Securing VoIP — Capture-Analysis Workbook

Sean Cheesman

About This Workbook

This workbook is the hands-on companion to *Securing VoIP at Every Layer* (Cheesman Press, 2026). It walks you through fifteen capture-analysis exercises drawn from the volume's 32 companion captures, plus three synthesis exercises that cross multiple captures. Each exercise has a defined scenario, a small set of tasks the reader actually performs in Wireshark, and an expected-findings block that documents what the analysis should reveal.

The exercises are self-paced. There is no scoring and no badge. The point is to build muscle memory for the analyst procedures that the book describes — so that when you open a real production capture, the filters, the menu paths, and the field-level reading are already familiar.

Prerequisites

- **Wireshark 4.x** installed. The exercises were written against Wireshark 4.6; earlier 4.x versions work, 3.x may have menu-path differences.
- **The Securing VoIP capture bundle.** Download `securing-voip-pcaps.zip` from seancheesman.com, extract anywhere convenient. The bundle's top-level folder is `securing_voip/` with five category subdirectories (`attacks/`, `case-files/`, `defenses/`, `signaling/`, `decrypt-demo/`).
- **The Securing VoIP Wireshark profile.** Download `Securing_VoIP_Wireshark_Profile.zip` from the same site. Install per the included `README_PROFILE.txt` — the workbook references filter shortcuts and color rules that this profile pre-loads. Exercises work without the profile, but you'll do more typing.
- **Optional:** Wireshark's GeoIP plugin (for the SIP-DDoS exercise) and OpenSSL CLI tools (for the capstone decryption exercise).

How to Use This Workbook

1. Read the **Scenario** — understand what you're looking at and why an analyst would care.
2. Work through the **Tasks** in order. Each task names a Wireshark filter, a menu path, or a calculation. The filter syntax matches the Securing VoIP profile's filter shortcuts; if you're not using the profile, type the filter manually.
3. Compare against the **Expected Findings** block. If your findings match, you've successfully replicated the analysis. If they don't, the most common causes are: wrong capture file open, dissector not engaged (Decode As → RTP/SRTP may be needed), or a typo in the filter.
4. Read the **Reflection** — the connection back to the chapter prose, real-world implications, and what to take with you to your own captures.

The Expected Findings appear immediately below each set of Tasks, separated by a horizontal rule and labeled. Self-paced learners can read the findings before attempting the tasks if that helps; the exercises are designed for either reading order.

Table of Contents

Chapter 1. Warm-Up: Bundle and Tooling Check

This warm-up confirms your tooling is configured correctly before you tackle the real exercises. If anything in the warm-up doesn't work, fix the setup before continuing — the exercises assume the warm-up steps work.

Scenario. You've extracted the Securing VoIP capture bundle and installed the Wireshark profile. Verify that Wireshark loads the captures, applies the profile's filter shortcuts, and dissects SIP correctly.

Tasks.

1. Open `securing_voip/signaling/signaling_via_chain_topology_leak.pcap` in Wireshark.
2. In the upper-left "Profile" indicator (or via Edit → Configuration Profiles), confirm "Securing VoIP" is the active profile. If not, select it.
3. Apply filter `sip` in the filter bar. The bar should turn green; you should see 2 SIP packets.
4. Click the first SIP packet. In the dissection pane, expand the **Session Initiation Protocol** layer. The request method should be `INVITE`.
5. Verify the custom columns from the profile are visible: between the standard **Length** column and the **Info** column, you should see **Delta**, **TLS Ver**, **TLS Cipher**, **Identity**, and **Auth** columns. They'll be empty for this capture (no TLS, no Identity header, no auth) but the column headers should be present.

Expected Findings. Two packets in the capture, both SIP. Packet 1 is the INVITE with three Via headers; Packet 2 is a 100 Trying response. Profile is "Securing VoIP" with the four custom security columns visible (empty for this particular capture). If the columns aren't present, the profile installation didn't take — re-check that the profile files are in

> the right `profiles/Securing VoIP/` directory for your OS and that you've selected the profile in Edit → Configuration Profiles.

If everything checks out, you're ready for the real exercises. Total time for warm-up: ~5 minutes.

Chapter 2. Section 1 — Reconnaissance & Signaling Attacks

2.1. Exercise 1: Identify the Scanner

Capture: `securing_voip/attacks/attacks_sipvicious_scan.pcap`

Scenario. Your SBC's syslog has flagged unusual SIP traffic from an external IP. You've captured a sample of the traffic for analysis. Confirm whether it's a scan, identify the tool if you can, and assess what the attacker has learned about your infrastructure so far.

Tasks.

1. Open the capture. Apply filter `sip`. Note the total packet count.
2. Examine the source and destination IPs. Is there a one-to-many pattern (single source, many destinations) or many-to-one?
3. Click any SIP request. Expand the SIP layer. Read the **User-Agent** header. Cross-reference against the attack-tool UA fingerprint table in the cheat sheet (or the Reconnaissance filters on Panel 1).
4. Filter `sip.User-Agent contains "friendly-scanner"`. Count matches.
5. Switch to filter `sip.Status-Code == 200`. Click any matching response and find the **Server** header. What software and version is the responding host running?

Expected Findings. 18 packets total: 12 SIP OPTIONS probes from `203.0.113.150` to consecutive destinations in `192.168.10.40-51`, plus 6 responses (mix of 200 OK and 404 Not Found from the targets that bothered to answer). Every probe carries `User-Agent: friendly-scanner` — the canonical SIPVicious svmap fingerprint. The 200 OK responses include `Server: Asterisk PBX 18.20.0` — the attacker now knows the

> target platform and version, and can pivot directly to a CVE database lookup. **Reflection:** this is the recon phase of an attack chain. The next probe an svmap-equipped attacker runs is svwar (extension enumeration) against the IPs that answered — see Exercise 2.

