black hat EUROPE 2021
november 10-11, 2021
BRIEFINGS

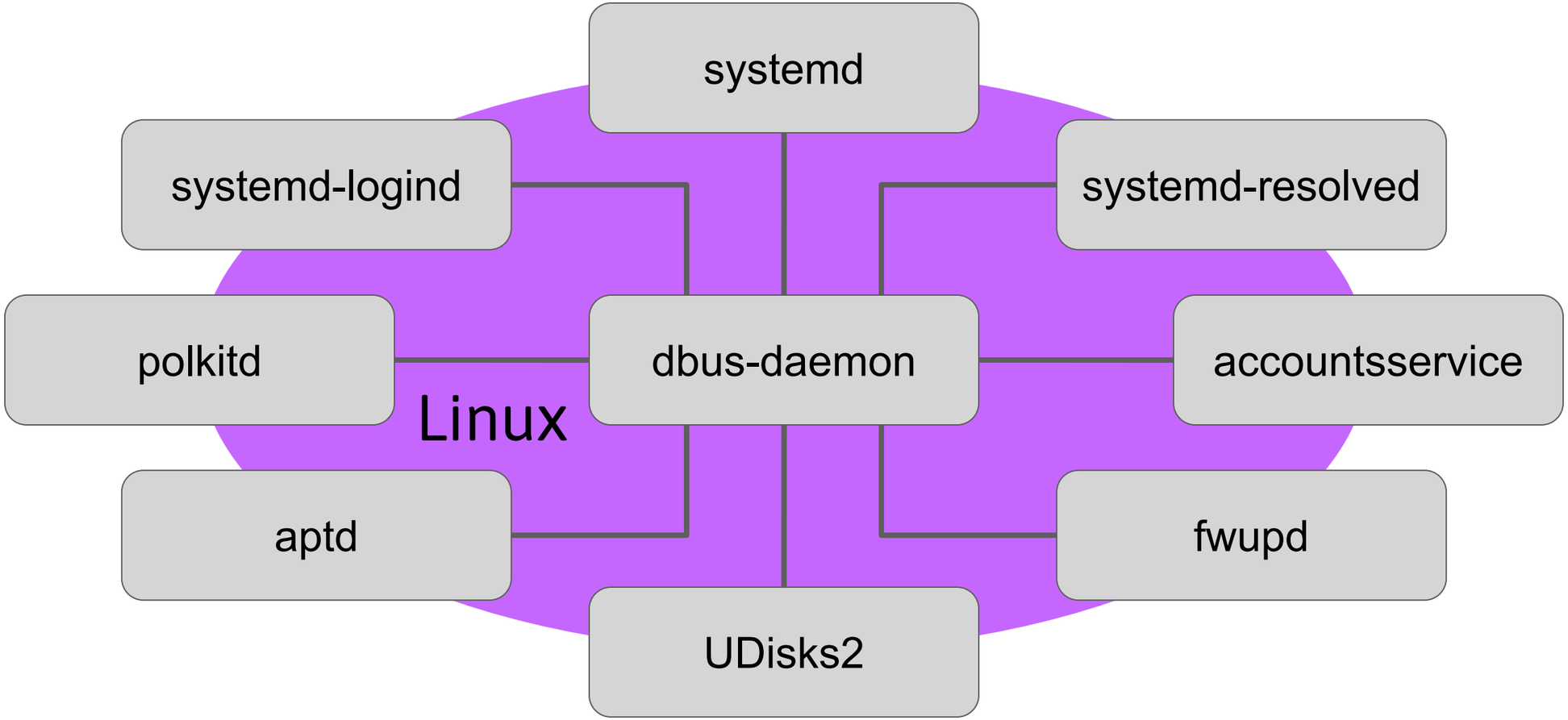
Message in a Broken Bottle: Exploring the Linux IPC Attack Surface

