RTEMS Security Assessment

NCC Group Security Services USA
Version 1.0 – August 15, 2025

Prepared By
Catalin Visinescu
Mark Paruzel
Jonathan Lindsay

1 Table of Contents

1 Table of Contents 2
2 Executive Summary 3
3 Adversarial Model 6
4 Table of Findings 8
5 Finding Details 9
6 Out of Date Software 53
7 Static Analysis & Fuzzing 55
8 Finding Field Definitions 56
9 Contact Info 58

2 Executive Summary

Synopsis

NCC Group conducted a time-boxed security assessment of the Real-Time Executive for Multiprocessor Systems (RTEMS) project. RTEMs is a real-time operating system geared towards small and medium-sized embedded systems, and has been deployed in a wide array of environments ranging from satellites and rovers to nuclear reactors and physics experiments. It is a multi-threaded OS that runs within a single address space with no separation of kernel and user space.

Scope

At its core, RTEMS provides standard operating system features, such as scheduling, message passing, etc. However, the project also incorporates numerous features from third-party libraries. As such, when reviewing this project, NCC Group focused mainly on ways an attacker could enter the system remotely, be it through RTEMS-developed services or third-party libraries. A broad review of the project was conducted, due to the large amount of files and services it contains. This included reviewing the following areas:

- **Networking:** the project supports a number of different networking stacks. The older networking stack, `libnetworking`, is based on FreeBSD 4.4 and is currently deprecated. Documentation recommends only using this stack for legacy projects. The replacement for this stack resides in the `rtems-libbsd` repository, which is based on FreeBSD 12.1. Due to the age of `libnetworking` and `rtems-libbsd`, a number of vulnerabilities are already known and published, thus consultants focused primarily on determining whether the networking stack received back-ported fixes, or if it was still vulnerable to existing attacks. Lastly, RTEMS also supports integration with `LwIP`, which is a lightweight networking library. This project was lightly reviewed for open issues and CVEs, but the code was largely not scrutinized for vulnerabilities as it is slated for a future release.
- **System Services:** The system comes with a number of services, or daemons, available to be built as part of a BSP. These includes DHCP, FTP/TFTP, NFS, Telnet/Shell, IPSec, Mongoose HTTP, DNS, and PPP. Many of these services are themselves open source projects, and consultants evaluated these projects for existing and new vulnerabilities.
- **File System:** RTEMS supports a number of file systems, such as FAT, JFFS2/YAFFS2, IMFS, NFS, and their own RTEMS File System (RFS). Some of these are open source projects, and consultants reviewed those projects for open vulnerabilities.
- **Device Drivers:** Consultants looked at various drivers, such as SPI and I2C handling, as well as network and device drivers. The RTEMS source tree also contains drivers developed by Freescale, which are part of the Linux source tree.
- **Operating System:** The RTEMS kernel and application run in the same memory space, and thus any security vulnerabilities in the system likely impact the kernel as well. Critical parts of the RTEMS OS were reviewed, such as scheduling, message and event passing, memory barriers, and the heap implementation.

This research project consisted primarily of code review, focusing on the following repositories and versions:

- https://github.com/RTEMS/rtems
- https://github.com/RTEMS/rtems-libbsd
- https://git.rtems.org/rtems-lwip

NCC Group targeted RTEMS `tag 5.3` and RTEMS-libbsd `tag 5.1-freebsd-12`.

Limitations

Due to the size of RTEMS, not all components were subject to review. The following areas were not reviewed:

- **FreeBSD:** RTEMS pulls in a portion of FreeBSD to provide features that the OS does not currently have. This includes networking, NVMe, OpenMP, USB, SD/MMC, and graphics-related functionality such as framebuffers. This code is largely unmodified from the original FreeBSD code, with the RTEMS developers only providing wrappers for certain core functionality. As a result, consultants did not review the code in depth since FreeBSD is a popular and well known open source project that is subject to frequent review from the open source community.
- **LwIP:** This networking library is slated to be used with the next major release of RTEMS, v6.0. Consultants did not review the source code in depth since it is currently not used by RTEMS v5.3.
- **Hardware Support:** Some BSP boilerplate code was briefly scrutinized, but due to the number of board types supported by RTEMS, the majority of board support code was not reviewed. Consultants focused primarily on remote attacks.
- **Drivers:** Several drivers are sourced from either FreeBSD or Linux and incorporated without modification. These components were not reviewed in depth since they are also subject to review from the larger open source community.

Key Findings

The assessment uncovered a set of common operating system vulnerabilities. The most notable findings were:

Stack Corruption in BOOTP/DHCP: Remote code execution could be attained through the BOOTP/DHCP process. This does not require an attacker to possess valid credentials, only network connectivity with the device. This is discussed further in finding "Stack Corruption Processing Malformed BOOTP/DHCP Responses".

XDR Serialization Buffer Overflow: Specially crafted RPC messages could overflow length checks, resulting in system instability or allowing the attacker to exfiltrate data through out-of-bounds reads. This is documented in finding "XDR Serialization Vulnerable to Buffer Overflows".

Default Root Account Used: RTEMS comes preconfigured to use a weak password with the *root* account. Default credentials provide an attacker with a trivial way to enter and compromise the system. This is further detailed in finding "Default Root Account Created With No Password".

Strategic Recommendations

Outdated Software: A common theme across this project is its reliance on outdated third-party software. Numerous CVEs were reported against the various components used by RTEMS, such as XDR parsing, DHCP, libnetworking, to name a few. In practice, not all of these components are enabled or used in a default build, which should limit exposure to unpatched vulnerabilities. However, the fact remains that the RTEMS project contains outdated dependencies in its repository and provides documentation on how to use it; *outdated or vulnerable components weaken the overall security of RTEMS*. NCC Group recommends caution when using services and features listed in Out of Date Software. If these features are required, their functionality should be sourced from an up-to-date project that patches vulnerable code in a timely manner. Consultants also suggest moving all components that are vulnerable or not up to date into a separate repository. This would increase the work required to include insecure open source projects as part of the RTEMS

build, thus removing any avenue for users to accidentally include vulnerable code through an oversight in BSP configuration.

Encrypted Communication: Adding support for TLS would eliminate the ability for an attacker to inject or manipulate data during transit. A widely used library for this would be MbedTLS[1] or BearSSL[2]. Likewise, replace the FTP daemon with one that supports SFTP. BOOTP code should be removed entirely as it has been largely replaced by DHCP.

Avoid libnetworking, Update libbsd: Both libnetworking and the libbsd networking stack are no longer being updated, with libbsd (based on FreeBSD 12.1) being marked End of Life in November of 2019. Both stacks have a number of CVEs listed against them, and patches may not be backported in a timely manner, if at all. However, the libbsd stack is preferred by the community and NCC Group suggests that users of this network stack engage with the RTEMS community to request and obtain security patches as they arrive. RTEMS developers are encouraged to backport fixes, or port the latest version of FreeBSD (v13.3 as of writing) into the project.

1. https://github.com/Mbed-TLS/mbedtls
2. https://bearssl.org/

3 Adversarial Model

RTEMS is a highly configurable open source project, where users can choose to include certain features from the project as part of their product. These features include a networking stack, multi-threading, telnet and shell access, FTP service, file system support, to name a few. It is clear that RTEMS developers invested considerable effort in making this project easy to configure and develop on, by leaning on existing open source projects to provide the functionality of these features.

Given that RTEMS can be configured to run on a wide array of processors, the adversarial model would change depending on type of device (medical device or satellite in orbit), the security features of the hardware, and the enabled services. RTEMS is developed in a way that maximizes portability and ease of use, leaving security as an exercise for developers incorporating this OS into their project. Developers are seldom security experts, which could leave their product vulnerable to attackers.

Access Method

Remote Adversary: These are agents situated in a position that would allow communication with the RTEMS-enabled device, such as on a network or via another compromised device. These actors are motivated to attack any open services that RTEMS exposes, such as Telnet, FTP, or DHCP. Since these services run within the same memory space as the kernel and applications, any vulnerability therein would cause a denial of service, or something critical such as remote code execution. Some of these services require credentials to be entered, but have otherwise weak security and lack user and process segregation. Attackers could use the lax security of RTEMS as a jumping point to other devices on the network. Addressing these threats involves the upgrade or removal of insecure services, which is a large development undertaking that may also be hindered by incompatible licensing, such as in the case of Mongoose.

Local Adversary: A local adversary can interact with the device software through the existing interfaces such as USB, WiFi, Bluetooth, etc. For instance, they could be exploiting a vulnerability in RTEMS software already running on the device and compromise the device without opening the device and physically tampering with the internals. Unlocked UART shells and debugging tools are sometimes used by attackers to get a foot hold on the system they are trying to attack.

Physical Adversary: A physical adversary is someone with direct physical access to RTEMS-enabled devices. They could read unencrypted storage, extract firmware, perform side-channel attacks, attempt to compromise security using fault injection, modify firmware using JTAG/SWD. An interposer on any of the hardware busses could also allow attackers to intercept and replace data as it is being transmitted; effectively initiating a man-in-the-middle attack.
Cloning devices and the installation of hardware and software Trojans are of particular concern, especially on safety-critical devices.

Technical Capability

RTEMS-powered devices may be attacked by entities with different levels of expertise.

Opportunistic Adversary: These are commonly called script kiddies, they use off-the-self tools with little or no customization. Due to the nature of RTEMS-powered devices, it is more likely that they would attack telecommunication of medical devices that are more easily accessible.

Resource-limited Adversary: These actors have limited budget and skills, can only invest limited time and money to compromise a target. These adversaries are likely to attack industrial control systems as well as the other types of devices mentioned above.

Advanced Persistent Threat Adversary: Well funded organizations, sometimes backed by nation states or sponsored by terrorist organizations, they have the money and time for long-term operations. These adversaries are likely to attack all the types of devices mentioned above, as well as military and aerospace devices that either require special equipment (satellite dishes to communicate with satellites) or special knowledge and capabilities (obtaining a military devices and reverse engineer it).

Level of Trust

These describe whether the attacker is inside the trusted boundary.

Internal Adversary: These actors are part of the RTEMS development team, or part of an organization that uses RTEMS. Attacks are limited to changes to the RTEMS source code or build system where a vulnerability could be injected as part of a legitimate software design cycle. The RTEMS team has put great effort into documenting their project, but a potential internal adversary could provide guidance within the documentation that results in a weaker security posture.

Former Insider: These adversaries are someone with lots of knowledge and potentially recent copies of the data that can guide a successful attack. They could maintain friendly relationships with current insiders, in order to siphon information.

Trusted Third-Party Adversary: An individual, such as a contractor, given some level of trust to perform software development or maintenance that abuses that trust. They may have access to source code and build systems, potentially inserting malicious updates or removing security mitigations that compromise the device security. Nation-sponsored individuals may be inserted into organizations such as NASA, in order to steal intellectual property, or corrupt equipment such as RTEMS-powered satellites.