2.2. Exercise 2: Map the Extension Enumeration

Capture: `securing_voip/attacks/attacks_svwar_extension_enum.pcap`

Scenario. Following on from the svmap reconnaissance, the same source IP is now sending REGISTER requests to one of your PBXes. Determine what they're doing, identify which extensions on the PBX exist, and assess whether any have been compromised.

Tasks.

1. Open the capture. Apply filter `sip.Method == "REGISTER"`. Note the total REGISTER count and the source IP.
2. Sort by the **To** column (you may need to add it: Edit → Preferences → Columns → +Custom: `sip.to.user`). Look at the To-URI user values. Is there a pattern?
3. Filter `sip.Status-Code == 401`. These are the server's auth challenges. Count matches.
4. Filter `sip.Status-Code == 404`. These are the server's "no such extension" responses. Count matches.
5. Examine the 401 responses' source extensions. List which extensions returned 401 (i.e., exist on the server).
6. Check whether any REGISTER includes an **Authorization** header — has the attacker tried to actually authenticate, or just probed?

Expected Findings. 50 REGISTERs from `203.0.113.150`, walking extensions 1000 through 1049 sequentially. 9 of them return 401 Unauthorized (real extensions: 1003, 1007, 1011, 1020, 1024, 1031, 1035, 1042, 1048); the other 41 return 404 Not Found (non-existent). No REGISTER carries an Authorization header — the attacker hasn't tried to authenticate yet. The 401-vs-404 split itself is the enumeration channel. **Reflection:** the attacker now has a precise account list to feed into a credential-spray, Digest brute-force, or social-engineering campaign — without ever triggering account-lockout protection (which only activates on bad-password attempts, not on credential-less probes). The structural defense is to flatten the response semantics: respond 401 to every REGISTER regardless of whether the extension exists, deferring the existence check to the credential validation step. See Ch 2.2 §"Digest Authentication Pitfalls."

2.3. Exercise 3: Harvest the Digest Hash

Capture: `securing_voip/attacks/attacks_digest_auth_brute_force.pcap`

Scenario. An attacker on a coffee-shop Wi-Fi has captured a colleague's SIP REGISTER exchange. The colleague's phone uses Digest authentication over UDP. Extract the cryptographic material the attacker would feed into hashcat for offline cracking.

Tasks.

1. Open the capture. Apply filter `sip`. Step through all four packets in order.
2. Identify the four packets by type: which is the credential-less REGISTER, which is the server's 401 challenge, which is the authenticated REGISTER, which is the 200 OK?
3. Click the 401 challenge. Expand the SIP layer; find the **WWW-Authenticate** header.

Note the realm, nonce, algorithm, and qop values.

4. Click the authenticated REGISTER. Find the **Authorization** header. Note the username, response hash, cnonce, nc, and uri values.
5. Construct the hashcat input string for SIP-Digest (hashcat mode 11400):

   `$sip$<user><realm>*REGISTER*<uri>*<nonce>*<cnonce>*<nc>*<qop>*<response>*MD5`

6. The capture's password is `Welcome123!` (deliberate, for verification). Run the constructed hashcat string against a wordlist containing that password and confirm it cracks. (Optional — requires hashcat installed.)

Expected Findings. 4 packets total. Packet 1 is the credential-less REGISTER for user `alice`. Packet 2 is the 401 challenge with realm=`voice.example.com`, nonce=`abc123def456ghi789`, algorithm=MD5, qop=`auth`. Packet 3 is the authenticated REGISTER with the response hash computed against password `Welcome123!`. Packet 4 is the 200 OK. The complete hashcat input string is reconstructible from packets 2 and 3 alone — the attacker doesn't need anything else. **Reflection:** this is the structural argument for SIP-over-TLS. Every input to the password hash (except the password) is on the wire in cleartext. With TLS, the entire exchange is inside an encrypted channel and the attacker captures only ciphertext. Without TLS, password complexity has to do all the work — and "do all the work" against modern GPUs means 12+ characters with mixed case, numbers, and symbols. Anything shorter cracks in minutes. See Ch 2.1 §"SIP Digest Credential Brute-Forcing" and Ch 2.2 §"TLS for SIP Signaling."

Chapter 3. Section 2 — Media-Plane Analysis

3.1. Exercise 4: Reconstruct the Eavesdropped Audio

Capture: `securing_voip/attacks/attacks_rtp_eavesdrop_g711.pcap`

Scenario. A compromised SPAN port on the executive-floor switch was found mirroring voice-VLAN traffic to a third destination. This capture is a representative slice of what the SPAN port had been collecting. Reconstruct the call and assess what the attacker captured.

Tasks.

1. Open the capture. The first thing Wireshark will probably show is "UDP" rather than "RTP" — the heuristic dissector in the profile should engage, but if it doesn't, right-click any packet → Decode As → set the current value for the UDP port to RTP.
2. Apply filter `rtp`. Confirm packets are now dissected as RTP. Read the **Payload type** field: which codec?
3. Navigate Telephony → RTP → RTP Streams. The dialog lists every RTP stream Wireshark found.
4. Select both streams (Ctrl-click), click "Play Streams." Wireshark mixes the two streams into a single playback timeline.
5. Listen to the audio. (If you don't have audio output, click "Save Audio" to export as a `.au` file you can play later.)
6. Statistics → Packet Lengths with filter `rtp`. What's the size distribution? Single peak or bimodal?

