Journal of Network and Computer Applications 254 (2026) 104546

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Research paper

Covert L2/L3 tunneling via SIP signaling on embedded hardware: Attack, evasion, and detection

Gorka Gorrochategui *, Unai Zulaika, Pablo Garaizar

University of Deusto, Bilbao, Spain

ARTICLE INFO

Dataset link: Additional Material - Source Code, Static Binaries, Videos, PCAPs, SIPp dataset (Original data)

Keywords:
Covert channels
SIP
VoIP
NIDS
RTC
Red team

ABSTRACT

Covert channels exploiting application-layer protocols represent a persistent threat in enterprise environments, particularly when targeting endpoints that remain outside the scope of endpoint detection and response solutions, such as embedded Voice over Internet Protocol (VoIP) phones. Existing SIP-based covert channels either embed low-bandwidth steganographic data within call-lifecycle signaling messages (Mazurczyk and Szczypiorski, 2008; Mehić et al., 2014; Tsiatsikas et al., 2015), or achieve higher capacity by requiring active Real-time Transport Protocol (RTP) media streams (Schmidt et al., 2018; Saenger et al., 2020). However, to our knowledge, no prior work implements continuous Layer 2/Layer 3 tunneling over standalone SIP signaling outside of call contexts, nor validates such a channel on real embedded hardware against multiple network intrusion detection systems. We present a covert channel that encapsulates arbitrary Ethernet frames and IP packets within SIP OPTIONS messages, which are lightweight requests typically used for NAT keep-alive and capability probing, without establishing any voice call. A portable proof-of-concept with TUN/TAP virtual interfaces and ChaCha20 symmetric encryption has been implemented and deployed on two ARM-based IP phones from different vendors, showing that even resource-constrained embedded devices can serve as covert network egress nodes. Empirical evasion testing against Suricata, Snort 3, Zeek, and a FortiGate firewall with full Unified Threat Management (UTM) services reveals that the channel achieves a secure operating bandwidth of 128 kbit/s without triggering alerts on any of the tested platforms. Analysis of the tested rulesets and parsers reveals the absence of entropy-based or header-length anomaly rules for Session Initiation Protocol (SIP), accounting for the detection gap. To address this, we implement a lightweight statistical detection mechanism that analyzes non-standard SIP header entropy, showing that the covert channel can be identified through targeted analysis of header content. The gap is not inherent to SIP: current rulesets inspect syntax but not header content, length, or entropy. Adding any of these features is sufficient to detect the channel, as shown by the Zeek script and Suricata 8 rule provided as supplementary material. A single entropy threshold can itself be evaded through low-entropy padding. We therefore show that multi-feature and anomaly-based detectors recover identification of the channel where the scalar test fails. We also contribute a synthetic benign SIP corpus, generated with SIPp to emulate multiple devices and signaling scenarios, released as supplementary material for evaluating false positives beyond the single legacy dataset previously available.

1. Introduction

Voice over IP endpoints such as SIP-based IP phones represent an insufficiently characterized attack surface in enterprise networks. These devices run embedded GNU/Linux distributions with proprietary vendor firmware, operate without host-level monitoring agents, and fall entirely outside the scope of Endpoint Detection and Response (EDR) solutions and Security Information and Event Management (SIEM) correlation. Enterprise firewalls and SBC are routinely configured to permit their SIP signaling traffic for legitimate business communications, creating conditions in which protocol-compliant covert channels can operate undetected by Network Intrusion Detection System (NIDS) and Deep Packet Inspection (DPI) engines.

Several prior works have explored covert communication within the VoIP ecosystem. Mazurczyk and Szczypiorski (2008) performed a theoretical analysis of steganographic techniques applicable to SIP header fields and tokens during the signaling phase of a VoIP call, estimating a capacity of approximately 2.4 kb per call direction. Mehić et al. (2014) studied the maximum rate of SIP messages injectable during an established call without triggering Snort IDS alerts. Tsiatsikas

* Corresponding author.
E-mail addresses: gorka.gorrochategui@deusto.es (G. Gorrochategui), unai.zulaika@deusto.es (U. Zulaika), garaizar@deusto.es (P. Garaizar).

https://doi.org/10.1016/j.jnca.2026.104546
Received 13 April 2026; Received in revised form 11 June 2026; Accepted 6 July 2026
Available online 15 July 2026

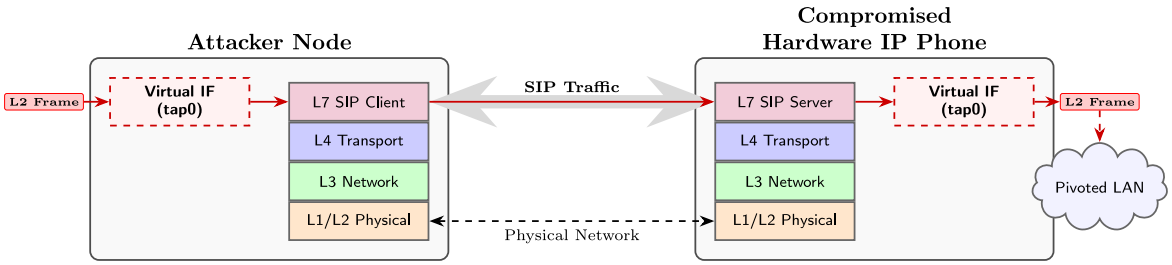

Fig. 1. Architecture of the SIP covert channel.

et al. (2015) demonstrated a botnet command and control channel embedded in SDP fields carried within SIP messages. At higher bandwidth, Schmidt et al. (2018) and Saenger et al. (2020) achieved data tunneling through active RTP media streams using silence suppression, with the latter providing a virtual network interface for TCP/IP tunneling, originally proposed in a privacy-enhancing context. From a taxonomic perspective, Wendzel et al. (2015) categorized network covert channel techniques into hiding patterns, providing a framework to assess the novelty of proposed methods. Within this taxonomy, the approach presented in this paper corresponds to the *Add Redundancy* pattern applied to SIP signaling. None of the reviewed approaches implements continuous Layer 2/Layer 3 tunneling over standalone SIP signaling outside of call contexts, nor validates such a channel on real embedded hardware against multiple intrusion detection platforms.

This paper addresses that gap by encapsulating arbitrary Ethernet frames and IP packets within SIP `OPTIONS` messages, as illustrated in Fig. 1. The `OPTIONS` method is a lightweight, asynchronous request typically used for Network Address Translation (NAT) keep-alive and capability probing (Rosenberg et al., 2002). It does not require media negotiation and does not establish a call dialog. By exploiting these properties, the proposed channel generates traffic that is structurally indistinguishable from legitimate SIP infrastructure activity. To validate this approach, a portable, statically compiled, user-space Proof of Concept (PoC) has been implemented and deployed on a real ARM-based IP phone.

The main contributions of this work are:

- A SIP `OPTIONS`-based covert channel that performs complete L2/L3 tunneling without a call context, encapsulating both Ethernet frames and IP packets.
- A portable, statically linked proof-of-concept[1] using Linux TUN/TAP interfaces and ChaCha20 symmetric encryption, validated on real ARM-based IP phones (Yealink T19P E2 and Grandstream GXP1630), demonstrating that resource-constrained embedded VoIP devices can serve as covert network egress nodes for L2 tunneling.
- An empirical evasion assessment against four security platforms spanning three detection paradigms: signature-based engines (Suricata with ET Open ruleset and Snort 3 with Cisco Talos Registered rules, both with SIP application-layer inspection enabled), metadata-oriented network monitoring (Zeek with SIP protocol analyzer), and a commercial Fortinet firewall with full UTM services.
- A practical entropy-based detection mechanism implemented as a Zeek script that analyzes SIP header payload entropy on a per-endpoint basis, showing that the covert channel can be identified through statistical analysis of header content. This detector can be evaded with low-entropy padding. We therefore show that multi-feature behavioral profiling and a one-class anomaly detector restore identification of the channel where the scalar entropy test fails. We also build and release a synthetic benign SIP corpus generated with SIPp, emulating multiple devices and signaling scenarios, with legitimate non-standard headers, to evaluate false positives beyond the single legacy dataset available. On this corpus, the multi-feature detector raises an order of magnitude fewer false positives than the scalar rule.

[1] The source code and other tools are provided as supplementary material.

2. Related work

The use of application-layer protocols to tunnel IP traffic for offensive purposes is well established. DNS tunneling (e.g., IODINE Ekman and Andersson, 2006–2023), HTTP-based tunneling, and ICMP or UDP encapsulation are common techniques in penetration testing and Red Team operations. However, these carriers have received extensive scrutiny and are increasingly covered by signature-based detection rules in modern NIDS.

Within the VoIP domain, several works have explored covert channels carried over RTP media streams. Open-source tools such as STEGOSIP (Pinna, 2011) have been developed to create IP tunnels using RTP media payload as a carrier. However, this tool, like other media-based channels, fundamentally depends on an active voice call with RTP media flow to operate. Similarly, Schmidt et al. (2018) proposed embedding covert data in fake RTP packets generated during silence intervals by exploiting Voice Activity Detection (VAD), achieving a practical tradeoff between steganographic bandwidth and stealthiness. Building on that approach, Saenger et al. (2020) developed a framework that exposes a virtual network interface for tunneling TCP/IP traffic through VAD-injected RTP packets, supporting full-duplex communication, originally proposed in a privacy-enhancing context. Their work is the closest to ours in terms of tunneling capability, but it fundamentally depends on an active voice call with RTP media flow. All RTP-based approaches share this constraint: they require an ongoing media session, which implies call establishment, generates Call Detail Records (CDRs), and is subject to call-duration monitoring and telephony fraud detection systems.

A smaller body of work has investigated covert channels within SIP signaling itself, independently of the media plane. Mazurczyk and Szczypiorski (2008) performed a theoretical analysis of steganographic capacity in SIP and Session Description Protocol (SDP) fields during the signaling phase of a VoIP call, identifying exploitable tokens such as `branch`, `tag`, `Call-ID`, and SDP session descriptors. They estimated a total capacity of approximately 2.4 kb per call direction across five signaling messages, and noted that `OPTIONS` messages could be intentionally invoked during the conversation phase to increase bandwidth. However, their contribution is analytical: no implementation, no tunneling, and no detection evaluation were provided. Their `OPTIONS` remark refers to an extra message sent within an active call to enlarge the same token-level embedding. The present work uses standalone `OPTIONS` requests, outside any call dialog, as the sole carrier of a full L2/L3 tunnel. Mehić et al. (2014) studied the maximum number of SIP messages that can be injected during an established VoIP call without triggering Snort IDS alerts, calculating the resulting steganographic bandwidth. Their channel also operates within the context of an active call and is limited to message-level data transfer. Tsiatsikas et al. (2015) demonstrated a botnet command and control channel that

encodes instructions in SDP fields (`sess-id`, `sess-version`, and `a=ptime`) carried within SIP messages. While their approach does not strictly require active media, it uses SDP bodies typically associated with call establishment (INVITE/200 OK exchanges) and is limited to low-bandwidth discrete C2 commands rather than arbitrary traffic tunneling.