Supply Chain Position

Supply Chain Adversary: These are adversaries which reside within a manufacturing environment, distribution chain, or intermediary suppliers of third-party products that RTEMS uses. These attacks are present when a malicious actor within the manufacturing environment is able to inject a vulnerability into hardware. This could be a vulnerable component, such as an insecure boot ROM, but it also could be the original developer of a third-party library. In recent memory, the NPM package manager[3] was found to be hosting a number of third-party libraries which were found to be vulnerable, largely because these libraries were not properly vetted. RTEMS does not use a package manager, but it is beholden to a number of third-party libraries for certain features, most notably the networking stack and services.

Update Channel Adversaries: These are adversaries positioned in a delivery path that allows them to intercept and modify software updates. The level of complexity required for a successful attack varies, with aerospace and military equipment being most difficult due to the use of special (and expensive) communication equipment, not easily available.

3. https://snyk.io/blog/npm-security-preventing-supply-chain-attacks/

4 Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors.

Title	Status	ID	Risk
Stack Corruption Processing Malformed BOOTP/ DHCP Responses	New	74P	High
Default Root Account Created With No Password	New	XDY	High
XDR Serialization Vulnerable to Buffer Overflows	New	3TV	High
Unsupported Reserved TCP Flags Overlap With Firewall Flags	New	AY2	Medium
`tarfs` and `tar` Code Is Vulnerable to Path Traversal Attacks	New	DTW	Medium
Stack Sanity Checker Uses Known Static Pattern	New	GMA	Low
Problematic Login Mechanism for Telnet	New	66M	Low
`ftpd` Code May Become a Security Issue	New	H72	Low
`ftp` Susceptible to Path Traversal Attacks	New	J49	Low
Shell Login Mechanism Susceptible to Timing Attacks	New	TMF	Low
TCP Ports Are Assigned Sequentially	New	NXP	Low
Out of Date Software Used by RTEMS	New	BGW	Low
`pppd` Vulnerable to Integer Overflow When Parsing Options	New	TCB	Low
Deprecated IKEv1 Still In Use	New	RY3	Low
Allocations of Zero Size May Hang The System	New	AUK	Low
Undefined Behaviour in BSP Firmware	New	GP7	Info
Issues Could Be Found With the Use of Static Code Analysis Tools	New	QP6	Info
Secure Coding Practices Are Not Followed Universally	New	JYU	Info

5 Finding Details

High

Stack Corruption Processing Malformed BOOTP/DHCP Responses

Overall Risk	High	**Finding ID**	NCC-E000848-74P
Impact	High	**Component**	libnetworking
Exploitability	Medium	**Category**	Data Validation
		Status	New

Impact

Exploitation of this issue would provide an attacker with remote code execution within the RTEMS kernel during the BOOTP/DHCP process. Providing an attacker is positioned on a network to respond to BOOTP/DHCP requests they can completely compromise a system running RTEMS. Beyond having access to a network with RTEMS available, exploitation requires no additional authentication or authorization.

Description

The current RTEMS networking stack supports both BOOTP and DHCP processing to allow an RTEMS deployment to dynamically configure network configuration. Relevant responses are received by the BOOTP and DHCP processing implementations with the provided domain host names being added to `/etc/hosts` by root filesystem processing code used to manage known hosts. The fully qualified name for the host is constructed in a stack buffer; however, the name is constructed using `snprintf()` without taking account of the size of the first component of the name. As such, a malicious response could be provided that would overflow this buffer, allowing stack data to be overwritten, potentially allowing arbitrary code execution. Exploitation of this issue would provide complete compromise of an RTEMS deployment.

Although the detailed implementation is different, both the BOOTP and DHCP code function in the same way: use BSD-derived code to make a BOOTP/DHCP request, and then custom code to process the response. The requests are implemented in:

rtems\kernel\rtems\cpukit\libnetworking\nfs\bootp_subr.c:943
rtems\kernel\rtems\cpukit\libnetworking\rtems\rtems_dhcp.c:880

After initial processing, if the host specified is to be added to the system's hosts file, the function `rtems_rootfs_append_host_rec()` is called:

```
/*
 * Write hosts record.
 */

int
rtems_rootfs_append_host_rec (in_addr_t cip,
                              const char    *cname,
                              const char    *dname)
{
  char           buf[128];
  char           *bufp = buf;
  const char     *bufl[1];
  struct in_addr ip;

  ip.s_addr = cip;
```

```
  if (cname && strlen (cname))
  {
    char addrbuf[INET_ADDRSTRLEN];

    snprintf (bufp, sizeof (buf), "%s\t\t%s", inet_ntoa_r (ip, addrbuf), cname);
    bufp += strlen (buf);

    if (dname && strlen (dname))
    {
      snprintf (bufp, sizeof (buf), "\t\t%s.%s", cname, dname);
      bufp += strlen (buf);
    }

    strcat (buf, "\n");

    bufl[0] = buf;

    if (rtems_rootfs_file_append ("/etc/hosts", MKFILE_MODE, 1, bufl) < 0)
      return -1;
  }
  else
  {
    printf ("rootfs hosts rec append, no cname supplied\n");
    return -1;
  }

  return 0;
}
```

Figure 1: cpukit/libnetworking/rtems/mkrootfs.c

As can be seen, if both domain name and host name are provided they are written to the target stack buffer using `snprintf()`. However, the size of the stack buffer is used for both calls - if a domain name is specified (i.e. the string pointer is not NULL and does not point to NULL), the size of the buffer passed for the second call to `snprintf()` can never be correct. Some of the buffer will have been used writing the host entry. Host name entries, for example, are limited to a maximum of 255 characters in length - this can easily be used to exhaust the provided stack buffer, with the domain name entry then being written from the final byte in the buffer for an additional 128 bytes (including NULL terminator).

Given the formatting used when rendering the domain name entry and the limited amount of memory available to attempt an exploit, exploitation of this issue would be non-trivial.

Invalid Characters & Minimum Length Checks

Hostnames and Fully Qualified Domain Names (FQDN) have a limited set of valid characters, comprised of alphanumeric characters (A-Z, 0-9), a hyphen, and a period. Special characters, such as a letter with an umlaut, may be transcoded into a subset of ASCII character so that it is valid[4]. Both hostnames and FQDN must have a minimum length of 1[5].

Checks validating the hostname and domain name are lacking from the DHCP/BOOTP code, allowing an attacker to craft DHCP messages that contain invalid characters. The following

4. https://en.wikipedia.org/wiki/Punycode
5. https://www.rfc-editor.org/rfc/rfc1123

code checks that the hostname is within the 255 character limit, but does not check for invalid characters and a length less than 1.

```
      case 12:
        /* Host name */
        if (len >= MAXHOSTNAMELEN) {
         printf ("dhcpc: hostname >= %d bytes\n", MAXHOSTNAMELEN);
         len = MAXHOSTNAMELEN-1;
        }
        if (sethostname (p, len) < 0) {
         printf ("dhcpc: can't set host name");
        }

        // ... snip.

        break;
```

Figure 2: cpukit/libnetworking/rtems/rtems_dhcp.c

Likewise, domain names only exhibit a check for minimum length by verifying that the first character is not NULL. Checks for maximum length and invalid characters are absent.

```
      case 15:
        /* Domain name */
        if (p[0])
        {
         rtems_bsdnet_domain_name = strdup (p);
        }
        break;
```

Figure 3: cpukit/libnetworking/rtems/rtems_dhcp.c

No checks are present in `rtems_rootfs_append_host_rec()` either to verify the data is correct, and data is appended to the hosts file verbatim.

Recommendation

The `rtems_rootfs_append_host_rec()` should either correctly determine the space available in the stack buffer used to construct the host entry, or should dynamically construct the required buffer. The latter would be preferable, since the buffer currently used is insufficient to construct valid entries with host names longer than approximately 124 bytes but shorter than 256 bytes. Moreover, check that the host and domain names only contain valid characters, and that length requirements are enforced (as per RFC-1123 and RFC-952).

Location

- rtems\cpukit\libnetworking\rtems\mkrootfs.c:169

High

Default Root Account Created With No Password

Overall Risk	High	**Finding ID**	NCC-E000848-XDY
Impact	High	**Component**	system
Exploitability	High	**Category**	Configuration
		Status	New

Impact

The default RTEMS user database contains weak credentials for the *root* account. If these credentials are not changed as part of the deployment process, they may provide an attacker with a trivial way to try to compromise any authenticated services or interfaces exposed by an RTEMS deployment.

Description

The core RTEMS platform includes libc-specific support, including implementations for required functions such as `getpwnam()` for processing the local user password database. To support this functionality, the supporting code creates a hardcoded default database containing only a root account with no password set. Weak default credentials are not guaranteed to be changed once deployed, and provide an attacker with a quick and easy avenue of attack for any RTEMS service or feature which they can use these credentials to authenticate to. In particular, root access allows complete compromise of an entire deployment.

```
/**
 *  Initialize useable but dummy databases
 */
static void pwdgrp_init(void)
{
  /*
   * Do the best to create this directory.
   */
  mkdir("/etc", S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH | S_IXOTH);

  /*
   *  Initialize /etc/passwd
   */
  init_file("/etc/passwd", "root::0:0::::\n");

  /*
   *  Initialize /etc/group
   */
  init_file("/etc/group", "root::0:\n");
}
```

Removing these default credentials means modifying the codebase for a given project using RTEMS, or updating the user database after deployment.

Recommendation

The creation of a root account should be performed in a configurable and secure manner as part of the deployment process. The absence of weak default credentials means that there is no opportunity to use them or leave them unchanged.

Location

- rtems\cpukit\libcsupport\src\pwdgrp.c:60

High

XDR Serialization Vulnerable to Buffer Overflows

Overall Risk	High	**Finding ID**	NCC-E000848-3TV
Impact	High	**Component**	rpc
Exploitability	High	**Category**	Data Validation
		Status	New

Impact

An attacker can craft an RPC message that overflows length checks, resulting in system instability or allowing the attacker to exfiltrate data through out-of-bounds reads.

Description

The SUNRPC library uses eXternal Data Representation[6] (XDR) as the data format for RPC commands. XDR has a well defined format, consisting of well defined types and structure. The RPC code uses unsigned integers, of type `u_int`, to track lengths of fields and structures. Bounds checks are common in the RPC code, however in a few instances, arithmetic on or using these length values is performed prior to bounds checks. Below, we list a couple of locations where improper handling of length values leads to a vulnerability.