Expected Findings. 200 packets total, all RTP. Payload type 0 = G.711 PCMU. Two streams (one each direction, distinct SSRCs). Wireshark's

> Play Streams reconstructs intelligible audio with no decryption step. Packet Lengths shows a single peak around 200 bytes (Ethernet + IP + UDP + 12-byte RTP header + 160-byte G.711 payload), confirming no SRTP overhead — this is pure plaintext RTP. **Reflection:** this capture is the proof that vanilla RTP has zero confidentiality. The bits on the wire **are** the audio. A bimodal length distribution at 200 / 214 bytes would have indicated SRTP coexisting with plain RTP (the 14 extra bytes = SRTP auth tag + index); the single peak here confirms no SRTP. See Ch 2.3 §"Plaintext RTP Eavesdropping" and Ch 2.4 §"SRTP."

3.2. Exercise 5: Extract the Credit Card

Capture: `securing_voip/attacks/attacks_dtmf_extraction_payment.pcap`

Scenario. This capture is from a payment-IVR call where the caller entered a 16-digit credit-card number via DTMF. The call traversed a misconfigured SBC that did not mask DTMF. Extract the card number from the capture.

Tasks.

1. Open the capture. Apply filter `rtp.p_type == 101`. This isolates the RFC 4733 telephone-event packets.
2. Count the matching packets. Each digit produces 5 events (per RFC 4733), so the count should be a multiple of 5.
3. Navigate Telephony → RTP → Stream Analysis. Select the caller-to-IVR stream. Click the **DTMF** tab. Wireshark decodes the digit sequence directly.
4. Alternative manual approach: click any telephone-event packet, expand the RTP layer, examine the 4-byte payload. Byte 0 is the digit code (0-9 for digits, 10 for *, 11 for #). Read the sequence by stepping through the events.

5. Note the PAN value the analysis reveals. (It's a documentation test number — 4111-2222-3333-4444 — not a Luhn-valid live card.)
6. As a sanity check: filter `rtp.p_type == 0` to isolate the audio packets. How many seconds of audio prompt + silence are in the capture before the DTMF events begin?

Expected Findings. 80 telephone-event packets (16 digits × 5 events each). DTMF tab shows the digit sequence `4 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4` = PAN `4111222233334444`. The audio packets include ~30 packets of IVR prompt audio before the DTMF starts and ~20 packets of additional audio after. Byte-0 of any telephone-event payload directly encodes the digit. **Reflection:** this capture is the threat model PCI-DSS DTMF masking exists to defeat. The masked-egress version of the same call is `defenses/defense_dtmf_masking_pci.pcap` — open that next to compare what the SBC's egress should have looked like (byte 0 stomped to 0x0F, volume zeroed). See Ch 2.3 §"DTMF Tone Extraction" and Ch 4.1 §"DTMF Masking and Suppression."

3.3. Exercise 6: Read the Master Key in Cleartext

Capture: `securing_voip/defenses/defense_sdes_inline_key.pcap`

Scenario. This capture is labeled "defense" but actually demonstrates the failure mode of SDES key exchange over cleartext SIP. Locate the SRTP master keys, decode them, and assess what an on-path observer could do with them.

Tasks.

1. Open the capture. Apply filter `sdp.media_attr contains "crypto"`. Identify the two packets that match.

2. Click the INVITE. Expand the SDP body. Find the `a=crypto:` attributes. How many are present? What crypto suites do they offer?
3. Copy the base64-encoded value after `inline:` (just one of them, e.g., the suite-1 offer).
4. Base64-decode the inline string. How many bytes do you get? Per RFC 4568, that should be 16 bytes of master key + 14 bytes of master salt = 30 bytes.

```
echo "<base64-inline-string>" | base64 -d | xxd
```

5. Click the 200 OK. Repeat: find the answerer's `a=crypto:` line. Decode their inline key.
6. Examine the transport. Is the SIP traffic on UDP/5060 (cleartext) or TCP/5061 (TLS-protected)?

Expected Findings. INVITE offers two crypto suites: `AES_CM_128_HMAC_SHA1_80` and `AES_CM_128_HMAC_SHA1_32`, each with its own inline key. 200 OK answers with the first suite and its own master key. All three base64 inline strings decode to 30 bytes (16-byte master key + 14-byte master salt) — the complete keying material an attacker needs to feed into srtp-decrypt and recover the audio. The SIP runs over cleartext UDP/5060 — the deliberate misconfiguration. **Reflection:** SDES is **encryption only if the signaling carrying it is also encrypted**. The compare-and-contrast capture is `decrypt-demo/tls_srtp_decrypt_demo.pcap` (Exercise 15) — same SDES key transport, but inside a TLS-protected SIPS channel, with the analyst's keylog file as the controlled-decryption channel. See Ch 2.4 §"SDES: The Convenient Trap."

Chapter 4. Section 3 — Network-Layer Attacks

4.1. Exercise 7: Detect the ARP Spoof

Capture: `securing_voip/attacks/attacks_arp_spoof_voice_vlan.pcap`

Scenario. A phone on the voice VLAN reported intermittent registration loss. You captured at the phone's switch port for analysis. Determine whether a MITM attack is in progress, identify the attacker's MAC, and confirm the impact on traffic flow.

Tasks.