From a taxonomic perspective, Wendzel et al. (2015) introduced a pattern-based classification of network covert channel techniques, enabling systematic comparison and novelty assessment of hiding methods. Within this framework, the approach presented in this paper corresponds to the *Add Redundancy* pattern: a custom SIP header is introduced to carry the covert payload, using the extensibility of the SIP protocol as defined in RFC 3261 (Rosenberg et al., 2002). The same authors later proposed a unified description method (Wendzel et al., 2016) for network information hiding techniques, which we adopt to position our work within the broader landscape. For a comprehensive survey of VoIP steganography methods and their detection, we refer the reader to Mazurczyk (2013).

Detection research has moved in parallel. Zillien and Wendzel (2023) revisit the two most widely used detection methods in the literature, ϵ-similarity and the compressibility score, and compare them against two machine-learning detectors of recent years, SnapCatch and GAS. All four assume a timing channel and operate on inter-packet intervals, so none of them transfers to storage channels that carry payload inside header fields. For storage channels of the *Add Redundancy* pattern, the useful signal is in the content and length of the anomalous header, not in its timing, which is the regime our detection script targets (Section 6.2). Wendzel et al. (2025) later add two new categories to the taxonomy, history and amplified channels, that point to legitimate traffic rather than carrying the payload themselves. Our work sits outside both: the payload is embedded directly in a non-standard SIP header.

Detection work aimed specifically at the SIP control plane is comparatively sparse. The closest is Koziak et al. (2021), who extend Zeek to flag covert transmission by tracking how often the values of standard SIP headers such as `Call-ID` and `CSeq` change, a behavioral, frequency-based signal. Sattolo and Jaskolka (2020) evaluate statistical complexity tests for storage channels in general. More recent steganalysis uses machine and deep learning, but works at the media or general-network layer rather than on signaling: Moghadasi and Dehghani (2026) detect codec-domain steganography in G.729 audio, and Joseph et al. (2023) classify covert flows in conferencing media traffic. None of these inspects the content of individual SIP headers. Our detector (Section 6.2) is complementary: rather than header-change frequency or media-stream statistics, it keys on the name, length, and byte entropy of each header, catching a single anomalous non-standard header in one message.

Recent construction work has stayed on the media plane. Alishavandi and Fakhredanesh proposed MKIPS (Alishavandi and Fakhredanesh, 2021), a storage channel that hides data in the Master Key Identifier field of SRTP packets at about 400 bit/s. Like other media-plane methods, it presupposes an established encrypted call and embeds in the media stream, not in signaling. Our channel instead carries a complete L2/L3 tunnel inside standalone `OPTIONS` requests, with neither a call nor a media stream. The recent surveys of Wu et al. (2021) and Khadse and Dakhane (2025) catalog these VoIP and network covert channels, but neither goes beyond token-level SIP steganography for short messages. None of the recent construction works was tested against a production NIDS or validated on real embedded hardware. We address both here, on IP phones from two vendors, against Suricata, Snort 3, Zeek, and FortiGate.

Table 1 summarizes the key differences between the proposed approach and prior work along the dimensions suggested for systematic comparison of SIP/VoIP covert channels.

As Table 1 shows, prior work in SIP-based covert channels is limited to theoretical capacity analyses, low-bandwidth steganographic embedding within call-lifecycle messages, or discrete command transfer for botnet control. Approaches that achieve practical tunneling bandwidth rely on active RTP media streams. SIP/SDP-based covert channels, RTP-based tunnels, and virtual-interface VoIP tunnels have all been studied previously. To our knowledge, the specific combination demonstrated here has not been shown before: the present work combines complete L2/L3 tunneling over standalone SIP signaling, with no dependency on call establishment, validated against four detection platforms covering signature-based, metadata-oriented, and commercial UTM approaches, and deployed on real embedded hardware.

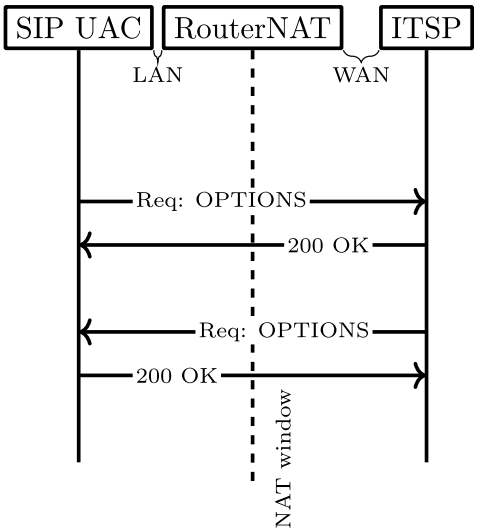

Fig. 2. SIP NAT window.

3. Fundamentals: SIP and TUN/TAP in the Linux kernel

3.1. Overview of SIP

SIP is an application-layer protocol used to establish, modify, and terminate multimedia sessions. Among its available methods, the `OPTIONS` method provides a lightweight, asynchronous request/response exchange used for capability probing and NAT session maintenance. This method is typically not associated with call establishment and rarely triggers fraud detection mechanisms or anomalies in CDRs.

The architectural characteristics of the `OPTIONS` method render it particularly suitable for implementing covert channels. It does not require media negotiation, eliminating the need for additional RTP ports, and can be transmitted outside existing SIP dialogs as a standalone transaction that creates no persistent session state (Rosenberg et al., 2002). This allows it to traverse corporate firewalls where SIP signaling is allowed. Because no call is established, detection mechanisms based on CDRs analysis are rendered ineffective, as no anomalous call durations or specific destinations are generated for monitoring by telephony fraud detection systems. The generated traffic also blends with legitimate network activity, as SIP `OPTIONS` messages produce regular and predictable patterns actively employed to maintain the NAT window open. Fig. 2 illustrates this behavior.

An additional property of SIP that this work employs is its header extensibility. RFC 3261 permits user agents to include custom header fields in any SIP message; proxies are required to forward unrecognized headers unchanged, and receiving user agents are expected to discard headers they do not recognize (Rosenberg et al., 2002). This mechanism provides a standards-compliant means of embedding arbitrary payloads within `OPTIONS` messages without violating protocol syntax, as detailed in Section 4.

Table 1
Comparison of VoIP and SIP-based covert channel approaches.

Work	Carrier	Active call	Channel type	Reported bandwidth	NIDS evaluation	Evasion result
Mazurczyk and Szczypiorski (2008)	SIP/SDP fields	Yes[a]	Header steg.	~2.4 kb/call	None	N/A
Mehić et al. (2014)	SIP headers	Yes	Header steg.	0.46–1.86 Mb/s[b]	Snort	Yes[c]
Tsiatsikas et al. (2015)	SDP fields	Partial[d]	Payload steg.	Not reported	None	N/A
Schmidt et al. (2018)	RTP (VAD)	Yes	RTP payload tunnel	0.38–10.7 kb/s[e]	None	N/A
Saenger et al. (2020)	RTP (VAD)	Yes	RTP network tunnel	~19.2 kb/s	None	N/A
Alishavandi and Fakhredanesh (2021)	SRTP MKI field	Yes	Protocol steg.	~400 bit/s	None	N/A
This work	**SIP OPTIONS**	No	**SIP signal. tunnel**	**128 kb/s**	**Suricata, Snort 3, Zeek, FortiGate**	**Yes (128 kb/s)**

[a] Theoretical analysis; no implementation provided.
[b] Theoretical bandwidth from max. SIP/SDP payload (~58 kB per RFC 3261 §18.1.1) at 1–4 ms g/s.
[c] Evasion up to 4 SIP messages/s; higher rates triggered Snort anomaly alerts.
[d] Uses SDP in INVITE/200 OK for session establishment; does not require active media
[e] Range across codec and injection parameter configurations (Schmidt et al., 2018).

3.2. User-space networking via TUN/TAP

The Linux kernel TUN/TAP interfaces allow user-space applications to read and inject packets as if they originated from a conventional network interface, providing complete control over L2/L3 traffic without requiring application-specific kernel modules. TUN/TAP has been available since Linux kernel version 2.4, released in 2001 (Krasnyansky, 1999–2000).

The PoC employs three capabilities of this subsystem: binding with the TUN/TAP interface for bidirectional frame handling, non-blocking multiplexed I/O between the kernel and the SIP encapsulation component, and routing and bridging support for reaching arbitrary internal hosts.

This design allows forwarding complete IP traffic (ICMP, TCP, UDP, mDNS, SSH, among others) through the SIP covert channel, enabling an attacker to reach arbitrary hosts and operate directly at Layer 2 on the target network segment, including link-local traffic such as ARP and mDNS.

4. PoC design

4.1. Threat model

The covert channel assumes an attacker who has obtained remote code execution on an enterprise IP phone and can deploy a statically linked user-space binary on it. The target device must run a Linux kernel with TUN/TAP support and must not be subject to host-level monitoring. The attacker also controls a remote endpoint running the same software, reachable via a network path that permits SIP traffic. No assumptions are made about the presence or absence of NIDS; evasion capability is evaluated empirically in Section 5. Table 3 demonstrates that these prerequisites, namely documented remote code execution vulnerabilities and TUN/TAP support, are met by IP phones from at least three major vendors.

4.2. Software architecture

The software component architecture is built upon a virtual network interface using the Linux TUN/TAP driver, together with an encapsulation module that packages frames within SIP OPTIONS messages. It uses the PJSIP protocol stack exclusively as a transport mechanism and incorporates lightweight symmetric encryption based on ChaCha20 (Bernstein, 2008). An I/O loop is implemented to multiplex traffic between the SIP layer and the kernel, ensuring full-duplex communication.

4.3. Encapsulation flow

Each outgoing IP packet from the virtual TUN/TAP interface is processed through the following flow:

1. Reading the packet in user space as sent by the kernel.
2. Encrypting the packet using ChaCha20, followed by Base64 encoding (required because SIP is a text-based protocol as per RFC 3261).
3. Embedding the encoded payload within a custom SIP header.
4. Transmitting the packet via `SIP OPTIONS` method to the remote end-point.

The encapsulation process can be formally expressed as follows:

$$P' = \mathcal{E}_{ChaCha20}(P, k)$$

$$M_{sip} = \text{Header} \parallel \text{Base64}(P')$$

where P represents the raw packet data, k is the Pre-Shared Key (PSK), P' is the encrypted payload, and M_{sip} is the final SIP message containing the encapsulated traffic.

The reverse process (reception and decapsulation of data) involves:

1. Parsing incoming SIP `OPTIONS` requests.
2. Extraction of the custom header payload.
3. Base64 decoding and ChaCha20 decryption.
4. Injection of the reconstructed frame into the TUN/TAP interface for kernel routing.

Algorithm 1 details the encapsulation process that takes place when traffic from the TUN/TAP interface needs to be sent through the SIP channel.

Algorithm 1 SIP Encapsulation and Transmission.

Require: Packet data pkt, length len
Require: Endpoint ep, destination dst
Require: Pre-shared key PSK
Ensure: Packet transmitted via SIP OPTIONS
1: $enc \leftarrow \textsc{ChaCha20}(pkt, len, PSK)$
2: $b64 \leftarrow \textsc{Base64}(enc)$ ▷ $\approx \lceil len \times 4/3 \rceil$ bytes
3: $req \leftarrow \textsc{SipOptions}(ep, dst)$
4: $hdr \leftarrow \textsc{Header}(\text{“Defined”}, b64)$
5: $\textsc{AddHeader}(req, hdr)$
6: $tsx \leftarrow \textsc{NewTransaction}(req)$
7: $\textsc{Send}(tsx)$

Note that ChaCha20, being a stream cipher, preserves the original packet length after encryption. Consequently, the length len is not transmitted: the receiver recovers it directly from the size of the Base64-decoded payload, which equals the ciphertext and hence the

original plaintext length. This is why *len* is an input to encryption in Algorithm 1 but is not needed for decryption in Algorithm 2. However, the subsequent Base64 encoding expands the payload by approximately 33%, a factor that directly impacts the effective Maximum Transmission Unit (MTU).

