



GHOST  
PROTOCOL

WE

THE FIELD LIBRARY

DEEP TRACK

WHERE IT MEETS THE WORLD

# Hardware and the Edges

The physical layer, IoT and embedded, radio  
and wireless, physical access and side  
channels, and the edges.



Ghost Protocol (Pvt) Ltd · Colombo · [ghosts.lk](http://ghosts.lk)

Security internship training · The Field Library · Edition 1 · 2026

THE GHOST PROTOCOL FIELD LIBRARY · DEEP TRACK

# Hardware and the Edges

The physical layer, IoT and embedded, radio and wireless, physical access and side channels, and the edges.

Ghost Protocol (Pvt) Ltd · Colombo

The Field Library, Edition 1 · 2026

Copyright 2026 Ghost Protocol (Private) Limited. All rights reserved.

Sold as a PDF for the buyer's own reading. Please do not share or resell the file.

# The library

The Field Library is a whole shelf, not one book. The Core Path is the spine everyone reads. The rest is depth and breadth, taken on as the work calls for it. The shelf is complete now, and you are reading the one marked here.

## A · THE CORE PATH

---

O The Oath BUILT

I Groundwork BUILT

II The Web BUILT

III The Wider Field BUILT

IV The Work BUILT

V The Craft BUILT

## B · DEEP TRACKS

---

AI and Agents BUILT

Reading Code BUILT

Internal and Active Directory BUILT

The Web, Deeper BUILT

Cloud and Containers BUILT

Mobile and APIs BUILT

Reverse Engineering BUILT

Hardware and the Edges YOU ARE HERE

## C · THE OPERATIONS

---

Red and Purple Team BUILT

OSINT and Recon BUILT

Report Writing Mastery BUILT

The Consulting Craft BUILT

Bug Bounty BUILT

## D · THE OTHER SIDE

---

Detection and Response BUILT

Threat Hunting and Forensics BUILT

Hardening and Architecture BUILT

## E · THE GROUND RULES

---

The Law, Deeper BUILT

Standards and Compliance BUILT

Ethics and Disclosure BUILT

## F · COMPANIONS

---

The Reference Pack BUILT

The Lab Workbook BUILT

Case Files BUILT

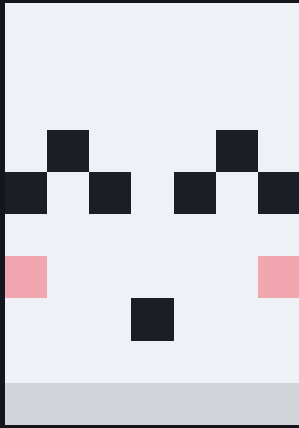
The Bestiary BUILT

The Glossary and the Ledger BUILT

# Contents

- Opening ..... 5
- 1** The Physical Layer ..... 6
- 2** IoT and Embedded ..... 10
- 3** Radio and Wireless ..... 15
- 4** Physical Access and Side Channels ..... 19
- 5** The Edges ..... 23
- Worked answers ..... 29

## Where the solder begins



The scope this time was not a website or a server. It was a small plastic device on the desk, with a cable and a blinking light. Blink circled it, unsure where the login box was.

BLINK "How do I test it? There is no page to open."

SOL "Software ends where the solder begins, and this is the other side of that line. A thing you can hold plays a different game: its firmware, its debug ports, the radio it speaks, the chip you can read with a clip. And the rule here is tighter than anywhere, because now you are in a room. You touch hardware only with written authorisation, and often with a signed letter in your pocket that says you are allowed to be standing there."

Worm's eye was on the letter, not the device.

WYRM "carry the letter."

So the book is the physical layer: embedded devices and their firmware, the radio and the wireless, the careful ethics of physical access, and the edges where the network finally meets the real world.

## CHAPTER ONE

# The Physical Layer

Every other volume treats the machine as something you reach over a wire. This one picks it up. The chapter sets the ground for the shelf: why a physical device is its own attack surface, what that surface is made of, and the posture a tester keeps when the target is an object on the bench rather than a service on a port.

