Chapter 2: Service Enrichment and Device Fingerprinting

Identification begins where discovery leaves off: when simply finding a device is not enough to understand what it is. While a host record indicates something is on the network, enrichment digs deeper, uncovering the device's true identity, measuring how much we can trust that claim, and highlighting what we still do not know.

Learning Objectives

By the end of this chapter, students should be able to:

1. Explain why service enrichment belongs to diagnostic analytics.
2. Distinguish a Chapter 1 host-existence claim from a Chapter 2 device-identity claim.
3. Compare MAC OUI, hostname, port signature, service product, HTTP, TLS, JARM, RTSP, ONVIF, SSDP, mDNS, Dynamic Host Configuration Protocol (DHCP)/router, Fingerbank, and TCP fingerprint evidence.
4. Describe how the Breakwater enrichment adapters run concurrently while preserving failure isolation.
5. Explain why confidence should rise only when stronger or materially different evidence supports the claim.
6. Trace a camera and a Cast device through enrichment, fingerprinting, and CPE construction.
7. Identify where enrichment must stop and where Chapter 3 vulnerability assessment begins.
8. State the limits of the current implementation without fabricating certainty.

2.1 Why Identity Comes Next

Discovery hands analysts a roster of devices, but enrichment steps in to reveal what those devices truly are and how much faith we can place in each identity claim.

The sequence of these steps matters. If a device slips through the gaps in Chapter 1, enrichment cannot bring it back. Mistaken identities ripple into vulnerability matching and attack-path analysis, spawning false alarms or hiding real threats. An inaccurate CPE can create phantom CVEs, while a missing one can let real vulnerabilities go undetected. Wildcard CPEs, if too broad, flood reports with noise instead of insight.

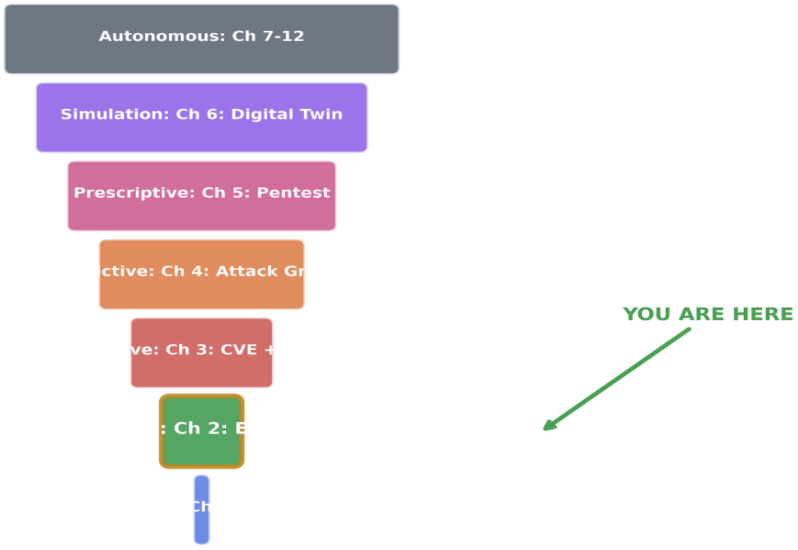


Figure 2.1. Analytics taxonomy. Chapter 2 occupies the diagnostic layer between Chapter 1's descriptive inventory and Chapter 3's detective vulnerability assessment.

The analytical boundary is simple:

Chapter 1: Discovery	What devices appear to exist?	Host record with evidence provenance
Chapter 2: Enrichment	What are those devices?	Identity artifact with confidence and CPE candidates
Chapter 3: Assessment	What weaknesses apply?	Candidate or confirmed vulnerability findings

Table 2.1. *Chapter boundary. Chapter 2 is the bridge between measured presence and vulnerability reasoning.*

Many tools blur those analytical boundaries, spitting out IP addresses, vendor names, service strings, and CVE lists in a single confident-looking line. But more detail does not always mean more truth. These outputs can blend stale ARP data, vague MAC prefixes, generic HTTP titles, and guesswork CPEs into a result that looks certain but is anything but. Chapter 2 urges a more discerning eye, teaching us to separate solid evidence from shaky assumptions.

2.2 The Running Case: A Google-Labeled Host

The running device begins with a sparse Chapter 1 handoff:

> ip: 192.168.86.42
>
> mac: d4:f5:47:xx:xx:xx
>
> vendor_hint: Google Inc.
>
> open_ports: [80, 443]
>
> discovery_sources: [arp_cache, router_arp, tcp] This record confirms the device is counted, but it is not enough for real management. A Google OUI hints at the maker, not the model. Ports 80 and 443 show that HTTP and TLS are open, but keep the device's true identity hidden. Without a hostname, the analyst must search for a name to call it.

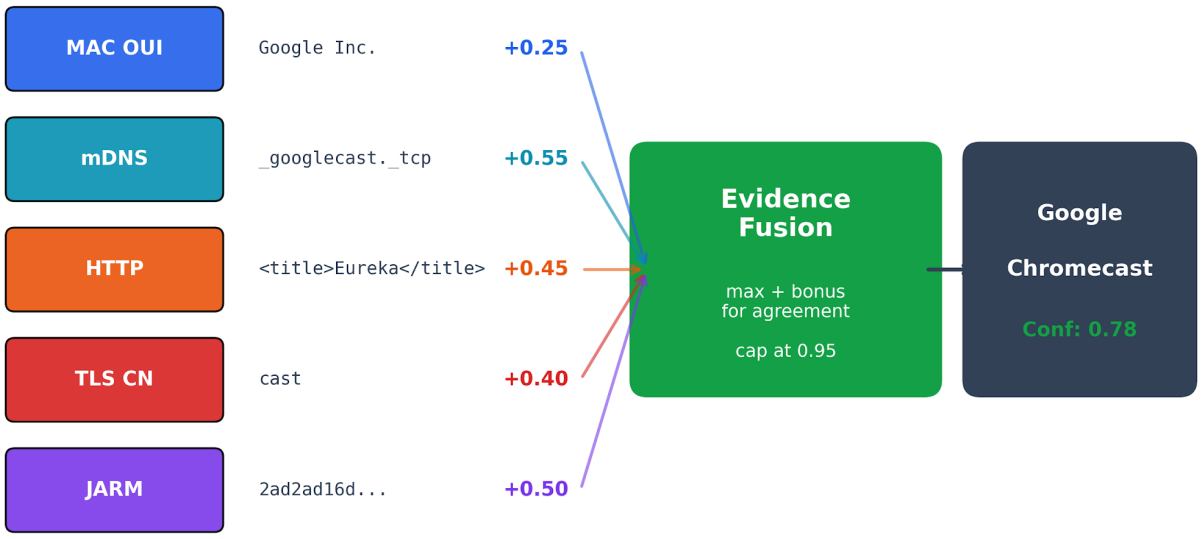
Enrichment's mission is to build a case from many clues, not just one. For example, if mDNS reveals _googlecast._tcp, the device might be a Chromecast or another Cast-enabled gadget. If the HTTP title spells out Chromecast, the case for that identification becomes stronger. On the other hand, a plain self-signed TLS certificate adds very little confidence. If JARM groups the host with other Google Cast devices, the confidence rises, but only to a certain point, since JARM observes TLS behavior rather than the device itself. In the course of this process, transparency is essential: the output must clearly indicate what evidence supports the claim, what evidence is missing, what is still uncertain, and whether the result is sufficiently reliable for Chapter 3. Every item should enable analysts to answer: "If Chapter 3 relies on this identity, what risks remain?"

This leads directly to the next section: Evidence and Trust Boundaries. Here, we examine how different types of evidence vary in reliability and how their intrinsic structural independence strengthens or weakens the ultimate identity claim.

Chapter 2: Service Enrichment and Device Fingerprinting

Service enrichment is detective work under a cloud of uncertainty. Analysts sift through open ports, service names, banners, certificates, multicast names, protocol responses, and even the lack of noise of missing replies. Some clues are solid, others are hazy, stale, or easily faked.

Figure 2.21: Evidence Fusion — Multi-Source Identification



Multiple weak signals combine into a strong identification
Agreement between sources boosts confidence beyond any single source

Figure 2.2. Evidence fusion across protocols. Independent signals converge on a single identity claim.

What matters most is structural independence. A handful of HTTP strings from the same web stack is less persuasive than a group of agreements from different protocols—an HTTP title, an RTSP banner, and an ONVIF response all pointing to the same camera family. When protocols agree, it means independent parts of the device are telling the same story. It is easy to fake a single title, but much harder to forge a collection of matching evidence across titles, certificates, RTSP behavior, ONVIF data, and port signatures.