Underflow in `xdrmem_getbytes()` and `xdrmem_putbytes()`

Untrusted content is received in External Data Representation (XDR) Serialization code. Note that member `x_handy` is of unsigned type.

```
typedef struct __rpc_xdr {
...
  char *    x_public;  /* users' data */
  void *    x_private;  /* pointer to private data */
  char *    x_base;    /* private used for position info */
  u_int     x_handy;  /* extra private word */
} XDR;

/* Block for the RPC reply to an outstanding
 * transaction.
 * The caller is woken by the RPC daemon either
 * upon reception of the reply or on timeout. */
enum clnt_stat rpcUdpRcv(RpcUdpXact xact)
{
    int          refresh;
    XDR      reply_xdrs;
    struct rpc_msg    reply_msg;
    rtems_status_code  status;
    rtems_event_set    gotEvents;
    refresh = 0;

    do {
        /* block for the reply */
        status = rtems_event_receive(RTEMS_RPC_EVENT,
           RTEMS_WAIT | RTEMS_EVENT_ANY,
           RTEMS_NO_TIMEOUT, &gotEvents);
        ASSERT( status == RTEMS_SUCCESSFUL );
```

6. https://en.wikipedia.org/wiki/External_Data_Representation

```
        if (xact->status.re_status) {
#ifdef MBUF_RX
            /* add paranoia */
            ASSERT( !xact->ibuf );
#endif
            return xact->status.re_status;
        }

#ifdef MBUF_RX
        xdrmbuf_create(&reply_xdrs, xact->ibuf, XDR_DECODE);
#else
        xdrmem_create(&reply_xdrs, xact->ibuf->buf, xact->ibufsize, XDR_DECODE);
#endif
....
```

Figure 4: rtems/cpukit/libfs/src/nfsclient/src/rpcio.c

The `xdrmem_create()` function saves the untrusted data for later use:

```
void xdrmem_create( XDR *xdrs, char * addr, u_int size, enum xdr_op op )
{
    xdrs->x_op = op;
    xdrs->x_ops = ((uintptr_t)addr & (sizeof(int32_t) - 1))
        ? &xdrmem_ops_unaligned : &xdrmem_ops_aligned;
    xdrs->x_private = xdrs->x_base = addr;
    xdrs->x_handy = size;
}
```

Figure 5: rtems/cpukit/librpc/src/xdr/xdr_mem.c

Later, the code behaves correctly by returning a NULL pointer if `xdrs->x_handy` is less than `len`:

```
static int32_t* xdrmem_inline_aligned( XDR *xdrs, u_int len )
{
    int32_t *buf = 0;
    if (xdrs->x_handy >= len) {
        xdrs->x_handy -= len;
        buf = (int32_t *)xdrs->x_private;
        xdrs->x_private += len;
    }
    return (buf);
}
```

Figure 6: rtems/cpukit/librpc/src/xdr/xdr_mem.c

However, two functions, `xdrmem_getbytes()` and `xdrmem_putbytes()` incorrectly handle the input, in case more then available bytes are required:

```
static bool_t xdrmem_getbytes(XDR *xdrs, caddr_t addr, u_int len)
{
    if ((xdrs->x_handy -= len) < 0)
        return (FALSE);
    memcpy(addr, xdrs->x_private, len);
    xdrs->x_private += len;
    return (TRUE);
}

static bool_t xdrmem_putbytes(XDR *xdrs, const char *addr, u_int len)
```


```
{
    if ((xdrs->x_handy -= len) < 0)
        return (FALSE);
    memcpy(xdrs->x_private, addr, len);
    xdrs->x_private += len;
    return (TRUE);
}
```

Figure 7: rtems/cpukit/librpc/src/xdr/xdr_mem.c

In the code above, `xdrs->x_handy -= len` may cause an underflow, since `xdrs->x_handy` is a unsigned integer. As a result, the calculated value in `xdrs->x_handy` could be a large positive value, which means the `return (FALSE);` may never execute. On the following line a buffer overflow could occur when `memcpy()` operation will run.

This vulnerability is tracked via CVE-2003-0028[7] and affects SUNRPC and all its derivations.

Overflow in `xdr_array()`

Similar to the previous snippet, the following code highlights insufficient checks to prevent an overflow when `nodesize` is calculated. The `elsize` parameter is never checked for size prior to multiplication. The result of a maliciously constructed XDR message could cause the system to use up a large portion of memory, impacting other services running on the device, leading to a possible denial of service.

```
bool_t
xdr_array(
  register XDR *xdrs,
  caddr_t *addrp,    /* array pointer */
  u_int *sizep,    /* number of elements */
  u_int maxsize,    /* max numberof elements */
  u_int elsize,    /* size in bytes of each element */
  xdrproc_t elproc)  /* xdr routine to handle each element */
{
  // ... snip.
  register u_int c;  /* the actual element count */
  register u_int nodesize;
  // ... snip.

  c = *sizep;
  if ((c > maxsize) && (xdrs->x_op != XDR_FREE)) {
    return (FALSE);
  }
  nodesize = c * elsize;

        // ... snip.

      *addrp = target = mem_alloc(nodesize);
      if (target == NULL) {
        (void) fprintf(stderr,
         "xdr_array: out of memory\n");
        return (FALSE);
      }
```

Figure 8: cpukit/librpc/src/xdr/xdr_array.c

7. https://www.cvedetails.com/cve/CVE-2003-0028/

While consultants did not find that this function was used in a way that would compromise the system, the possibility remains that the vulnerability could be triggered by future code modifications. This vulnerability is tracked by CVE-2002-0391[8] and has been found in products that use XDR code from SUNRPC or its derivations.

Recommendation

Overflow conditions on unsigned integers should be checked prior to any arithmetic. Compiler warnings should be enabled using `-Wall` to flag these types of errors during development. This type of check has been encountered in many functions within the XDR parsing code, and NCC Group recommends scrutinizing the code base for similar conditionals.

Update `xdrmem_getbytes` and `xdrmem_putbytes` to behave like `xdrmem_inline_aligned`:

```
From:
    if ((xdrs->x_handy -= len) < 0)

To:
    if (xdrs->x_handy >= len) {
        xdrs->x_handy -= len;
```

Update `xdr_array` to validate all input bounds:

```
From:
    nodesize = c * elsize;

To:
    nodesize = c * elsize;
    if ((c != 0) && (nodesize / c != elsize)) {
        return (FALSE);
    }
```

8. https://www.cvedetails.com/cve/CVE-2002-0391/

Medium

Unsupported Reserved TCP Flags Overlap With Firewall Flags

Overall Risk	Medium	**Finding ID**	NCC-E000848-AY2
Impact	High	**Component**	ipfw
Exploitability	Medium	**Category**	Other
		Status	New

Impact

An attacker could craft a TCP packet with the ECN Echo flag set, in order to bypass firewall filtering.

Description

The firewall filtering mechanism uses the flag `IP_FW_TCPF_ESTAB` to track whether a packet is from an established connection. This flag overlaps with the ECN Echo flag as defined by RFC-2481[9]. The networking code predates the RFC and thus, does not handle the newly added flags.

```
#define IP_FW_TCPF_ESTAB 0x40
```

Figure 9: cpukit/libnetworking/netinet/ip_fw.h

The RFC defines ECN as bit 6, which amounts to `0x40`, same as the code above.

```
Bits 6 and 7 in the IPv4 TOS octet are designated as the ECN field.
Bit 6 is designated as the ECT bit, and bit 7 is designated as the CE
bit.
```

Figure 10: RFC-2481, Section 5.

The `tcpflg_match()` function below collapses the firewall flags and tcp flags into one single bitmap, where the ECN flag and `IP_FW_TCPF_ESTAB` overlap. The flag is later checked against the firewall tcp flags, and the packet is allowed through.

```
static int
tcpflg_match(struct tcphdr *tcp, struct ip_fw *f)
{
  u_char    flg_set, flg_clr;

  if ((f->fw_tcpf & IP_FW_TCPF_ESTAB) &&
      (tcp->th_flags & (IP_FW_TCPF_RST | IP_FW_TCPF_ACK)))
    return 1;

  flg_set = tcp->th_flags & f->fw_tcpf;
  flg_clr = tcp->th_flags & f->fw_tcpnf;

  if (flg_set != f->fw_tcpf)
    return 0;
  if (flg_clr)
```

9. https://datatracker.ietf.org/doc/html/rfc2481

```
    return 0;

  return 1;
}
```

Figure 11: cpukit/libnetworking/netinet/ip_fw.c

Consultants have noticed that the `i386-rtems5/include/tcp.h` header contains the new ECN flags, however the firewall code has not been updated to use said flags.

```
#define TH_FIN 0x01
#define TH_SYN 0x02
#define TH_RST 0x04
#define TH_PUSH 0x08
#define TH_ACK 0x10
#define TH_URG 0x20
#define TH_ECE 0x40
#define TH_CWR 0x80
```

Figure 12: include/netinet/tcp.h

This vulnerability was originally reported against FreeBSD 4.2 and prior, and is tracked via CVE-2001-0183[10].

Recommendation

A fix[11] is available for the firewall rules in FreeBSD 4.2 and later. Consider porting the patch into the RTEMS firewall code.

10. https://vuldb.com/?id.16562
11. https://github.com/freebsd/freebsd-src/commit/c0ec10a75a5e43d23af275d2977093bc397accc6?diff=unified

Medium

`tarfs` and `tar` Code Is Vulnerable to Path Traversal Attacks

Overall Risk	Medium	**Finding ID**	NCC-E000848-DTW
Impact	Medium	**Component**	tar
Exploitability	High	**Category**	Access Controls
		Status	New

Impact

RTEMS software that processes untrusted data in tar format, is vulnerable to path traversal, which can compromise the device running RTEMS.

Description

TAR (**T**ape **AR**chive) is a popular format[12] for data that is supported by RTEMS. In particular, modern software reads and writes using the updated UStar TAR format (Unix Standard TAR). Data in this format could be used to deliver software updates, or simply archive data to be stored or sent remotely. It consists of a header that is 512 bytes long that contains metadata about the file (file name, size, attributes), followed by the content of the file padded to the next 512 bytes.

`Untar_ProcessHeader()` (not shown here) is used to parse the header file and populate the following structure:

```
typedef struct {
  char *file_path;
  char *file_name;
  char link_name[UNTAR_FILE_NAME_SIZE];
  unsigned long mode;
  unsigned long file_size;
  unsigned long nblocks;
  unsigned char linkflag;
  const rtems_printer *printer;
} Untar_HeaderContext;
```

The aforementioned function is called from the following functions:

- `Untar_FromFile_Print()` - reads tar from file then process
- `Untar_FromMemory_Print()` - whole content is in memory, just process
- `Untar_FromChunk_Print()` - process the content from memory chunk
- `rtems_tarfs_load()` - process a tarfs image

None of these functions, nor in the `Untar_ProcessHeader()` function, handle relative file paths such as "../../../evil_file.sh". For instance, a tar archive that contains a software update could be used to leave the current directory and write files at some sensitive location. This is particularly dangerous on RTEMS as there is no separation between kernel and user space. Therefore, sensitive locations cannot be restricted.

12. https://en.wikipedia.org/wiki/Tar_(computing)

Recommendation

Ensure the file name (first 100 bytes in the header) cannot be used to escape the current folder where the operation takes place. Sequences of characters "../" should not be allowed as part of the file name. Only allow a subset of characters as part of the filename, e.g. alphanumeric and '_'. [13]

Additionally, in `Untar_ProcessHeader()`, define the following structure and cast the header[14] data so that offsets are no longer manually calculated.

```
struct posix_header
{        /* byte offset */
  char name[100];    /*   0 */
  char mode[8];      /* 100 */
  char uid[8];      /* 108 */
  char gid[8];      /* 116 */
  char size[12];    /* 124 */
  char mtime[12];    /* 136 */
  char chksum[8];    /* 148 */
  char typeflag;    /* 156 */
  char linkname[100];    /* 157 */
  char magic[6];    /* 257 */
  char version[2];    /* 263 */
  char uname[32];    /* 265 */
  char gname[32];    /* 297 */
  char devmajor[8];    /* 329 */
  char devminor[8];    /* 337 */
  char prefix[155];    /* 345 */
        /* 500 */
};

// Currently the "linkname" above is obtained as follows.
// This approach is error prone
strlcpy(ctx->link_name, &bufr[157], sizeof(ctx->link_name));
```

Location

- `rtems/cpukit/libfs/src/imfs/imfs_load_tar.c`
- `rtems/cpukit/libmisc/untar/untar.c`

13. https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html]
14. https://github.com/Distrotech/tar/blob/distrotech-tar/src/tar.h

Low

Stack Sanity Checker Uses Known Static Pattern

Overall Risk	Low	**Finding ID**	NCC-E000848-GMA
Impact	Low	**Component**	system
Exploitability	Undetermined	**Category**	Security Improvement Opportunity
		Status	New

Impact

Stack guard does not achieve the goal of detecting potentially malicious overflows as it can easily be bypassed by a motivated attacker.