1. Open the capture. Apply filter `arp`. Note the total ARP packet count and the time distribution.
2. Look for `arp.duplicate-address-detected` — Wireshark's automatic flag when an IP is claimed by multiple MACs.
3. Identify which two MACs claim `192.168.10.1` (the voice-VLAN gateway). One is legitimate, one is the attacker.
4. Look at the ARP opcode distribution: how many requests (opcode 1) vs replies (opcode 2)? A normal exchange is 1:1; a spoof shows many more replies than requests.
5. Find the first SIP REGISTER from the phone (filter `sip.Method == "REGISTER"`). Examine the Ethernet destination MAC. Is it the legitimate gateway MAC or the attacker's MAC?
6. Find the second SIP REGISTER (after the ARP burst). Examine the Ethernet destination MAC again. Has it changed?

Expected Findings. 24 packets total. Packets 1-2 are a legitimate ARP request/reply between the phone (`00:1b:21:11:22:33`) and the real gateway (`00:1c:0e:aa:bb:cc`). Packet 3 is the phone's first SIP REGISTER

> addressed to the real gateway MAC. Packets 4-23 are 20 unsolicited ARP replies from `00:50:56:ab:cd:ef` claiming to be `192.168.10.1`, alternating broadcast and targeted variants. Packet 24 is the next SIP REGISTER — same source/destination IPs, but the Ethernet destination MAC is now `00:50:56:ab:cd:ef`. The phone's ARP cache has been poisoned. **Reflection:** gratuitous ARP replies (no preceding request) are the signature. Dynamic ARP Inspection (DAI) on the switch drops these at line rate when paired with DHCP snooping. See Ch 2.5 §"ARP Spoofing" and the Cisco IOS DAI configuration pattern in the same chapter.

4.2. Exercise 8: Catch the VLAN-Hopping Probe

Capture: `securing_voip/attacks/attacks_vlan_hopping_voip_hopper.pcap`

Scenario. A small captured trace from an unfamiliar device plugged into a conference-room switch port. Determine what the device is, how it discovered the voice VLAN, and confirm whether it succeeded in reaching the voice VLAN.

Tasks.

1. Open the capture. There are only 5 packets total — small enough to read end-to-end.
2. Click packet 1. Identify the protocol. Expand the protocol layer. What information does it reveal?
3. Click packet 3. Look at the Ethernet II header. Is this an untagged or 802.1Q-tagged frame? If tagged, what VLAN ID? What's the priority (CoS) value?
4. Look at the SIP payload of packet 3. Find the User-Agent header. What tool produced this packet?
5. Click packet 5. Determine whether the voice VLAN accepted the probe. What's the response, and what does the **Server** header reveal about the responding system?

Expected Findings. Packet 1 is a CDP advertisement from `SW-FLOOR2-EDGE` on port `GigabitEthernet0/24`, with a **VoIP VLAN Reply** TLV (type 0x000E) revealing voice VLAN 200. Packet 3 is an 802.1Q-tagged frame on VLAN 200 with priority 5 (voice CoS), carrying a SIP OPTIONS to the call manager with `User-Agent: voiphopper/2.04`. Packet 5 is CUCM's 200 OK on the voice VLAN with `Server: Cisco-CUCM12.5`, confirming the hop succeeded. **Reflection:** the structural enablers are (a) CDP advertising voice VLAN to anything that listens and (b) the switch port accepting 802.1Q-tagged frames from an unauthorized device. 802.1X authentication on the access port closes both holes at once — the unauthenticated device never reaches the link layer, so neither CDP nor VLAN tagging matter. See Ch 2.5 §"VoIP Hopper / CDP-LLDP Exploitation."

Chapter 5. Section 4 — Fraud Patterns

5.1. Exercise 9: Recognize the IRSF Fingerprint

Capture: `securing_voip/attacks/attacks_irsf_full_chain.pcap`

Scenario. Your carrier has alerted you to unusual outbound traffic from your SBC. The capture is a 60-second window of the alert period. Determine whether the traffic is legitimate, identify the destinations, and quantify the fraud exposure.

Tasks.

1. Open the capture. Apply filter `sip.Method == "INVITE"`. Note the INVITE count.
2. Cycle through these per-country-code filters, counting INVITEs in each:

```
sip.Method == "INVITE" and sip.To contains "+232"   (Sierra Leone)
sip.Method == "INVITE" and sip.To contains "+53"    (Cuba)
sip.Method == "INVITE" and sip.To contains "+371"   (Latvia)
sip.Method == "INVITE" and sip.To contains "+252"   (Somalia)
sip.Method == "INVITE" and sip.To contains "+960"   (Maldives)
```

3. Pick any single call's Call-ID. Filter on it. Walk the call's full lifecycle (INVITE → 100 → 200 → ACK → BYE → 200). Note the duration between INVITE and BYE — that's the billable call length.
4. Open Telephony → SIP Statistics → Summary. What's the average call duration across all calls?
5. Reflect: how long would 50 calls of this average duration at carrier termination rates of ~$2/minute to these destinations cost the customer?

Expected Findings. 50 INVITEs, ~10 each to +232, +53, +371, +252, +960. Each call has full lifecycle through 200 OK / ACK / BYE. Average call duration ~60 seconds. The country-code distribution is the IRSF fingerprint — none of these are destinations a typical North American business calls. At ~$2/min for 60 seconds × 50 calls = ~$100 in the captured window; an attacker who maintains this volume for 8 hours (8 hours × 60 min × 50 concurrent ÷ 60 sec/call) generates several thousand dollars of carrier charges. **Reflection:** the structural defense is destination-country geo-blocking with a default-deny posture for high-risk country codes. Calls to a real Sierra Leone customer are a documented exception added to an allowlist — explicit business decision, not implicit policy gap. See Ch 3.1 §"IRSF Mechanics" and §"Geo-Blocking and Destination Whitelisting."