MAC OUI	Manufacturer hint	Broad, randomized, spoofed, or tied to a module vendor
Hostname	Human or vendor naming clue	User-controlled, stale, or absent
Open ports	Probe routing and service opportunity	Ports do not prove product identity

nmap -sV	Service product and version hints	Generic probes miss IoT specificity
HTTP	Title, server header, authentication realm	Easy to spoof; often names the web stack
TLS certificate	Subject, issuer, subject alternative name (SAN), self-signed pattern	May identify vendor practice rather than device class
JARM	TLS implementation grouping	Groups shared stacks, libraries, or configurations
RTSP	Camera or media-server banner	Often incomplete or authentication-limited
ONVIF	Structured camera self-report	Strong, but still self-reported
SSDP/mDNS	Ecosystem and service advertisements	Link-local, unauthenticated, and sometimes noisy
DHCP/router	Lease name, MAC, and vendor-class hints	Router-specific and often incomplete
Fingerbank	Optional external classification	Requires a configured external service trust
TCP fingerprint	OS-family and stack hints	Low-confidence heuristics and permission-sensitive

Table 2.2. *Evidence sources and limits. Enrichment improves identity only when the limits remain visible.*

> The fingerprint service draws clear boundaries with ordered levels and confidence caps. A lone MAC OUI cannot outvote a detailed camera response. A service product string cannot overrule a clash between stronger protocols. ONVIF can inspire high confidence by offering manufacturer, model, firmware, and serial number, but it is not perfect—devices can misreport, lag, or even lie outright.

2.4 Implementation Anatomy

Breakwater does not rely on a single probe. Instead, it uses a fleet of small, specialized probes and lets the fingerprint service piece together the story. Orchestration happens in router.py, where enrich_host_batch() builds target lists, checks if the subnet is local or remote, launches adapter batches in parallel, merges their findings, builds CPEs, and hands everything to the fingerprint service. If one adapter fails, evidence quality drops, but the host remains in play.

The current fan-out returns twelve result sets across the following source files:

ARP	apps/api/app/scanning/arp_adapter.py	local link-layer corroboration
mDNS unicast	apps/api/app/scanning/mdns_adapter.py	known-host mDNS service data
mDNS browse	apps/api/app/scanning/mdns_adapter.py	passive link-local advertisements
SSDP	apps/api/app/scanning/ssdp_adapter.py	UPnP metadata and device XML
TLS	apps/api/app/scanning/tls_adapter.py	certificate identity fields
HTTP	apps/api/app/scanning/http_adapter.py	titles, server headers, authentication realms
Fingerbank	apps/api/app/scanning/dhcp_adapter.py	optional MAC/DHCP external classification
DHCP/router	apps/api/app/scanning/dhcp_router_adapter.py	router-side lease evidence
RTSP	apps/api/app/scanning/rtsp_adapter.py	camera and media-server banners
ONVIF	apps/api/app/scanning/onvif_adapter.py	structured camera identity
JARM	apps/api/app/scanning/jarm_adapter.py	TLS implementation hash
TCP fingerprint	apps/api/app/scanning/tcp_fingerprint_adapter.py	broad stack behavior

Table 2.3. *Adapter map. The file names are the implementation anchors for the enrichment evidence described in this chapter.*

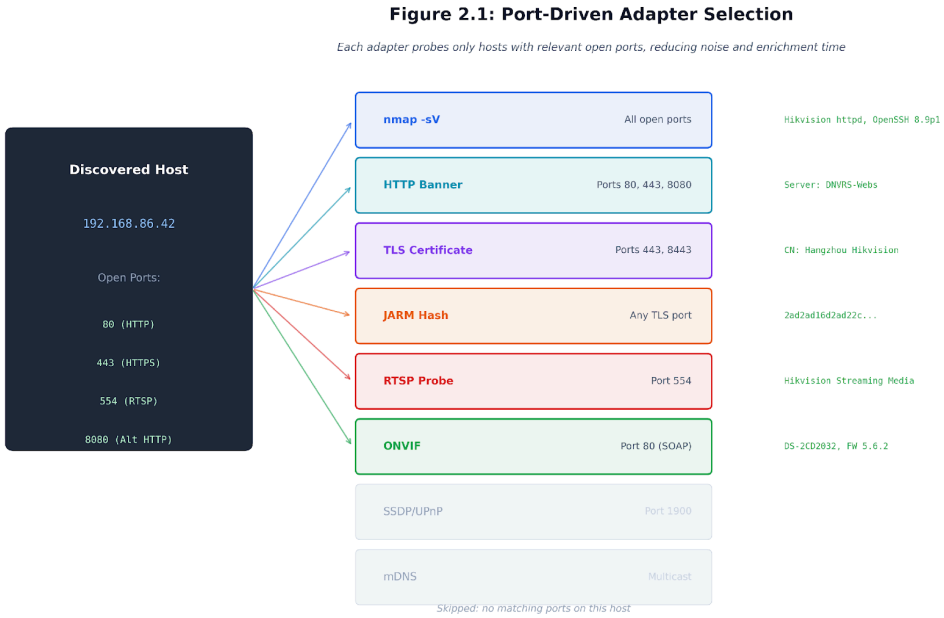


Figure 2.3. Port-driven adapter selection. Confirmed open ports decide which expensive protocol probes are worth attempting.

Choosing probes based on open ports makes the system nimble. If a host has no TLS port, JARM is skipped. No RTSP or ONVIF ports means camera probes are left out. Remote subnets skip local multicast traffic because it never crosses routers. These choices keep performance high and measurements clean by avoiding pointless data.

The adapter contract defines how each enrichment module interacts with the scanner. Specifically, an adapter:

- Accepts a host list as input,
- Returns a mapping of IP addresses to probe evidence as output,
- Handles errors internally by isolating exceptions so that one adapter's failure does not disrupt others,
- Can be turned on or off through configuration settings.

This contract enables modular enrichment and simplifies integration, making each adapter predictable and replaceable.

This contract matters in both the lab and the wild. ONVIF responses might be absent, SSDP XML broken, TLS handshakes may stall, or cameras might reject RTSP requests. The right move is to record what is missing, keep going with what you have, and never invent data to fill the blanks. In the lab, if an adapter fails or does not return evidence, students should clearly document which evidence is missing, note any resulting uncertainty in the identity claim, and proceed with the other available adapters. This approach averts confusion and makes certain that uncertainty is visible in lab results. In operational situations, it is best practice to explicitly tag hosts with missing or inconsistent evidence, mark them for follow-up review, and use automated alerts for patterns of persistent gaps or conflicts. Organizations deploying at scale can schedule periodic evidence audits to reexamine uncertain identities

and clear out false positives. By building these safeguards into enrichment workflows, security teams maintain situational awareness and avoid letting overlooked evidence propagate hidden risks downstream.

Figure 2.12: Graceful Degradation — Adapter Independence

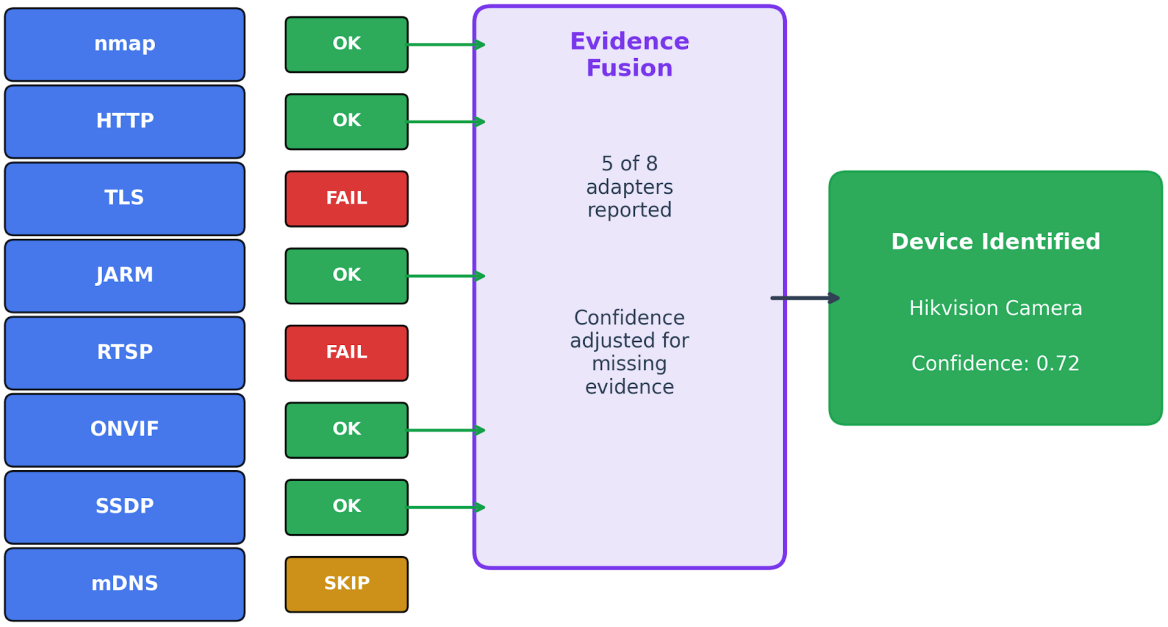


Figure 2.4. Graceful degradation. Missing adapters reduce evidence depth but do not remove the host or force a guessed identity.