Description

Stack guard protects against stack overflows by first setting a value unknown to the applications (cookie), just before the stack boundary. If at any time during the device operation the cookie value has changed, a stack overflow has occurred and actions such as device reset can be enforced. For the stack protection to work properly, the cookie must be random and difficult to guess. Otherwise, if that value is known, an attacker can craft a buffer overflow that maintains the value in the cookie.

In RTEMS, stack sanity checker uses a static pattern for the cookie, trivial to craft by an attacker:

```
#define CPU_STACK_CHECK_PATTERN_INITIALIZER \
  { \
    0xFEEDF00D, 0x0BAD0D06, /* FEED FOOD to  BAD DOG */ \
    0xDEADF00D, 0x600D0D06  /* DEAD FOOD but GOOD DOG */ \
  }
```

While this may be acceptable for development to find stack overflows, it's not enough to deter an attacker on production devices.

Recommendation

Rather than using the same static pattern on all devices, the RTEMS software should ideally take advantage of a future BSP function that returns a stack cookie. The function would ideally provide a true random value, or if random entropy is not available on the processor, use a value derived on some device-unique identifier, not known to the attacker. Alternatively, entropy can be injected to and stored on the device at manufacturing time by the OEM vendor. Entropy should ideally be periodically updated. Saved entropy is sensitive and must be stored securely.

Location

`cpukit/libmisc/stackchk/check.c`

Low

Problematic Login Mechanism for Telnet

Overall Risk	Low	**Finding ID**	NCC-E000848-66M
Impact	Medium	**Component**	telnetd
Exploitability	High	**Category**	Authentication
		Status	New

Impact

An attacker with the knowledge of the default password could gain access to a system with telnet enabled, access data, or make changes to the system that would allow for further compromise. An attacker may also guess or compute the password using a brute-force attack on the cryptographic algorithm employed, or through detecting differences in execution time during password comparison.

Description

Default Credentials

Default credentials were found in the telnet daemon source code, along with a function to check user entered credentials against the password. As default credentials are freely available online through product/service documentation and in the RTEMS source repository, they would be one of the passwords that an attacker would try as part of an attack.

```
/* #undef TELNETD_DEFAULT_PASSWD */
/* Default password: 'rtems' */
#define TELNETD_DEFAULT_PASSWD "tduDcyLX12owo"
```

Figure 13: cpukit/include/rtems/passwd.h

Outdated Cryptographic Algorithm in Use

The default password of 'rtems' is encrypted using DES and a random salt. The `__des_crypt_r` follows the traditional *crypt(3)* implementation, limiting the password to 8 bytes (zero padded) and a 2-byte salt. The DES algorithm has also been found to be cryptographically insecure[15], though attacks are mainly limited to brute-force computations.

```
bool rtems_telnetd_login_check(
  const char *user,
  const char *passphrase
)
{
  char *pw = getenv( "TELNETD_PASSWD");
  char cryptbuf [21];
  char salt [3];

  if (pw == NULL || strlen( pw) == 0) {
    #ifdef TELNETD_DEFAULT_PASSWD
      pw = TELNETD_DEFAULT_PASSWD;
    #else
      return true;
    #endif
  }

  strncpy( salt, pw, 2);
```

15. https://en.wikipedia.org/wiki/Data_Encryption_Standard#Security_and_cryptanalysis

```
  salt [2] = '\0';

  return strcmp(
    __des_crypt_r( passphrase, salt, cryptbuf, sizeof( cryptbuf)),
    pw
  ) == 0;
}
```

Figure 14: cpukit/telnetd/check_passwd.c

Comparison of Password Vulnerable to Timing Attacks

Comparing the result of a cryptographic operation using functions such as `strcmp`, `strncmp`, and `memcmp`, can expose the comparison operation to timing side-channel attacks[16]. These functions return once a character that does not match is encountered, and thus leaking information regarding where the comparison process fails.

Recommendation

An extensive review should be undertaken to identify any pieces of code that utilize default credentials. Once identified, credentials should be removed, if possible, or replaced with a configuration step that allows for the substitution of the default credentials with one supplied by the user.

Telnet sends data in plaintext, allowing an attacker to sniff packets in transit. Ideally, telnet should be disabled and replaced with a secure shell (SSH).

To combat timing side-channel attacks, consider implementing a safe version of `strncmp` and `memcmp` for comparing the results of cryptographic operations. This function should have a constant runtime and should not return early if a comparison fails. See the `CRYPTO_memcmp()`[17] function in OpenSSL.

Consider replacing DES Crypt with Argon2 (or Argon2id), or if FIPS 140-2 compliance is a concern, use PBKDF2 with a work factor of over 600,000[18].

16. https://blog.mi.hdm-stuttgart.de/index.php/2016/06/28/side-channel-attacks/
17. https://github.com/openssl/openssl/blob/OpenSSL_1_1_1-stable/crypto/cryptlib.c#L443
18. https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Low

`ftpd` Code May Become a Security Issue

Overall Risk	Low	**Finding ID**	NCC-E000848-H72
Impact	Medium	**Component**	ftpd
Exploitability	Low	**Category**	Authentication
		Status	New

Impact

RTEMS developers may use the ftp test suites which do not implement authentication as sample code for their ftpd implementation, leaving the device vulnerable to data extraction and potential malware injection.

Description

The *ftp* protocol is considered insecure and it is no longer recommended to be used. However, on inexpensive devices that do not support crypto operations, or on devices that operate on private networks, it may still be acceptable to use.

In the code below the `login` member of the `rtems_ftpd_configuration` structure points to a function that is used to authenticate the user by verifying if the username and password provided are correct:

```
typedef bool (*rtems_shell_login_check_t)(
  const char * /* user */,
  const char * /* passphrase */
);

struct rtems_ftpd_configuration
{
   rtems_task_priority     priority;           /* FTPD task priority  */
   unsigned long           max_hook_filesize;  /* Maximum buffersize  */
                                  /*    for hooks        */
   int                     port;               /* Well-known port     */
   struct rtems_ftpd_hook  *hooks;             /* List of hooks       */
   char const              *root;              /* Root for FTPD or 0 for / */
   int                     tasks_count;        /* Max. connections    */
   int                     idle;               /* Idle timeout in seoconds
                                  or 0 for no (inf) timeout */
   int                     access;             /* 0 - r/w, 1 - read-only,
                                  2 - write-only,
                                  3 - browse-only */
   rtems_shell_login_check_t login;            /* Login check or 0 to ignore
                                  user/passwd. */
   bool                    verbose;            /* Say hello! */
};

/* Configuration table */
static struct rtems_ftpd_configuration *ftpd_config;
```

The RTEMS codebase provides two test suites for the ftp daemon, and that code is likely to be used by device developers using RTEMS as starting point for their implementation. Note that in both cases the structure is stored in a global variable `rtems_ftpd_configuration`

(therefore all members are initialized to zero) and then `login` function pointer is not set. As such, this approach skips user authentication.

```
struct rtems_ftpd_configuration rtems_ftpd_configuration = {
  /* FTPD task priority */
  .priority = 100,
  /* Maximum buffersize for hooks */
  .max_hook_filesize = 0,
  /* Well-known port */
  .port = 21,
  /* List of hooks */
  .hooks = NULL,
  /* Root for FTPD or NULL for "/" */
  .root = NULL,
  /* Max. connections */
  .tasks_count = 4,
  /* Idle timeout in seconds  or 0 for no (infinite) timeout */
  .idle = 5 * 60,
  /* Access: 0 - r/w, 1 - read-only, 2 - write-only, 3 - browse-only */
  .access = 0
};
```

As such, the developers using ftpd with RTEMS are left in an a insecure-by-default state:

```
static void ftpd_daemon(rtems_task_argument args RTEMS_UNUSED)
{
    ...
            if (ftpd_config->login)
              info->auth = false;
            else
              info->auth = true;
```

All commands follow the pattern below, therefore sensitive commands are unprotected:

```
static void command_size(FTPD_SessionInfo_t *info, char const* fname)
{
  struct stat stbuf;
  char buf[FTPD_BUFSIZE];

  if(!info->auth)
  {
    send_reply(info, 550, "Access denied.");
    return;
  }

  ...
}
```

Recommendation

Implement a skeleton login function in the ftpd code that follows strict security guidelines, for instance protection against timing attacks when comparing the username and password. The code that retrieves the password from local storage may have different implementations on each platform, and should contain a `#error Function Not Implemented` line to force the user to implement it before the code can be used.

Location

- `rtems/cpukit/include/rtems/ftpd.h`

- `rtems/cpukit/ftpd/ftpd.c`
- `rtems/cpukit/ftpd/ftpd-init.c`
- `rtems-libbsd/testsuite/ftpd01/test_main.c`
- `rtems-libbsd/testsuite/ftpd02/test_main.c`

Low

`ftp` Susceptible to Path Traversal Attacks

Overall Risk	Low	**Finding ID**	NCC-E000848-J49
Impact	Medium	**Component**	ftpd
Exploitability	Low	**Category**	Access Controls
		Status	New

Impact

An ftp client may be able to access data that normally should not be allowed, or update critical files that compromise the device.

Description

The code supports changing the *ftpd* root directory. This allows all file system operations (creating and deleting folders, uploading and downloading files) to be relative to the specified root directory path. Note however that, as explained in the `chroot` man pages,[19] this is not a security feature that implements a sandboxing mechanism:

> This call changes an ingredient in the pathname resolution process and does nothing else. In particular, it is not intended to be used for any kind of security purpose, neither to fully sandbox a process nor to restrict filesystem system calls.

```
static void session(rtems_task_argument arg)
{
  FTPD_SessionInfo_t  *const info = (FTPD_SessionInfo_t  *)arg;
  bool chroot_made = false;

  while (1)
  {
    rtems_event_set set;

    rtems_event_receive(FTPD_RTEMS_EVENT, RTEMS_EVENT_ANY, RTEMS_NO_TIMEOUT,
      &set);

    chroot_made = chroot_made
      || (rtems_libio_set_private_env() == RTEMS_SUCCESSFUL
        && chroot(ftpd_root) == 0);
...
```

An attacker that was able to obtain the ftp password, say by sniffing the network and capturing the plain text username and password, can use `../../../secret_file` to escape the ftp root folder and:

- store new files and update existing files that could compromise the device
- retrieve sensitive files that should not normally be accessible, for instance configuration files

```
static void command_store(FTPD_SessionInfo_t *info, char const *filename)
{
...
  {
```

19. https://man7.org/linux/man-pages/man2/chroot.2.html

```
    /* Data transfer to regular file or /dev/null. */
    int fd = 0;

    if(!null)
      fd = creat(filename, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
...
    if(info->xfer_mode == TYPE_I)
    {
...
    }
    else if(info->xfer_mode == TYPE_A)
    {
...
    }
...


static void command_retrieve(FTPD_SessionInfo_t  *info, char const *filename)
{
...
  if (0 > (fd = open(filename, O_RDONLY)))
  {
    send_reply(info, 550, "Error opening file.");
    return;
  }

  if (fstat(fd, &stat_buf) == 0 && S_ISDIR(stat_buf.st_mode))
...
  s = data_socket(info);

  if (0 <= s)
  {
    int n = -1;
    if(info->xfer_mode == TYPE_I)
    {
      while ((n = read(fd, buf, FTPD_DATASIZE)) > 0)
      {
        if(send(s, buf, n, 0) != n) break;
        yield();
      }
    }
    else if (info->xfer_mode == TYPE_A)
    {
...
    }
...
```

Recommendation

Ensure the filename does not contain the “..” sequence.