Kevin Backhouse
GitHub Security Lab

Overview

- D-Bus message passing system
  - architecture overview
  - security principles
  - polkit
- source code examples
- common bugs
- vulnerability demos

D-Bus demo


systemd
systemd-logind
systemd-resolved
polkitd
dbus-daemon
accountsservice
Linux
aptd
fwupd
UDisks2

Seth Vargo @sethvargo · Oct 19, 2019

Linux: do one thing and do it well

SystemD: nah

33 251 1.3K

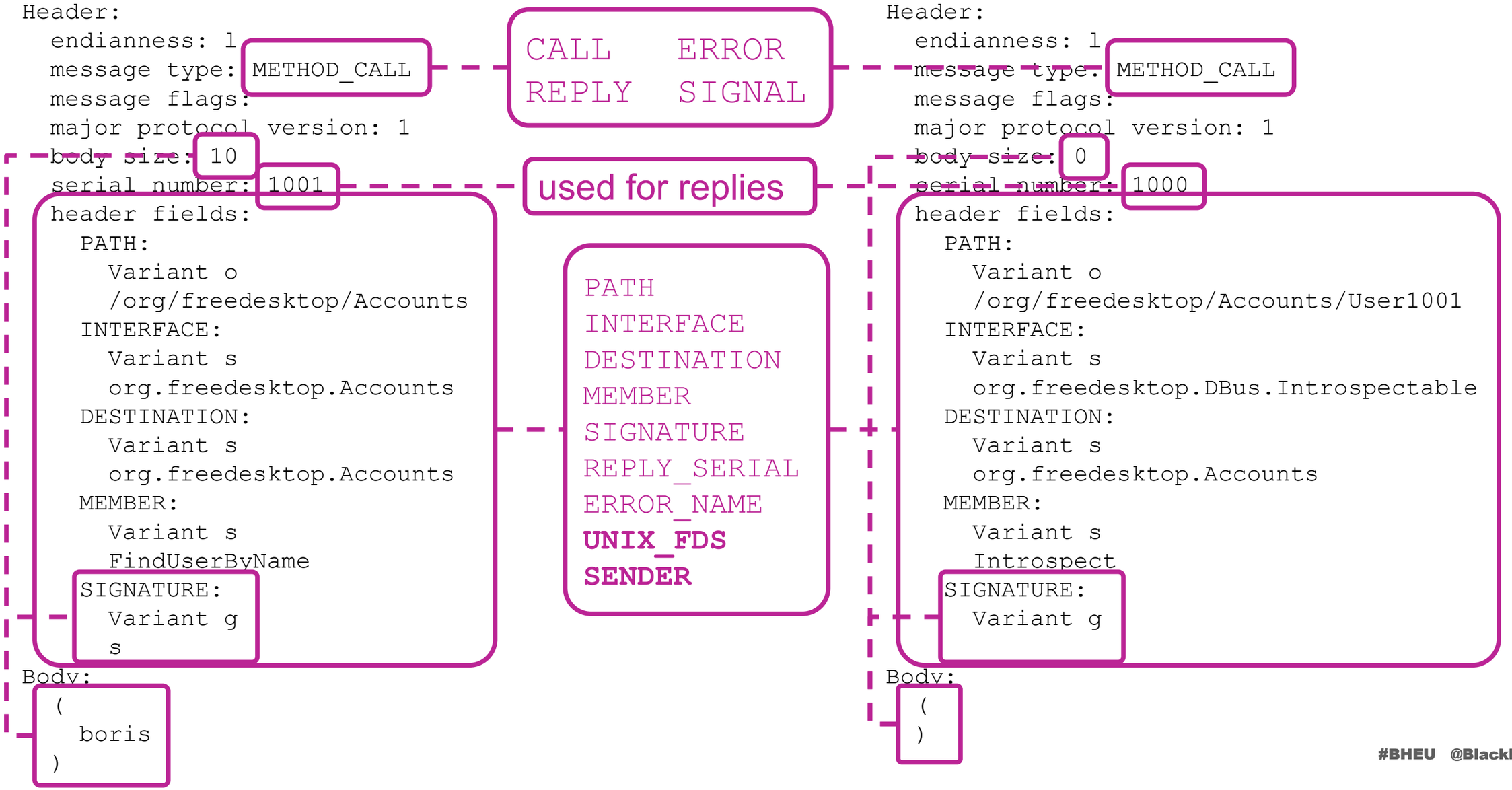
Jeremi M. Gosney
@jmgosney

Replying to @sethvargo and @IanColdwater

I'd just like to interject for a moment. What you're referring to as Linux, is in fact, the Linux Subsystem for SystemD, or as I've recently taken to calling it, LSD. Linux is not an operating system unto itself, but rather another free component of fully functioning SystemD.

4:12 AM · Oct 19, 2019 · Twitter for Android

24 Retweets **4** Quote Tweets **138** Likes


Header:
endianness: l
message type: METHOD_CALL
message flags:
major protocol version: 1
body size: 10
serial number: 1001
header fields:
PATH:
Variant o
/org/freedesktop/Accounts
INTERFACE:
Variant s
org.freedesktop.Accounts
DESTINATION:
Variant s
org.freedesktop.Accounts
MEMBER:
Variant s
FindUserByName
SIGNATURE:
Variant g
s
Body:
(
boris
)
CALL
ERROR
REPLY
SIGNAL
used for replies
PATH
INTERFACE
DESTINATION
MEMBER
SIGNATURE
REPLY_SERIAL
ERROR_NAME
UNIX_FDS
SENDER
Header:
endianness: l
message type: METHOD_CALL
message flags:
major protocol version: 1
body size: 0
serial number: 1000
header fields:
PATH:
Variant o
/org/freedesktop/Accounts/User1001
INTERFACE:
Variant s
org.freedesktop.DBus.Introspectable
DESTINATION:
Variant s
org.freedesktop.Accounts
MEMBER:
Variant s
Introspect
SIGNATURE:
Variant g
Body:
(
)

```
Header:
  endianness: l
  message type: METHOD_CALL
  message flags:
  major protocol version: 1
  body size: 0
  serial number: 4097
  header fields:
    PATH:
      Variant o
      /org/freedesktop/DBus
    INTERFACE:
      Variant s
      org.freedesktop.DBus
    DESTINATION:
      Variant s
      org.freedesktop.DBus
    MEMBER:
      Variant s
      Hello
    SIGNATURE:
      Variant g

Body:
  (
  )
```

```
Header:
  endianness: l
  message type: METHOD_RETURN
  message flags: NO_REPLY_EXPECTED
  major protocol version: 1
  body size: 12
  serial number: 1
  header fields:
    DESTINATION:
      Variant s
      :1.3591
    REPLY_SERIAL:
      Variant u
      4097
    SIGNATURE:
      Variant g
      s
    SENDER:
      Variant s
      org.freedesktop.DBus
Body:
  (
  :1.3591
  )
```

unique bus name

well-known name

D-Bus security features

- secure message delivery to correct recipient
- well-known names:
  - ownership rules
- unique bus names:
  - reliable way to check credentials
  - no more PID recycling vulnerabilities
- sending rules

The many config files of D-Bus

Accounts service config files:

```
/usr/share/dbus-1/system-services/org.freedesktop.Accounts.service
/usr/share/dbus-1/interfaces/org.freedesktop.Accounts.xml
/usr/lib/systemd/system/accounts-daemon.service
/etc/dbus-1/system.d/org.freedesktop.Accounts.conf
```

(Plus more later, when we get to polkit.)