Before adapters are launched, the router faces three essential decisions.

First, it decides whether the target is local or remote. Link-local evidence is valuable but limited. ARP cache reads, mDNS browsing, SSDP probing, and DHCP-router correlation make sense on a local segment. They do not have the same meaning across a routed subnet. The code path in _is_remote_subnet() uses scanner_config.scan_topology, exposed through BREAKWATER_SCAN_TOPOLOGY, to force local, force remote, or auto-detect overlap with the local network. In remote mode, the router skips link-layer adapters instead of treating their absence as a device property.

Second, it derives port lists from the service data it already has. TLS probes target services Nmap marked as SSL or HTTPS, plus known TLS management ports like 443, 5001, and 8443. HTTP probes target services with HTTP indicators or common web-management ports such as 80, 81, 88, 443, 631, 5000, 5001, 8008, 8080, 8081, 8088, 8181, 8443, 8880, 8888, 9000, and 9091. Camera probes use a narrower list: RTSP is attempted on 554, 8554, and 10554, while ONVIF candidates come from 80, 8080, 8899, and 3702. These values come from apps/api/app/scanning/router.py. They are practical defaults for the device classes Breakwater tries to identify, not a claim about all networks.

Third, it launches independent adapter batches with asyncio.gather(). A slow ONVIF response must not block HTTP titles from being collected. A malformed UPnP document should not erase TLS certificate evidence. A JARM timeout should not prevent a strong RTSP banner from reaching the fingerprint service. The adapter result is a dictionary keyed by IP address. Missing keys mean no useful evidence from that adapter, not that the host disappeared.

The progressive scanner scales this method up. Discovery, enrichment, and identification each have their own queues and counters, allowing the scanner to handle multiple hosts at once. Analysts do not have to wait for the whole subnet to finish; results flow in as they are ready. For teaching, each phase is clear: discovery fills the queue, enrichment adds evidence, and identification makes the call. Partial results are welcome if they show their stage and what is still missing.

2.5 Service and Protocol Evidence

Adapters make more sense when sorted by the kind of evidence they gather.

2.5.1 Nmap Service Detection

Nmap -sV acts as a wide-net service identifier. It scans open ports, sends protocol probes, and compares the replies to its database. This works well for familiar services like OpenSSH, nginx, Apache, MySQL, and many embedded web servers. Still, it struggles with IoT devices that hide behind generic web stacks or custom protocols.

Figure 2.5: nmap Service Version Detection Flow

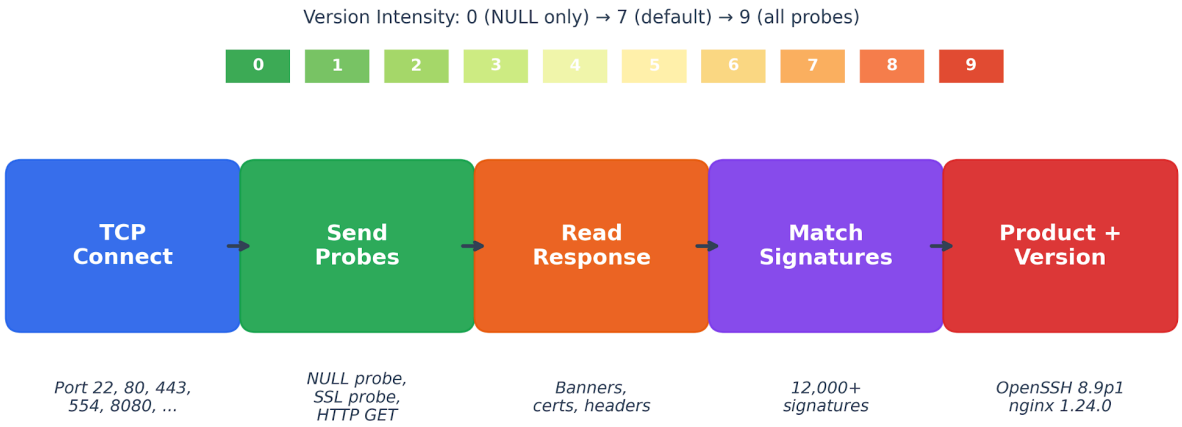


Figure 2.5. nmap service detection at work. A version probe reaches an open port, the response is matched against the probe database, and the result becomes a service-product hypothesis.

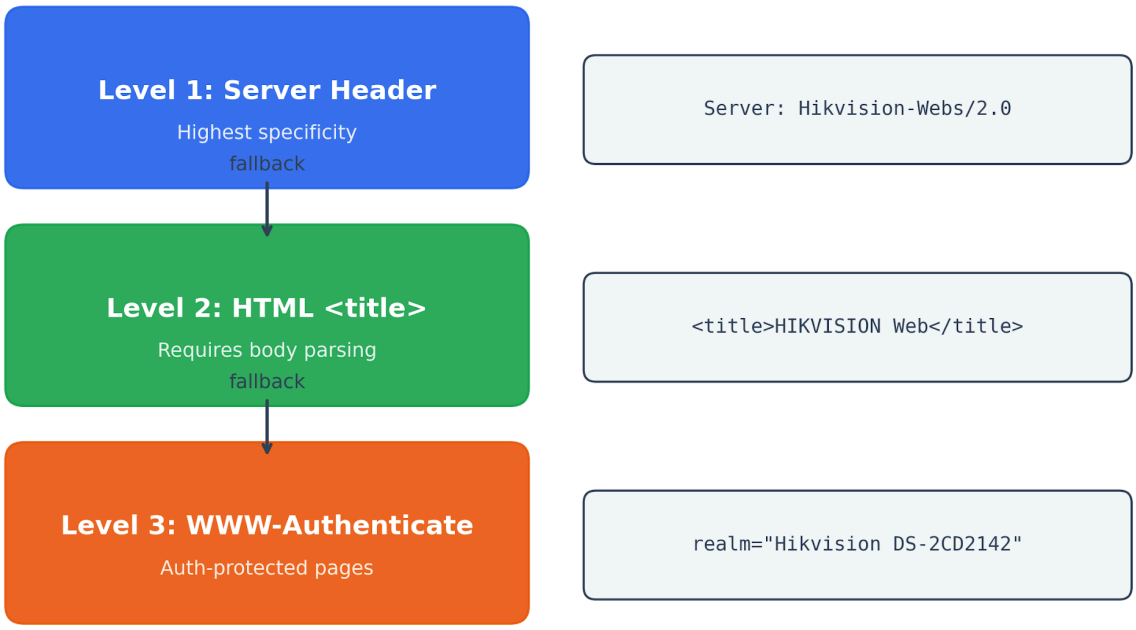
The Breakwater adapter dials back Nmap's version intensity for targeted scans. With service detection on, it uses -sT -sV --version-intensity 2. If running as root, it adds OS guessing and stretches the timeout. This strikes a balance: enough service evidence for identity work, but not so much that every enrichment run turns into a marathon scan.

Before moving forward, the router sweeps away known false positives. The _clean_nmap_false_positives() function filters out Nmap products like Neato Botvac connected, since generic service detection can mislabel common embedded HTTPS servers—especially in devices like Google Nest. Generic detection is helpful, but it should never be the only voice in the room.

2.5.2 HTTP and TLS

HTTP enrichment hunts for quick clues: titles, server headers, and authentication realms. On home and small office networks, these are often the fastest way to spot a Synology DiskStation, a Hikvision camera, or a Cast interface—even if the service product is bland. The enrichment record hands the HTTP title and server header to the fingerprint service. Realms help classify responses, but they only appear in the final identity if they are actually present.