Location

`rtems/cpukit/ftpd/ftpd.c`

Low

Shell Login Mechanism Susceptible to Timing Attacks

Overall Risk	Low	**Finding ID**	NCC-E000848-TMF
Impact	Low	**Component**	shell
Exploitability	Low	**Category**	Timing
		Status	New

Impact

An attacker may be able to guess the shell password by timing the responses from RTEMS-based devices.

Description

The function `rtems_shell_login_check()` is used to validate the provided credentials by:

- retrieving the data from the local password database file using `getpwnam_r()` [20]
- calculating the hash of the provided `passphase` using `crypt_r()` [21]
- validate the two are identical, using `strcmp()` [22]

Note that `strcmp()` returns when the first different character is detected, therefore the function is vulnerable to timing-attacks.[23] An intruder with sufficient time and resources can take advantage of parallel processing to craft hashes using the known `crypt_r()` algorithm to match one, then two, then three and so on bytes of the password database hash. Given enough tries (and the fact that the number of attempts is not limited) an attacker could eventually obtain the secret passphrase.

```
bool rtems_shell_login_check(
  const char *user,
  const char *passphrase
)
{
  char buf[256];
  struct passwd *pw_res;
  struct passwd pw;
  int eno;
  bool ok;

  eno = getpwnam_r(user, &pw, &buf[0], sizeof(buf), &pw_res);

  /* Valid user? */
  if (eno == 0 && strcmp(pw.pw_passwd, "*") != 0) {
    if (strcmp(pw.pw_passwd, "") == 0) {
      ok = true;
    } else if (strcmp(pw.pw_passwd, "x") == 0) {
      /* TODO: /etc/shadow */
      ok = false;
    } else {
```

20. https://www.freebsd.org/cgi/man.cgi?getpwnam_r
21. https://www.freebsd.org/cgi/man.cgi?crypt_r
22. https://www.freebsd.org/cgi/man.cgi?strcmp
23. https://nachtimwald.com/2017/04/02/constant-time-string-comparison-in-c/

```
      struct crypt_data data;
      char *s;

      s = crypt_r(passphrase, pw.pw_passwd, &data);
      ok = strcmp(s, pw.pw_passwd) == 0;
    }
  } else {
    ok = false;
  }
...
}
```

Additionally, the result of `crypt_r()` is not validated, and could be `NULL` if the `cf_default_func()` is changed to return such a value. While this is not an issue currently, users of RTEMS are encouraged to customize the build to their needs and may make assumptions about how return values are handled, thus leading to an undiscovered vulnerability. When `NULL` is passed to `strcmp()`, the behavior is undefined and may cause the system to crash.

Recommendation

Use a constant time comparison function.[24] Validate the value of `s` before use to ensure it is not `NULL` .

24. https://www.openssl.org/docs/man1.1.1/man3/CRYPTO_memcmp.html

Low

TCP Ports Are Assigned Sequentially

Overall Risk	Low	**Finding ID**	NCC-E000848-NXP
Impact	Low	**Component**	libnetworking
Exploitability	Medium	**Category**	Uncategorized
		Status	New

Impact

An attacker could deduce which port numbers the FTP server will open following a PASV command from an authorized client. The attacker would be able to connect to the port before the authorized used does, gaining access to the filesystem.

Description

Upon receiving a PASV command, the FTP server opens a new data socket and listens for a connection from the client. When binding to the new socket, a call into `bind()` is issued, with the port specified as `0`.

```
static void
command_pasv(FTPD_SessionInfo_t *info)
{

    // ... snip

    struct sockaddr_in addr;
    socklen_t addrLen = sizeof(addr);

    addr = info->ctrl_addr;
    addr.sin_port = htons(0);

    if (0 > bind(s, (struct sockaddr *)&addr, addrLen))
```

Figure 15: cpukit/ftpd/ftpd.c

A call into `bind()` results in a syscall that leads into `in_pcbbind()`. The portion of this function responsible for assigning a new unused port number, seen below, does so sequentially.

```
int
in_pcbbind(struct inpcb *inp, struct mbuf *nam)
{
        // ... snip.

  if (lport == 0) {

              // ... snip.

    if (first > last) {
      /*
       * counting down
       */
      count = first - last;

      do {
        if (count-- <= 0)  /* completely used? */
```

```
        return (EADDRNOTAVAIL);
      --*lastport;
      if (*lastport > first || *lastport < last)
       *lastport = first;
      lport = htons(*lastport);
    } while (in_pcblookup(inp->inp_pcbinfo,
       zeroin_addr, 0, inp->inp_laddr, lport, wild));
  } else {
    /*
     * counting up
     */
    count = last - first;

    do {
      if (count-- <= 0)  /* completely used? */
       return (EADDRNOTAVAIL);
      ++*lastport;
      if (*lastport < first || *lastport > last)
       *lastport = first;
      lport = htons(*lastport);
    } while (in_pcblookup(inp->inp_pcbinfo,
       zeroin_addr, 0, inp->inp_laddr, lport, wild));
  }
```

Figure 16: cpukit/libnetworking/netinet/in_pcb.c

Predictable port numbers may affect many other services running on the system. An attacker could determine the next available port used by services. This vulnerability is tracked by CVE-1999-0074[25].

Recommendation

Port numbers should be randomly allocated when calls to `bind()` specify `0` as the port number. Since this involves code that could affect the entire system in other ways, a more localized countermeasure would be to randomly assign port numbers to PASV connections in the `command_pasv()` function.

25. https://nvd.nist.gov/vuln/detail/CVE-1999-0074

Low

Out of Date Software Used by RTEMS

Overall Risk	Low	**Finding ID**	NCC-E000848-BGW
Impact	Low	**Component**	system
Exploitability	Undetermined	**Category**	Patching
		Status	New

Impact

Outdated third-party software contains vulnerabilities that can help an attacker successfully compromise an RTEMS-based device.

Description

Note that the list below is not complete, only a few samples are shown here to show the widespread use of outdated software.

PPP

The `pppd` source is taken from `ftp://cs.anu.edu.au/pub/software/ppp` and the latest used version is 2.3.11 (December 1999). NCC Consultants were unable to connect to the FTP address to determine if updates are available. A clone of the project exists[26], which is actively maintained, and the latest version is 2.4.9.

Mongoose HTTP

The `mghttpd` source dates from 2013 and sources indicate version 3.9. Since then, the Mongoose HTTP server code has been taken over by Cesanta Software[27] which actively maintains the code base. A number of vulnerabilities[28] have been filed against Mongoose, which may impact this specific version.

NFS Client

The NFS client in RTEMS supports NFS v2, standardized in RFC-1094. This version does not support any authentication mechanism or ACL and relies on the permission model of the filesystem for security. Those features are available in NFS v4[29].

SUN RPC

The `librpc` service is based on SUN RPC v4.0, dated from 1989. A number of vulnerabilities have been filed against SUN RPC and its derivations[30]. However, with the exception of finding "XDR Length Check Can Overflow", many CVEs do not apply to the RTEMS codebase due to the age of the RPC code and its configuration.

xz Decompression Software

The code has not been updated from 2011, and is missing important patches.[31]

zlib Compression Library

RTEMS uses version 1.2.5 from Apr 19, 2010 and is affected by at least one known vulnerability. [32]

26. https://github.com/ppp-project
27. https://github.com/cesanta
28. https://www.cvedetails.com/vulnerability-list/vendor_id-16334/product_id-41402/Cesanta-Mongoose.html
29. https://web.mit.edu/rhel-doc/5/RHEL-5-manual/Deployment_Guide-en-US/s1-nfs-security.html
30. https://www.cvedetails.com/google-search-results.php?q=sunrpc&sa=Search
31. https://github.com/torvalds/linux/commit/4f8d7abaa413c34da9d751289849dbfb7c977d05
32. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-37434

Recommendation

While this list contains only a few examples, a thorough investigation must be performed to find all source code that is out of date. A process should be established to track the versions of all third party dependencies and third party code bundled with the project, monitor vulnerabilities reported against these software components, and to update these components periodically and/or when vulnerabilities have been fixed.

Low

`pppd` Vulnerable to Integer Overflow When Parsing Options

Overall Risk	Low	**Finding ID**	NCC-E000848-TCB
Impact	High	**Component**	pppd
Exploitability	Undetermined	**Category**	Patching
		Status	New

Impact

An options file with numerous characters can cause a length variable, tasked with tracking the number of characters read, to overflow. This overflow can cause the system to crash when the length variable is used to index an array.

Description

Parsing options from a file relies on the `getword()` function to read out and parse each word within the file. Reads are performed one character at a time, with special logic dedicated to handle new line characters, comments, and escaped characters. The logic therein keeps track of the length of each word using a length variable `len`. If the character is valid, it is appended to the `word` variable, so long as the variable has enough space.

While the check is necessary, it does not stop the `len` variable from being incremented, allowing it to increment despite `word` being full. The variable will overflow (and become negative) if a sufficiently large option file is provided. A negative `len` would pass the if-statement and index into `word` using a negative number.

```
        int c, len, escape;

        // ... snip.

        len = 0;

        // ... snip.

      /*
       * Store the resulting character for the escape sequence.
       */
      if (len < MAXWORDLEN-1)
    word[len] = value;
      ++len;

        // ... snip.

  /*
   * An ordinary character: store it in the word and get another.
   */
  if (len < MAXWORDLEN-1)
      word[len] = c;
  ++len;
```

Figure 17: cpukit/pppd/options.c

This vulnerability was first disclosed to the maintainers of Paul's PPP project[33] as CVE-2014-3158[34] (fixed in version v2.4.9).

Recommendation

Increment the length of the word buffer within the same scope as character assignment[35]. If possible, upgrade the `pppd` implementation to the latest version, or consider migrating to Paul's PPP project, which is currently maintained.

