fs/smb/client/cifsfs.c
index 20cafdff508106..97985347102792 100644
--- a/fs/smb/client/cifsfs.c
+++ b/fs/smb/client/cifsfs.c
@@ -546,6 +546,30 @@ static int cifs_show_devname(struct seq_file *m, struct dentry *root)
        return 0;
 }

+static void
+cifs_show_upcall_target(struct seq_file *s, struct cifs_sb_info *cifs_sb)
+{
+       if (cifs_sb->ctx->upcall_target == UPTARGET_UNSPECIFIED) {
+               seq_puts(s, ",upcall_target=app");
+               return;
+       }
+
+       seq_puts(s, ",upcall_target=");
+
+       switch (cifs_sb->ctx->upcall_target) {
+       case UPTARGET_APP:
+               seq_puts(s, "app");
+               break;
+       case UPTARGET_MOUNT:
+               seq_puts(s, "mount");
+               break;
+       default:
+               /* shouldn't ever happen */
+               seq_puts(s, "unknown");
+               break;
+       }
+}
+
 /*
```

```
  * cifs_show_options() is for displaying mount options in /proc/mounts.
  * Not all settable options are displayed but most of the important
@@ -562,6 +586,7 @@ cifs_show_options(struct seq_file *s, struct dentry *root)
        seq_show_option(s, "vers", tcon->ses->server->vals->version_string);
        cifs_show_security(s, tcon->ses);
        cifs_show_cache_flavor(s, cifs_sb);
+       cifs_show_upcall_target(s, cifs_sb);

        if (tcon->no_lease)
                seq_puts(s, ",nolease");

diff --git a/fs/smb/client/cifsglob.h b/fs/smb/client/cifsglob.h
index 5041b1ffc244b0..63d194ebbd7d9b 100644
--- a/fs/smb/client/cifsglob.h
+++ b/fs/smb/client/cifsglob.h
@@ -153,6 +153,12 @@ enum securityEnum {
        Kerberos,               /* Kerberos via SPNEGO */
 };

+enum upcall_target_enum {
+       UPTARGET_UNSPECIFIED, /* not specified, defaults to app */
+       UPTARGET_MOUNT, /* upcall to the mount namespace */
+       UPTARGET_APP, /* upcall to the application namespace which did the mount */
+};
+
 enum cifs_reparse_type {
        CIFS_REPARSE_TYPE_NFS,
        CIFS_REPARSE_TYPE_WSL,
@@ -1084,6 +1090,7 @@ struct cifs_ses {
        struct session_key auth_key;
        struct ntlmssp_auth *ntlmssp; /* ciphertext, flags, server challenge */
        enum securityEnum sectype; /* what security flavor was specified? */
+       enum upcall_target_enum upcall_target; /* what upcall target was specified? */
        bool sign;              /* is signing required? */
        bool domainAuto:1;
        bool expired_pwd;  /* track if access denied or expired pwd so can know if need to update */

diff --git a/fs/smb/client/connect.c b/fs/smb/client/connect.c
index 0ce2d704b1f3f8..0a97228c06b1e7 100644
--- a/fs/smb/client/connect.c
+++ b/fs/smb/client/connect.c
@@ -2339,6 +2339,26 @@ cifs_get_smb_ses(struct TCP_Server_Info *server, struct smb3_fs_context *ctx)

        ses->sectype = ctx->sectype;
        ses->sign = ctx->sign;
+
+       /*
+        *Explicitly marking upcall_target mount option for easier handling
+        * by cifs_spnego.c and eventually cifs.upcall.c
+        */
+
+       switch (ctx->upcall_target) {
+       case UPTARGET_UNSPECIFIED: /* default to app */
+       case UPTARGET_APP:
+               ses->upcall_target = UPTARGET_APP;
+               break;
+       case UPTARGET_MOUNT:
+               ses->upcall_target = UPTARGET_MOUNT;
+               break;
+       default:
+               // should never happen
+               ses->upcall_target = UPTARGET_APP;
+               break;
+       }
+
        ses->local_nls = load_nls(ctx->local_nls->charset);

        /* add server as first channel */
```

```
diff --git a/fs/smb/client/fs_context.c b/fs/smb/client/fs_context.c
index 5c5a52019efada..c87879e4739b1a 100644
--- a/fs/smb/client/fs_context.c
+++ b/fs/smb/client/fs_context.c
@@ -67,6 +67,12 @@ static const match_table_t cifs_secflavor_tokens = {
 	{ Opt_sec_err, NULL }
 };
 