Figure 2.6: HTTP Banner Extraction — Three-Level Cascade

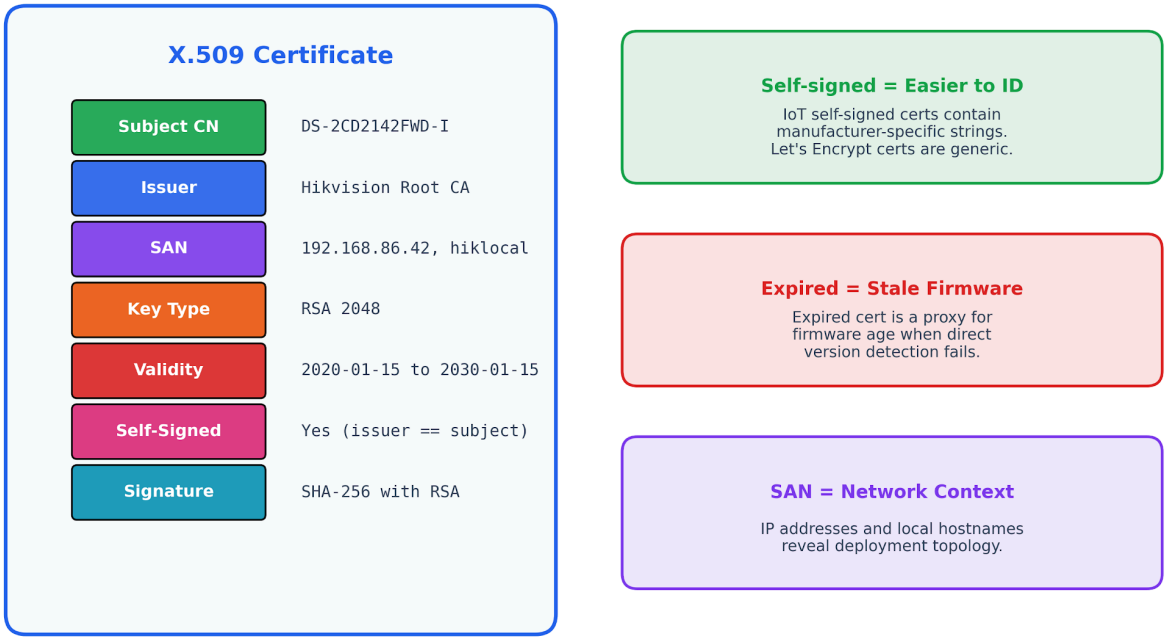


Body read limit: 8,192 bytes — catches >95% of IoT web interfaces

Figure 2.6. HTTP banner cascade. Web response strings are collected from bounded HTTP probes, and only the supported fields are included in the flattened enrichment. HTTP evidence is easy to misinterpret. A title might show a web framework instead of the device name. A server header could say "lighttpd" or "nginx" without revealing the actual product. Sometimes, a redirect hides the useful page. The adapter reads only the first 8192 characters of the response body and does not crawl the whole web interface. It uses httpx.AsyncClient allows redirects and follows up to three redirects. These limits are set in apps/api/app/scanning/http_adapter.py.

TLS certificate analysis delivers a fresh angle. The adapter pulls out the subject common name, alternative names, issuer details, and checks if the certificate is self-signed. While self-signed certificates usually raise eyebrows in web security, in IoT enrichment, they can serve as a fingerprint—vendors often stamp devices with unique factory certificates or issuer names.

Figure 2.7: TLS Certificate Fields for Device Identification



The TLS adapter turns off certificate verification while gathering metadata, letting it connect to devices that trusted public stores would reject. This does not mean the certificate is trusted—trust comes later. For now, the scanner's job is to collect and tag identity evidence, not to pass judgment.

2.5.3 JARM

JARM fingerprints devices by how they behave during TLS handshakes, not by what is on their certificates. The adapter sends 10 ClientHello messages and combines the responses into a unique hash. Devices with the same TLS stack and settings can end up sharing a JARM fingerprint, even if their certificates are worlds apart.

Figure 2.8: JARM — 10-Probe TLS Fingerprinting

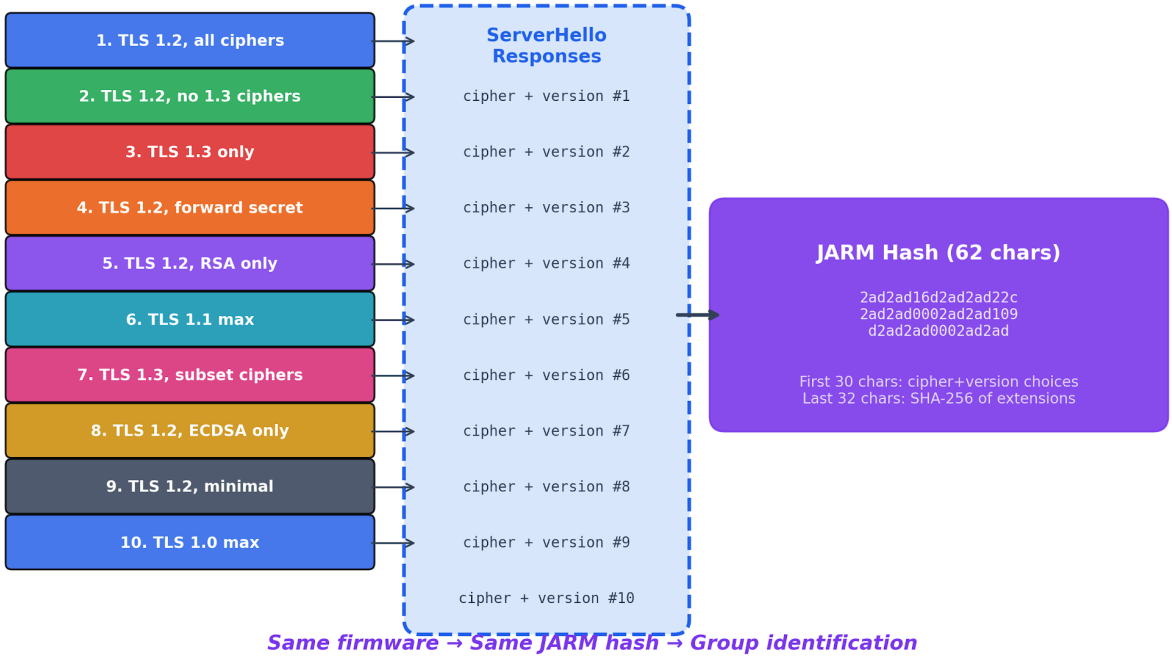


Figure 2.7. JARM probe construction. The jarm_adapter.py script sets up ten distinct TLS probe flavors by varying TLS versions, cipher suites, extension orders, and introducing GREASE values. It maintains a local record of known hashes for device families such as Synology, Plex, Home Assistant, Ubiquiti, Google Nest/Home, Axis, and Reolink. When a matching hash is identified, it is introduced into the fingerprint cascade with a confidence score of 0.70.

Figure 2.15: JARM Grouping Propagation

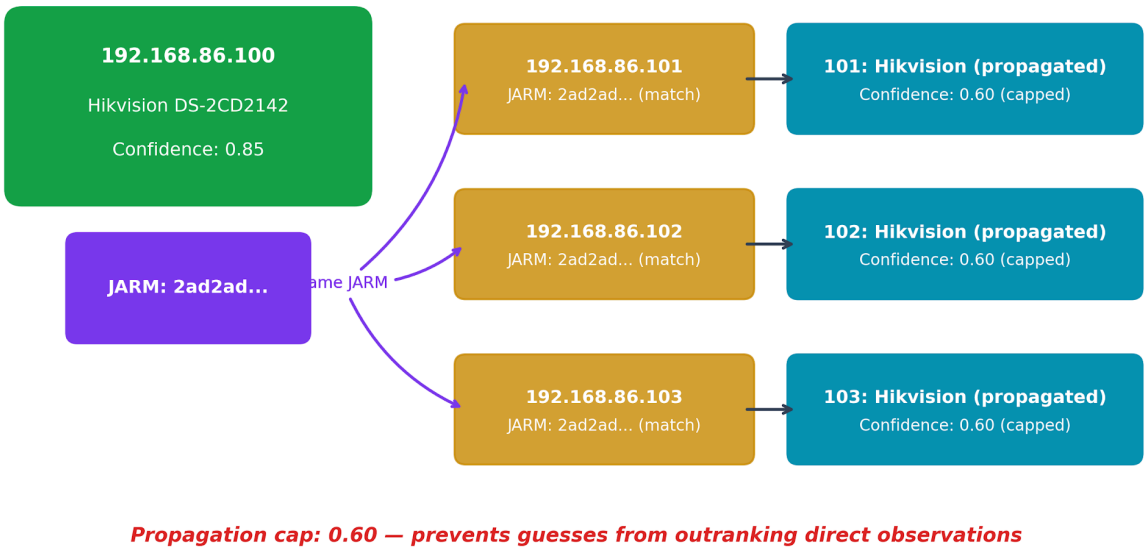


Figure 2.8. JARM grouping. One strongly identified TLS host can lend a capped hypothesis to unidentified peers with the same TLS behavior.

JARM also enables grouping. After the initial scan, the fingerprint service groups hosts by their jarm_hash. If a weak or unknown host shares a hash with a well-identified host, the service can copy the vendor, device type, and model from the best-identified host, but limits confidence to 0.60. JARM can suggest a group hypothesis but cannot guarantee certainty for every device. Students should use JARM to ask, “Which hosts behave like the device we already understand?” rather than “Which hosts are the same product?” A collision may occur because products share code, configuration, or a vendor platform. Each grouped host still requires corroboration before Chapter 3 treats its CPE as model-level evidence. In media management, they can reveal identity details that generic service probes often miss.