33. https://github.com/ppp-project/ppp
34. https://nvd.nist.gov/vuln/detail/CVE-2014-3158
35. https://github.com/ppp-project/ppp/commit/7658e8257183f062dc01f87969c140707c7e52cb

Low

Deprecated IKEv1 Still In Use

Overall Risk	Low	**Finding ID**	NCC-E000848-RY3
Impact	Undetermined	**Component**	libbsd - ipsec-tools
Exploitability	Low	**Category**	Cryptography
		Status	New

Impact

IKEv1 in Aggressive Mode sends the hash digest of the preshared key (PSK) used on the system in plain text. Likewise, a man-in-the-middle attack during a Main Mode handshake would also reveal the PSK digest, even if the packets are encrypted. While this does not break the scheme immediately, it allows an adversary with large computation power to attack the hash digest using a bruteforce or dictionary attack. PSKs with low entropy would be at higher risk.

Description

The Internet Key Exchange (IKE) v1 protocol was developed to facilitate secure setup of a VPN connection. IKEv1 is based upon the Internet Security Association and Key Management Protocol (ISAKMP), which provides two modes of operation during the first phase of the handshake:

- **Main Mode** - this mode uses a series of six transmissions between the client and server as part of it's handshake. The first four packets contain Diffie-Hellman key data to establish a session key. The last two data transmissions are encrypted using the session key derived from the previous data exchange.
- **Aggressive Mode** - this mode combines the handshake into a total of three data transmissions. The response from the remote machine includes a hash of the PSK, transmitted in plaintext before the session key is established.

An attacker sniffing **Aggressive Mode** traffic would be able to gather all data needed to calculate the digest, but would not be able to retrieve the preshared key itself. However, since all the data required to compute the digest is within the cleartext message (minus the PSK), the attacker could attempt a bruteforce or dictionary attack[36] on the digest itself as a means to discover the key. It is also possible to attack **Main Mode** and gain access to the PSK hash digest. However, since the data is encrypted with a session key, the attacker would need to place a machine-in-the-middle and sniff data during the DH handshake to read out the PSK digest.

In some cases, a PSK needs to be installed on multiple machines in order for those machines to communicate securely. Shared key material runs the risk of being leaked through a vulnerability on any system that employs it. A leak of this type would compromise the security of all machines that use the key, regardless of which mode the clients use[37].

IKEv1 is Deprecated

As of October 2022, the IKEv1 protocol has been deprecated[38], with IKEv2 being recommended as a replacement. The `ipsec-tools` project contains the `racoon` library, which only implements IKEv1. This leaves RTEMS users with an obsolete library as the main IPSEC

36. https://security.stackexchange.com/questions/76444/what-are-the-practical-risks-of-using-ike-aggressive-mode-with-a-pre-shared-key
37. https://www.usenix.org/conference/usenixsecurity18/presentation/felsch
38. https://www.ietf.org/archive/id/draft-ietf-ipsecme-ikev1-algo-to-historic-07.html

option. IKEv2 requires more computational resources than IKEv1 due to the cryptographic algorithms employed, and may not be the ideal tool for a constrained, embedded system. IKEv2 also deprecates a number of ciphers that are known to be vulnerable.

Recommendation

Due to the IKEv1 protocol being deprecated by the IETF, it is recommended that IKEv2 is used as the default, using an up-to-date library such as racoon2[39]. Alternatives to IKEv2 include Strongswan and Wireguard.

IKEv1 can still be used **only if** absolutely necessary, and **only if** a PSK is not used for authentication, such as RSA-based. PSKs with enough entropy (256 bits) would be improbable to crack with today's hardware. However, PSKs are still vulnerable to being leaked from other devices that use it.

Location

- `rtems-libbsd/ipsec-tools/src/*`

39. https://github.com/zoulasc/racoon2

Low

Allocations of Zero Size May Hang The System

Overall Risk	Low	**Finding ID**	NCC-E000848-AUK
Impact	Low	**Component**	system
Exploitability	Undetermined	**Category**	Denial of Service
		Status	New

Impact

RTEMS provides it's own implementation of `malloc()` that strictly defines zero-length allocations, resulting in a NULL pointer return value. This, however, would cause `rtems_bsdnet_malloc()` to loop indefinitely if the specified size is zero, resulting in a denial of service of the device.

Description

The implementation of `malloc()` clearly defines the result of allocating zero bytes. The highlighted section below shows that zero, or `NULL`, is returned if the `size` parameter is zero.

```
void *malloc(
  size_t  size
)
{
  void        *return_this;

  /*
   * Validate the parameters
   */
  if ( !size )
    return (void *) 0;

  return_this = rtems_heap_allocate_aligned_with_boundary( size, 0, 0 );
  if ( !return_this ) {
    errno = ENOMEM;
    return (void *) 0;
  }

  return return_this;
}
```

The `rtems_bsdnet_malloc()` function, shown below, wraps a call to `malloc()` in an infinite loop. This assumes that if `malloc()` fails for any reason, it should be retried until the memory is allocated. In the case where `size` is zero, the call to `malloc()` will fail return a `NULL` pointer and fail to exit the loop. Instead, it will retry and fail infinitely.

```
void *
rtems_bsdnet_malloc (size_t size, int type, int flags)
{
  void *p;
  int try = 0;

  for (;;) {
    uint32_t nest_count;
```

```
    p = malloc (size);
    if (p || (flags & M_NOWAIT))
      return p;
    nest_count = rtems_bsdnet_semaphore_release_recursive ();
    if (++try >= 30) {
      rtems_bsdnet_malloc_starvation();
      try = 0;
    }
    rtems_task_wake_after (rtems_bsdnet_ticks_per_second);
    rtems_bsdnet_semaphore_obtain_recursive (nest_count);
  }
}
```

It should be noted, this only applies when `flags` does not contain `M_NOWAIT`, which is not always the case. As such, this finding is marked as having a `Low` risk factor due to how this function is used in the overall project.

Recommendation

Remove or adjust the infinite loop in `rtems_bsdnet_malloc()` so that repeated failures are not just logged, but also trickle down to the calling functions. Consider returning the `NULL` value when `rtems_bsdnet_malloc_starvation()` is triggered rather than retrying.

Info

Undefined Behaviour in BSP Firmware

Overall Risk	Informational	**Finding ID**	NCC-E000848-GP7
Impact	None	**Component**	bsp
Exploitability	None	**Category**	Configuration
		Status	New

Impact

Code may be interpreted differently by different compilers, resulting in unexpected behaviour.

Description

At least one instance was found in the BSP firmware where an undefined behaviour causes `size_cfg.upper_wrprot_block` to be set to `0xFFFF`. The value of `rdid` could be controlled by an attacker such as through an implanted interposer. The value of `cap_code` is BCD encoded, and the value of `cap_decoded` is attacker controlled and can be any value between `0` and `99`.

From the C standard:

> The behavior is undefined if the right operand is negative, or greater than or equal to the length in bits of the promoted left operand.

Note that when the value of `qspi_device_size` is calculated as `1 << (cap_decoded + 6)`, the left shift right operand can be up to `99 + 6 = 105` times, which represents an undefined behaviour.

Additionally, note that using signed data types is not recommended with bitwise operators[40]. For instance `cap_code` should be `unsigned int` as bitwise operator are `cap_code >> 4` and `cap_code & 0xf`.

```
ALT_STATUS_CODE alt_qspi_enable(void)
{
...
    uint32_t rdid = 0;

    // Query device capabilities
    // This requires QSPI to be enabled.

    if (status == ALT_E_SUCCESS)
    {
        status = alt_qspi_device_rdid(&rdid);
    }

    if (status == ALT_E_SUCCESS)
    {
        // NOTE: The size code seems to be a form of BCD (binary coded decimal).
        //   The first nibble is the 10's digit and the second nibble is the 1's
        //   digit in the number of bytes.

        // Capacity ID samples:
```

40. https://wiki.sei.cmu.edu/confluence/display/c/INT13-C.+Use+bitwise+operators+only+on+unsigned+operands

```
        //  0x15 :   16 Mb =>   2 MiB => 1 << 21 ; BCD=15
        //  0x16 :   32 Mb =>   4 MiB => 1 << 22 ; BCD=16
        //  0x17 :   64 Mb =>   8 MiB => 1 << 23 ; BCD=17
        //  0x18 :  128 Mb =>  16 MiB => 1 << 24 ; BCD=18
        //  0x19 :  256 Mb =>  32 MiB => 1 << 25 ; BCD=19
        //  0x1a
        //  0x1b
        //  0x1c
        //  0x1d
        //  0x1e
        //  0x1f
        //  0x20 :  512 Mb =>  64 MiB => 1 << 26 ; BCD=20
        //  0x21 : 1024 Mb => 128 MiB => 1 << 27 ; BCD=21

        // NCC Group - cap_code should not be signed type
        int cap_code = ALT_QSPI_STIG_RDID_CAPACITYID_GET(rdid);

        if ( ((cap_code >> 4) > 0x9) || ((cap_code & 0xf) > 0x9))
        {
            // If a non-valid BCD value is detected at the top or bottom nibble, chances
            // are that the chip has a problem.

            dprintf("DEBUG[QSPI]: Invalid CapacityID encountered: 0x%02x.\n", cap_code);
            status = ALT_E_ERROR;
        }
        else
        {
            int cap_decoded = ((cap_code >> 4) * 10) + (cap_code & 0xf);
            // NCC Group - 1 shifted more than 31 times, may result in a 0
            qspi_device_size = 1 << (cap_decoded + 6);
            dprintf("DEBUG[QSPI]: Device size = 0x%" PRIx32 ".\n", qspi_device_size);
        }
    }

    // Configure the device size and address bytes

    if (status == ALT_E_SUCCESS)
    {
        ALT_QSPI_DEV_SIZE_CONFIG_t size_cfg =
        {
            .block_size         = ALT_QSPI_DEVSZ_BYTESPERSUBSECTOR_RESET,  // 0x10  => 2^16 =
            ↳ 64 KiB
            .page_size          = ALT_QSPI_DEVSZ_BYTESPERDEVICEPAGE_RESET, // 0x100 => 256 B
            .addr_size          = ALT_QSPI_DEVSZ_NUMADDRBYTES_RESET,       // 0x2   => 3 bytes
            ↳ or 0x00ffffff mask.
            .lower_wrprot_block = 0,
            // NCC Group - could be up to 0xFFFFFFFF >> 16 = 0xFFFF
            .upper_wrprot_block = (qspi_device_size - 1) >> 16,
            .wrprot_enable      = ALT_QSPI_WRPROT_EN_RESET
        };

        status = alt_qspi_device_size_config_set(&size_cfg);
    }
```

Since `wrprot_enable` is set to `ALT_QSPI_WRPROT_EN_RESET` (which is `0`), the call to `alt_qspi_device_size_config_set()` is not using the value of `upper_wrprot_block` and therefore this finding is reported as Informational.

```
ALT_STATUS_CODE alt_qspi_device_size_config_set(const ALT_QSPI_DEV_SIZE_CONFIG_t * cfg)
{
...
    if (cfg->wrprot_enable)
    {
        alt_write_word(ALT_QSPI_LOWWRPROT_ADDR, cfg->lower_wrprot_block);
        alt_write_word(ALT_QSPI_UPPWRPROT_ADDR, cfg->upper_wrprot_block);
    }
...
```

Recommendation

- The value of `cap_code` must be restricted to acceptable values: 0x15 to 0x19, 0x20, and 0x21, as shown in the comment in the C code above. This eliminates the undefined behaviour.
- Altera BSP firmware must be reviewed, and additional instances of where undefined behaviour may occur must be removed. Simple C-language static analyzers can help with this task, for instance cppcheck.[41]
- Variables `cap_code` and `cap_decoded` must be changed to type `uint32_t`.
- The code does not appear to use the resulting `wrprot_enable`. If code is no longer used, remove it to decrease the code complexity.
- Test code when compiled us undefined behaviour sanitizers.[42]

Location

`rtems/bsps/arm/altera-cyclone-v/contrib/hwlib/src/hwmgr/alt_qspi.c`

41. https://cppcheck.sourceforge.io/
42. https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html

Info

Issues Could Be Found With the Use of Static Code Analysis Tools

Overall Risk	Informational	**Finding ID**	NCC-E000848-QP6
Impact	None	**Component**	system
Exploitability	None	**Category**	Configuration
		Status	New

Impact

The issues presented in this finding were found using static analysis tools. While these do not currently represent vulnerabilities, it shows that RTEMS codebase could benefit from better static code analysis scrutiny.