The decapsulation process, shown in Algorithm 2, handles incoming SIP OPTIONS requests and extracts the tunneled traffic. Upon reception, the request method is verified and the presence of the custom `Defined` header is validated; its absence indicates a legitimate SIP OPTIONS probe (for example, a NAT keep-alive) or a malformed packet, in which case a standard 200 OK response is returned without further processing. The presence of this custom header is the only distinguishing factor between covert messages and ordinary SIP signaling. When the header is present, the payload is Base64-decoded and ChaCha20-decrypted using the PSK, and the resulting frame is written to the TUN/TAP interface for kernel routing. A 200 OK response is returned in all cases, remaining indistinguishable from a standard OPTIONS reply.

Algorithm 2 SIP Decapsulation and Injection.

Require: Received SIP request $rdata$
Require: TUN/TAP write pipe $pipe_{from}$
Require: Pre-shared key PSK
Ensure: Decapsulated packet injected into kernel
$header \leftarrow$ FIND-HEADER($rdata$, "Defined")
if $header =$ NULL **then**
  RESPOND($rdata$, 200, "Ok")
  return
end if
$payload \leftarrow$ EXTRACT-HEADER-VALUE($header$)
$decoded \leftarrow$ BASE64-DECODE($payload$)
$packet \leftarrow$ CHACHA20-DECRYPT($decoded, PSK$)
write($pipe_{from}, packet$, len($packet$)) ▷ Send to main loop
RESPOND($rdata$, 200, "OK")

4.4. Cross-compilation for embedded systems

To ensure compatibility with ARM-based IP phones running minimalist Linux distributions, the PoC employs a static compilation strategy that eliminates dynamic library dependencies. Specifically, the PJSIP library is statically linked against uClibc to reduce the executable's footprint, adhering to the constraints of a Buildroot-type minimalist root filesystem. This cross-compilation process generates a self-contained binary targeting the ARM EABI5 architecture, as demonstrated by the following file signature:

```
$ file unicovertchannel
unicovertchannel: ELF 32-bit LSB executable,
ARM, EABI5 version 1 (SYSV), statically linked,
for GNU/Linux 3.2.0, stripped
```

4.5. Lightweight encryption and obfuscation

Entry-level IP phones have tight storage and memory budgets, so linking against OpenSSL or similar crypto libraries is not an option: the resulting static binary would not fit in the target firmware. Asymmetric schemes and full block-cipher implementations were therefore ruled out in favor of ChaCha20, a stream cipher with a small code footprint and a simple implementation in standard C that compiles in directly. Its ciphertext exhibits high entropy, which hinders pattern-based detection even after Base64 encoding.

No lossless compression is applied before encryption. The tunneled payloads are arbitrary network frames, much of which is already encrypted or compressed by higher-layer protocols and is therefore close to incompressible. A generic compressor would yield negligible savings while adding code size and CPU cost on a resource-constrained device. Since the channel optimizes for stealth rather than maximal throughput, this stage is omitted.

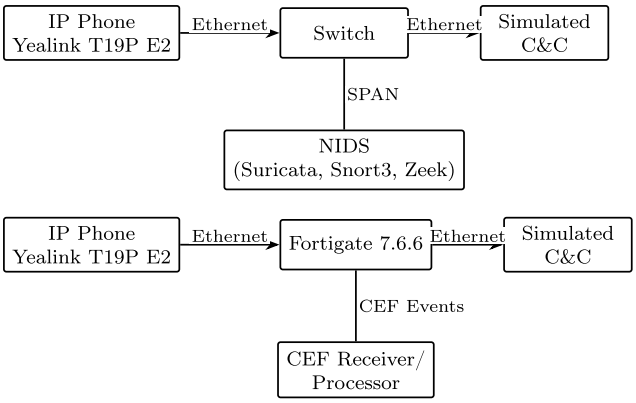

Fig. 3. Laboratory setup.

Table 2
Testbed components and configuration.

Component	Version	Configuration
Yealink T19P E2	FW 53.84.0.15	Target device (ARM EABI5)
Grandstream GXP1630	FW 1.0.4.132	Second target device (ARM)
Suricata	8.0.4	ET Open ruleset, SIP decoder enabled
Snort 3	3.1.11.0	Talos Registered ruleset (retrieved 15 Mar 2026)
Zeek	8.1.2	SIP analyzer, default config
FortiWifi 40F	FortiOS 7.6.6 (build 3652)	App Control (SIP), IPS `high_security` profile

5. Evaluation

We deployed a controlled experimental environment to empirically validate the effectiveness and stealth capabilities of the proposed SIP-based covert channel. This section details the testbed architecture, the specific components used, and the results obtained regarding detection evasion and operational performance.

5.1. Test scenario

The network topology implemented for the system evaluation is schematically illustrated in Fig. 3. To ensure reproducibility and validity of the evasion tests, specific hardware and software versions were selected, as summarized in Table 2.

The compromised end-device is a Yealink T19P E2 IP phone. The security monitoring infrastructure comprises four platforms covering the main defensive approaches in use today: signature-based (Suricata with the SIP application-layer decoder explicitly enabled, and Snort 3 with its native SIP inspector), metadata-oriented monitoring (Zeek with its built-in SIP protocol analyzer), and perimeter security (FortiWifi 40F with Application Control restricted to SIP traffic and Intrusion Prevention System (IPS) configured with the `high_security` profile to simulate a hardened enterprise environment).

5.2. Real-world validation

5.2.1. Initial access

To validate the system in a representative hardware environment, initial access to the Yealink T19P E2 terminal was secured by exploiting CVE-2023-43959 (National Vulnerability Database, NIST, 2023), a high-severity (CVSS 8.8) OS command injection vulnerability permitting authenticated arbitrary code execution via the device's diagnostic interface. Following standard post-exploitation methodology, the deployment sequence involved establishing a remote shell via the device's native Busybox `netcat` utility, followed by the retrieval and execution of the covert channel binary pre-compiled for the target ARM EABI5 architecture.

Table 3
Portability assessment across major IP phone vendors.

Vendor	Architecture	TUN/TAP support	Known RCE
Yealink	ARM	Validated[a]	CVE-2023-43959 (National Vulnerability Database, NIST, 2023)
Snom	ARM/MIPS	Documented (Snom Technology, 2024)	Advisory 20150113-0 (SEC Consult Vulnerability Lab, 2015)
Grandstream	ARM	Validated[a]	CVE-2020-5738 (National Vulnerability Database, 2020), CVE-2026-2329 (Fewer, 2026)

[a] Directly validated through PoC deployment in this work.

5.2.2. Post-compromise covert channel deployment

Once initial access was obtained, the PoC was deployed on the device, establishing a pivot point into the simulated corporate network. The covert channel itself is conceptually independent from the specific vulnerability used for initial access: it requires only a Linux kernel with TUN/TAP support and the ability to execute a statically linked user-space binary. These conditions are met by IP phones from multiple manufacturers widely deployed in enterprise environments, as summarized in Table 3.

All three vendors ship devices running embedded Linux with OpenVPN client support, which implies that the kernel is compiled with TUN/TAP enabled. TUN/TAP availability was directly confirmed through PoC deployment on both the Yealink T19P E2 and the Grandstream GXP-1630. Snom support is confirmed by vendor documentation. All three also have documented remote code execution vulnerabilities that could serve as initial access vectors. The channel is therefore not tied to a single device or CVE: any embedded Linux IP phone with TUN/TAP support and a usable remote code execution vector is a viable host.

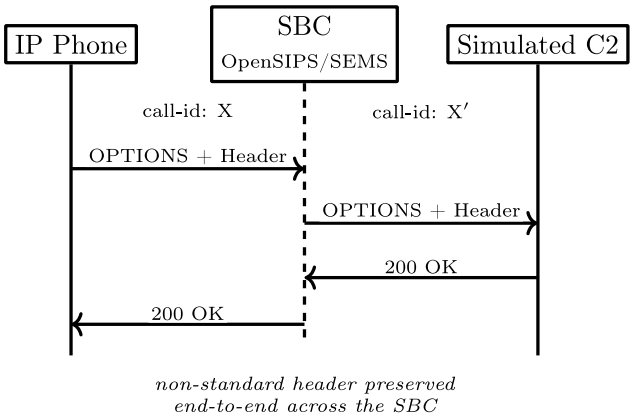

Fig. 4. Covert channel survival across an SBC: with OpenSIPS topohiding and SEMS B2BUA in default configuration (no header filtering), the non-standard SIP header inside OPTIONS is preserved end-to-end.

5.2.3. SBC traversal

To evaluate channel survivability when traversing a Session Border Controller, the traffic was routed through two representative SBCs, covering both intermediary classes. The first was an OpenSIPS 4.0.0-rc1 instance acting as a transparent SBC, configured with the `topology_hiding` module enabled for all SIP methods, including OPTIONS, to simulate realistic topology-hiding policies applied by enterprise SBCs. The second was a SEMS 2.1.0 instance running the `sbc` module as a signaling back-to-back user agent (B2BUA), which terminates each leg and reconstructs the outgoing message (assigning a distinct Call-ID per leg), the case in which a custom header is most likely to be stripped. In both setups all non-standard SIP headers arrived intact at the receiver under the default module configuration, and the covert channel operated correctly end-to-end. Neither test included explicit header-stripping or whitelist-based filtering policies; an SBC configured to drop non-standard header fields would discard the carrier header and break the channel, a limitation discussed in Section 6.5. Because the covert channel rides on OPTIONS messages, which carry no SDP body, the SDP-rewriting normalization applied by some SBCs does not affect it. Fig. 4 illustrates the preservation of the non-standard header end-to-end across the SBC. The OpenSIPS and SEMS configurations used in these tests are provided as supplementary material.

5.3. Evasion capability assessment

The detection test results were systematically evaluated across the four security platforms described in Section 5.1. Table 4 summarizes the evasion capability assessment. All tested platforms classified the covert traffic as legitimate SIP signaling. Suricata, configured with the Emerging Threats Open ruleset and SIP application-layer decoder, generated no alerts. Initial testing with Snort 3 revealed that the Cisco Talos Registered ruleset includes session-level anomaly detection rules not present in the ET Open set. Specifically, the use of a unique Call-ID per encapsulated packet triggered the built-in SIP inspector rule 140:27 (`"maximum dialogs within a session reached"`, classified as Potentially Bad Traffic) within a single session. After the PoC was modified to reuse a single Call-ID, consistent with the behavior of legitimate NAT keep-alive exchanges where a persistent logical session is maintained, no further alerts were generated. This finding demonstrates that while Talos rules provide broader coverage than ET Open for SIP session semantics, the covert channel can be adapted to match expected dialog patterns. Zeek 8.1.2, operating with its default SIP protocol analyzer, recorded all transactions in `sip.log` as standard OPTIONS/200 OK exchanges. The `notice.log` file was not generated, indicating zero anomalies detected by Zeek's built-in policy scripts. Zeek's SIP analyzer does not record non-standard headers in its log output: the custom header carrying the covert payload was completely absent from `sip.log`. This means that even manual inspection of Zeek's metadata logs would not reveal the presence of the covert channel. No community packages for SIP-based covert channel detection exist in the Zeek package ecosystem at the time of writing. To establish the detection boundary, the channel was evaluated at incremental transmission rates. No alerts were generated at or below 2 pps (128 kb/s tunnel throughput, achieved using a large virtual interface MTU as described in Section 5.4.5). The 3 pps condition, however, already produced alerts from sid:27899, confirming that 2 pps is the upper rate compatible with undetected operation. At peak throughput (5.2 Mb/s), a volumetric alarm was triggered exclusively in the Snort 3 Registered ruleset, as expected. No alerts were generated by Suricata,

Table 4
Detection results across security platforms at 128 kb/s operational bandwidth.