RTSP starts with an OPTIONS request. Many cameras respond without prompting for credentials, often disclosing a Server header or an authentication realm that reveals the vendor, firmware, or product family. While a generic HTTP title might just say "Login," an RTSP realm can shout "IP Camera" or drop a vendor’s name.

Figure 2.9: RTSP Probe Sequence for Camera Identification

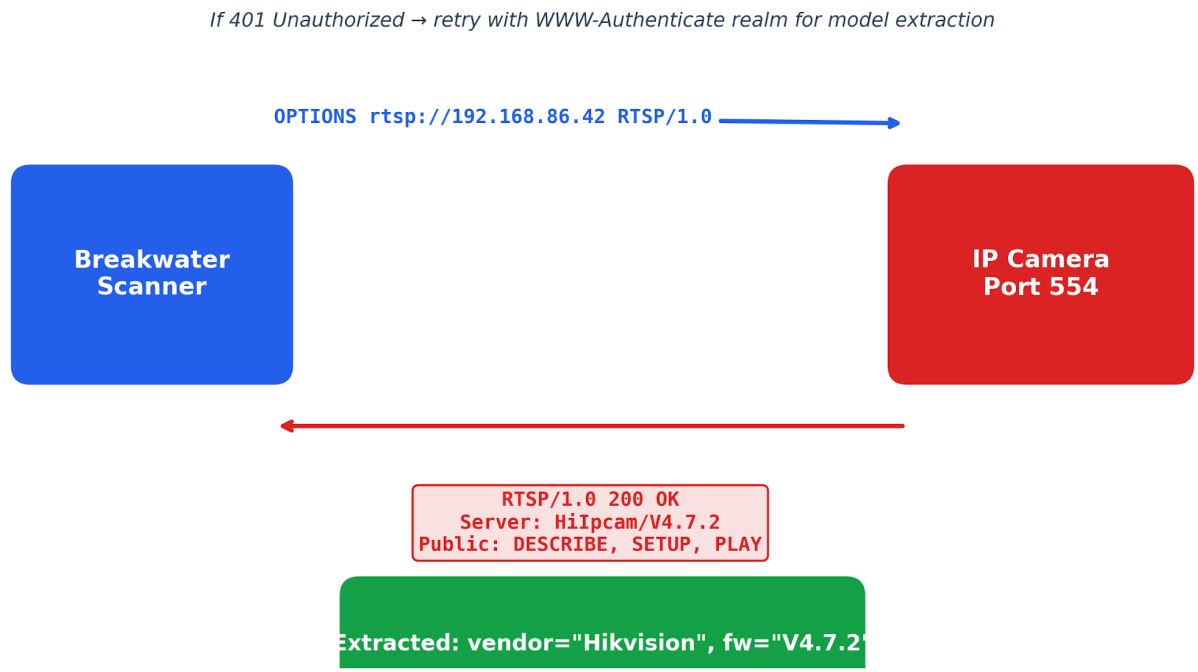


Figure 2.9. RTSP probe sequence. A lightweight OPTIONS request extracts methods, server headers, and sometimes an authentication realm.

ONVIF provides more information than a simple banner if the device responds. Its SOAP GetDeviceInformation call can return manufacturer, model, firmware version, serial number, and hardware ID in a structured XML format. The adapter in apps/api/app/scanning/onvif_adapter.py uses credentials from scanner_config.onvif_username and scanner_config.onvif_password, which can be set via the BREAKWATER_ environment prefix. By default, these fields are empty. Enrichment does not automatically attempt default credentials; testing defaults occurs later. The adapter authenticates when credentials are configured, sends a GetDeviceInformation request, parses the response, and emits a structured identity tuple.

ONVIF can inspire high confidence, but only up to a point. Its information is self-reported: firmware details might be old, OEM libraries might fill fields in odd ways, and a compromised device can simply make things up. ONVIF deserves trust, but only so far.

2.5.5 SSDP, UPnP, and mDNS

SSDP and mDNS show how devices introduce themselves on local networks. SSDP can hand over a UPnP device description packed with manufacturer, model, friendly name, serial number, device type, and service list. mDNS broadcasts service types like _googlecast._tcp, _airplay._tcp, HomeKit, IPP, RTSP, Sonos, RAOP, and SSH.

Figure 2.10. SSDP and UPnP enrichment. A multicast discovery response yields device-description XML that provides manufacturer, model, and service-list information.

The SSDP adapter relies on defusedxml.ElementTree rather than the standard XML parser, since it deals with XML from untrusted devices. Defusedxml acts as a shield, blocking malicious devices from launching XXE attacks against the scanner.

mDNS takes two routes: probe_mdns_batch() targets known IPs when port 5353 is open, while browse_mdns() listens quietly for local advertisements, catching devices that are not yet in the host map. The adapter even checks for macOS LAN interfaces, since VPN tunnels can hijack multicast routes. If the scanner listens on the wrong interface, it might miss devices entirely.

Multicast comes with built-in boundaries. mDNS only speaks on the local segment and cannot cross routers. If mDNS is silent on a remote subnet, it does not mean the device lacks Bonjour services—it may simply mean the scanner is listening from the wrong side of the fence.

2.5.6 Link-Layer, External, and Stack-Behavior Signals

Four smaller evidence paths sit beside the more visible protocol adapters.

> ARP and DHCP/router data help flesh out the host record before identification begins—arp_adapter.py fills in missing MAC addresses, vendor names, and hostnames from local ARP data. dhcp_router_adapter.py queries routers for hostnames and vendor hints via SNMP and SSH, and uses lease parsing and ARP as backups. These sources reveal what the local network knows, but cannot, alone, pin down the product's true identity.

Fingerbank is an optional tool. The dhcp_adapter.py script sends MAC addresses and optional DHCP fingerprint data to the Fingerbank API only if scanner_config.fingerbank_enabled is set and an API key is available. The results are helpful as external classification hints, but they do not have the same weight as local evidence from ONVIF or RTSP.

> TCP fingerprinting is kept on a short leash. tcp_fingerprint_adapter.py sends a raw probe to check TTL, window size, MSS, and TCP options to guess the OS family or device type. It is off by default, needs Scapy and raw sockets, and only gets a 0.35 confidence score. It is a behavioral hint, not a product name.

2.6 The Fingerprint Cascade

Once the adapters have gathered their evidence, the fingerprint service begins evaluating and ranking each identity clue. The fingerprint cascade is structured so that generic or broad hints are considered first, while more detailed, structured, and cross-protocol evidence shapes the final claim as evidence accumulates. The service uses rule tables for elements like hostnames, service products, RTSP server strings, port combinations, certificate issuers, HTTP titles, mDNS service types, DHCP vendor classes, and vendor-device mappings. These rule tables are critical because they clearly distinguish general clues from specific, actionable claims. This step-by-step cascade guarantees that the final identity claim is based on the strongest and most independent evidence available, preventing weaker or less reliable hints from controlling the result. The following examples show how the cascade protects the chapter from making overly confident claims:

1. A MAC OUI can support a vendor hint, but it cannot distinguish a Google Chromecast from a Nest thermostat.
2. Ports 8000 and 554 together support a Hikvision camera hypothesis, but the claim remains weaker than ONVIF.
3. _googlecast._tcp strongly suggests a Cast-capable media endpoint, but the rule must still account for televisions and displays that embed Cast support.
4. A certificate issuer named Synology may support network-attached storage (NAS) identity, but the certificate alone does not prove the exact model.

The Apple SSH guard is a good example of careful rule-making. macOS devices can show both AirPlay and SSH. Without this guard, AirPlay evidence might mislabel a MacBook as a media player. The fingerprint service checks for SSH first, making sure _airplay._tcp or _raop._tcp do not jump the line. The key is that a specific mix of signals should always take precedence over a misleading single clue, preserving the source of each claim.

Think of the cascade as a ladder with four rungs of specificity.

The first level is vendor hinting. MAC prefixes, OUI lookup, router leases, and external lookup services can indicate that a device is associated with vendors such as Google, Apple, Hikvision, or Synology. This level is useful for triage and grouping, but it is rarely sufficient for making a model-specific claim. It is also the level most affected by MAC randomization, locally administered addresses, recycled modules, and devices that embed network capabilities from other products.

The second rung is service-class inference. Port combinations, generic service products, and stack behavior can suggest that a host is functioning as a camera, printer, NAS, media player, router, or workstation. The rule table includes camera-like port sets and Google Nest/Home control ports. These rules are valuable because many IoT devices exhibit service patterns revealing their general type even when their names remain hidden. However, ports only reveal the services offered, not the device's true business function.