+static const match_table_t cifs_upcall_target = {
+	{ Opt_upcall_target_mount, "mount" },
+	{ Opt_upcall_target_application, "app" },
+	{ Opt_upcall_target_err, NULL }
+};
+
 const struct fs_parameter_spec smb3_fs_parameters[] = {
 	/* Mount options that take no arguments */
 	fsparam_flag_no("user_xattr", Opt_user_xattr),
@@ -178,6 +184,7 @@ const struct fs_parameter_spec smb3_fs_parameters[] = {
 	fsparam_string("sec", Opt_sec),
 	fsparam_string("cache", Opt_cache),
 	fsparam_string("reparse", Opt_reparse),
+	fsparam_string("upcall_target", Opt_upcalltarget),
 
 	/* Arguments that should be ignored */
 	fsparam_flag("guest", Opt_ignore),
@@ -248,6 +255,29 @@ cifs_parse_security_flavors(struct fs_context *fc, char *value, struct smb3_fs_c
 	return 0;
 }
 
+static int
+cifs_parse_upcall_target(struct fs_context *fc, char *value, struct smb3_fs_context *ctx)
+{
+	substring_t args[MAX_OPT_ARGS];
+
+	ctx->upcall_target = UPTARGET_UNSPECIFIED;
+
+	switch (match_token(value, cifs_upcall_target, args)) {
+	case Opt_upcall_target_mount:
+		ctx->upcall_target = UPTARGET_MOUNT;
+		break;
+	case Opt_upcall_target_application:
+		ctx->upcall_target = UPTARGET_APP;
+		break;
+
+	default:
+		cifs_errorf(fc, "bad upcall target: %s\n", value);
+		return 1;
+	}
+
+	return 0;
+}
+
 static const match_table_t cifs_cacheflavor_tokens = {
 	{ Opt_cache_loose, "loose" },
 	{ Opt_cache_strict, "strict" },
@@ -1450,6 +1480,10 @@ static int smb3_fs_context_parse_param(struct fs_context *fc,
 		if (cifs_parse_security_flavors(fc, param->string, ctx) != 0)
 			goto cifs_parse_mount_err;
 		break;
+	case Opt_upcalltarget:
+		if (cifs_parse_upcall_target(fc, param->string, ctx) != 0)
+			goto cifs_parse_mount_err;
+		break;
 	case Opt_cache:
 		if (cifs_parse_cache_flavor(fc, param->string, ctx) != 0)
 			goto cifs_parse_mount_err;
@@ -1627,6 +1661,11 @@ static int smb3_fs_context_parse_param(struct fs_context *fc,
 	}
```

```
        /* case Opt_ignore: - is ignored as expected ... */

+       if (ctx->multiuser && ctx->upcall_target == UPTARGET_MOUNT) {
+               cifs_errorf(fc, "multiuser mount option not supported with upcalltarget set as 'mount'\n");
+               goto cifs_parse_mount_err;
+       }
+
        return 0;

  cifs_parse_mount_err:
```

```
diff --git a/fs/smb/client/fs_context.h b/fs/smb/client/fs_context.h
index 890d6d9d4a592f..67b7fc48ac583c 100644
--- a/fs/smb/client/fs_context.h
+++ b/fs/smb/client/fs_context.h
@@ -61,6 +61,12 @@ enum cifs_sec_param {
        Opt_sec_err
 };

+enum cifs_upcall_target_param {
+       Opt_upcall_target_mount,
+       Opt_upcall_target_application,
+       Opt_upcall_target_err
+};
+
 enum cifs_param {
        /* Mount options that take no arguments */
        Opt_user_xattr,
@@ -114,6 +120,8 @@ enum cifs_param {
        Opt_multichannel,
        Opt_compress,
        Opt_witness,
+       Opt_is_upcall_target_mount,
+       Opt_is_upcall_target_application,

        /* Mount options which take numeric value */
        Opt_backupuid,
@@ -157,6 +165,7 @@ enum cifs_param {
        Opt_sec,
        Opt_cache,
        Opt_reparse,
+       Opt_upcalltarget,

        /* Mount options to be ignored */
        Opt_ignore,
@@ -198,6 +207,7 @@ struct smb3_fs_context {
        umode_t file_mode;
        umode_t dir_mode;
        enum securityEnum sectype; /* sectype requested via mnt opts */
+       enum upcall_target_enum upcall_target; /* where to upcall for mount */
        bool sign; /* was signing requested via mnt opts? */
        bool ignore_signature:1;
        bool retry:1;
```