/usr/lib/systemd/system/accounts-daemon.service

```
[Unit]
Description=Accounts Service

# In order to avoid races with identity-providing services like SSSD or
# winbind, we need to ensure that Accounts Service starts after
# nss-user-lookup.target
After=nss-user-lookup.target
Wants=nss-user-lookup.target

[Service]
Type=dbus
BusName=org.freedesktop.Accounts
ExecStart=/usr/lib/accountsservice/accounts-daemon
Environment=GVFS_DISABLE_FUSE=1
Environment=GIO_USE_VFS=local
```

/etc/dbus-1/system.d/org.freedesktop.Accounts.conf

```
<busconfig>
  <!-- Only root can own the service -->
  <policy user="root">
    <allow own="org.freedesktop.Accounts"/>
  </policy>

  <policy context="default">
    <allow send_destination="org.freedesktop.Accounts"/>
    <allow send_destination="org.freedesktop.Accounts"
           send_interface="org.freedesktop.DBus.Properties"/>
    <allow send_destination="org.freedesktop.Accounts"
           send_interface="org.freedesktop.DBus.Introspectable"/>
    <allow send_destination="org.freedesktop.Accounts.User"
           send_interface="org.freedesktop.DBus.Properties"/>
    <allow send_destination="org.freedesktop.Accounts.User"
```

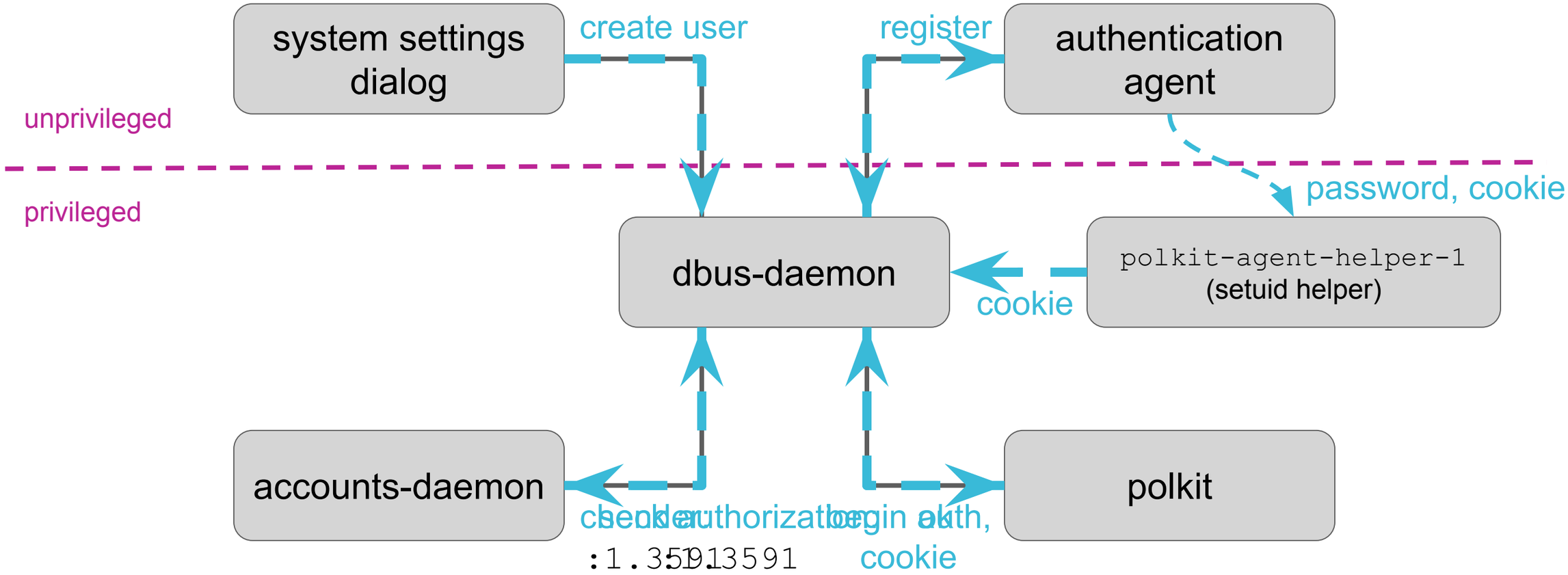
/usr/share/dbus-1/system.d/org.freedesktop.login1.conf

```
<busconfig>
  <policy user="root">
    <allow own="org.freedesktop.login1"/>
    <allow send_destination="org.freedesktop.login1"/>
    <allow receive_sender="org.freedesktop.login1"/>
  </policy>

  <policy context="default">
    <deny send_destination="org.freedesktop.login1"/>

    <allow send_destination="org.freedesktop.login1"
           send_interface="org.freedesktop.DBus.Introspectable"/>

    <allow send_destination="org.freedesktop.login1"
           send_interface="org.freedesktop.DBus.Peer"/>

    <allow send_destination="org.freedesktop.login1"
           send_interface="org.freedesktop.DBus.Properties"
```