The third rung is protocol-stated identity. HTTP titles, authentication realms, RTSP banners, mDNS types, SSDP descriptions, TLS issuers, and ONVIF info all put words on the wire. Each protocol has its peculiarities: an HTTP title might be a template, an RTSP realm may reveal a family but not a model, and ONVIF can deliver structured details. This level is strongest when several surfaces agree, and weakest when a single, easily-edited string carries the whole claim.

The fourth rung is the normalized output. Here, the fingerprint result is shaped into a stable artifact: vendor, device type, model or family, confidence, match method, evidence, conflict notes, and CPE candidates. This is where a scanner row becomes an analytic statement. If the output loses the match method, Chapter 3 cannot tell where the CPE came from. If it drops the conflict note, Chapter 3 might mistake a disputed firmware version for a sure thing.

> This multi-tiered approach guards against a common analytical trap: letting one evidence source drown out the rest. Enrichment is not a strict hierarchy—the strength of each source depends on context. ONVIF is great for cameras, but not for printers. mDNS helps spot device ecosystems, but not firmware versions. TLS certificates hint at vendor habits, but rarely reveal device function. Port signatures guide probing, but cannot pin down a product.

2.7 Confidence as Communication

Confidence summarizes the quality of the evidence, the order of decisions, and the amount of uncertainty into a single value that other systems can use.

The implemented ceilings are concrete:

Fingerprint DB MAC prefix	0.85	Stored fingerprint match
ONVIF manufacturer/model	0.85	Structured camera self-report

SSDP manufacturer/model	0.75	UPnP device description
mDNS service type	0.75	Service-type rule match
Fingerbank score at or above 70	0.75	Optional external classification
Hostname rule	0.70	Hostname, mDNS name, or SSDP friendly name
HTTP title rule	0.70	HTTP title in flattened enrichment
JARM known hash	0.70	Local known-hash table
Exact port signature	0.65	Stored port signature
RTSP server header	0.65	Camera or media-server header
TLS certificate upgrade	0.65	Device-specific TLS issuer/CN over generic type
Service product string	0.60	Nmap product or service text
RTSP authentication realm	0.60	RTSP realm pattern
Fingerbank score 50-69	0.60	Optional external classification
JARM grouping	<= 0.60	Propagation from an identified peer with the same JARM
Subset port signature	0.55	Stored signature subset
DHCP vendor class	0.50	Router-side DHCP evidence
Fingerbank score 30-49	0.50	Optional external classification

Fingerbank score below 30	0.45	Optional external classification
HTTP server header	0.40	Generic server-header clue
TCP fingerprint	0.35	TTL, window, and device-hint heuristic
OUI vendor/device-type hint	0.25	Vendor prefix and broad device-type map

Table 2.4. *Implemented confidence ceilings. The values come from FingerprintService.identify_hosts() and _map_fingerbank_score() in apps/api/app/fingerprint/service.py; the TLS upgrade row comes from the post-processing path that upgrades generic tier-2 matches when device-specific issuer or common-name rules match.*

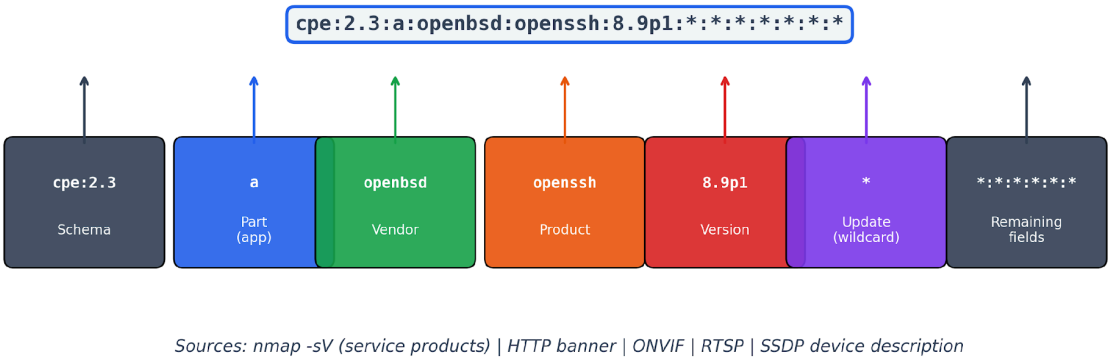
The table is not a statistical promise—it is a handshake on how the system behaves. Breakwater sets clear confidence ceilings but does not claim that a 0.75 score is gospel across a large, hand-checked fleet. True calibration would need labeled devices, repeatable scans, scoring scripts, and error analysis. Analysts should treat these as ceilings, not as precise odds.

To improve accuracy and trustworthiness over time, organizations should consider periodic calibration exercises. This can include validating scanner-generated identity claims against ground-truth datasets, conducting sample spot-checks of device inventories, or using automated tools to compare results with manually confirmed devices. Consistent feedback loops between enrichment outputs and observed reality help teams raise their confidence ceilings and catch emerging sources of error. Incorporating such validation procedures into production workflows helps preserve trust in automated scores and allows more defensible vulnerability assessments as networks evolve.

Confidence climbs when evidence is strong, specific, and comes from independent sources. OUI plus mDNS plus HTTP beat OUI alone. ONVIF, RTSP, and a matching port signature make a stronger case. Repeating weak signals does not help—a single detailed model response is worth more than many generic banners. For instance, if the OUI indicates Google but mDNS advertises AirPlay, the system ought to favor the more specific protocol and explicitly document the conflict. The rationale for the confidence score is as important as the score itself. Future analysts need clarity about whether low confidence results from missing, contradictory, generic, externally sourced, or self-reported evidence.

2.8 CPE construction belongs in Chapter 2 because CPE starts as an identity result, not just a vulnerability lookup. Chapter 3 will find vulnerabilities, while Chapter 2 decides whether the product string is sturdy enough to proceed to the next step.

Figure 2.16: CPE 2.3 String Anatomy



Sources: nmap -sV (service products) | HTTP banner | ONVIF | RTSP | SSDP device description

Figure 2.11. CPE anatomy. Vendor, product, part, and version fields translate identity evidence into the naming scheme consumed by vulnerability databases.

The cpe_builder.py script assembles CPE candidates from service products, HTTP headers, ONVIF details, SSDP fields, RTSP strings, TLS issuers, and fallback identity data. The output must always remember its roots—a CPE built from an ONVIF model and firmware is higher quality than a wildcard CPE based solely on vendor and device type. Both have their place, but Chapter 3 needs to know which is which.

exact self-report	explicit version observed in structured protocol data	strong candidate, still self-reported
normalized	version parsed and cleaned from a banner or service string	useful, but review the parsing context
family-only	product known, version missing	broad recall, high false-positive risk
wildcard	vendor or family only	admissible only with explicit uncertainty
conflicting	sources disagree	Requires a conflict note before Chapter 3

Table 2.5. *Version-confidence classes. These classes interpret the CPE builder's output for downstream use; they are not a separate scoring API.*

The golden rule: keep uncertainty in plain sight. A wildcard CPE is fine for general awareness, but not as proof of a vulnerability. If CPEs conflict, the system should not just grab the newest or simplest one. Chapter 2's job is done only when uncertainty is clear enough for Chapter 3 to handle with care.

The CPE builder has two main tasks: create enough candidates for Chapter 3 to find the right vulnerabilities, and label them clearly so weak matches can be flagged or ignored. build_cpe_from_identity() steps in when a device has a

strong identity but no version info. This only works if the next step remembers the candidate came from identity, not from a specific software version.

2.9 Worked Trace: Camera and Cast Device

The IoT simulator gives Chapter 2 two striking contrasts: a Hikvision camera bristling with local protocol surfaces, and a Google Chromecast whose strongest identity clue comes from the Cast ecosystem, not a classic management banner. A host moves from discovery output through enrichment adapters, fingerprinting, confidence scoring, and CPE construction.

The home-network host in Section 2.2 builds intuition, but lacks a known answer key. That is why the working trace uses the simulator, which provides labeled teaching examples in ground-truth.json. Here, 26 devices live on 172.30.0.0/24. The trace does not pretend every adapter will always deliver; it shows what the code can gather, what the lab says is present, and in which identity one ought to remain cautious.

2.9.1 Hikvision Camera

The lab camera at 172.30.0.10 is recorded as:

```
    ip: 172.30.0.10
    type: ip_camera
    vendor: Hikvision
    model: DS-2CD2143G2-I
    firmware: V5.5.52 build 200915
    mac: c0:56:e3:00:00:01
protocols:
  - http/80
  - rtsp/554
  - tcp/8000
  - ssdp
known_cves:
  - CVE-2021-36260
  - CVE-2017-7921
```

That ground truth is not the scanner's output—it is the answer key, used to judge if the scanner's identity claim holds water. The scanner still has to earn its label through real evidence.