5.2. Exercise 10: Spot the Wangiri Cadence

Capture: `securing_voip/attacks/attacks_wangiri_pattern.pcap`

Scenario. Customer-service is reporting employees getting strange missed calls from international numbers. You captured 5 minutes of inbound SIP at the SBC during the reported window. Determine whether this is wangiri, identify the country-code pattern, and propose a detection rule.

Tasks.

1. Open the capture. Apply filter `sip.Method == "INVITE"`. How many inbound INVITEs?
2. Apply filter `sip.Method == "CANCEL"`. How many CANCELs? Compare against the INVITE count.
3. Pick any single call's Call-ID. Filter on it. Calculate the time delta between INVITE and

CANCEL. Is it greater than or less than 2 seconds?

4. Repeat the delta calculation for 4-5 different calls. Is the delta consistent?
5. Filter `sip.Method == "INVITE"` and examine the From-URI country codes. Are they consistent with the IRSF country codes from Exercise 9?
6. Look at the To-URIs (destination DIDs). Are they targeting one DID repeatedly, or many DIDs once each?

Expected Findings. 80 INVITEs, 80 CANCELs (1:1 correspondence). Time delta between any INVITE and its matching CANCEL is consistently 0.8-1.2 seconds — below normal user pickup latency. From-URIs span the same overseas country codes as IRSF: +232 (Sierra Leone), +252 (Somalia), +371 (Latvia), +960 (Maldives), +53 (Cuba). To-URIs target 80 different customer DIDs sequentially. **Detection rule:** alert when inbound INVITE → CANCEL on the same Call-ID has a time delta < 2 seconds AND the source CLI matches a high-risk country code. **Reflection:** wangiri lures victims to call back the displayed CLI; the callback connects to premium-rate revenue-share infrastructure. The structural enforcement is identical to IRSF geo-blocking but on the inbound side. See Ch 3.1 §"Wangiri and Callback Schemes."

Chapter 6. Section 5 — Management-Plane Compromise

6.1. Exercise 11: Reconstruct the AMI Session

Capture: `securing_voip/attacks/attacks_asterisk_ami_exposed.pcap`

Scenario. An attacker reached an Asterisk Manager Interface on TCP/5038 from outside the management network. You captured the session. Reconstruct what happened, identify the credentials used, and quantify the damage.

Tasks.

1. Open the capture. Apply filter `tcp.port == 5038`. Right-click any packet → Follow → TCP Stream. The AMI conversation now reads as plain ASCII dialog.
2. What's the server banner (first line)? What platform and version does it disclose?
3. Find the **Action: Login** frame. Read the **Username** and **Secret** values directly from the stream.
4. Find the **Action: Originate** frame. What's the channel being used (CallerID identity)? What's the destination extension?
5. Cross-reference the destination's country code against the IRSF/wangiri lists from Exercises 9 and 10. Recognize the destination?
6. Find the **OriginateResponse** event. Did the originate succeed (Response: Success)?

Expected Findings. Server banner: `Asterisk Call Manager/8.0.2` — cleartext platform fingerprint. Login: Username `admin` / Secret `AsteriskAMI2024!` — in plaintext on the wire. Originate channel `SIP/1042` (impersonating extension 1042), destination `+37127991122` (Latvia — wangiri country code). OriginateResponse confirms success.

> **Reflection:** AMI is not a "management protocol that needs authentication" — it's a **remote code execution primitive** (the Originate + dialplan System() combination shells out as the asterisk user). Treating it as RCE rather than auth changes how aggressively you protect it: `bindaddr = 127.0.0.1` in `manager.conf` removes it from the network entirely. See Ch 3.2 §"Asterisk Manager Interface and ARI Abuse."

6.2. Exercise 12: Extract the Provisioning Config

Capture: `securing_voip/attacks/attacks_tftp_provisioning_sniff.pcap`

Scenario. An attacker with passive access to the voice VLAN captured a phone's boot-time provisioning fetch. The provisioning server uses TFTP. Reconstruct the fetched config file and identify what credentials it exposes.

Tasks.

1. Open the capture. Apply filter `tftp`. Step through the packets.
2. Identify the filename in the Read Request (RRQ). What naming convention does it follow?
3. Right-click any TFTP packet → Follow → UDP Stream. Wireshark reassembles the multi-block file transfer into the original config.
4. Read the assembled XML. Identify **every** credential disclosed. Look for the SSH password, the SIP authentication password, and the phone's local PIN/password.
5. Note the call manager hostname and IP, plus any port configuration that reveals the SIP infrastructure.

Expected Findings. 7 packets total. RRQ for `SEP001b21112233.cnf.xml` (Cisco SEP-MAC convention). Three DATA blocks reassemble into the config XML. Credentials disclosed: `<sshPassword>Cisco12345</sshPassword>` (SSH access to the phone itself), `<authPassword>SIPaccess2024!</authPassword>` (SIP register credentials for extension 1042), `<phonePassword>987654</phonePassword>` (local phone-menu PIN). Plus call manager hostname `cucm-01.example.com`, IP `10.0.1.10`, SIP ports 5060/5061. **Reflection:** the attacker only needs to be present during a phone boot — power cycles, config refreshes, firmware updates, and network outages all trigger a re-fetch. Across hundreds of phones, the cumulative probability approaches 100% over time. HTTPS provisioning with mutual TLS closes this entire class of attack. See Ch 3.2 §"TFTP Sniffing" and §"Config-Server Hardening."