---

A web app is a process you talk to through a browser. A network service is a port you talk to through a socket. Both hide the machine. This shelf is about the machine itself: the board, the chips, the ports on the edge of it, the radio coming out of it, and the room it sits in. None of that is exotic, and most of it is simpler than the web. It is also the surface where the oldest rule in security keeps coming true, so we start by saying the rule plainly, then take a device apart in our heads before we take one apart on the bench.

## Why hardware is its own surface

---

There is a long-standing rule in this trade: once someone has unrestricted physical access to a machine, it is no longer only your machine. Software defences assume the attacker is on the far side of an interface you control. Physical access removes that assumption. The attacker is no longer sending requests to your front door. They are holding the building, and they can open the back of it with a screwdriver.

That changes what controls mean. A login prompt stops a remote caller. It does not stop someone who can pull the storage chip off the board and read it on a bench. A firewall rule shapes what crosses the network. It does nothing about a debug header soldered to the board that drops you to a root shell with no password at all. The security of a device is only as good as the assumption that nobody will ever hold it, and for a great many devices, phones left on trains, routers in cupboards, sensors bolted to a wall outdoors, meters on the side of a building, that assumption is simply false. The device was shipped into the world, and the world includes people with screwdrivers.

So hardware earns a place in this Library for a reason the web does not need to argue. The device is a trust boundary that the designer drew and the physical world ignores. Your job on this shelf is to see where that boundary was drawn, and to find the places where holding the object lets you step across it.

## What a device adds to the surface

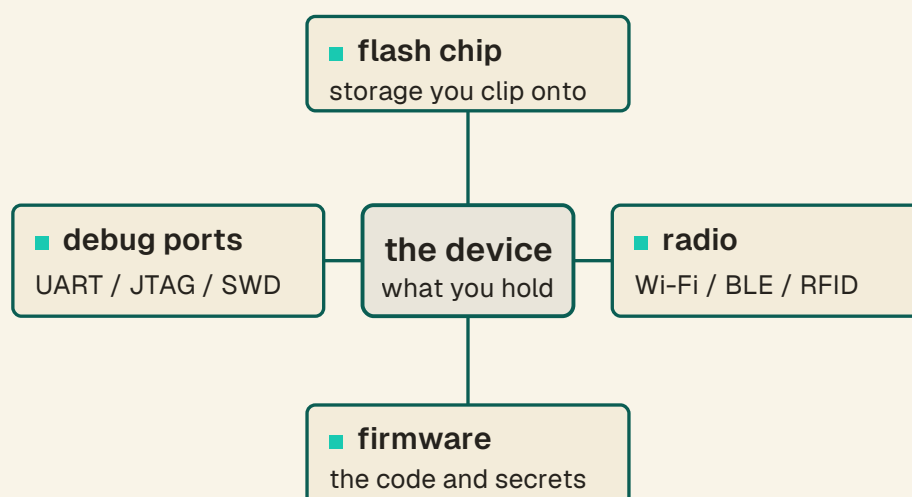
---

Take any connected device, a router, an IP camera, a smart lock, a point-of-sale terminal, a car's infotainment unit, and open it. The same handful of surfaces keep appearing, because they are how the thing was built and tested. Learn to name them on sight.

- **Debug and programming interfaces.** The ports the engineers used to bring the board up and flash it: a serial console (UART), a chip-level debug port (JTAG or SWD). They are frequently left enabled

in shipped hardware, because disabling them is an extra step nobody was paid to take. Chapter Two is about finding and reading them.