Discovery hands Chapter 2 an IP address, a MAC address, and a set of open ports. The router spots port 80 and sends the host to HTTP probing. It sees 554 and triggers RTSP probing. Port 8000 allows the fingerprint service to apply the Hikvision port-combo rule when 554 is also present. Since the host is local, SSDP is in play. Port 80 also makes the host an ONVIF candidate, but the simulator does not have to return ONVIF info. That difference matters: the ONVIF adapter can try, but ONVIF evidence is never a sure thing.

OUI	c0:56:e3 Hikvision prefix	broad vendor support
nmap/open ports	80, 554, 8000	service routing and camera-class support
HTTP	title or server header if exposed	Vendor support when branded
RTSP	OPTIONS on 554	camera/media-server evidence
SSDP	UPnP description is returned	manufacturer/model support
ONVIF	attempted when candidate ports are present	high value only if the device answers

Table 2.6. *Hikvision trace. The table separates eligible probes from the evidence actually observed.*

The table draws a line between what probes could run and what evidence actually appeared—this is where many scanner reports stumble. A sloppy report might claim, "port 80 exists, so ONVIF confirmed the camera." The code does not back that up. probe_onvif_batch() might return nothing if the library is missing, the port is blocked, credentials are required, or the simulator skips ONVIF. The correct record states that ONVIF did not contribute and never fills in the ONVIF fields in the answer key.

A defensible enriched record for the lab camera might look like this:

```
  ip: 172.30.0.10
  vendor: Hikvision
  device_type: camera
  model_or_family: Hikvision IP Camera / DS-2CD family
  confidence: 0.75
  match_method: port_combo_or_rtsp_server
evidence:
  oui: c0:56:e3
  open_ports: [80, 554, 8000]
  rtsp_port: 554
  ssdp: present_if_description_returned
not_observed:
  onvif: no usable GetDeviceInformation response
  version_confidence: family-only
  chapter_3_admissibility: query camera-family CPEs with review
```

If ONVIF returns a manufacturer, model, and firmware string in a real deployment, the identity can be upgraded:

```
vendor: Hikvision
device_type: camera
model: DS-2CD2143G2-I
firmware: V5.5.52 build 200915
confidence: 0.85
match_method: onvif
version_confidence: exact_self_reported
```

The two artifacts have different downstream meanings. The first is a strong claim about the camera family. It can route the host into camera-specific review and broad CPE matching, but should not be treated as a confirmed firmware-level result. The second is a model-level self-report with a firmware string. It can support narrower CPE candidates while still carrying the ONVIF self-report caveat.

The same device also illustrates why Chapter 2 should not absorb Chapter 3. The ground truth lists CVE-2021-36260 and CVE-2017-7921, but Chapter 2 does not declare those vulnerabilities just because the model resembles a Hikvision camera. It produces CPE candidates and evidence quality. Chapter 3 decides whether a CVE applies, whether the version is in range, whether a tool corroborates it, and whether a credential or endpoint test is authorized.

2.9.2 Google Cast Device

The lab Chromecast at 172.30.0.20 is recorded as:

```
ip: 172.30.0.20
type: media_player
vendor: Google
model: Chromecast
firmware: 1.68.382489
mac: 54:60:09:00:00:01
protocols:
  - http/8008
  - tcp/8009
  - tls/8443
  - mdns
notes: Eureka info endpoint exposes device metadata. mDNS announces _googlecast._tcp.
```

This device is a better example than a generic Google host because it shows the difference between vendor identity and product-family identity. A Google OUI alone could represent many products. _googlecast._tcp narrows the claim to a Cast-capable endpoint. HTTP on 8008 lets the scanner read an interface associated with the Cast stack, including Eureka metadata when exposed by the device. TLS on 8443 allows certificate and JARM probing. TCP 8009 contributes to the service surface but, by itself, does not prove an exact model.

OUI	54:60:09 Google prefix	broad vendor hint
HTTP	8008 eligible for title/header collection	Cast support if a branded response appears
TLS	8443 are eligible for the certificate and JARM	supporting implementation evidence
mDNS	_googlecast._tcp	strong ecosystem signal
TCP	8009 open	service-surface support

Table 2.7. *Google trace. The result may be strong at the device family level even when the exact hardware model is unavailable.*

The correct output is neither a Google device nor an overconfident exact model. A defensible artifact might say:

```
ip: 172.30.0.20
vendor: Google
device_type: media_player
family: Chromecast / Google Cast
confidence: 0.75
match_method: mdns_service_type
supporting_sources:
  - oui
  - mdns
  - http_if_branded
  - tls_or_jarm_if_present
limits:
  - exact hardware model depends on HTTP/Eureka metadata availability
  - Cast-capable televisions can share ecosystem signals
chapter_3_admissibility: family-level CPE only
```

The implemented mDNS service-type rule bounds the confidence in that example. If a stronger SSDP or HTTP record supplies a concrete model, such as Chromecast Ultra, the identity may be more specific. If mDNS is missing because the scanner is remote, confidence should decrease. The system should not hide that difference because the lab answer key knows the truth.

A degraded output might look like this:

```
ip: 172.30.0.20
```

```
    vendor: Google
    device_type: unknown_or_media_player
    confidence: 0.25
    match_method: oui_vendor
evidence:
    oui: 54:60:09
    open_ports: [8008, 8009, 8443]
missing:
    mdns: not visible from routed scanner
    http_title: no branded response
    jarm_match: none
    chapter_3_admissibility: no model-specific CPE
```

The difference between the two outputs is the key lesson. Inside the rich local case, _googlecast._tcp plus the Cast port pattern supports a media-player family claim. In the degraded case, the same device may remain a Google-labeled host with only a few ports open. The lower score reflects the limited evidence available. The main takeaway is that the reliability of the identity claim depends directly on the quality and independence of supporting evidence. Students should recognize that developed output must accurately communicate both its strengths and its limits.

The camera shows that rich service exposure does not automatically mean exact identity. HTTP, RTSP, SSDP, and camera port patterns can support a strong camera-family claim even when ONVIF is absent. If ONVIF answers, the claim improves. If it does not, the artifact must say so.

The Chromecast shows that a device can be well identified without a traditional management protocol. mDNS and Cast-specific ports can carry the family claim. HTTP and TLS can help, but they may not expose the exact hardware model. The result is still operationally useful because media devices have different risk controls from routers, cameras, workstations, and printers.

Both examples reinforce the same handoff rule: Chapter 2 must provide Chapter 3 with the most defensible identity claim available, along with the supporting evidence and the appropriate level for CPE admissibility. Chapter 2 should not translate ground truth directly into observations, nor should it treat missing adapter results as negative attributes of the device. Additionally, it must avoid equating a broad family-level CPE with an exact version match. This exact delineation ensures identity claims remain accurate and that uncertainty is clearly communicated downstream.

The chapter's evidence is code-grounded, fixture-grounded, and test-grounded. It is not a field benchmark.

Implementation	apps/api/app/scanning/router.py	adapter fan-out, local/remote gating, port-derived target lists, merge behavior	does not prove field accuracy

Implementation	adapter files under apps/api/app/scanning/	protocol-specific collection behavior	Adapter success depends on topology, credentials, libraries, and device response
Implementation	apps/api/app/fingerprint/service.py	rule tables, confidence ceilings, Apple SSH guard, JARM grouping	values are engineered ceilings, not calibrated probabilities
Implementation	apps/api/app/scanning/cpe_builder.py	CPE generation paths and identity fallback	Provenance classes are interpretive guidance, not a separate API field
Fixture	student-lab/ground-truth.json	labeled simulated devices and protocol profiles	answer key, not scanner observation
Lab	student-lab/phase2-lab.md and walkthrough	student reasoning tasks and identity-output contract	instructional exercises, not measurements
Tests	apps/api/tests/test_adapters.py and related adapter tests	AirPlay/SSH guard, mDNS rules, JARM grouping, CPE construction, adapter failure isolation	unit and integration behavior, not population accuracy

Table 2.8. *Provenance map. The chapter's claims are limited to what these artifacts can support.*

This distinction matters because timing and accuracy claims require raw run artifacts. A measured accuracy table would need the scan command, fixture state, expected answer set, scoring script, timestamp, and failure cases. Without those artifacts, the evidence can explain the implementation rationale, what the lab fixture contains, and which tests exercise that behavior. It cannot support a field identification rate.

Enrichment helps when it clarifies evidence. It becomes unsafe when it makes weak evidence appear certain. Cloud-only devices may expose little local identity beyond OUI and narrow discovery evidence.

1. MAC randomization can weaken vendor-based inference.
2. Generic embedded web servers can hide the product behind the server stack.
3. ONVIF may require credentials or may return stale firmware fields.
4. JARM can group products that share TLS libraries or configurations.
5. Multicast evidence usually disappears across routed boundaries.
6. A scanner without the right interface, route, privilege, or dependency may see only part of the device.