Description

Dangling Else

The actual code is displayed below. The developer's intention is shown by the line indentation:

```
if (socket_send(viftable[vifi].v_rsvpd, m, &rsvp_src) < 0)
        if (rsvpdebug)
            printf("rsvp_input: Failed to append to socket\n");
    else
        if (rsvpdebug)
            printf("rsvp_input: send packet up\n");
```

Figure 18: rtems\cpukit\libnetworking\netinet\ip_mroute.c

In C, the `else` statement is associated with the closest `if`. In reality the code implements the following behavior:

```
if (socket_send(viftable[vifi].v_rsvpd, m, &rsvp_src) < 0)
        if (rsvpdebug)
            printf("rsvp_input: Failed to append to socket\n");
        else
            if (rsvpdebug)
                printf("rsvp_input: send packet up\n");
```

This issue cannot be exploited, but it shows code quality deficiencies.

Memory Allocations Are Not Validated Before Use

Throughout the codebase the RTEMS code ensures that `realloc()` function is not returning `NULL` before the buffer is used.

```
void *bootp_strdup_realloc(char *dst,const char *src)
{
  size_t len;
  if (dst == NULL) {
    /* first allocation, simply use strdup */
  if (src) dst = strdup(src);
  }
  else {
    /* already allocated, so use realloc/strcpy */
    len = src ? strlen(src) + 1 : 0;
    /* src == NULL tells us to effectively free dst */
```

```
    dst = realloc(dst,len);
    if (dst != NULL) {
        strcpy(dst,src);
    }
  }
  return dst;
}
```

Figure 19: rtems/cpukit/libnetworking/nfs/bootp_subr.c

There are however functions that perform memory reallocations indirectly through `bootp_strdup_realloc` that are not checking the output before use. If the `realloc()` call in `bootp_strdup_realloc()` fails, `rtems_bsdnet_domain_name` will be set to `NULL`. The code below shows that output is used without verification, potentially causing a crash.

```
static void processOptions (unsigned char *optbuf, int optbufSize)
{
...
    switch (code) {
...
    case 15:
      /* Domain name */
      if (p[0]) {
        rtems_bsdnet_domain_name = bootp_strdup_realloc(rtems_bsdnet_domain_name,(char *)p);
        printf("Domain name is %s\n", rtems_bsdnet_domain_name);
      }
...
}
```

Figure 20: rtems/cpukit/libnetworking/nfs/bootp_subr.c

A similar example is found in `callrpc()`:

```
int callrpc( char *host, int prognum, int versnum, int procnum,
    xdrproc_t inproc, char *in, xdrproc_t outproc, char *out )
{
...
    if (crp->oldhost == NULL) {
        crp->oldhost = malloc(MAXHOSTNAMELEN);
        crp->oldhost[0] = 0;
        crp->socket = RPC_ANYSOCK;
}
...
```

Note that there are many other instances similar in nature, however, for brevity only a few are shown in this finding.

Recommendation

Low-hanging fruit issues that can be found using static code analysis tools should be fixed. Ensure that running these tools is integrated in the build process and issues reported are fixed in a timely manner.

Dangling Else

Use brackets to correct the code behaviour:

```
if (socket_send(viftable[vifi].v_rsvpd, m, &rsvp_src) < 0) {
        if (rsvpdebug)
            printf("rsvp_input: Failed to append to socket\n");
    }
```

```
    else {
        if (rsvpdebug)
            printf("rsvp_input: send packet up\n");
    }
```

Memory Allocations Are Not Validated Before Use

Update the code to only use the output of `bootp_strdup_realloc()` if it is not NULL.

Info

Secure Coding Practices Are Not Followed Universally

Overall Risk	Informational	**Finding ID**	NCC-E000848-JYU
Impact	Medium	**Component**	system
Exploitability	Low	**Category**	Security Improvement Opportunity
		Status	New

Impact

Unsafe code makes it more likely for an attacker to exploit an RTEMS-based device.

Description

Secure coding practices are a broad topic that deals with writing code in such a way that mistakes are less likely to crop up, and thus are less likely to be exploited. A number of mistakes shown below could be remedied through secure coding guidelines, static analysis tools, and in-depth reviews of submitted patches. This is not an exhaustive list of these types of issues, but serves to inform the development team that such mistakes can crop up.

Masking Error Codes

A fairly common flaw found in the RTEMS codebase is passing failed return values to calling code. This hides the error that was encountered, either by not returning the code or by reassigning the return code to another value. In the following case, the variable `rc` is used to track return codes from calling functions, but if an error is detected, the error code is not passed to the calling function. Instead, the while loop is exited and a success value (`0`) is returned.

```
int
rtems_rfs_bitmap_map_alloc (rtems_rfs_bitmap_control* control,
                            rtems_rfs_bitmap_bit      seed,
                            bool*                     allocated,
                            rtems_rfs_bitmap_bit*     bit)
{
  rtems_rfs_bitmap_bit upper_seed;
  rtems_rfs_bitmap_bit lower_seed;
  rtems_rfs_bitmap_bit window;     /* may become a parameter */
  int                  rc = 0;

  // ... snip ...
  while ( // ... snip ... ) {
    // ... snip ...
    if (upper_seed < control->size)
    {
      *bit = upper_seed;
      rc = rtems_rfs_search_map_for_clear_bit (control, bit, allocated,
                                               window, 1);
      if ((rc > 0) || *allocated)
        break;
    }

    if (lower_seed >= 0)
    {
```

```
      *bit = lower_seed;
      rc = rtems_rfs_search_map_for_clear_bit (control, bit, allocated,
                                              window, -1);
      if ((rc > 0) || *allocated)
        break;
    }
    // ... snip ...
  }
  // ... snip ...

  return 0;
}
```

Figure 21: rtems/cpukit/libfs/src/rfs/rtems-rfs-bitmaps.c

Integer Overflow/Underflow

Integer overflow or underflow is a common flaw found when calculating a value that is too large to fit into the current data type. In the following example, the number of elements is calculated based on the input `_bits`, which itself could be any number. In the case where `_bits` is zero, the highlighted code will subtract one and underflow the variable value. The resulting calculation would be a very large number, or if `_bits` was set to one, the calculation would result in zero.

```
/**
 * The basic element of a bitmap. A bitmap is manipulated by elements.
 */
typedef uint32_t rtems_rfs_bitmap_element;

/**
 * Return the number of bits for the number of bytes provided.  The search
 * element and the element must have the same number of bits.
 */
#define rtems_rfs_bitmap_element_bits() \
  rtems_rfs_bitmap_numof_bits (sizeof (rtems_rfs_bitmap_element))

/**
 * Return the number of elements for a given number of bits.
 */
#define rtems_rfs_bitmap_elements(_bits) \
  (((_bits) - 1) / rtems_rfs_bitmap_element_bits()) + 1)
```

Figure 22: rtems/cpukit/include/rtems/rfs/rtems-rfs-bitmaps.h

The following code uses the above macro and passes the result to `malloc()`.

```
int
rtems_rfs_bitmap_open (rtems_rfs_bitmap_control* control,
                       rtems_rfs_file_system*    fs,
                       rtems_rfs_buffer_handle*  buffer,
                       size_t                    size,
                       rtems_rfs_buffer_block    block)
{
  size_t elements = rtems_rfs_bitmap_elements (size);

  // ... snip ...

  elements = rtems_rfs_bitmap_elements (elements);
```

```
  control->search_bits = malloc (elements * sizeof (rtems_rfs_bitmap_element));

  // ... snip ...
}
```

Figure 23: rtems/cpukit/libfs/src/rfs/rtems-rfs-bitmaps.c

Normally, passing zero into `malloc()` is considered undefined behaviour[43]. However in this case, the implementation of `malloc()` takes into account a zero size and will return `NULL` to signal an error. While this particular case does not cause a vulnerability, it remains a silent bug that may become vulnerable as code is changed. Functions that wrap memory allocation can have different logic and may not test for exceptional conditions, such as in the case of finding "Allocations of Zero Size May Hang The System".

Use of Uninitialized Variables

Though more prevalent in non-RTEMS code, failing to initialize memory or variables prior use can lead to unexpected results.[44]
In the following code, the variable `ino` is used within a `printf()` call before it is initialized and can cause a leak of the stack memory content (4 bytes). In more serious situations use of uninitialized variables could trigger a system crash, or cause non-deterministic behaviour. NCC Group did not look into these types of issues further as they can be easily found using static code analysis tools.

```
typedef uint32_t rtems_rfs_ino;

static int
rtems_rfs_rtems_file_open (rtems_libio_t* iop,
                           const char*    pathname,
                           int            oflag,
                           mode_t         mode)
{
  rtems_rfs_file_system* fs = rtems_rfs_rtems_pathloc_dev (&iop->pathinfo);
  rtems_rfs_ino          ino;
  rtems_rfs_file_handle* file;
  int                    flags;
  int                    rc;

  flags = 0;

  if (rtems_rfs_rtems_trace (RTEMS_RFS_RTEMS_DEBUG_FILE_OPEN))
    printf("rtems-rfs: file-open: path:%s ino:%" PRId32 " flags:%04i mode:%04" PRIu32 "\n",
           pathname, ino, flags, (uint32_t) mode);

  rtems_rfs_rtems_lock (fs);

  ino = rtems_rfs_rtems_get_iop_ino (iop);

  // ... snip ...
```

Figure 24: rtems/cpukit/libfs/src/rfs/rtems-rfs-rtems-file.c

Copying Memory Without Known Bounds

The code below shows a coding pattern present throughout the codebase. It shows that `ctermid()` copies a fixed-size buffer without knowing the size of the destination buffer. This function is currently only called from the tests, and does not pose a vulnerability. However,

43. https://stackoverflow.com/questions/10684595/behaviour-of-malloc0
44. https://cwe.mitre.org/data/definitions/457.html

using functions that do not know the bounds of the buffer they operate on can lead to buffer overruns.

```
static char *ctermid_name = "/dev/console";

/**
 *  ctermid() - POSIX 1003.1b 4.7.1 - Generate Terminal Pathname
 */
char *ctermid( char *s )
{
  if ( !s )
    return ctermid_name;

  /*
   *  We have no way of knowing the length of the user provided buffer.
   *  It may not be large enough but there is no way to know that. :(
   *  So this is a potential buffer owerrun that we can do nothing about.
   */
  strcpy( s, ctermid_name );
  return s;
}
```

Figure 25: rtems/cpukit/libcsupport/src/ctermid.c

Inconsistent Use of Return Variables

The C language does not have a consistent way of returning multiple types of values; developers need to chose whether to `return` a variable or to pass it back via an output parameter. In some cases, developers try to return a number of different types using `return`. This creates confusion when error codes and data are returned in the same place, leaving humans (rather than machines) to infer and understand all possible return codes that a function can emit.