Chapter 7. Section 6 — Incident Response

7.1. Exercise 13: Investigate the Saturday Toll Fraud

Capture: `securing_voip/case-files/case_4-2-1_toll_fraud.pcap`

Scenario. You're the IR responder for a 60-attorney law firm. The FreePBX-based SBC was compromised on a Saturday morning while the office was empty; the carrier alerted you to $52K of fraudulent charges 6 hours into the attack. The pcap is the SBC's wire capture from 02:00 to 09:00. Reconstruct the attack timeline and identify the entry point.

Tasks.

1. Open the capture. Apply filter `sip.Method == "REGISTER" and ip.src == 203.0.113.42`. This isolates the attacker's traffic.
2. Sort the REGISTERs by To-URI. What's the pattern? Note the attacker's progression through the extension space.
3. Find the first 200 OK in response to one of the attacker's REGISTERs. Which extension was compromised? What's the timestamp?
4. Filter `sip.Method == "INVITE" and ip.src == 10.0.2.5` (the SBC) — these are the calls placed **after** the successful compromise. How many INVITEs in the capture? What's the time range?
5. Run the per-country-code filters from Exercise 9 against the attacker's INVITEs. What's the destination distribution?
6. Open Telephony → SIP Statistics → Summary. What's the total billable call duration?

Expected Findings. The attacker's REGISTERs walk ext 1001-1099 with ~50 attempts per extension (4,950+ total REGISTERs). The successful 200

> OK is for ext 1024 at 02:11 — the compromise. From 02:14 onward, INVITEs flow to +232, +371, and +882 destinations. Total billable duration is in the hundreds of minutes (the 890-minute figure from Ch 4.2 corresponds to the full 6-hour attack window). **Reflection:** the attacker's exhaustive REGISTER pattern is the signature — and is detectable by rate-limiting any single source IP to (say) 5 REGISTERs/min. The successful compromise hinged on extension 1024's voicemail PIN matching its extension number (the FreePBX default). Two layered defenses defeat this attack class: (1) credential-complexity enforcement at provisioning time, (2) rate-limiting at the SBC. See Ch 4.2 §"Case File 4.2-1: The Saturday Toll Fraud."

7.2. Exercise 14: Investigate the Registration Hijack

Capture: `securing_voip/case-files/case_4-2-2_reg_hijack.pcap`

Scenario. You're investigating a HIPAA-regulated healthcare clinic. Patients have reported speaking with someone who claimed to be their physician but collected sensitive information; the physicians have no record of those calls. You suspect registration hijack. Confirm the attack pattern and identify the affected extensions.

Tasks.

1. Open the capture. Apply filter `sip.Method == "REGISTER"`. Identify the affected extension by looking at the To-URI of REGISTERs and noting any extension with multiple source IPs.
2. For the affected extension, identify the legitimate phone's source IP and the attacker's source IP.
3. Both REGISTERs return 200 OK. Why does the call manager accept both? (Look at the deployment context — Cisco UCM with shared-line behavior enabled.)

4. Apply filter `sip.Method == "INVITE" and sip.To contains "<affected extension>"`. Find an incoming call. Examine how the UCM forks the INVITE to both registered Contacts.
5. For the forked INVITE, which endpoint sends the 200 OK first? Examine the time delta.
6. After the attacker's endpoint answers, filter `rtp and ip.dst == <attacker IP>` to see the RTP stream flowing to the attacker. Is there a return stream (attacker → patient)?

Expected Findings. Affected extension shows two simultaneous registrations: one from the clinic's internal network (legitimate phone), one from a DigitalOcean IP (`203.0.113.71`, the attacker). Both succeed because Cisco UCM's shared-line behavior permits multiple registrations per extension. Inbound INVITEs fork to both Contacts; the attacker's endpoint, being on a faster path from the DigitalOcean datacenter than from the clinic's internal network through their PBX, typically wins the race-condition for answering. RTP flows from patient → attacker; the attacker's return stream is muted or silence. **Reflection:** the dual-registration anomaly is the on-the-wire detection signal. Configure the call manager to allow only one registration per extension where shared-line behavior isn't operationally required; geo-fence registrations to expected ranges (alert on REGISTER from outside the clinic's normal geographies). The HIPAA exposure here is the **content** of the intercepted calls — PHI — which triggers the 60-day breach notification timer. See Ch 4.2 §"Case File 4.2-2: Registration Hijack and Silent Call Interception."

Chapter 8. Section 7 — Capstone: TLS+SRTP Decryption

8.1. Exercise 15: Decrypt a Full SIPS+SRTP Call

Capture: `securing_voip/decrypt-demo/tls_srtp_decrypt_demo.pcap` **Companion files:** `sslkeylog.txt`, `self_signed_ca.pem`, `server.crt`, `README.txt` (all in the `decrypt-demo/` directory)

Scenario. You have a TLS-protected SIPS call with SRTP-encrypted media, plus the session keys from the analyst's controlled decryption channel. Decrypt the capture end-to-end: read the SIP signaling, extract the SRTP master key from the now-readable SDP, decrypt the SRTP audio, play the call.

Tasks.

1. Open the capture **first without configuring decryption**. Apply filter `tls.handshake`. Confirm you see a TLS 1.3 handshake. Apply filter `tcp.port == 5061`. Examine the Application Data records — they're opaque ciphertext.
2. Now configure decryption. Edit → Preferences → Protocols → TLS → "(Pre)-Master-Secret log filename" → point at `decrypt-demo/sslkeylog.txt`. Click OK.
3. Re-apply filter `sip`. The SIPS messages should now appear as decrypted SIP. Step through: identify the REGISTER, INVITE, 200 OK, ACK, BYE.
4. Click the INVITE. Expand the SDP body. Find the `a=crypto:` line. Copy the base64 value after `inline:`.
5. Find the RTP streams. Apply filter `rtp`. Note the SSRCs (two of them).
6. Edit → Preferences → Protocols → SRTP → add an entry: paste the base64 inline key as the master key, enter the SSRCs. Click OK.
7. Apply filter `rtp`. The packets should now show as decrypted RTP. Telephony → RTP → RTP Streams → select both → Play Streams.

