author Andrew Donnellan <ajd@linux.ibm.com> 2020-08-20 14:45:12 +1000
committer Michael Ellerman <mpe@ellerman.id.au> 2020-10-06 23:22:27 +1100
commit bd59380c5ba4147dcbaad3e582b55ccfd120b764 (patch)
tree 3c4e5103b502d8ff8c271b3ef20185c3f8178d40
parent 3b6c3adbb2fa42749c3d38cfc4d4d0b7e096bb7b (diff)
download linux-bd59380c5ba4147dcbaad3e582b55ccfd120b764.tar.gz

diff options
context: 3
space: include
mode: unified

powerpc/rtas: Restrict RTAS requests from userspace

A number of userspace utilities depend on making calls to RTAS to retrieve
information and update various things.

The existing API through which we expose RTAS to userspace exposes more
RTAS functionality than we actually need, through the sys_rtas syscall,
which allows root (or anyone with CAP_SYS_ADMIN) to make any RTAS call they
want with arbitrary arguments.

Many RTAS calls take the address of a buffer as an argument, and it's up to
the caller to specify the physical address of the buffer as an argument. We
allocate a buffer (the "RMO buffer") in the Real Memory Area that RTAS can
access, and then expose the physical address and size of this buffer in
/proc/powerpc/rtas/rmo_buffer. Userspace is expected to read this address,
poke at the buffer using /dev/mem, and pass an address in the RMO buffer to
the RTAS call.

However, there's nothing stopping the caller from specifying whatever
address they want in the RTAS call, and it's easy to construct a series of
RTAS calls that can overwrite arbitrary bytes (even without /dev/mem
access).

Additionally, there are some RTAS calls that do potentially dangerous
things and for which there are no legitimate userspace use cases.

In the past, this would not have been a particularly big deal as it was
assumed that root could modify all system state freely, but with Secure
Boot and lockdown we need to care about this.

We can't fundamentally change the ABI at this point, however we can address
this by implementing a filter that checks RTAS calls against a list
of permitted calls and forces the caller to use addresses within the RMO
buffer.

The list is based off the list of calls that are used by the librtas
userspace library, and has been tested with a number of existing userspace
RTAS utilities. For compatibility with any applications we are not aware of
that require other calls, the filter can be turned off at build time.

Cc: stable@vger.kernel.org
Reported-by: Daniel Axtens <dja@axtens.net>
Signed-off-by: Andrew Donnellan <ajd@linux.ibm.com>
Signed-off-by: Michael Ellerman <mpe@ellerman.id.au>
Link: https://lore.kernel.org/r/20200820044512.7543-1-ajd@linux.ibm.com