deny by default

polkit demo

polkit architecture



polkit config files

- /usr/share/polkit-1/actions/
- /var/lib/polkit-1/localauthority

/usr/share/polkit-1/actions/org.freedesktop.accounts.policy

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE policyconfig PUBLIC "-//freedesktop//DTD PolicyKit Policy Configuration 1.0//EN"
"http://www.freedesktop.org/standards/PolicyKit/1.0/policyconfig.dtd">
<policyconfig>
  <vendor/>
  <vendor_url/>
  <icon_name>stock_person</icon_name>
  <action id="org.freedesktop.accounts.change-own-user-data">
    <description gettext-domain="accounts-service">Change your own user data</description>
    <message gettext-domain="accounts-service">Authentication is required to change your own user data</message>
    <defaults>
      <allow_any>yes</allow_any>
      <allow_inactive>yes</allow_inactive>
      <allow_active>yes</allow_active>
    </defaults>
  </action>
  <action id="org.freedesktop.accounts.user-administration">
    <description gettext-domain="accounts-service">Manage user accounts</description>
    <message gettext-domain="accounts-service">Authentication is required to change user data</message>
    <defaults>
      <allow_any>auth_admin</allow_any>
      <allow_inactive>auth_admin</allow_inactive>
      <allow_active>auth_admin_keep</allow_active>
    </defaults>
```

/var/lib/polkit-1/localauthority/10-vendor.d/gnome-initial-setup.pkla

```
[Allow the gnome-initial-setup user to mount disks, set the locale, keyboard, date/time, control the network and create users without prompting]
Identity=unix-user:gnome-initial-setup
Action=org.freedesktop.udisks2.filesystem-mount-system;org.freedesktop.hostname1.*;org.freedesktop.NetworkManager.*;org.freedesktop.locale1.*;org.freedesktop.accounts.*;org.freedesktop.timedate1.*;org.freedesktop.realmd.*;org.freedesktop.RealtimeKit1.*
ResultAny=no
ResultInactive=no
ResultActive=yes
```

D-Bus code examples

Main libraries:

- dbus
- python3-dbus
- glib2
- systemd

What they do:

- parse/serialize D-Bus messages
- listen on a (Unix) socket for messages
- dispatch incoming method calls, method replies, and signals

example: python3-aptdaemon

```
@dbus_deferred_method(APTDAEMON_DBUS_INTERFACE,
                      in_signature="as", out_signature="s",
                      sender_keyword="sender")
def InstallPackages(self, package_names, sender):
    """Fetch and install the given packages from the repositories.

    <SNIP>

    Requires the ``org.debian.apt.install-or-remove-packages``
    :ref:`PolicyKit privilege <policykit>`.

    :param package_names: Packages to be upgraded
    :type package_names: as

    :returns: The D-Bus path of the new transaction object which
        performs this action.
    """
    log.info("InstallPackages() was called: %s" % package_names)
    self._check_package_names(package_names)
    return self._create_trans(enums.ROLE_INSTALL_PACKAGES, sender,
                              packages=(package_names, [], [], [], [], []))
```

```
@dbus_deferred_method(APTDAEMON_DBUS_INTERFACE,
                      in_signature="as", out_signature="s",
                      sender_keyword="sender")
def InstallPackages(self, package_names, sender):
    """Fetch and install the given packages from the repositories.