In the code below, two variables are used to track return values; `sc` and `esc`. The former is used temporarily to check the return value from individual sub-functions, while the latter is used as the final return code. In the highlighted sections below, if the call to `create_logical_disk_name()` fails due to a memory error and returns `NULL`, the value of `sc` is copied into `esc`, and ultimately returned. If the call to `rtems_bdpart_get_disk_data()` returns a successful return code, `esc` will return a success as well. The issue here seems to stem from the author copy-pasting a common idiom (`esc = sc;`) into error handling conditionals.

```
rtems_status_code rtems_bdpart_register(
  const char *disk_name,
  const rtems_bdpart_partition *pt,
  size_t count
)
{
  rtems_status_code sc = RTEMS_SUCCESSFUL;
  rtems_status_code esc = RTEMS_SUCCESSFUL;

  // ... snip ...

  /* Get disk data */
  sc = rtems_bdpart_get_disk_data( disk_name, &fd, &dd, &disk_end);
  if (sc != RTEMS_SUCCESSFUL) {
    return sc;
  }
```

```
  /* Create logical disk name */
  logical_disk_name = create_logical_disk_name(
    disk_name,
    &logical_disk_marker
  );
  if (logical_disk_name == NULL) {
    esc = sc; //// NCC: sc is RTEMS_SUCCESSFUL here.
    goto cleanup;
  }

  // ... snip ...

cleanup:
  // ... snip ...
  return esc;
}
```

Figure 26: rtems/cpukit/libblock/src/bdpart-register.c

Recommendation

Masking Error Codes: Return values should not be intercepted and changed by the calling function. By doing so, the caller will not receive contextual information regarding the error, while a more nefarious bug could wrongly return a success where an error code is warranted, thus masking an error.

Integer Overflow/Underflow: Many integer over/underflows can be caught using static analysis, but not all. Consider adding `assert()` calls into the code to catch mathematical operations that could overflow[45]. Any static analysis tools that instrument the source code would be better at catching such bugs that span many functions/macros.

Use of Uninitialized Variables: Reading from uninitialized variables is undefined behaviour according to the C specification. While this itself is not a security vulnerability, it could become one as code changes over time. NCC Group recommends the usage of static analysis tools and strict compiler flags such as `-Wuninitialized`. Not all variables need to be initialized, however it is good practice that can avoid potential security issues as a result.

Copying Memory Without Known Bounds: Avoid using memory and string functions that do not take the length of the buffers as a parameter. There are a number of functions that could be used instead, such as `strlcpy`. Perform a `grep strcpy` operation and find other instances of similar functions.

Inconsistent Use of Return Variables: Consider using a single return variable per function rather than multiple ones. A common idiom is to check if a return code is an error, and `goto` a cleanup step before returning said error value. Functions that return multiple values should use an output parameter to send data back to the calling function, while leaving the return statement to return success or error. Consultants found that this idiom was *mostly* followed in the code RTEMS code, but not universally. Enforcing a consistent coding style across the codebase will catch many of these types of errors.

45. https://stackoverflow.com/questions/57938488/overflow-assertions-on-various-c-data-type-which-ones-are-guaranteed-to-be-tr

6 Out of Date Software

Other than the core operating system itself, many services available to users are pulled from open source projects, of which many are out of date or have been deprecated. The following is not a complete list of open source dependencies, only a few samples are shown here to show widespread use of outdated libraries. However, the fact remains that these libraries are not patched in a timely manner to mitigate against potential attacks.

Networking

There are two main networking libraries available to RTEMS v5.3. The first is the legacy networking stack, called `libnetworking`, based on or around FreeBSD 4.4, and the second is based on FreeBSD 12.1, and is called `libbsd`. RTEMS users are able to select the legacy network stack using the `--enable-networking` parameter during configuration. The newer network stack has a different build step detailed in that repo's README[46] file.

For `libnetworking`, NCC Group was not able to find a definitive version that is used within RTEMS, however the code is similar to that of FreeBSD 4.4. While using old source code is not indicative of a vulnerability, a large number of high risk CVEs exist against this networking stack that have not been resolved. NCC Group echoes RTEMS guidance that this network stack is not used for any projects going forward.

The newer networking stack is based on FreeBSD 12.1, which was released in November, 2019[47], and was categorized as End of Life (EoL) in early 2021. In other words, this networking stack has not seen any updates since it's EoL, and a number of high-risk CVEs have been issued against this version of FreeBSD, which we were able to confirm.

PPP

The `pppd` source is taken from `ftp://cs.anu.edu.au/pub/software/ppp` and the latest used version is 2.3.11 (December 1999). NCC Consultants were unable to connect to the FTP address to determine if updates are available. However, some patches to this version were ported. A clone of the project exists[48], which is actively maintained, and the latest version is 2.5.0.

Mongoose HTTP

The `mghttpd` source dates from 2013 and sources indicate version 3.9. Since then, the Mongoose HTTP server code has been taken over by Cesanta Software[49] which actively maintains the code base. A number of vulnerabilities[50] have been filed against Mongoose, which may impact this specific version. The RTEMS team indicated on their website that the Mongoose library is out of date due to incompatibilities with the licensing model, stating *"Recent versions of the server have moved to an RTEMS-incompatible license and thus it is not up-to-date in RTEMS. It is anticipated that it will be removed at some future date"*[51]. While this library has not yet been removed, NCC Group recommends to avoid this version.

NFS Client

The NFS client in RTEMS supports NFS v2, standardized in RFC-1094. This version does not support any authentication mechanism or ACL and relies on the permission model of the filesystem for security. NFS v4 adds the concept of user separation. However, if a newer version of NFS is ported, RTEMS would not be able to enforce user separation as the entire OS and it's application run within the same memory space.

46. https://github.com/RTEMS/rtems-libbsd/blob/5.1-freebsd-12/README.md
47. https://www.freebsd.org/releases/
48. https://github.com/ppp-project
49. https://github.com/cesanta
50. https://www.cvedetails.com/vulnerability-list/vendor_id-16334/product_id-41402/Cesanta-Mongoose.html
51. https://devel.rtems.org/wiki/Developer/Mongoose_Web_Server

IPSec

IPSec and Racoon (IKE v1 client) were introduced as part of the `rtems-libbsd` networking stack. This project has been officially abandonned since 2014, however recent security patches[52] have been back ported. Both Racoon v1 and the standard it implements, IKE v1, have been deprecated[53] by the IETF. IKE v1 contains a fundemental flaw with Aggressive Mode, which is a feature of this protocol. The IETF and Racoon developers recommend using a more recent version of the protocol, currently implemented by Racoon2.

SUN RPC

The `librpc` service is based on SUN RPC v4.0, dated from 1989. A number of vulnerabilities have been filed against SUN RPC and its derivations[54]. However, with the exception of finding "XDR Length Check Can Overflow", many CVEs do not apply to the RTEMS codebase due to the age of the RPC code and its configuration. RPC is primarily used by the NFS client.

DHCP

DHCP functionality is provided by the `dhcpd` daemon, v6.2.1, written by Roy Marples[55]. At time of writing, the current version of this library is v10.0.6. The list of fixed issues on github does not go back further than 2018, and so NCC Group consultants were not able to get a complete listing of fixed vulnerabilities. No CVEs for this version of the library were found.

DNS

The `libbsd` repository contains mDNSResponder v878-270.2[56], which was released in 2021. Since then, many newer version of this library have been published and the current version is v2200.0.8. While NCC Group did not find any open or unpatched CVEs in this code, it is still recommended to upgrade to the latest version as the difference between the two version is substantial.

zlib Compression Library

RTEMS uses version 1.2.5 of zlib, which was released on Apr 19, 2010. This version is affected by a number of CVEs, such as CVE-2022-37434[57], CVE-2018-25032[58], and CVE-2023-45853[59]. The latest version of zlib is v1.3.1[60].

xz Decompression Software

RTEMS uses the `xz-embedded` library, which is a subset of the `xz-utils` project. Both project are actively maintained and incorporate fixes in a timely manner. This particular version dates from approximately 2013, and is based on the `xz-embedded-20130513` release. While NCC Group did not find security issues with this version, it is recommended to upgrade this component as it has received bug fixes[61] since then.

52. https://www.freshports.org/security/ipsec-tools
53. https://www.ietf.org/archive/id/draft-ietf-ipsecme-ikev1-algo-to-historic-07.html
54. https://www.cvedetails.com/google-search-results.php?q=sunrpc&sa=Search
55. https://github.com/NetworkConfiguration/dhcpcd
56. https://github.com/apple-oss-distributions/mDNSResponder/tree/rel/mDNSResponder-878
57. https://github.com/madler/zlib/issues/723
58. https://github.com/madler/zlib/issues/605
59. https://github.com/madler/zlib/issues/868
60. https://github.com/madler/zlib/releases
61. https://github.com/torvalds/linux/commit/4f8d7abaa413c34da9d751289849dbfb7c977d05

7 Static Analysis & Fuzzing

A number of static analysis tools are currently used by the RTEMS development team, such as Coverity[62] and clang-analyzer. These tools largely do not instrument the code to provide the analyzer with feedback about the scan. As such, static analysis of this type is limited in its coverage. Moreover, these tools were primarily used on RTEMS core functionality; consultants were unable to find documentation if such tools were run on all services.

Fuzzing is another method of finding bugs while the code is running. Many advanced fuzzers, such as AFL[63], instrument the binary before running it. This instrumentation provides feedback to the fuzzer and the information is used to guide the fuzzer in crafting inputs. Since RTEMS incorporates other open source projects, such as DHCP support, fuzzers also exist to test these types of network services.

Recommendation: Employing static analysis tools during a build is essential for catching bugs as they arise. However, the types of static analysis tools used will result in findings of variable quality. For a more in-depth scan, consider fuzzing various components in isolation. RTEMS developers should decouple code as best as possible to achieve this. For the various services supported by RTEMS, each of those are great candidates for fuzzing. For network services, such as DHCP, consider using Fuzzowski[64] or similar network-based fuzzers.

62. https://scan.coverity.com/projects/rtems
63. https://github.com/AFLplusplus/AFLplusplus
64. https://github.com/nccgroup/fuzzowski

8 Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Rating	Description
Critical	Implies an immediate, easily accessible threat of total compromise.
High	Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.
Medium	A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.
Low	Implies a relatively minor threat to the application.
Informational	No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

Rating	Description
High	Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.
Medium	Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.
Low	Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

Rating	Description
High	Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.

Rating	Description
Medium	Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.
Low	Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Category Name	Description
Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.

9 Contact Info

The team from NCC Group has the following primary members:

- Catalin Visinescu – Consultant
- Mark Paruzel – Consultant
- Jonathan Lindsay – Consultant