Platform	Paradigm	Alert at < 128 kb/s	Alert condition
Suricata 8 + ET Open	Signature + volumetric	No	None identified
Snort 3 + Cisco Talos Registered	Signature + volumetric	No	sid:27899 at >4 OPTIONS/s[a]
FortiGate 7.6.6 (UTM)	Commercial	No	None identified

[a] Rule sid:27899: `detection_filter track by_src, count 100, seconds 25`.

Table 5
Shannon entropy evasion via low-entropy padding: detection rate as a function of padding size and MTU.

Pad (bytes)	MTU (bytes)	Entropy (bits/byte)	Zeek alerts /headers	Detect. (%)
0	1400	5.74	1473/1473	100.0
12	1400	5.56	1645/1666	98.7
12	300	5.50	2457/2467	99.6
40	100	4.18	0/2712	0.0
60	100	3.67	0/6757	0.0

Table 6
Secure operating throughput on Yealink T19P E2 with ChaCha20 encryption (no alerts triggered on any platform).

Metric	Value
Runs (N)	30
Duration per run	30 s
Mean throughput	128 kb/s
Std. deviation (σ)	<1 kb/s

Zeek, or the FortiGate firewall at any throughput level. While the FortiGate firewall generated no security alerts at any tested throughput level, its traffic logs did record the covert exchanges. The forwarded sessions were logged under the `traffic:forward accept` event category. The Application Control engine correctly identified the flows through its SIP application signature: the corresponding CEF events were emitted with `FTNTFGTappid` set to 34 640, `FTNTFGTapp` set to `SIP`, and `FTNTFGTappcat` set to `VoIP`. That is, the FortiGate's deep packet inspection engine classified the covert traffic as standard SIP signaling at the application layer, raising no policy violation or anomaly condition despite the presence of non-standard header fields carrying the covert payload. To assess whether the ChaCha20 layer is itself necessary for evasion, we re-ran the proof-of-concept with encryption disabled. Two end-to-end sessions were captured on the Grandstream GXP-1630 to cover representative payload profiles: an interactive Telnet shell (plaintext ASCII) and an SSH session reaching a remote host through a SOCKS5 proxy running on the phone (SSH ciphertext). Replaying both captures through the same detection chain confirms that ChaCha20 is not required for signature-based DPI or NIDS evasion: Snort 3 produces only the same generic `sid:27899` VOIP OPTIONS information-gathering notice already triggered on the encrypted run, plus two spurious `sid:1394` INDICATOR-SHELLCODE x86 inc ecx NOOP hits on the SSH capture, an artefact of the Base64 encoding step: null padding bytes in the SSH binary packet protocol encode as `A` characters (`0x41`) in Base64, and a run of 24 such bytes produces 32 consecutive `A`s, exceeding the 31-character match threshold of the rule. This alert would not appear in unencoded SSH traffic, and Suricata 8 raises zero alerts on either capture. For the entropy-based detector developed in Section 6.2, the outcome is less clear-cut: at the calibrated threshold of 4.75 bits/byte the SSH capture is still flagged, while the Telnet capture falls below the threshold (≈4.55 bits/byte) and evades detection. The captures and runner scripts for the three detectors are provided as supplementary material. To assess the robustness of the entropy detector against deliberate evasion, we evaluated a low-entropy padding bypass. The proof-of-concept was extended with a configurable prefix and suffix of repeated `A` bytes (0×41) appended outside the Base64-encoded ciphertext block in the carrier header. Four configurations were tested against the same Zeek detector at its operating threshold of 4.75 bits/byte; Table 5 summarizes the results.

With 60-byte padding on each side and a tunnel MTU of 100 bytes, the mean Shannon entropy drops to 3.67 bits/byte and the detector generates zero alerts. The evasion carries a concrete operational cost: tunnel throughput falls from 128 kb/s to 11.7 kb/s, approximately an 11-fold reduction. The captures and Zeek runner script are provided as supplementary material. The robustness of detection under this evasion, using multi-feature and anomaly-based detectors, is analyzed in Section 6.4. These results show that Talos session-tracking rules are sensitive to message volume but not to header content, and that the 128 kb/s operational bandwidth remains well below any detection threshold across all tested platforms.

5.4. Performance analysis

The performance evaluation of the covert channel was conducted on the target Yealink T19P E2 hardware to assess its practical viability for Red Team operations. All measurements were performed directly over the tunnel interface using tools statically cross-compiled for the ARM EABI5 architecture of the target device. The SIP encapsulation layer uses stateless PJSIP transactions rather than full User Agent (UA) dialog handling, reducing per-packet overhead by eliminating dialog state maintenance, retransmission timers, and transaction tracking logic.

5.4.1. Bandwidth evaluation

Table 6 presents the secure operating throughput measured using iperf3, statically compiled for the target ARM EABI5 architecture. Each experimental run consisted of a 30-second TCP transfer over the tunnel interface operating at the 128 kb/s secure bandwidth configuration, with no alerts triggered on any of the tested platforms. A total of 30 independent runs were performed.

Throughput was consistent across all 30 runs, with negligible variance ($\sigma < 1$ kb/s) at the reporting resolution of iperf3, explained by the fixed encapsulation overhead and the constant processing capacity of the embedded hardware under sustained load. A throughput of 128 kb/s is sufficient for interactive remote access, file exfiltration, and network pivoting operations, and exceeds the bandwidth requirements of common post-exploitation tools including SSH, SOCKS proxying, and command-and-control frameworks.

5.4.2. Latency measurements

Table 7 summarizes the round-trip time (Round-Trip Time (RTT)) overhead introduced by the SIP encapsulation mechanism. Each experimental run consisted of 100 ICMP echo requests transmitted over the tunnel interface. A total of 30 independent runs were performed.

With a measured RTT of 5.20 ms and low variance ($\sigma = 0.10$ ms), the covert channel introduces minimal latency overhead, well within the requirements for interactive shell sessions and real-time system monitoring tools. No packet loss was observed across all 30 runs, confirming the reliability of the tunnel for sustained operational use.

To confirm vendor-agnostic feasibility, the same evaluation was repeated on a Grandstream GXP-1630 (firmware 1.0.4.132), accessed via CVE-2020-5738 (National Vulnerability Database, 2020) and CVE-2026-2329 (Fewer, 2026). Mean receiver-side throughput reached 783 kb/s (30 runs) and mean RTT was 23.42 ms ($\sigma = 0.64$ ms) with 0.00%

Table 7
Round-trip latency on Yealink T19P E2 with ChaCha20 encryption.

Metric	Value
Runs (N)	30
Packets per run	100
Mean RTT	5.20 ms
Std. deviation (σ)	0.10 ms
95% CI	[5.16, 5.23] ms
Min (run average)	4.97 ms
Max (run average)	5.37 ms
Mean packet loss	0.00%

packet loss across 30 runs, confirming correct operation of the channel on a second vendor's embedded Linux IP phone.

To validate operation beyond the isolated laboratory environment, the channel was additionally tested over a real internet path in the phone-to-PC direction. Two endpoints separated by 24 routing hops (verified by traceroute) were connected with the TAP interface MTU set to 1500 bytes, the standard Ethernet value, to avoid IP fragmentation over the WAN path. Baseline RTT (no tunnel, 100 ICMP probes) was 31.282 ms with 0% loss. Through the tunnel the RTT was 55.124 ms (an overhead of 23.8 ms), also with 0% loss. Mean throughput over 30 independent iperf3 runs was 268 kb/s, exceeding the 128 kb/s stealth operating rate and confirming that the channel operates correctly under real-world internet routing conditions.

5.4.3. Resource overhead and long-term stability

To assess long-term stability and resource overhead, the implant was run continuously for one hour on the Grandstream GXP-1630 under sustained tunnel load, with CPU consumption sampled every 30 s by reading `/proc/pid/stat` and `/proc/stat` directly on the device. Both traffic directions were tested with iperf3 TCP. Fig. 5 shows the results. In the PC-to-phone direction (decapsulation), process CPU stabilized at approximately 75% of the single core; in the phone-to-PC direction (encapsulation), it reached approximately 79%. The 4-percentage-point difference reflects the additional cost of constructing SIP OPTIONS headers and encoding the payload on the constrained ARMv7 processor. System idle time dropped to 0% in both tests, confirming that the implant fully saturates the CPU at maximum throughput. Despite the sustained load, memory consumption remained constant at 1064 kB resident set size in both runs with no growth over the full hour. Zero packet drops were recorded on the TAP interface throughout both sessions.

We also checked the effect on a concurrent voice call on the same GXP-1630. With the implant carrying a continuous encrypted tunnel load of approximately 900 kb/s, ten G.711 calls placed through the phone showed a received-RTP packet loss of 0.06%, against 0.04% for the same ten calls with the tunnel disabled. The difference is within the background loss of the link and is not significant at this sample size; SIP registration was unaffected and all calls established and tore down normally under load. We make no strong claim from a test of this size, only that perceptible call quality did not appear to be affected at a realistic tunnel rate. The captures are provided as supplementary material.

As an indicative measure of energy cost, the AC wall power feeding the PoE injector was read with a consumer plug-in meter of 0.1 W display resolution. The idle phone drew 2.6 W, rising to 2.8 W with the tunnel saturating the CPU under continuous iperf3. Repeating the full test with ChaCha20 disabled yielded the same 2.8 W, which indicates that the cipher adds no power distinguishable at the meter's resolution and that the implant as a whole accounts for roughly 0.2 W over idle. Given the modest precision of the instrument and the wall-side measurement point, these figures are indicative rather than exact.

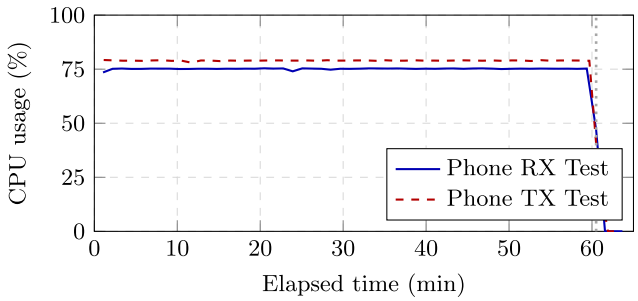

Fig. 5. Implant CPU usage on the Grandstream GXP-1630 during one-hour iperf3 TCP tests (both traffic directions), sampled every 30 s via `/proc`. The dotted line marks the end of the iperf session.