8. What do you hear?

Expected Findings. TLS 1.3 handshake on TCP/5061 between Alice (192.0.2.45) and Bob (192.0.2.46). Without decryption, all of it is opaque. With the keylog file, the SIP exchange decrypts: REGISTER → 200 OK; INVITE with SDP containing `a=crypto:1 AES_CM_128_HMAC_SHA1_80 inline:<base64-key>` → 200 OK with the answerer's own inline key → ACK; BYE → 200 OK. After configuring the SRTP master key in Wireshark's SRTP preferences, the RTP stream decrypts and Play Streams reconstructs a pair of pure tones (Alice's 600 Hz, Bob's 900 Hz — the synthetic demo payload). **Reflection:** this is the layered-defense story made concretely verifiable. SDES key transport is safe when wrapped in TLS — the SDES key never appears on the wire as plaintext. The analyst's keylog file is the controlled-decryption channel that lets authorized monitoring work without compromising on-the-wire security. Compare against `defenses/defense_sdes_inline_key.pcap` (Exercise 6) where the same SDES pattern runs on cleartext SIP — exact same SDP body, but visible to any passive observer. See Ch 2.4 §"SDES" and §"DTLS-SRTP."

Chapter 9. Synthesis Exercises

The three exercises in this section cross multiple captures and ask you to build the kind of multi-source analysis that real incident-response and detection-engineering work requires. There are no single right answers — the Expected Findings sketch one defensible approach, but your detection rules and conclusions may legitimately differ from the suggested ones.

9.1. Synthesis A: Build a SIPVicious Detection Ruleset

Captures used:

- `attacks/attacks_sipvicious_scan.pcap`
- `attacks/attacks_svwar_extension_enum.pcap`
- `attacks/attacks_digest_auth_brute_force.pcap`

Scenario. Your SIEM team wants a detection ruleset specifically for SIPVicious-pattern attacks. Build a ruleset that catches each phase of the SIPVicious attack chain (recon, enumeration, credential capture) with reasonable false-positive rates.

Tasks.

1. Review each of the three captures briefly. Identify the **one** signal in each capture that most reliably distinguishes the attacker from legitimate traffic.
2. For each signal, decide between a per-packet rule (catches every attacker packet, may alert-fatigue) and a threshold rule (catches the **pattern**, fewer alerts).
3. Document each rule in plain English, plus a Wireshark-display-filter equivalent that would let an analyst verify the rule's match against any future capture.
4. Identify what the ruleset **misses** — what would a smarter attacker change to evade it?

Suggested Approach.

Recon rule: User-Agent containing `friendly-scanner` (or the other known fork strings: `sundayddr`, `sipsak`). Per-packet rule. Filter equivalent: `sip.User-Agent contains "friendly-scanner"`. Misses: an attacker who scrubs the User-Agent string (trivially done by anyone reading their svmap docs).

Enumeration rule: REGISTERs from a single source IP at greater than 5/second OR sequential To-URIs across more than 20 REGISTERs in 60 seconds. Threshold rule. Filter equivalent: open Statistics → Conversations → SIP, sort by Packets per source. Misses: an attacker who slows the scan (10 REGISTERs per minute over hours).

Credential-capture rule: SIP Digest exchanges on UDP (cleartext) — every successful Digest auth on UDP is a hash exfiltration opportunity. Per-packet rule (informational). Filter equivalent: `sip.Authorization contains "Digest" and ip.proto == 17`. Misses: nothing — this is the structural failure that requires deploying TLS to fix, not a detection problem.

What's missed in aggregate: a sophisticated attacker uses residential proxies, randomizes User-Agent, slows the scan, and pivots through compromised devices in the customer's expected geography. None of the rules above catch that. The defense against that attacker is structural (TLS for signaling, strong passwords, account-lockout on credential failures) rather than signature-based.

9.2. Synthesis B: Compare SDES Cleartext to TLS-Wrapped SDES

Captures used:

- `defenses/defense_sdes_inline_key.pcap`
- `decrypt-demo/tls_srtp_decrypt_demo.pcap`

Scenario. These two captures use the same SRTP keying mechanism (SDES with `a=crypto:` inline keys) but at different security postures. Build a side-by-side comparison that demonstrates what TLS adds to the SDES picture.

Tasks.

1. Open both captures side-by-side (Wireshark supports multiple windows).
2. For each capture, locate the INVITE and find the `a=crypto:` inline key. In which capture is the key visible to a passive observer on the wire?
3. For each capture, examine the transport. What protocol carries the SIP? What transport carries the media?
4. Identify the **single difference** between the two captures that determines whether SDES is safe in each.
5. Build a one-paragraph explanation suitable for a non-technical stakeholder (executive, regulator) that explains why "SDES is safe when SIP is TLS-protected" — using these two captures as concrete evidence.

Suggested Approach.

In `defense_sdes_inline_key.pcap`, the SIP traffic runs on UDP/5060 — cleartext. The `a=crypto:` inline key is visible directly in the SDP body without any decryption step.