- **Storage you can read off the board.** The firmware lives in a flash chip, often a small eight-legged part you can clip onto and read without even removing it. Everything the device knows at rest is in there: the code, the configuration, sometimes keys and passwords. Chapter Two covers pulling it.
- **Radio.** If it is wireless it has an antenna, and the antenna is a door that needs no physical contact at all. Wi-Fi, Bluetooth, a proprietary link on an unlicensed band, an access card it reads. Chapter Three is the radio surface.
- **Physical ports a passer-by can reach.** A USB socket, an SD card slot, an exposed console port on the outside of the case. Anything a person can plug into without opening anything. Chapter Four covers what a hostile USB device does.
- **The process it sits inside.** Many devices are not the target so much as the way in: the controller on a factory line, the sensor feeding a building system. Where the device touches a physical process, the stakes change. Chapter Five is that edge.



The surfaces a held device adds: the firmware and the flash chip that stores it, the debug ports the engineers left on, and the radio. A remote attacker meets one interface; someone holding the device meets all of them at once.

Notice the shape of the list. A remote attacker sees the one interface the device chose to expose. Someone holding the device sees all of these at once, and most of them were never designed to be defended, because the designer assumed nobody would ever be holding it.

## Hardware is cheaper to attack than it looks

The picture of hardware hacking is an oscilloscope and a soldering station and a decade of practice. Some of it is that. Most of the findings that matter are not. A serial console that drops to a root shell is read with a four dollar adapter and a terminal program. A firmware image is pulled with a clip and a cheap programmer, or just downloaded from the vendor's own update page. Default credentials are tried by typing them. The tools on this shelf, the ones in the chapters ahead, cost tens of dollars, not

thousands, and that is the point. The barrier to the physical surface is low, which is exactly why it is worth a tester's attention and a defender's worry.

The expensive, specialist end of hardware exists, and this shelf names it where it starts: power and timing side channels in Chapter Four, silicon-level work it does not pretend to teach. We keep those at awareness. The working surface, the part a GP tester actually reaches for on an authorised engagement, is the cheap end, and it is where almost all of the real-world failures live.

## The tester's posture

Working on an object changes your habits in three concrete ways, and all three come from the same fact: the thing in front of you is physical, finite, and often the client's only one.

**Passive before active, even more than on the web.** Read before you write. Watch the serial console before you type into it. Dump the flash and read it on your bench before you change anything on the device. On the network a bad request usually just fails. On a board, a bad write can brick the device permanently, and you may be holding a unit the client cannot easily replace.

**Work on a unit you are allowed to destroy.** Hardware work carries real risk of damage: a slipped probe, a shorted pin, a flash write that does not complete. Before anything invasive, you establish which unit is yours to sacrifice. Testing on the client's one production device, the only camera on the live system, the meter still wired into the building, is how an engagement becomes an outage.

**Document the physical state.** Photograph the board before you touch it. Note which header is which, which way the chip faced, what you connected where. On the web the request history is your record. On a bench, your record is the photographs and notes you took, and the client will want the device back the way it came.

### ■ THE LINE

Holding a device is not permission to open it. Physical and hardware testing needs explicit written authorisation that names the device, and because this work happens on real objects in real places, it usually needs more paper than a remote test does: who owns the unit, whether you may open the case, whether you may risk damaging it, and where the work may be done. The law has not changed and it is not re-taught here: the Computer Crime Act No. 24 of 2007 and the full treatment of authorisation are in Volume 0. What this shelf adds is that the target is now a thing you can pick up, and picking up the wrong thing, a device that is not yours and not in scope, is the same offence it has always been. Chapter Four carries this line all the way onto a client's premises.



### SOL SAYS

The instinct a new tester brings to a board is the web instinct: poke it and see what happens. Unlearn it here. On a device the cost of a careless action is not a failed request, it is a dead unit and a client who no longer trusts you near their hardware. The professional move is slow on purpose. You read the chip before you write it, you work on the spare and not the live one, and you photograph the board before the first probe touches it. Patience is not timidity. It is the only way to come away with a finding and leave the device working.