    <SNIP>

    Requires the ``org.debian.apt.install-or-remove-packages``
    :ref:`PolicyKit privilege <policykit>`.

    :param package_names: Packages to be upgraded
    :type package_names: as

    :returns: The D-Bus path of the new transaction object which
        performs this action.
    """
    log.info("InstallPackages() was called: %s"  % package_names)
    self._check_package_names(package_names)
    return self._create_trans(enums.ROLE_INSTALL_PACKAGES, sender,
                              packages=(package_names, [], [], [], [], []))
```

polkit check authorization (python)

```
def _check_authorization(subject, action_id, timeout, bus, flags=None):
    def policykit_done(xxx_todo_changeme):
        (authorized, challenged, auth_details) = xxx_todo_changeme
        if authorized:
            deferred.callback(auth_details)
        elif challenged:
            deferred.errback(AuthorizationFailed(subject, action_id))
        else:
            deferred.errback(NotAuthorizedError(subject, action_id))
    if not bus:
        bus = dbus.SystemBus()
    # Set the default flags
    if flags is None:
        flags = CHECK_AUTH_ALLOW_USER_INTERACTION
    deferred = Deferred()
    pk = bus.get_object("org.freedesktop.PolicyKit1",
                        "/org/freedesktop/PolicyKit1/Authority")
    details = {}
    pk.CheckAuthorization(
        subject, action_id, details, flags, "",
        dbus_interface="org.freedesktop.PolicyKit1.Authority",
```

```
def _check_authorization(subject, action_id, timeout, bus, flags=None):
    def policykit_done(xxx_todo_changeme):
        (authorized, challenged, auth_details) = xxx_todo_changeme
        if authorized:
            deferred.callback(auth_details)
        elif challenged:
            deferred.errback(AuthorizationFailed(subject, action_id))
        else:
            deferred.errback(NotAuthorizedError(subject, action_id))
    if not bus:
        bus = dbus.SystemBus()
    # Set the default flags
    if flags is None:
        flags = CHECK_AUTH_ALLOW_USER_INTERACTION
    deferred = Deferred()
    pk = bus.get_object("org.freedesktop.PolicyKit1",
                        "/org/freedesktop/PolicyKit1/Authority")
    details = {}
    pk.CheckAuthorization(
        subject, action_id, details, flags,  "",
        dbus_interface="org.freedesktop.PolicyKit1.Authority",
        timeout=timeout,
        reply_handler=policykit_done,
        error_handler=deferred.errback)
    return deferred
```

example: systemd-logind

```
static int method_set_wall_message(sd_bus_message *message, void *userdata, sd_bus_error *error) {
        int r;
        Manager *m = userdata;
        char *wall_message;
        unsigned enable_wall_messages;
        r = sd_bus_message_read(message, "sb", &wall_message, &enable_wall_messages);
        if (r < 0)
                return r;
        <SNIP>
        r = bus_verify_polkit_async(message,
                                    CAP_SYS_ADMIN,
                                    "org.freedesktop.login1.set-wall-message",
                                    NULL,
                                    false,
                                    UID_INVALID,
                                    &m->polkit_registry,
                                    error);
        if (r < 0)
                return r;
        if (r == 0)
                return 1; /* Will call us back */
```

