← BACK TO BLOG

THREAT RESEARCH

BPFdoor in Telecom Networks: Sleeper Cells in the Backbone

Rapid7 Labs

Mar 26, 2026 | *Last updated on Mar 26, 2026* | 26 min read

VIEW THE WEBINAR

Executive overview

The strategic positioning of covert access within the world's telecommunication networks

A months-long investigation by Rapid7 Labs has uncovered evidence of an advanced China-nexus threat actor, Red Menshen, placing some of the stealthiest digital sleeper cells the team has ever seen in telecommunications networks. The goal of these campaigns is to carry out high-level espionage, including against government networks.

Telecommunications networks are the central nervous system of the digital world. They carry government communications, coordinate critical industries, and underpin the digital identities of billions of people. When these networks are compromised, the consequences extend far beyond a single provider or region. That level of access is, and should be, a national concern as it compromises not just one company or organization, but the communications of entire populations.

Over the past decade, telecom intrusions have been reported across multiple countries. In several cases, state-backed actors accessed call detail records, monitored sensitive communications, and exploited trusted interconnections between operators. While these incidents often appear isolated, a broader pattern is emerging.

Why telecom networks are strategic espionage targets

Telecommunications infrastructure provides a uniquely valuable strategic positioning.

Modern telecom networks are layered ecosystems composed of routing systems, subscriber management platforms, authentication services, billing systems, roaming databases, and lawful intercept capabilities. These systems rely on specialized signaling protocols such as SS7, Diameter, and SCTP to coordinate identity, mobility, and connectivity across national and international boundaries.

Persistent access within these environments enables far more than a conventional data breach. An adversary positioned inside the telecom core may gain visibility into subscriber identifiers, signaling flows, authentication exchanges, mobility events, and communications metadata. In the most concerning scenarios, this level of access could support long-term intelligence collection, large-scale subscriber tracking, and monitoring of sensitive communications involving high-value geopolitical targets.

Telecommunications networks sit at the intersection of identity, mobility, and global connectivity. Compromise at this layer carries national and international implications.

A structured campaign, not isolated incidents

What looks like discrete breaches increasingly resembles a repeatable campaign model designed to establish persistent access inside telecommunications infrastructure.

Our investigation uncovered a long-term and ongoing operation attributed to a China-nexus threat actor. Rather than conducting short-term intrusion activity, the operators appear focused on long-term positioning by embedding stealthy access mechanisms deep inside telecom and critical environments and maintaining them for extended periods.

In effect, attackers are placing sleeper cells inside the telecom backbone: dormant footholds positioned well in advance of operational use.

Across investigations and public reporting, we observe recurring elements: kernel-level implants, passive backdoors, credential-harvesting utilities, and cross-platform command frameworks. Together, these components form a persistent access layer designed not simply to breach networks, but to inhabit them.