5.4.4. Operational usability

To validate the practical usability of the covert channel for Red Team operations, a series of critical administrative and offensive security tasks were executed through the tunnel. Interactive SSH worked without visible lag: tab-completion was responsive and `htop` refreshed smoothly over the tunnel. High-volume text output was also usable, for example, `dmesg` scrolled through without buffering delays. Beyond basic administration, the tunnel supported post-exploitation activities, including file transfers and SOCKS proxy tunneling for lateral movement within the target network.

These tests confirm that the measured bandwidth and latency are sufficient for the operational scenarios considered in this work, including interactive shell sessions and network pivoting operations commonly employed in Advanced Persistent Threat (APT) campaigns and Red Team assessments. Because the tunnel operates at Layer 2, it also provides the network-level primitives required for a broader class of local-segment attack techniques, including credential harvesting via LLMNR/NBT-NS poisoning, ARP spoofing for man-in-the-middle scenarios, DHCPv6 spoofing, mDNS-based service enumeration, and SMB/NTLM relay attacks, though these were not individually exercised in this evaluation.

5.4.5. MTU and fragmentation analysis

The MTU of the virtual `tap0` interface directly determines the size of the Ethernet frames injected into the covert channel and, therefore, the payload capacity of each SIP OPTIONS message. Fig. 6 provides a visual representation of the MTU breakdown and the SIP header overhead derived from the academic VoIP dataset (Nassar et al., 2010). The SIP header overhead is approximately 335 bytes; combined with the IP/UDP framing (28 bytes), this reduces the available physical MTU (1500 bytes) to a theoretical SIP payload capacity of approximately 1137 bytes per message.

In the test environment, the `tap0` interface was deliberately configured with an MTU of 8000 bytes to minimize the number of SIP transactions required per unit of tunneled data, directly reducing the volumetric footprint that signature-based detection engines could use as a trigger. Each outgoing frame is encrypted with ChaCha20 (which preserves the original length) and subsequently Base64-encoded, expanding the payload by approximately 33% to a maximum of ≈10,667 bytes. This produces SIP messages with unusually large custom headers that would appear anomalous to any manual inspection, yet no tested platform generated alerts related to header length or message size. This exceeds the 1500-byte Ethernet MTU, causing IP-level fragmentation of the UDP packets carrying the SIP messages on the physical network. Intermediate enterprise SBCs and SIP proxies may additionally inject routing headers (e.g., `Via`, `Record-Route`) in transit, which are accommodated within the safety margin provided by the 65,535-byte UDP limit permitted by RFC 3261 (Rosenberg et al., 2002).

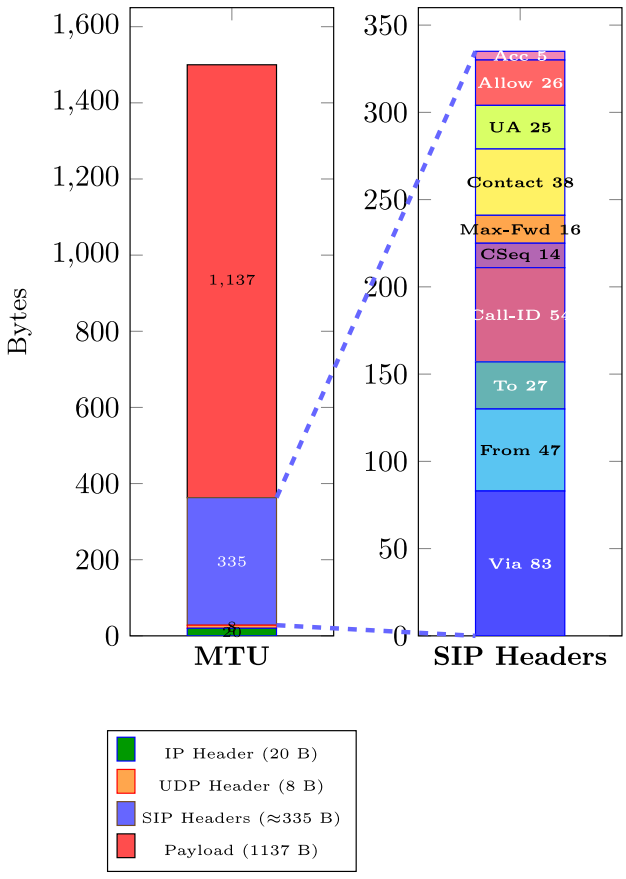

Fig. 6. MTU Breakdown (Left) and Detailed SIP Header overhead analysis (Right).
Source: Data adapted from Nassar et al. (2010).

Fragmentation does not impair covert channel operation because all tested NIDS platforms perform full IP reassembly before application-layer inspection. The reassembled SIP message remains syntactically valid and within the limits permitted by RFC 3261, which allows UDP SIP messages up to 65,535 bytes including IP and UDP headers (Rosenberg et al., 2002). No NIDS platform generated alerts due to message size or fragmentation. Specifically, none of the three evaluated signature and UTM platforms (Snort 3, Suricata 8, and FortiGate) raised an alert attributable to IP-level fragmentation of the SIP/UDP datagrams produced at MTU 8000. Each platform reassembled the fragments and classified the resulting messages as valid SIP signaling. Configuring a large MTU on `tap0` therefore serves a dual purpose: it maximizes throughput and reduces the volumetric detection surface, and it does not introduce additional detectability, as the resulting fragmented traffic is fully reassembled and classified as valid SIP signaling by all tested platforms.

The current implementation does not perform automatic MTU negotiation or path MTU discovery. The SIP header overhead shown in Fig. 6 assumes minimal URI lengths to maintain a lower traffic profile.

5.5. Adversarial adaptation: Stealth mode via rate limiting

While the secure operating bandwidth of 128 kb/s was demonstrated in the previous sections, specific Red Team scenarios prioritize stealth over speed to evade heuristic analysis. Since detection mechanisms may rely on identifying high-frequency OPTIONS patterns, we evaluated the feasibility of a "Stealth Mode" based on standard Linux traffic control tools. Using the user-space TUN/TAP architecture, the `iptables` packet filtering framework with the `limit` match module was used to rate-limit outgoing traffic on the tunnel interface. This configuration forces the covert channel to operate strictly within the bounds of legitimate NAT keep-alive traffic. The following configuration was validated in the test environment to limit transmission to approximately one encapsulated packet every 30 s (2/min):

```
iptables -A OUTPUT -o tap0  -m limit \
--limit 2/min --limit-burst 1 -j ACCEPT
iptables -A OUTPUT -o tap0 -j DROP
```

With this configuration applied, the channel's traffic footprint mirrors the temporal pattern of legitimate VoIP keep-alive mechanisms: one OPTIONS request every 30 s, consistent with the default interval configured in Kamailio Project (2024) and OpenSIPS (OpenSIPS Solutions, 2024). Although adhering to this constraint reduced the effective bandwidth to roughly 400 bps, insufficient for bulk file transfers, it remained fully functional for low-bandwidth command and control (C&C) heartbeats and high-latency shell commands. This confirms that an attacker can dynamically trade bandwidth for invisibility, bypassing frequency-based anomaly detection by mirroring the temporal patterns of standard SIP infrastructure.

Given that the viability of rate limiting as a stealth mechanism had been demonstrated, we decided to integrate it directly into the PoC itself rather than relying on kernel-level tools such as `iptables` or `iproute2 tc`, which are unlikely to be available on constrained firmware environments such as the Yealink T19P E2. The PoC therefore enforces a configurable inter-packet delay between successive OPTIONS transmissions at the covert channel logic level, making the stealth mode self-contained and fully portable across deployment environments. The same 30-second inter-packet interval validated with `iptables` was reproduced natively within the PoC, achieving an equivalent traffic footprint without any dependency on host network configuration.

6. Discussion

The proposed approach demonstrates that SIP control messages, typically considered benign by defensive systems, can be repurposed to exfiltrate information or establish arbitrary traffic tunnels. IP phone devices remain largely unmonitored by EDR tools, frequently run outdated firmware, and expose vulnerabilities that allow remote code execution, as evidenced in Section 5.2.

The developed prototype confirms that even resource-constrained embedded systems can host covert channels capable of evading DPI mechanisms and NIDS, enabling sustained unauthorized access to internal networks. This section analyzes the root causes of the detection failures, presents an empirically validated detection mechanism, and discusses the broader defensive implications.

6.1. Analysis of detection failure

To understand why the proposed covert channel evaded the tested NIDS, a static analysis of the active rulesets in Suricata, Snort 3, and FortiGate was performed, complemented by empirical observation of each platform's behavior during evasion testing.

Analysis of the Emerging Threats Open ruleset for Suricata 8 (Emerging Threats, 2025) reveals that SIP-related coverage is predominantly focused on known exploit strings (e.g., `friendly-scanner`, `SIPVicious`), specific CVE exploits targeting legacy PBX buffer overflows, and volumetric thresholds for REGISTER/INVITE floods. Regarding Snort 3 with the Cisco Talos Registered ruleset, our empirical evasion testing (Section 5.3) confirms a similar detection gap. The only signature triggered during initial testing was rule 140:27 ("maximum dialogs within a session reached"), a session-level tracking feature rather than a content-based detection. This necessitated adapting the PoC to normalize `Call-ID` usage to mimic legitimate NAT keep-alive patterns, after which no further alerts were generated.

The FortiGuard IPS signature database was additionally reviewed against the publicly accessible FortiGuard Threat Encyclopedia (FortiGuard Labs, 2026). The SIP-related entries identified therein are exclusively oriented towards known attack patterns: volumetric and rate-based signatures such as `SIP.Register.Brute.Force` (ID 47088), which triggers at five or more REGISTER requests within ten seconds, and `pplsip.SIP.Scanner` (ID 48756), which detects scanning activity from the Sippts tool at a rate of 200 requests per second; content-based signatures targeting specific protocol violations such as `SIP.Angle.Bracket.Request.URI` (ID 30006) and `SIP. Invite.Spoofing` (ID 28925); and application-specific Denial of Service (DoS) signatures such as `MS.Communicator.SIP. Invite.DoS` (ID 44286), which triggers on INVITE floods exceeding 100 requests within five seconds. No signature in the reviewed catalog addresses the OPTIONS method as a data carrier, whether through rate-based thresholds, non-standard header detection, or payload entropy analysis. This observation was further supported by querying the live IPS signature database on the test FortiWifi 40F, filtered by SIP protocol, which returned 57 signatures. Although the FortiGate management interface does not expose the full technical detail of each entry, inspection of the signature titles suggests that none of them targets the OPTIONS method as a data-carrying vector.

Overall, the consistent lack of alerts across all tested platforms points to a structural detection gap: no ruleset (ET Open, Cisco Talos, or FortiGuard) contains signatures addressing header length anomalies, non-standard header presence, or payload entropy analysis. Protocol misuse through syntactically valid but semantically anomalous headers falls entirely outside the current detection scope of signature-based engines.

6.2. Entropy-based detection with Zeek

Following the approach proposed by Koziak et al. (2021) for adapting Zeek to detect steganographic transmission, and building on the complexity-based detection tests evaluated by Sattolo and Jaskolka (2020) for storage-based covert channels, we implemented a lightweight Zeek script that combines three features to identify SIP-based covert communication.