Adversaries can also manipulate enrichment. Banners, HTTP titles, TLS certificates, and multicast advertisements are cheap to spoof. A compromised device can lie through ONVIF. A bridge can advertise services on behalf of another host. A locally administered MAC can defeat OUI assumptions.

To detect or mitigate spoofed evidence, analysts should cross-validate identity claims from multiple independent sources whenever possible. If observed identities conflict across protocols and over time, this can indicate tampering. Automated anomaly detection tools can help highlight behavior or attribute inconsistencies, such as a hostname that doesn't match the vendor suggested by the OUI or a banner appearing in an unexpected context. Whenever feasible, use structured evidence from protocols that require stronger authentication, or corroborate device identity with out-of-band data. These techniques help raise suspicion when an adversary attempts to forge or disguise device properties.

The practical defense is disagreement-aware evidence. A claim supported by OUI, HTTP, TLS, RTSP, ONVIF, and repeated observations is harder to fake than one supported by a single banner. A claim with a visible conflict note is safer than one that hides contradiction behind a clean label.

2.12 Student Lab Bridge

The Phase 2 lab asks students to inspect service enrichment as a reasoning problem, not endpoint hunting. The lab uses GET /v1/scanning/smart-scan/{scan_id}/results and asks students to separate direct observation, interpreted identity, and remaining uncertainty. That is the same reasoning standard the chapter requires. Grading will stress clarity of reasoning: a strong lab answer cites concrete evidence from the scan, explains how identity conclusions were drawn, and clearly states what is still unknown or ambiguous. Answers should avoid guessing or masking uncertainty, and instead provide explicit justification for each aspect of the identity claim. This approach ensures that students develop an evidence-based mindset aligned with the analytical goals of enrichment.

The lab commands are intentionally ordinary. Students authenticate against the local API, select a completed scan, inspect compact host views with jq, and write bounded identity judgments. To make the process clear, the essential lab steps are:

1. Authenticate to the local API.

2. Select a completed scan from the available results.

3. Inspect the scan data, focusing on compact host views (for example, using jq).

4. Judge and record identity claims for the hosts, supplying evidence and noting any unresolved uncertainties.

The goal is not to memorize adapter internals but to defend a claim such as:

```
identity_claim: Hikvision camera family
```

```
source_provenance:
  - open ports 80, 554, 8000
  - RTSP server header if observed
  - ONVIF device information if observed
    version_confidence_class: family-only_or_exact_self_reported
    ambiguity_flag: true_or_false
    downstream_admissibility: broad_family_query_or_model_level_query
```

The strong student's answer states what was observed, what was inferred, and what remains unresolved. A weaker answer collapses those layers into a tidy label. The tidy label is insufficient because it gives Chapter 3 no way to determine whether a CPE is admissible, broad, or unsafe to use.

2.13 What Chapter 2 Produces

At the end of enrichment, each host should carry a structured identity artifact:

```
  ip: 172.30.0.20
  vendor: Google
  device_type: media_player
  model_or_family: Chromecast / Google Cast
  confidence: 0.75
  match_method: mdns_service_type
evidence:
  oui: 54:60:09
  mdns: _googlecast._tcp
  http_8008: observed_if_present
  conflicts: []
  version_confidence: family-only
cpe_candidates:
  - value: cpe:2.3:o:google:chromecast_firmware:*:*:*:*:*:*:*:*
    source: identity_fallback
    admissibility: broad family-level query solely
```

This artifact is intentionally more detailed than a typical scanner output. It communicates to downstream systems the pipeline's identity assessment, the supporting rationale, and the limits of that assessment. The confidence score, source list, conflict notes, and version classification are essential controls that prevent Chapter 3 from treating all CPEs as equally reliable.

Beyond Chapter 3, these enriched identity artifacts are suitable for integration with external platforms such as SIEMs and vulnerability management tools. By exporting structured identity, confidence, and provenance data, organizations can automate alerting, correlate device profiles with incident events, and inform risk prioritization across their wider security stack. Direct API integrations allow these artifacts to contribute to consolidated asset management, facilitate automated ticketing for review cases, and enhance vulnerability monitoring pipelines. This functional flexibility increases the practical value of the enrichment process and embeds its outputs into real-time security workflows.

Chapter 3 begins only after this work is done. Vulnerability assessment needs CPEs, service versions, device types, open ports, credential-test eligibility, and evidence quality. It also needs restraint.

A Chapter 2 result can say three things Chapter 3 needs:

1. the best current identity claim and the evidence that supports it;
2. the version-quality class, including whether firmware is exact, normalized, missing, or conflicting;
3. The admissibility of each CPE candidate, including whether it is primary, fallback, or too broad for confident matching.

Those statements are the handoff contract. Chapter 3 may query vulnerability data, run validation tools, and test credentials where authorized. But Chapter 3 should never have to guess whether an identity came from ONVIF, a title string, a MAC prefix, a JARM group, or a broad fallback. Chapter 2 must carry that provenance forward.

2.14 Open Problems

Three open problems remain.

First, confidence calibration needs more labeled data. Breakwater assigns disciplined confidence ceilings, but a stronger empirical program would compare those scores against manually verified identities across larger and more diverse fleets.

Second, adversarial identity deserves deeper treatment. Most enrichment signals were designed for service discovery, not hostile settings. Future systems need better disagreement detection and stronger signals that are expensive to forge.

Third, identity sharing has privacy costs. Shared JARM clusters, banner rules, and device fingerprints would help defenders, but device inventories disclose sensitive organizational information. Publishing or aggregating such inventories can expose details about an organization's technology stack, network structure, and possible vulnerabilities. Prudent consideration is needed before sharing or correlating device identity data between environments.

The central principle is consistent: enrichment is beneficial when it clarifies evidence, but it becomes problematic when it transforms weak indicators into overly confident identity claims. Enrichment should always make uncertainty visible, allowing defenders to act on clear information, and never conceal doubt by presenting guesses as facts.

Summary

Chapter 2: Service Enrichment and Device Fingerprinting

Service enrichment functions as the diagnostic intermediary between discovery and assessment. It goes beyond solely appending banners to host records by converting partial protocol evidence into a bounded identity claim suitable for downstream evaluation.

The most robust Chapter 2 outcome is not the label that appears most confident, but rather the one that transparently presents its supporting evidence, methodology, confidence ceiling, conflicts, version quality, and CPE admissibility. This lucidity enables Chapter 3 to assess vulnerabilities without inheriting errors of concealed identity.

References

Althouse, J. (2021). "JARM: An Active TLS Fingerprinting Tool." *Salesforce Engineering Blog.*

Cheshire, S. & Krochmal, M. (2013). "Multicast DNS." RFC 6762. IETF.

Cheshire, S. & Krochmal, M. (2013). "DNS-Based Service Discovery." RFC 6763. IETF.

Dierks, T. & Rescorla, E. (2008). "The Transport Layer Security (TLS) Protocol Version 1.2." RFC 5246. IETF.

IEEE Standards Association. "IEEE OUI and Company ID Assignments."

Lyon, G. (2009). *Nmap Network Scanning.*

MITRE. "Common Platform Enumeration Dictionary."

NIST. (2011). "Common Platform Enumeration: Naming Specification Version 2.3." NIST IR 7695.

ONVIF. (2022). "ONVIF Core Specification Version 22.12."

Schulzrinne, H., Rao, A., Lanphier, R., Westerlund, M., & Stiemerling, M. (2016). "Real-Time Streaming Protocol Version 2.0." RFC 7826. IETF.

UPnP Forum. (2015). "UPnP Device Architecture 2.0."

Review Questions

1. A host has a Google OUI, ports 8008, 8009, and 8443, and an mDNS _googlecast._tcp advertisement. What identity claim is defensible, and what remains unresolved?
2. Why does the Breakwater nmap adapter use --version-intensity 2 in targeted service detection rather than relying on nmap's default intensity?
3. Explain why ONVIF carries high confidence yet should still be capped below certainty.
4. A MacBook exposes SSH and AirPlay. Why would an AirPlay-only rule be dangerous, and how does the Apple SSH guard reflect better evidence reasoning?
5. Compare TLS certificate inspection and JARM. What does each identify, and where can each mislead the analyst?
6. What information must travel with a CPE candidate before Chapter 3 should use it for vulnerability lookup?
7. Give one example of a device that should remain at family-level identity rather than model-level identity.

Cross-References

- **Chapter 1** supplies the host record, topology context, and discovery provenance.
- **Chapter 3** consumes CPE candidates, service versions, confidence, conflict notes, and admissibility judgments.
- **Chapter 4** uses bounded identity when constructing predictive attack paths.

- **Later chapters** may use certificate metadata, fleet grouping, and confidence signals, but those later uses do not change Chapter 2's scope.