```
static int method_set_wall_message(sd_bus_message *message, void *userdata, sd_bus_error *error) {
        int r;
        Manager *m = userdata;
        char *wall_message;
        unsigned enable_wall_messages;
        r = sd_bus_message_read(message, "sb", &wall_message, &enable_wall_messages);
        if (r < 0)
                return r;
        <SNIP>
        r = bus_verify_polkit_async(message,
                                    CAP_SYS_ADMIN,
                                    "org.freedesktop.login1.set-wall-message",
                                    NULL,
                                    false,
                                    UID_INVALID,
                                    &m->polkit_registry,
                                    error);
        if (r < 0)
                return r;
        if (r == 0)
                return 1; /* Will call us back */
        r = free_and_strdup(&m->wall_message, empty_to_null(wall_message));
        if (r < 0)
                return log_oom();
        m->enable_wall_messages = enable_wall_messages;
 done:
        return sd_bus_reply_method_return(message, NULL);
}
```

Common bugs

- ~~memory corruption~~
- privilege dropping mistakes:
  - accidental file existence disclosure
  - SIGSTOP denial of service
  - file system TOCTOU
- bad error handling

file existence disclosure: aptdaemon (CVE-2020-16128)

```
def _set_debconf(self, debconf_socket):
    """Set the socket of the debconf proxy.

    The worker process forwards all debconf commands through this
    socket by using the passthrough frontend. On the client side
    debconf-communicate should be connected to the socket.

    Can only be changed before the transaction is started.

    Keyword arguments:
    debconf_socket: absolute path to the socket
    """
    if self.status != enums.STATUS_SETTING_UP:
        raise errors.TransactionAlreadyRunning()
    if not os.access(debconf_socket, os.W_OK):
        raise errors.AptDaemonError("socket does not exist: "
                                    "%s" % debconf_socket)
    if not os.stat(debconf_socket)[4] == self.uid:
        raise errors.AptDaemonError("socket '%s' has to be owned by the "
                                    "owner of the "
                                    "transaction" % debconf socket)
```

```
def _set_debconf(self, debconf_socket):
    """Set the socket of the debconf proxy.

    The worker process forwards all debconf commands through this
    socket by using the passthrough frontend. On the client side
    debconf-communicate should be connected to the socket.

    Can only be changed before the transaction is started.

    Keyword arguments:
    debconf_socket: absolute path to the socket
    """
    if self.status != enums.STATUS_SETTING_UP:
        raise errors.TransactionAlreadyRunning()
    if not os.access(debconf_socket, os.W_OK):
        raise errors.AptDaemonError("socket does not exist: "
                                    "%s" % debconf_socket)
    if not os.stat(debconf_socket)[4] == self.uid:
        raise errors.AptDaemonError("socket '%s' has to be owned by the "
                                    "owner of the "
                                    "transaction" % debconf_socket)
    self.debconf = dbus.String(debconf_socket)
    self.PropertyChanged("DebconfSocket", self.debconf)
```

privilege dropping DoS: accountsservice (CVE-2020-16126)

```
static gboolean is_in_pam_environment (User *user, const gchar *property)
{
        gboolean ret = FALSE;
        const gchar *prefix;
        FILE *fp;
        g_autofree gchar *pam_env = NULL;

        if (g_strcmp0 (property, "Language") == 0)
                prefix = "LANG";
        else if (g_strcmp0 (property, "FormatsLocale") == 0)
                prefix = "LC_TIME";
        else
                return FALSE;

        pam_env = g_build_path ("/", accounts_user_get_home_directory (ACCOUNTS_USER (user)), ".pam_environment", NULL);

        if (!user_drop_privileges_to_user (user))
                return FALSE;
        if ((fp = fopen (pam_env, "r"))) {
                gchar line[50];
                while ((fgets (line, 50, fp)) != NULL) {
```

```
static gboolean is_in_pam_environment (User *user, const gchar *property)
{
        gboolean ret = FALSE;
        const gchar *prefix;
        FILE *fp;
        g_autofree gchar *pam_env = NULL;