6.2.1. Detection method

The detector operates on the `sip_header` event, which Zeek emits for every individual header in each SIP message, including non-standard headers that are not recorded in the default `sip.log` output. For each header, three conditions are evaluated:

1. **Non-standard header identification.** The header name is checked against a set of headers defined in RFC 3261 and common extensions. Standard headers are immediately discarded, regardless of their content. This eliminates false positives from legitimate headers that may exhibit high entropy: for instance, `Authorization` headers in the Nassar dataset reach entropy values of up to 5.00 bits/byte with lengths of 132–140 bytes, yet are correctly discarded as standard.
2. **Length threshold.** Headers shorter than 50 bytes are discarded, as they are unlikely to carry meaningful covert payloads.
3. **Shannon entropy analysis.** The entropy of the header value is computed using Zeek's `find_entropy()` built-in function. Headers with entropy below a configurable threshold are discarded.

A header that satisfies all three conditions simultaneously triggers a Notice in Zeek's alert framework. The rationale for the three-feature combination is that no single feature is sufficient in isolation. The Nassar dataset contains non-standard headers (`X-Asterisk-HangupCause`, `X-Asterisk-HangupCauseCode`) that pass Feature 1 but are correctly discarded by Feature 3 due to their low entropy (maximum 3.80 bits/byte). Conversely, `Authorization` headers pass Features 2 and 3 but are discarded by Feature 1. The combination of all three conditions ensures that no legitimate header in the evaluated baseline triggers an alert.

Table 8
Shannon entropy profiles of SIP headers measured over the covert traffic capture and the Nassar legitimate baseline (Nassar et al., 2010) (bits/byte).

Header	Type	Source	Entropy	Length
`Defined` (covert)	Non-std	Covert	5.15–6.00	72–10,688 B
`Authorization`	Std	Nassar	4.76–5.00	132–140 B
`Via`	Std	Nassar	4.44–4.96	58–98 B
`From`	Std	Nassar	4.12–4.74	32–49 B
`To`	Std	Nassar	3.80–4.77	18–45 B
`Call-ID`	Std	Nassar	3.39–4.07	45 B

Table 9
Detection performance across entropy thresholds ($N_{\text{covert}} = 1473$, $N_{\text{legit}} = 96{,}403$ headers).

Threshold (bits/byte)	TP	FP	Recall	Precision
3.0–5.0	1473	0	100%	100%
5.25	1372	0	93.1%	100%
5.5	939	0	63.7%	100%
5.75	882	0	59.9%	100%
6.0	0	0	0%	–

6.2.2. Entropy profiles

To calibrate the detector, the Shannon entropy and length of both covert and legitimate SIP headers were measured using a dedicated Zeek profiling script executed against both the covert traffic capture and the Nassar dataset. Table 8 summarizes the observed ranges for the most relevant headers.

The covert header `Defined` (the name is software configurable), carrying Base64-encoded ChaCha20 ciphertext, exhibits entropy in the 5.15–6.00 bits/byte range. The wide length range (72–10,688 bytes) reflects the `tap0` MTU of 8000 bytes configured in the test environment (Section 5.4.5). No legitimate header in the Nassar baseline simultaneously satisfies all three detection conditions, confirming that the three features together separate covert from legitimate traffic.

6.2.3. Detection results

The detector was evaluated against the covert traffic capture ($N_{\text{covert}} = 1473$ encapsulated packets) and the Nassar legitimate SIP baseline ($N_{\text{legit}} = 96{,}403$ headers, of which 894 are non-standard) (Nassar et al., 2010). Table 9 presents detection performance across entropy thresholds with the length threshold fixed at 50 bytes.

At the recommended threshold of 4.75 bits/byte, the detector achieves 100% recall and 100% precision: all 1473 covert headers are detected with zero false positives across 96,403 legitimate headers, including 894 non-standard headers present in the Nassar dataset. Precision remains at 100% across all thresholds because the three-feature combination ensures that no legitimate header triggers an alert. Recall remains at 100% up to a threshold of 5.0 bits/byte, then begins to degrade at 5.25 bits/byte as shorter covert payloads with lower entropy fall below the threshold. At 6.0 bits/byte no covert headers are detected, as Base64 encoding inherently bounds entropy below this value. These results are bounded by the evaluated corpus: the covert traffic originates from a single encapsulation configuration, and the legitimate baseline comprises a single deployment scenario. Performance on production networks with diverse SIP implementations, third-party header extensions, or alternative encryption schemes may differ, and broader validation is needed before generalizing these figures. Section 6.4 provides such broader validation on a synthetic corpus generated with SIPp, where the scalar rule does raise false

Table 10
False-positive rate of the scalar entropy rule (Zeek, 4.75 bits/byte) on two legitimate SIP baselines.

Baseline	Headers	FP	FP rate
Nassar 2010 (single deployment)	96,403	0	0.00%
SIPp synthetic (12 UAs, diverse non-std hdrs)	3,660	329	8.99%

positives on legitimate non-standard headers. The 100% precision reported here is therefore specific to the Nassar baseline.

Table 10 quantifies the gap between the two baselines at the operating threshold of 4.75 bits/byte. The contrast is the direct motivation for the multi-feature and anomaly-based detectors presented in Section 6.4.

These results demonstrate that the proposed covert channel, while invisible to all tested security platforms in their default configuration, can be identified through a lightweight Zeek script that performs targeted statistical analysis of SIP header content. The complete detection script is provided as supplementary material. The detection mechanism requires no modification to Zeek's core engine and can be deployed as a standard policy script. This identification is demonstrated on the dataset evaluated here. Its false-positive behavior on legitimate non-standard headers is examined in Section 6.4.

6.3. Native entropy detection with Suricata 8

The Zeek-based detector presented above requires a custom script operating outside the standard NIDS rule pipeline. A natural question is whether the same detection principle can be expressed within a signature-based engine, enabling deployment via standard rule distribution channels. Suricata 8.0.0, released in 2025, introduces a native `entropy` keyword that computes Shannon entropy directly within the rule evaluation engine, without requiring Lua scripting or external plugins. This section evaluates whether a single Suricata 8 rule can replicate the entropy-based detection achieved by the Zeek script.

6.3.1. SIP inspection capabilities and limitations

Suricata 8 provides 13 SIP-specific sticky buffers for rule matching (Open Information Security Foundation, 2025): `sip.method`, `sip.uri`, `sip.request_line`, `sip.stat_code`, `sip.stat_msg`, `sip.response_line`, `sip.protocol`, `sip.from`, `sip.to`, `sip.via`, `sip.user_agent`, `sip.content_type`, and `sip.content_length`. These cover only headers explicitly defined in RFC 3261 and common extensions. Unlike the HTTP protocol module, which offers a generic `http.header_names` keyword for enumerating arbitrary header names, the SIP module provides no equivalent mechanism. Consequently, a non-standard header such as the covert channel carrier used in this work cannot be targeted by name through Suricata's SIP sticky buffers.

This architectural difference has a direct impact on detection granularity. The Zeek detector analyzes the entropy of each individual header value via the `sip_header` event, isolating the covert payload from the surrounding message structure. In Suricata 8, since no sticky buffer can address the custom header, the `entropy` keyword must operate on the raw UDP payload of the entire SIP message, which includes both the high-entropy covert data and the lower-entropy standard headers (`Via`, `From`, `To`, `CSeq`, etc., totaling approximately 335 bytes of overhead). In messages with large covert payloads, the encrypted data dominates the entropy calculation; in shorter messages, the standard header overhead dilutes the overall entropy below the detection threshold.

Table 11
Suricata 8 entropy rule detection performance.

Threshold	Dataset	Packets	Alerts
4.75 bits/byte	Covert (OPTIONS)	1,473	1472
4.75 bits/byte	Nassar legitimate baseline	41,482	0
4.75 bits/byte	SIPp synthetic baseline	600	59
5.75 bits/byte	Covert (OPTIONS)	1,473	1450
5.75 bits/byte	Nassar legitimate baseline	41,482	0
5.75 bits/byte	SIPp synthetic baseline	600	33

6.3.2. Detection rule

Given the above constraints, the following rule was designed to detect high-entropy SIP OPTIONS requests without depending on the name of the carrier header:

```
alert sip any any -> any any ( \
  msg:"COVERT-SIP High entropy in SIP OPTIONS"; \
  content:"OPTIONS sip"; depth:11; \
  entropy: value >= 5.75; \
  classtype:policy-violation; \
  sid:9000001; rev:1;)
```

The rule anchors on the raw request line via `content:"OPTIONS sip"` (without a sticky buffer), ensuring the entropy calculation covers the full message payload. The threshold of 5.75 bits/byte was selected based on the entropy sweep results presented in Table 9.

6.3.3. Detection results

The rule was evaluated on Suricata 8.0.4 against the covert traffic capture and the Nassar legitimate SIP baseline (Nassar et al., 2010). Table 11 presents the results.

At the 4.75 bits/byte threshold, the rule detects 1472 out of 1473 covert OPTIONS messages (99.93% recall) with zero false positives across the entire Nassar dataset. Raising the threshold to 5.75 bits/byte reduces recall to 1450 detections (98.44%), with the 23 undetected messages corresponding to the shortest covert payloads (UDP length <500 bytes), where the standard SIP header overhead dilutes the overall message entropy below the threshold. At both thresholds, no false positives were observed on the Nassar baseline. The SIPp synthetic corpus behaves differently: because the `entropy` keyword scores the full message payload rather than the isolated header value, it cannot separate a legitimate `OPTIONS` request carrying several Base64-like extension headers from a covert carrier. On that corpus the rule raises 59 false positives at 4.75 and 33 at 5.75 bits/byte, out of 60 legitimate `OPTIONS` requests (Table 11), the coarse-measurement cost that motivates the multi-feature detector of Section 6.4. This contrasts with the Zeek detector, which at 5.75 bits/byte detects 882 covert headers out of 1473 (59.9% recall) but analyzes only the isolated header value rather than the full message.

6.3.4. Comparison with the Zeek detector

Table 12 summarizes the trade-offs between the two approaches.

The Zeek script isolates each header value individually, enabling a three-feature composite test that achieves 100% recall at lower thresholds (4.75 bits/byte) with zero false positives on the Nassar baseline. The Suricata 8 rule offers operational simplicity, a single signature distributable through standard rule management, at the cost of coarser entropy measurement. Lowering the Suricata threshold to 4.75 bits/byte raises recall to 99.93% while maintaining zero false positives on the Nassar baseline, closing most of the difference with the Zeek detector. However, the inability to isolate non-standard header values means that on production traffic with diverse SIP implementations, the risk of false positives at lower thresholds remains higher than with the Zeek three-feature approach. The SIPp synthetic corpus bears

Table 12
Comparison of entropy-based detection approaches.

Property	Zeek script	Suricata 8 rule
Entropy scope	Header value only	Full SIP message
Custom header access	Yes (`sip_header`)	No (raw payload)
Features combined	3 (name+len+entropy)	1 (entropy only)
Deployment	Policy script	Standard rule
FP at $\theta = 4.75$ (Nassar)	0	0
FP at $\theta = 5.75$ (Nassar)	0	0
FP at $\theta = 4.75$ (SIPp)	329/3660 hdrs	59/60 msgs
FP at $\theta = 5.75$ (SIPp)	0/3660 hdrs	33/60 msgs
Recall at $\theta = 4.75$	100% (1473 hdrs)	99.93% (1473 msgs)
Recall at $\theta = 5.75$	59.9% (1473 hdrs)	98.44% (1473 msgs)

this out. At 4.75 bits/byte the rule flags 59 of 60 legitimate `OPTIONS` requests, and even at 5.75 bits/byte it still raises 33, because scoring the full message keeps the aggregate entropy high once several Base64-like extension headers are present. The Zeek detector, which scores each header value in isolation, drops to zero false positives on the same corpus at 5.75 bits/byte (Table 12). Neither scalar approach is robust across both baselines at a single threshold, the gap addressed by the multi-feature detector of Section 6.4.