Diffstat

-rw-r--r--	arch/powerpc/Kconfig	13
-rw-r--r--	arch/powerpc/kernel/rtas.c	153

2 files changed, 166 insertions, 0 deletions

```
diff --git a/arch/powerpc/Kconfig b/arch/powerpc/Kconfig
index fe0e6b317cc26..6c76caa950e16 100644
--- a/arch/powerpc/Kconfig
+++ b/arch/powerpc/Kconfig
@@ -1004,6 +1004,19 @@ config PPC_SECVAR_SYSFS
          read/write operations on these variables. Say Y if you have
          secure boot enabled and want to expose variables to userspace.

+config PPC_RTAS_FILTER
+       bool "Enable filtering of RTAS syscalls"
+       default y
+       depends on PPC_RTAS
+       help
+         The RTAS syscall API has security issues that could be used to
+         compromise system integrity. This option enforces restrictions on the
+         RTAS calls and arguments passed by userspace programs to mitigate
+         these issues.
+
+         Say Y unless you know what you are doing and the filter is causing
+         problems for you.
+
 endmenu

 config ISA_DMA_API

diff --git a/arch/powerpc/kernel/rtas.c b/arch/powerpc/kernel/rtas.c
index 806d554ce3577..954f41676f692 100644
--- a/arch/powerpc/kernel/rtas.c
+++ b/arch/powerpc/kernel/rtas.c
@@ -992,6 +992,147 @@ struct pseries_errorlog *get_pseries_errorlog(struct rtas_error_log *log,
        return NULL;
 }

+#ifdef CONFIG_PPC_RTAS_FILTER
+
+/*
+ * The sys_rtas syscall, as originally designed, allows root to pass
+ * arbitrary physical addresses to RTAS calls. A number of RTAS calls
+ * can be abused to write to arbitrary memory and do other things that
+ * are potentially harmful to system integrity, and thus should only
+ * be used inside the kernel and not exposed to userspace.
+ *
+ * All known legitimate users of the sys_rtas syscall will only ever
+ * pass addresses that fall within the RMO buffer, and use a known
+ * subset of RTAS calls.
+ *
+ * Accordingly, we filter RTAS requests to check that the call is
+ * permitted, and that provided pointers fall within the RMO buffer.
+ * The rtas_filters list contains an entry for each permitted call,
+ * with the indexes of the parameters which are expected to contain
+ * addresses and sizes of buffers allocated inside the RMO buffer.
+ */
```

```
+struct rtas_filter {
+	const char *name;
+	int token;
+	/* Indexes into the args buffer, -1 if not used */
+	int buf_idx1;
+	int size_idx1;
+	int buf_idx2;
+	int size_idx2;
+
+	int fixed_size;
+};
+
+static struct rtas_filter rtas_filters[] __ro_after_init = {
+	{ "ibm,activate-firmware", -1, -1, -1, -1, -1 },
+	{ "ibm,configure-connector", -1, 0, -1, 1, -1, 4096 },  /* Special cased */
+	{ "display-character", -1, -1, -1, -1, -1 },
+	{ "ibm,display-message", -1, 0, -1, -1, -1 },
+	{ "ibm,errinjct", -1, 2, -1, -1, -1, 1024 },
+	{ "ibm,close-errinjct", -1, -1, -1, -1, -1 },
+	{ "ibm,open-errinct", -1, -1, -1, -1, -1 },
+	{ "ibm,get-config-addr-info2", -1, -1, -1, -1, -1 },
+	{ "ibm,get-dynamic-sensor-state", -1, 1, -1, -1, -1 },
+	{ "ibm,get-indices", -1, 2, 3, -1, -1 },
+	{ "get-power-level", -1, -1, -1, -1, -1 },
+	{ "get-sensor-state", -1, -1, -1, -1, -1 },
+	{ "ibm,get-system-parameter", -1, 1, 2, -1, -1 },
+	{ "get-time-of-day", -1, -1, -1, -1, -1 },
+	{ "ibm,get-vpd", -1, 0, -1, 1, 2 },
+	{ "ibm,lpar-perftools", -1, 2, 3, -1, -1 },
+	{ "ibm,platform-dump", -1, 4, 5, -1, -1 },
+	{ "ibm,read-slot-reset-state", -1, -1, -1, -1, -1 },
+	{ "ibm,scan-log-dump", -1, 0, 1, -1, -1 },