## ■ EXERCISE

On a device you own and are willing to open (an old router, a dead phone, a cheap IoT gadget from a drawer), take the case off and map it. Do not connect anything yet. Photograph the board, then write down what you can identify by eye: the largest chip (usually the processor or system-on-chip), any flash memory chip (often an eight-pin part near it), and any row of unpopulated holes or a small header, which is very often a debug or serial port. Label each on your photo. You are building the habit of reading a board before touching it, and you will connect to what you found in Chapter Two.

## ■ ON YOUR FEET

A client asks GP to assess a smart door controller they sell. They courier over one unit, the engineering sample, with a note: "test whatever you like, just send it back working." It is the only one you have. You already suspect the interesting findings are inside: the firmware on the flash chip, and a debug header you can see on the board. What do you do before you touch the device with anything, and what one thing do you ask the client for before you do anything that could leave the unit damaged or unable to boot?



## WHAT WYRM KEEPS

A device is a trust boundary the designer drew and the physical world ignores. Someone holding the object sees every surface at once, and most of them were never built to be defended.

- Physical access defeats assumptions that software controls rely on. Once a machine can be held, a login prompt or a firewall rule does not stop the chip being read or a debug port being used.
- A device adds five recurring surfaces: debug and programming interfaces, readable flash, radio, exposed physical ports, and the process the device sits inside. The rest of this shelf takes them one at a time.
- The working surface is cheap to attack. The findings that matter mostly come from tens of dollars of kit, not a lab. The expensive silicon end stays at awareness.
- The posture is read before write, work on a unit you may sacrifice, photograph and document the physical state, and never touch a device without written authorisation that names it. The law is Volume 0.

## CHAPTER TWO

# IoT and Embedded

The working heart of hardware testing. This chapter takes an embedded device apart the way a tester does: get the firmware and read it, find the debug ports the engineers left on, read the flash chip directly, try the credentials the vendor never changed, and look hard at how the device updates itself.

An embedded device is a small computer running a small operating system, usually a cut-down Linux or a lightweight real-time system, with its program stored in a flash chip and a set of ports the engineers used to build it. Testing one is mostly a hunt for two things: the firmware, because it holds the code and the secrets, and a way in, because the engineers almost always left one. This chapter is the four moves that get you both, and the one design flaw, the update path, that decides whether any of it stays fixed.

## Get the Firmware First

Almost every embedded finding starts from the firmware, the single image that holds the device's code, its configuration, its web interface, and sometimes its keys and passwords in plain sight. You want it before you do anything invasive, because reading it is low risk and tells you what the device is. There are three honest ways to get it, in order of preference.

**Download it.** Many vendors publish firmware updates on a support page. That is the whole image, handed to you, no hardware needed. Always try this first. **Intercept it.** If the device fetches updates over the network, you may be able to capture the image as it downloads (which also tells you whether the download was protected, a finding in itself, covered below). **Read it off the chip.** When there is no download, you pull it directly from the flash, which the later section covers. The point of the order is the posture from Chapter One: the least invasive method that works, every time.

### Reading an image with binwalk

Once you have the image file, `binwalk` is the standard first tool. It scans the image for the signatures of known formats, compression headers, filesystem magic bytes, bootloader structures, and tells you what is inside and where. A typical run reports something like a uImage header, a compressed kernel, and then a `squashfs` filesystem at some offset (SquashFS is a read-only compressed filesystem that embedded Linux devices use constantly).

```
# binwalk reads an image and reports what it finds, by offset
$ binwalk firmware.bin
```

| DECIMAL | HEXADECIMAL | DESCRIPTION                             |
|---------|-------------|-----------------------------------------|
| 0       | 0x0         | uImage header, ... Linux kernel         |
| 14560   | 0x38E0      | LZMA compressed data                    |
| 1310720 | 0x140000    | Squashfs filesystem, little endian, ... |

Extracting that filesystem (modern binwalk does this with `binwalk -e`, and `unsquashfs` unpacks a SquashFS) gives you the device's whole root filesystem on your bench as ordinary files. Now the hard-