The absence of a generic `sip.header` or `sip.header_names` keyword in Suricata represents a structural gap for SIP anomaly detection. Should future Suricata releases extend the SIP parser to expose arbitrary header names and values, as the HTTP module already does, the entropy keyword could be scoped to individual header values, potentially matching the Zeek detector's precision within the signature-based paradigm.

6.4. Multi-feature and anomaly-based detection

Table 5 shows that an adversary can defeat the entropy detector by padding the carrier header with low-entropy bytes, driving recall to zero under aggressive padding. To assess whether additional features recover that recall, and whether a model trained only on benign traffic can generalize to unseen covert variants, we evaluate two complementary detectors.

Both treat one SIP header as one sample and extract fifteen features from the header value only: header length, a Base64 character-class fingerprint (fractions of upper, lower, digit, +, /, =, and other characters), and three randomness measures (distinct-character ratio, normalized longest character run, and zlib compression ratio). Header name, the non-standard flag, and scalar Shannon entropy are excluded so the detectors cannot fall back on trivial signatures and remain robust to header renaming. These feature families are established discriminators for encoded and encrypted content. Byte entropy, compression ratio, and the distinct-character ratio are among the statistical complexity tests evaluated by Sattolo and Jaskolka (2020) for storage covert channels, and recent machine-learning covert-channel detectors still rely on the entropy and byte-sequencing of field content (Rahman et al., 2026). The Base64 character-class fingerprint targets the same signal that current DNS exfiltration detectors exploit through length, entropy, and character-class counts over query content that, as they report, is itself Base64-encoded (Ozery et al., 2024). The first detector is a supervised Random Forest trained on the Nassar benign baseline and the unpadded covert captures. Padded captures and the SIPp corpus were held out entirely from training. The second is a one-class Isolation Forest trained only on Nassar benign headers and never shown the covert channel. Table 13 summarizes both detectors alongside the scalar entropy rule across all evaluation conditions.

On unpadded traffic the RF and the entropy rule are equivalent: both reach 100% recall, so the classifier adds nothing on the baseline channel. The contribution appears under evasion: where the entropy rule drops to zero recall at aggressive padding (40 and 60 bytes), the RF maintains 99.6%, detecting the increase in the normalized longest character run and the drop in compression ratio introduced by the repeated padding bytes. The Isolation Forest at `contamination=0.01` behaves differently: standard carriers are anomalous relative to the Nassar benign distribution (ROC-AUC 0.978 across all captures) but their feature values do not reach the top-1% extremity cutoff, so recall on unpadded traffic is 0%. Aggressive padding (40 and 60 bytes at MTU 100) drives `longest_run_norm` and `compression_ratio` to values far outside the benign range; these captures are flagged at 100% recall. The model is therefore sensitive to structural extremity, not to covert-channel content as such.

Table 13
Scalar entropy rule, multi-feature Random Forest, and Isolation Forest across all conditions. Threshold 4.75 bits/byte. Padded captures and the SIPp corpus held out from training. False positives over request headers (baseline sizes in Table 10).

Condition	Entropy (Zeek@4.75)	RF multi-feature	ISO Forest (`c=0.01`)
Recall on held-out covert captures			
Unpadded (MTU 1400)	100%	100%	0%
Pad 40 B, MTU 100	0%	99.6%	100%
Pad 60 B, MTU 100	0%	99.6%	100%
False-positive rate on benign baselines			
Nassar 2010	0.00%	0.00%	—
SIPp synthetic	8.99%	0.82%	0.00%

On benign traffic, the scalar rule raises 8.99% false positives on the SIPp corpus (Table 10), almost all on legitimate non-standard headers outside the RFC 3261 whitelist. The RF reduces this to 0.82%, with residual errors confined to header values with uniformly random Base64 content that are indistinguishable from encrypted payload. The Isolation Forest raises no false positives at `contamination=0.01`. Supplementary material documents sensitivity to this parameter. These figures are indicative: the covert class comes from a single tool and the baselines from two controlled corpora, so generalization to production traffic requires further validation. Machine learning for VoIP covert channels has so far targeted timing and media-layer features (Mazurczyk, 2013); extending multi-feature content profiling to flow-level behavior is a natural direction for future work. Feature extraction and training code, together with the held-out captures, are provided as supplementary material.

6.5. Limitations and future directions

The evaluation presented in this work has several limitations that should be acknowledged. The covert channel was fully validated on two device models: the Yealink T19P E2 and the Grandstream GXP1630 (firmware 1.0.4.132), with portability to further vendors supported by the TUN/TAP and CVE analysis of Section 5.2. The legitimate SIP baseline used for false positive evaluation comprises 96,403 headers extracted from the academic VoIP dataset of Nassar et al. (2010), which, while extensive, originates from a controlled research environment dating to 2010. To the best of our knowledge, this remains the only publicly available labeled SIP dataset with full packet captures suitable for header-level analysis; the scarcity of public VoIP corpora Nazih et al. (2020), largely due to privacy constraints inherent to voice traffic, limits the availability of more recent alternatives. A baseline drawn from a current production VoIP deployment would nonetheless provide stronger confidence in the generalizability of the false positive rate to modern enterprise traffic patterns. As a step towards this, Section 6.4 evaluates the detectors on a synthetic corpus, generated with SIPp, that includes legitimate non-standard headers. That the corpus is not production traffic is an important limitation, but it exercises header diversity absent from the 2010 baseline.

The entropy-based detector, while highly effective against the current implementation, can be circumvented by an adaptive adversary that pads the carrier header with low-entropy data, as demonstrated in

Table 5. This evasion is addressed by the multi-feature and anomaly-based detectors of Section 6.4, which remain effective against padded carriers. Two additional scope boundaries should be noted. First, the evaluation was conducted with SIP signaling transmitted in cleartext over UDP, as configured in the test scenario described in Section 5.1. In deployments where SIP is transported over TLS, the covert channel would still operate but would be encapsulated within the encrypted stream, making it invisible to network-level inspection; in such environments, however, the same TLS encryption also prevents legitimate NIDS from analyzing SIP header content, making both the attack and the proposed entropy-based detection mechanism dependent on TLS termination points. In practice, however, this is not a limitation specific to the proposed detector but a consequence of NIDS placement. The detector remains applicable wherever SIP signaling is observable in cleartext to the monitor. One such placement is co-location with the SBC or SIP proxy, which typically terminates TLS to apply topology hiding, normalization, or routing logic, and therefore exposes plaintext SIP to a co-located sensor. Enterprise environments with TLS-inspecting middleboxes that decrypt and inspect SIP traffic in transit are also suitable. Empirical evaluation of the channel and detector with TLS as the SIP transport is left as future work. Second, SIP proxies and SBCs that normalize or strip unrecognized headers as part of their forwarding logic would disrupt the covert channel without any explicit detection mechanism, a passive countermeasure already present in some enterprise architectures.

Deploying the detection scripts released as supplementary material raises three practical questions: where to place the sensor, how much it costs to run, and how it feeds a SIEM. On placement, both the Zeek script and the Suricata 8 rule need access to plaintext SIP traffic. The recommended position is a passive port mirror or network TAP upstream of any TLS-terminating device. In TLS-protected environments, co-location with the SBC or SIP proxy is equivalent, since these devices expose plaintext SIP before re-encryption. On overhead, the Zeek script processes each SIP message once at the `sip_header` event. Entropy computation is $O(l)$ in header-value length and per-endpoint state is limited to a sliding message counter, so no per-flow reassembly is required. A quantitative benchmark under production SIP load remains an open validation item. On SIEM integration, Zeek raises a standard `Notice::LOG` entry (source and destination address, timestamp, header name, entropy value, exceeded threshold) that any SIEM with a Zeek connector ingests without extra configuration, and the Suricata 8 rule emits a standard EVE-JSON alert supported natively by all major SIEM platforms.

Future research should investigate flow-level machine learning approaches for behavioral profiling of SIP traffic, complementary to the content-based multi-feature analysis of Section 6.4, considering features such as message size distributions, inter-arrival time patterns, header field cardinality, and the ratio of signaling messages to media sessions (Mazurczyk, 2013).

6.6. Ethical considerations

All experiments were performed on a dedicated isolated testbed with no connectivity to production infrastructure. The Yealink T19P E2 and Grandstream GXP1630 units used in this work were acquired specifically for research purposes. Network traffic was confined to a physically isolated switched segment; no third-party communications were intercepted at any point.

CVE-2023-43959 was publicly disclosed in October 2023 and patched in the vendor's subsequent firmware release prior to the start of this work. Exploitation was performed solely against the research unit under controlled conditions. The covert channel binary and Zeek detection script are released as supplementary material to enable independent validation and to support defensive tooling development. No offensive capability beyond what is described in this paper is provided.

7. Conclusion

This paper demonstrated that SIP `OPTIONS` messages can carry arbitrary L2/L3 traffic without triggering any of the four tested security platforms, and that the same channel is detectable through entropy analysis of non-standard headers. The PoC was implemented as a portable, statically linked binary with TUN/TAP virtual interfaces and ChaCha20 encryption, and was deployed on a real Yealink T19P E2 IP phone with ARM EABI5 architecture. Performance measurements over 30 independent runs yielded a secure operating throughput of 128 kb/s with a mean round-trip latency of 5.2 ms, confirming that the channel supports interactive shell sessions, file transfers, and lateral movement operations on resource-constrained embedded hardware. The portability analysis demonstrated that the two prerequisites for deployment, namely a Linux kernel with TUN/TAP support and the ability to execute a statically linked binary, are met by IP phones from at least three major vendors (Yealink, Snom, and Grandstream), all of which have documented remote code execution vulnerabilities. Vendor-agnostic feasibility was confirmed empirically on a second device: the Grandstream GXP1630 (firmware 1.0.4.132) reached a mean throughput of 783 kb/s and a mean round-trip latency of 23.42 ms with zero packet loss across 30 runs, demonstrating that the channel is not limited to a single hardware platform.

Evasion testing against four security platforms covering the main defensive approaches (Suricata and Snort 3 for signature-based detection, Zeek for metadata-oriented monitoring, and FortiGate for commercial UTM) produced no alerts in their default configurations. Empirical behavioral analysis of the Suricata and Snort 3 SIP parsers, complemented by static analysis of the ET Open and Cisco Talos Registered rulesets, revealed the structural cause: current detection logic validates SIP syntax without analyzing header content, length distribution, or entropy. Testing with Snort 3 additionally uncovered that Talos rules include session-level dialog tracking absent from ET Open, requiring the channel to normalize its `Call-ID` usage to evade detection.

To bridge the gap between the identified attack and practical defense, an entropy-based detection mechanism was implemented as a Zeek policy script. The detector combines three features: non-standard header identification, length thresholds, and Shannon entropy analysis. Evaluated against covert and legitimate traffic captures, it achieved 100% precision and 100% recall at an entropy threshold of 4.75 bits/byte, with zero false positives across all tested thresholds. This holds for the Nassar baseline; on a synthetic corpus generated with SIPp the scalar rule raises false positives on legitimate non-standard headers, and a multi-feature classifier reduces that false-positive rate by an order of magnitude (Section 6.4). This result demonstrates that while the covert channel is invisible to current commercial and open-source security platforms, targeted statistical analysis of SIP header content can reliably identify it. A native Suricata 8 rule using its entropy keyword achieved comparable results (99.9% recall at the same threshold) without requiring custom scripting.