> In `decrypt-demo/tls_srtp_decrypt_demo.pcap`, the SIP traffic runs on TCP/5061 inside TLS 1.3. Without the keylog file, the entire SIP exchange is opaque ciphertext; the `a=crypto:` inline key is **invisible** on the wire.
>
> **The single difference:** whether SIP transits TLS. The SDES protocol mechanism is identical in both captures — same crypto-suite, same inline-key format, same key length. What changes is the **channel that delivers the SDES key from offerer to answerer**. TLS makes that channel confidential; without TLS, the channel is open.
>
> **Stakeholder paragraph:** "SDES key exchange for VoIP media encryption is only as secure as the signaling channel that carries the keys. Our SDES deployment uses SIPS (TLS-protected SIP) on every hop — the media-encryption keys are never visible on the network. The two captures demonstrate this: in the protected configuration, the keys never appear in any wire trace; in the misconfigured equivalent, they appear directly in the SDP body of the call-setup messages, decoding to 30 bytes of binary that an on-path observer can use to decrypt the entire call. Our infrastructure runs only the protected configuration; the misconfigured equivalent is shown here as the counter-example, not the production reality."

9.3. Synthesis C: Trace an IRSF Attack Chain End-to-End

Captures used:

- `attacks/attacks_sipvicious_scan.pcap`
- `attacks/attacks_svwar_extension_enum.pcap`
- `attacks/attacks_irsf_full_chain.pcap`

- `case-files/case_4-2-1_toll_fraud.pcap`

Scenario. You're explaining the IRSF attack chain to a new SOC analyst. Use the four captures to walk the full lifecycle from reconnaissance through monetization. Identify the detection opportunity at each phase and document the defensive control that would have closed it.

Tasks.

1. Map each of the four captures to a phase in the IRSF kill chain (recon → enumeration → credential capture/compromise → monetization).
2. For each phase, identify the **earliest** detection opportunity — the packet (or pattern) that would have surfaced the attack if monitoring had been in place.
3. For each phase, document the defensive control whose presence would have prevented the attack from progressing to the next phase.
4. Identify which capture represents the **post-compromise** monetization phase — the calls placed **after** a successful credential compromise — and compute what the customer's exposure was during the captured window.

Suggested Approach.

Phase 1 (Recon): `attacks_sipvicious_scan.pcap` — svmap OPTIONS sweep. Earliest detection: first OPTIONS packet with `User-Agent: friendly-scanner`. Defensive control: SBC ACL deny of inbound SIP from unknown sources OR User-Agent allowlist enforcement.

Phase 2 (Enumeration): `attacks_svwar_extension_enum.pcap` — sequential REGISTER probes. Earliest detection: 5+ REGISTERs from a single source in 60 seconds. Defensive control: SBC rate-limit on

REGISTERs per source IP.

Phase 3 (Credential compromise): `case_4-2-1_toll_fraud.pcap` packets 1-4950 — REGISTER brute-force, eventual success at ext 1024 with PIN matching the extension number. Earliest detection: 50+ REGISTER attempts on the same extension. Defensive control: account-lockout policy after N failed auth attempts + credential-complexity enforcement at provisioning time.

Phase 4 (Monetization): `case_4-2-1_toll_fraud.pcap` packets 4951+ AND `attacks_irsf_full_chain.pcap`. Earliest detection: first outbound INVITE to a high-risk-country-code destination. Defensive control: geo-blocking at the SBC with default-deny for IRSF country codes.

Captured-window exposure: `attacks_irsf_full_chain.pcap` covers 50 calls of ~60-second average duration to +$1-5/min destinations. ~$50-150 in the captured window; extrapolated across the full 6-hour attack of `case_4-2-1`, the documented $52K loss reconciles.

Key takeaway: every phase has both a detection opportunity and a structural defensive control. The attack succeeded against an organization that had neither at any phase. Defense in depth means deploying at least one (preferably both) at every phase.

Chapter 10. Going Further: Your Own Captures

The exercises in this workbook use synthetic captures that were built to demonstrate specific patterns cleanly. Real production captures are messier — more sources, more concurrent calls, more noise, less neatly-structured attack patterns. To apply what you've learned here to your own environment:

1. **Capture your own SIP signaling.** Run `tcpdump -i <interface> -w your-sip.pcap port 5060 or port 5061` on your SBC's interface during a normal business hour. You'll see a much larger capture (hundreds to thousands of packets) than any of the exercises here.
2. **Capture media.** For SRTP-enabled environments, set `SSLKEYLOGFILE` on your SIP server before capturing — the keys will export to that file and Wireshark can decrypt the resulting traffic per the procedure in Exercise 15.
3. **Apply the same filters.** Every filter in this workbook works against any SIP/RTP capture, not just the synthetic ones. Try running the SIPVicious detection filters from Synthesis A against your own captures — you may be surprised what shows up.
4. **Build the statistics workflows.** The recipes in Panel 2 of the cheat sheet (svwar enumeration via Conversations, plaintext-RTP-vs-SRTP via Packet Lengths, etc.) apply directly to your own captures. Try each one once against a known-good capture from your environment to confirm the workflow.
5. **Set up the Securing VoIP profile as your default for VoIP-security work.** Switch between it and Vol I's diagnostic profile via Edit → Configuration Profiles based on the task.

The captures in this volume are the demonstration. The skills they build are the deliverable.

Companion Materials

- *Securing VoIP at Every Layer* — the main volume; this workbook's exercises reference its chapter walkthroughs throughout.
- `securing-voip-pcaps.zip` — the capture bundle used by every exercise.
- `Securing_VoIP_Wireshark_Profile.zip` — the analyst profile referenced in the warm-up.
- `wireshark_securing_voip_cheatsheet.pdf` — printed quick-reference card.

All are available from the companion site at seancheesman.com.

Workbook revision 1.0 — May 2026 — Cheesman Press