        <SNIP>

        pam_env = g_build_path ("/", accounts_user_get_home_directory (ACCOUNTS_USER (user)),".pam_environment", NULL);

        if (!user_drop_privileges_to_user (user))
                return FALSE;
        if ((fp = fopen (pam_env, "r"))) {
                gchar line[50];
                while ((fgets (line, 50, fp)) != NULL) {
                        if (g_str_has_prefix (line, prefix)) {
                                ret = TRUE;
                                break;
                        }
                }
                fclose (fp);
        }
        user_regain_privileges ();

        return ret;
}
```

Bad error handling: gdm3 (CVE-2020-16125)

```
static void
look_for_existing_users_sync (GdmDisplay *self)
{
        GdmDisplayPrivate *priv;
        GError *error = NULL;
        GVariant *call_result;
        GVariant *user_list;

        priv = gdm_display_get_instance_private (self);
        priv->accountsservice_proxy = g_dbus_proxy_new_sync (priv->connection,
                                                             0, NULL,
                                                             "org.freedesktop.Accounts",
                                                             "/org/freedesktop/Accounts",
                                                             "org.freedesktop.Accounts",
                                                             NULL,
                                                             &error);

        if (!priv->accountsservice_proxy) {
                g_warning ("Failed to contact accountsservice: %s", error->message);
                goto out;
        }
```

```
static void
look_for_existing_users_sync  (GdmDisplay *self)
{

        <SNIP>

        call_result = g_dbus_proxy_call_sync (priv->accountsservice_proxy,
                                              "ListCachedUsers",
                                              NULL,
                                              0,
                                              -1,
                                              NULL,
                                              &error);

        if (!call_result) {
                g_warning ("Failed to list cached users: %s", error->message);
                goto out;
        }

        g_variant_get (call_result, "(@ao)", &user_list);
        priv->have_existing_user_accounts = g_variant_n_children (user_list) > 0;
        g_variant_unref (user_list);
        g_variant_unref (call_result);

out:

        g_clear_error (&error);
}
```

Demo: gdm3/accountsservice LPE (CVE-2020-16125, CVE-2020-16126)

Bad error handling: polkit (CVE-2021-3560)

```
static gboolean
polkit_system_bus_name_get_creds_sync (PolkitSystemBusName           *system_bus_name,
                                       guint32                       *out_uid,
                                       guint32                       *out_pid,
                                       GCancellable                  *cancellable,
                                       GError                       **error)
{
  gboolean ret = FALSE;
  AsyncGetBusNameCredsData data = { 0, };
  GDBusConnection *connection = NULL;
  GMainContext *tmp_context = NULL;

  connection = g_bus_get_sync (G_BUS_TYPE_SYSTEM, cancellable, error);
  if (connection == NULL)
    goto out;

  data.error = error;

  tmp_context = g_main_context_new ();
  g_main_context_push_thread_default (tmp_context);
```

```
g_dbus_connection_call (connection,
                        "org.freedesktop.DBus",        /* name */
                        "/org/freedesktop/DBus",       /* object path */
                        "org.freedesktop.DBus",        /* interface name */
                        "GetConnectionUnixProcessID", /* method */
                        g_variant_new ("(s)", system_bus_name->name),
                        G_VARIANT_TYPE ("(u)"),
                        G_DBUS_CALL_FLAGS_NONE,
                        -1,
                        cancellable,
                        on_retrieved_unix_uid_pid,
                        &data);

  while (!((data.retrieved_uid && data.retrieved_pid) || data.caught_error))
    g_main_context_iteration (tmp_context, TRUE);

  if (out_uid)
    *out_uid = data.uid;
  if (out_pid)
    *out_pid = data.pid;
  ret = TRUE;
out:
  if (tmp_context)
    {
      g_main_context_pop_thread_default (tmp_context);
      g_main_context_unref (tmp_context);
    }
  if (connection != NULL)
    g_object_unref (connection);
  return ret;
```

Demo: polkit LPE (CVE-2021-3560)

Conclusion

- D-Bus is an IPC messaging system widely used by system services on Linux
- Design is good, but non-trivial. Complexity can cause accidents.
- Common bugs:
  - privilege dropping mistakes
  - bad error handling