Two directions merit further investigation. The entropy-based detector proposed here operates per-header on cleartext SIP; extending it to handle high-throughput environments will require either hardware-accelerated entropy computation or flow-level behavioral profiling, complementary to the content-based multi-feature approach of Section 6.4, using features such as inter-arrival times, message size distributions, and the signaling-to-media session ratio. Separately, the encapsulation technique may transfer to other signaling protocols within the Real-Time Communications (RTC) ecosystem that share similar properties of lightweight out-of-band messaging and are commonly permitted through enterprise firewalls.

Abbreviations

APT	Advanced Persistent Threat
C&C	Command and Control

CDRs	Call Detail Records
DoS	Denial of Service
DPI	Deep Packet Inspection
EDR	Endpoint Detection and Response
IPS	Intrusion Prevention System
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NIDS	Network Intrusion Detection System
PoC	Proof of Concept
PSK	Pre-Shared Key
RTC	Real-Time Communications
RTP	Real-time Transport Protocol
RTT	Round-Trip Time
SBC	Session Border Controller
SDP	Session Description Protocol
SIEM	Security Information and Event Management
SIP	Session Initiation Protocol
UA	User Agent
UTM	Unified Threat Management
VAD	Voice Activity Detection
VoIP	Voice over Internet Protocol

CRediT authorship contribution statement

Gorka Gorrochategui: Writing – original draft, Data curation, Conceptualization. **Unai Zulaika:** Writing – review & editing, Methodology. **Pablo Garaizar:** Writing – review & editing, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We gratefully acknowledge the support of the Basque Government for the Deustek research group.

Data availability

full source code, scripts, configs.

Additional Material - Source Code, Static Binaries, Videos, PCAPs, SIPp dataset (Original data) (google drive shared .zip (supplementary data))

References

Alishavandi, A.M., Fakhredanesh, M., 2021. MKIPS: MKI-based protocol steganography method in SRTP. ETRI J. 43 (3), 561–570.

Bernstein, D.J., 2008. ChaCha, a variant of Salsa20. In: Workshop Record of SASC (State of the Art of Stream Ciphers), Vol. 8. pp. 3–5, [Online]. Available: https://cr.yp.to/chacha/chacha-20080128.pdf.

Ekman, E., Andersson, B., 2006–2023. Iodine: IP-over-DNS tunnel. [Online]. Available: https://code.kryo.se/iodine/.

Emerging Threats, 2025. Emerging threats open ruleset for suricata. Proofpoint. [Online]. Available: https://rules.emergingthreats.net/open/suricata/.

Fewer, S., 2026. CVE-2026-2329: Critical Unauthenticated Stack Buffer Overflow in Grandstream GXP16xx Series VoIP Phones. Rapid7 Labs, [Online]. Available: https://www.rapid7.com/blog/post/ve-cve-2026-2329-critical-unauthenticated-stack-buffer-overflow-in-grandstream-gxp1600-voip-phones-fixed/.

FortiGuard Labs, 2026. FortiGuard threat encyclopedia: Intrusion prevention signatures. Fortinet.

Joseph, O., Elmalech, A., Hajaj, C., 2023. Detecting parallel covert data transmission channels in video conferencing using machine learning. Electronics 12 (5), 1091.

Kamailio Project, 2024. Modules' Documentation: Nathelper (NAT Traversal Helper). Kamailio SIP Server Documentation, [Online]. Available: https://www.kamailio.org/docs/modules/stable/modules/nathelper.html.

Khadse, M.A., Dakhane, D.M., 2025. A review on network covert channel construction and attack detection. Concurr. Comput.: Pr. Exp. 37, e8316.

Koziak, T., Wasielewska, K., Janicki, A., 2021. How to make an intrusion detection system aware of steganographic transmission. In: Proc. European Interdisciplinary Cybersecurity Conference. EICC 2021, ACM, Virtual Event, Romania, pp. 77–82.

Krasnyansky, M., 1999–2000. Universal TUN/TAP Device Driver. Linux Kernel Documentation, [Online]. Available: https://www.kernel.org/doc/Documentation/networking/tuntap.txt.

Mazurczyk, W., 2013. VoIP steganography and its detection—a survey. ACM Comput. Surv. 46 (2), 20.

Mazurczyk, W., Szczypiorski, K., 2008. Covert channels in SIP for VoIP signalling. In: Jahankhani, H., Revett, K., Palmer-Brown, D. (Eds.), Global E-Security. In: Communications in Computer and Information Science, vol. 12, Springer-Verlag, Berlin, Heidelberg, pp. 65–72.

Mehić, M., Mikulec, M., Voznak, M., Kapicak, L., 2014. Creating covert channel using SIP. In: Proc. Multimedia Communications, Services and Security. MCSS 2014, In: Communications in Computer and Information Science, vol. 429, Springer, pp. 182–192.

Moghadasi, H.A., Dehghani, H., 2026. Assessment of a VoIP steganalysis method based on statistical analysis and deep neural network. Sci. Rep. 16, 1517.

Nassar, M., State, R., Festor, O., 2010. Labeled VoIP data-set for intrusion detection evaluation. In: Proceedings of the 16th EUNICE/IFIP International Workshop on Networked Services and Applications. EUNICE 2010, Springer, pp. 97–106.

National Vulnerability Database, 2020. CVE-2020-5738. NVD, [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2020-5738.

National Vulnerability Database, NIST, 2023. CVE-2023-43959: YeaLink SIP-T19P-E2 OS Command Injection. National Vulnerability Database, NIST, [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2023-43959.

Nazih, W., Hifny, Y., Elkilani, W.S., Dhahri, H., Abdelkader, T., 2020. Countering DDoS attacks in SIP based VoIP networks using recurrent neural networks. Sensors 20 (20), 5875.

Open Information Security Foundation, 2025. SIP keywords. Suricata 8.0.0 User Guide. [Online]. Available: https://docs.suricata.io/en/suricata-8.0.0/rules/sip-keywords.html.

OpenSIPS Solutions, 2024. OpenSIPS documentation: nathelper module. OpenSIPS Manual. [Online]. Available: https://opensips.org/docs/modules/3.2/nathelper.html.

Ozery, Y., Nadler, A., Shabtai, A., 2024. Information-based heavy hitters for real-time DNS data exfiltration detection. In: Proc. Network and Distributed System Security Symposium. NDSS 2024, Internet Society, San Diego, CA, USA.

Pinna, E., 2011. Stegosip: TCP Tunnel over RTP/SIP. GitHub Repository, [Online]. Available: https://github.com/epinna/stegosip.

Rahman, M.W.U., Lin, Y.-Z., Weeks, C., Ruddell, D., Gabriellini, J., Hayes, B., Hariri, S., Satam, P., Ziegler, Jr., E.V., 2026. AI/ML based detection and categorization of covert communication in IPv6 network. Cybersecurity 9, 33.

Rosenberg, J., et al., 2002. SIP: Session Initiation Protocol. RFC 3261, IETF.

Saenger, J., Mazurczyk, W., Keller, J., Caviglione, L., 2020. Voip network covert channels to enhance privacy and information sharing. Future Gener. Comput. Syst. 111, 96–106.

Sattolo, T.A.V., Jaskolka, J., 2020. Evaluation of statistical tests for detecting storage-based covert channels. In: Hölbl, M., Rannenberg, K., Welzer, T. (Eds.), ICT Systems Security and Privacy Protection. IFIP SEC 2020, In: IFIP Advances in Information and Communication Technology, vol. 580, Springer, Cham, pp. 17–31.

Schmidt, S., Mazurczyk, W., Kulesza, R., Keller, J., Caviglione, L., 2018. Exploiting IP telephony with silence suppression for hidden data transfers. Comput. Secur. 79, 17–32.

SEC Consult Vulnerability Lab, 2015. Multiple critical vulnerabilities in snom IP phones. Advisory 20150113-0, [Online]. Available: https://sec-consult.com/vulnerability-lab/advisory/multiple-critical-vulnerabilities-in-snom-ip-phones/.

Snom Technology, 2024. Configuring VPN on Snom Deskphones. Snom Service Hub, [Online]. Available: https://service.snom.com/display/wiki/Configuring+VPN+on+Snom+Deskphones.

Tsiatsikas, Z., Anagnostopoulos, M., Kambourakis, G., Lambrou, S., Geneiatakis, D., 2015. Hidden in plain sight: SDP-based covert channel for botnet communication. In: Proc. Trust, Privacy and Security in Digital Business. TrustBus 2015, In: Lecture Notes in Computer Science, vol. 9264, Springer, pp. 48–59.

Wendzel, S., Mazurczyk, W., Zander, S., 2016. Unified description for network information hiding methods. J. Univers. Comput. Sci. 22 (11), 1456–1486.

Wendzel, S., Schmidbauer, T., Zillien, S., Keller, J., 2025. DYST (did you see that?): An amplified covert channel that points to previously seen data. IEEE Trans. Dependable Secur. Comput. 22 (1), 614–631.

Wendzel, S., Zander, S., Fechner, B., Herdin, C., 2015. Pattern-based survey and categorization of network covert channel techniques. ACM Comput. Surv. 47 (3), 50.

Wu, Z., Guo, J., Zhang, C., Li, C., 2021. Steganography and steganalysis in Voice over IP: A review. Sensors 21 (4), 1032.
Zillien, S., Wendzel, S., 2023. Weaknesses of popular and recent covert channel detection methods and a remedy. IEEE Trans. Dependable Secur. Comput. 20 (6), 5156–5167.

Gorka Gorrochategui is the founder and CEO of Irontec, a technology company he established in 2003 that specializes in telecommunications and cybersecurity solutions with a strong commitment to open source software. Under his leadership, Irontec has developed and released numerous open source projects, contributing actively to the global open source community. He holds a BSc and MSc in Cybersecurity. He has led key departments within his organization, including Real-Time Communications and Operator Services, as well as Offensive Cybersecurity. With over 20 years of experience bridging the gap between industry and academia, he serves as a guest lecturer at the University of Deusto, where he teaches Ethical Hacking. His research and professional work focus on offensive security and ethical hacking, combining practical industry expertise with academic rigor.

Dr. Unai Zulaika Zurimendi is a Research Associate at DeustoTech, University of Deusto, and a member of the DEUSTEK research group. He earned his PhD in 2022, specializing in Explainable Artificial Intelligence (XAI) and Knowledge Graph algorithms. His research focuses on AI and Natural Language Processing (NLP) for healthcare, specifically clinical data extraction and interoperability within European projects like IDEA4RC and PROTECT CHILD. His work explores interpretable link prediction and the development of NLP-powered pipelines for semantic data extraction.

Dr. Pablo Garaizar is an Associate Professor at the University of Deusto and a member of the DEUSTEK research group. He holds a PhD in Computer Engineering and a BSc in Psychology. He has over 20 years of experience lecturing Programming, High Performance Computing, Ethical Hacking and Cloud System Architectures subjects, and researching on the development of Computational Thinking in novice programmers, Internet based research experiments and computer networks.