+	{ "ibm,set-dynamic-indicator", -1, 2, -1, -1, -1 },
+	{ "ibm,set-eeh-option", -1, -1, -1, -1, -1 },
+	{ "set-indicator", -1, -1, -1, -1, -1 },
+	{ "set-power-level", -1, -1, -1, -1, -1 },
+	{ "set-time-for-power-on", -1, -1, -1, -1, -1 },
+	{ "ibm,set-system-parameter", -1, 1, -1, -1, -1 },
+	{ "set-time-of-day", -1, -1, -1, -1, -1 },
+	{ "ibm,suspend-me", -1, -1, -1, -1, -1 },
+	{ "ibm,update-nodes", -1, 0, -1, -1, -1, 4096 },
+	{ "ibm,update-properties", -1, 0, -1, -1, -1, 4096 },
+	{ "ibm,physical-attestation", -1, 0, 1, -1, -1 },
+};
+
+static bool in_rmo_buf(u32 base, u32 end)
+{
+	return base >= rtas_rmo_buf &&
+		base < (rtas_rmo_buf + RTAS_RMOBUF_MAX) &&
+		base <= end &&
+		end >= rtas_rmo_buf &&
+		end < (rtas_rmo_buf + RTAS_RMOBUF_MAX);
+}
+
+static bool block_rtas_call(int token, int nargs,
+			    struct rtas_args *args)
+{
+	int i;
+
+	for (i = 0; i < ARRAY_SIZE(rtas_filters); i++) {
+		struct rtas_filter *f = &rtas_filters[i];
+		u32 base, size, end;
+
+		if (token != f->token)
+			continue;
+
+		if (f->buf_idx1 != -1) {
+			base = be32_to_cpu(args->args[f->buf_idx1]);
+			if (f->size_idx1 != -1)
+				size = be32_to_cpu(args->args[f->size_idx1]);
+			else if (f->fixed_size)
+				size = f->fixed_size;
+			else
+				size = 1;
+
+			end = base + size - 1;
+			if (!in_rmo_buf(base, end))
+				goto err;
+		}
+
+		if (f->buf_idx2 != -1) {
+			base = be32_to_cpu(args->args[f->buf_idx2]);
+			if (f->size_idx2 != -1)
+				size = be32_to_cpu(args->args[f->size_idx2]);
+			else if (f->fixed_size)
+				size = f->fixed_size;
+			else
+				size = 1;
+			end = base + size - 1;
+
+			/*
+			 * Special case for ibm,configure-connector where the
+			 * address can be 0
+			 */
+			if (!strcmp(f->name, "ibm,configure-connector") &&
+			    base == 0)
+				return false;
+
+			if (!in_rmo_buf(base, end))
+				goto err;
+		}
+
+		return false;
+	}
+
+err:
+	pr_err_ratelimited("sys_rtas: RTAS call blocked - exploit attempt?\n");
+	pr_err_ratelimited("sys_rtas: token=0x%x, nargs=%d (called by %s)\n",
+			   token, nargs, current->comm);
+	return true;
+}
+
+#else
+
+static bool block_rtas_call(int token, int nargs,
+			    struct rtas_args *args)
+{
+	return false;
+}
+
+#endif /* CONFIG_PPC_RTAS_FILTER */
+
```

```
 /* We assume to be passed big endian arguments */
 SYSCALL_DEFINE1(rtas, struct rtas_args __user *, uargs)
 {
@@ -1029,6 +1170,9 @@ SYSCALL_DEFINE1(rtas, struct rtas_args __user *, uargs)
        args.rets = &args.args[nargs];
        memset(args.rets, 0, nret * sizeof(rtas_arg_t));

+       if (block_rtas_call(token, nargs, &args))
+               return -EINVAL;
+
        /* Need to handle ibm,suspend_me call specially */
        if (token == ibm_suspend_me_token) {

@@ -1090,6 +1234,9 @@ void __init rtas_initialize(void)
        unsigned long rtas_region = RTAS_INSTANTIATE_MAX;
        u32 base, size, entry;
        int no_base, no_size, no_entry;
+#ifdef CONFIG_PPC_RTAS_FILTER
+       int i;
+#endif

        /* Get RTAS dev node and fill up our "rtas" structure with infos
         * about it.
@@ -1129,6 +1276,12 @@ void __init rtas_initialize(void)
 #ifdef CONFIG_RTAS_ERROR_LOGGING
        rtas_last_error_token = rtas_token("rtas-last-error");
 #endif
+
+#ifdef CONFIG_PPC_RTAS_FILTER
+       for (i = 0; i < ARRAY_SIZE(rtas_filters); i++) {
+               rtas_filters[i].token = rtas_token(rtas_filters[i].name);
+       }
+#endif
 }

 int __init early_init_dt_scan_rtas(unsigned long node,
```