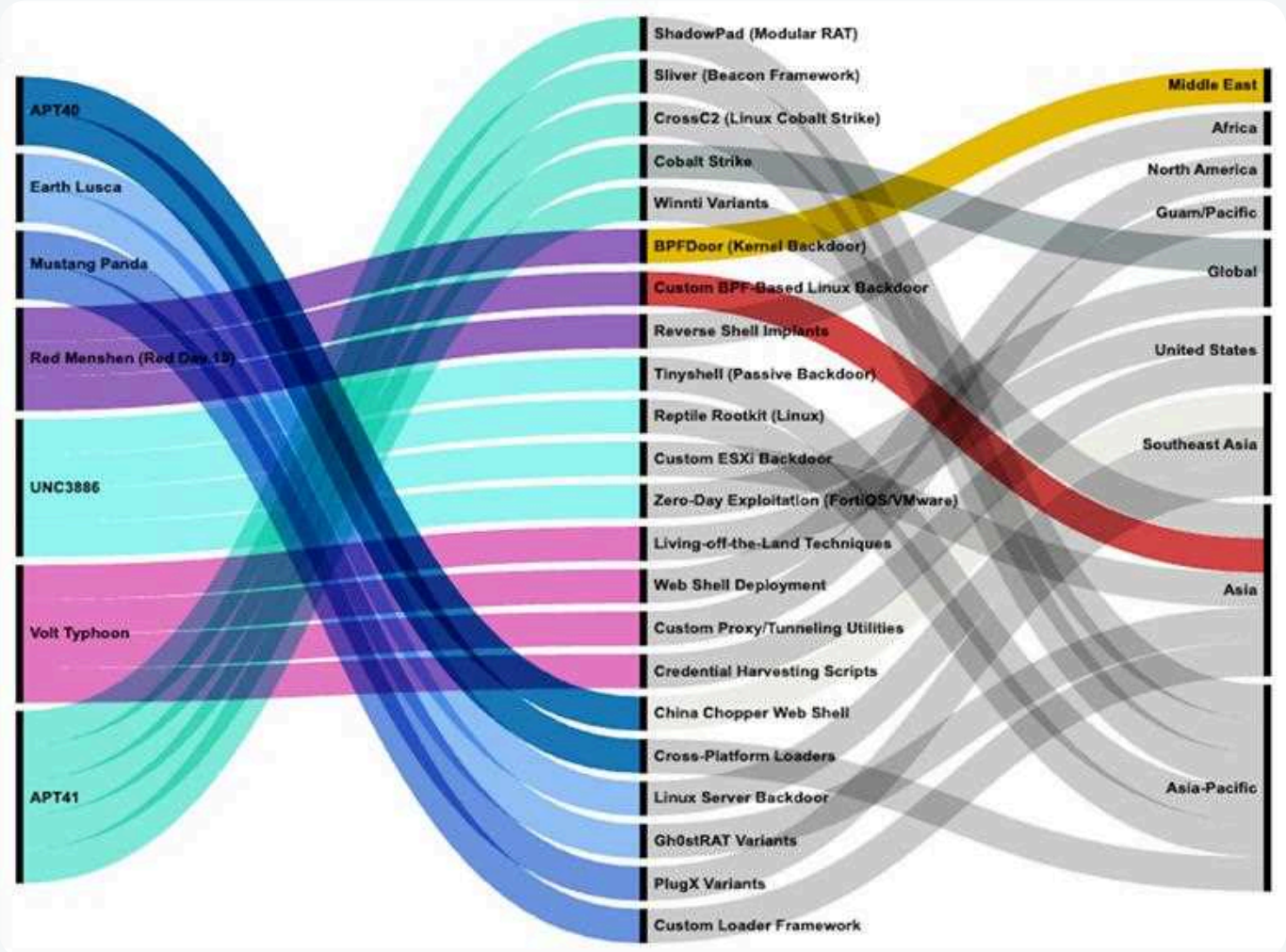

Figure 1: Actors, tools and regions in which specific threat groups target the telecom sector

How BPFdoor enables covert, deep-seated persistence

At the center of this activity is BPFdoor, a stealth Linux backdoor engineered to operate within the operating system kernel.

Unlike conventional malware, BPFdoor does not expose listening ports or maintain visible command-and-control channels. Instead, it abuses Berkeley Packet Filter (BPF) functionality to inspect network traffic directly inside the kernel, activating only when it receives a specifically- crafted trigger packet. There is no persistent listener or obvious beaconing. The result is a hidden trapdoor embedded within the operating system itself.

This approach represents a shift in stealth tradecraft. By positioning below many traditional visibility layers, the implant significantly complicates detection, even when defenders know what to look for.

Our research indicates BPFdoor is not an isolated tool, but part of a broader intrusion model targeting telecom environments at scale.

How attackers gain initial access to telecom environments

These findings reflect a broader evolution in adversary tradecraft. Attackers are embedding implants deeper into the computing stack — targeting operating system kernels and infrastructure platforms rather than relying solely on user-space malware.

Telecom environments — combining bare-metal systems, virtualization layers, high-performance appliances, and containerized 4G/5G core components — provide ideal terrain for low-noise, long-term persistence. By blending into legitimate hardware services and container runtimes, implants can evade traditional endpoint monitoring and remain undetected for extended periods.

For defenders, the implications are significant. Many organizations lack visibility into kernel-level operations, raw packet-filtering behavior, and anomalous high-port network activity on Linux systems. Addressing this threat requires expanding defensive visibility beyond the traditional perimeter to include deeper inspection of operating system behavior and infrastructure layers.

Sharing intelligence responsibly

Our investigation to identify potential victims is ongoing and, where potential compromise has been discovered, we have notified affected parties through relevant authorities or direct communication with our customers.

As part of our responsible research process, we have collaborated with government partners and national CERTs to share findings and indicators associated with this activity. When our analysis identified infrastructure that may have been impacted, we proactively notified the relevant organizations and provided detection guidance to assist with investigation and response while the research was still underway.

Rapid7 Intelligence Hub customers have access to the full technical details and indicators of compromise within the platform, including Surricata rules. Those rules are also available through AWS Marketplace, where we offer our curated AWS firewall rule sets.

Technical analysis

The sections that follow examine how modern telecommunications networks are structured, how initial access is established, and how BPFdoor and related tooling enable infrastructure-level persistence inside the telecom backbone.

Modern telecom network structure

To understand why telecom environments are such attractive strategic targets, it helps to visualize their layered architecture (Figure 2). At the outer edge sit customer-facing services and access infrastructure: mobile base stations (RAN), fiber aggregation routers, broadband gateways, DNS services, SMS-controllers, roaming gateways, security appliances like firewalls, proxies, VPNs, and internet peering points. These edge systems connect into the operator’s IP core and transport

backbone, where high-capacity routers and switches move massive volumes of voice, data, and signaling traffic across regions and international borders.

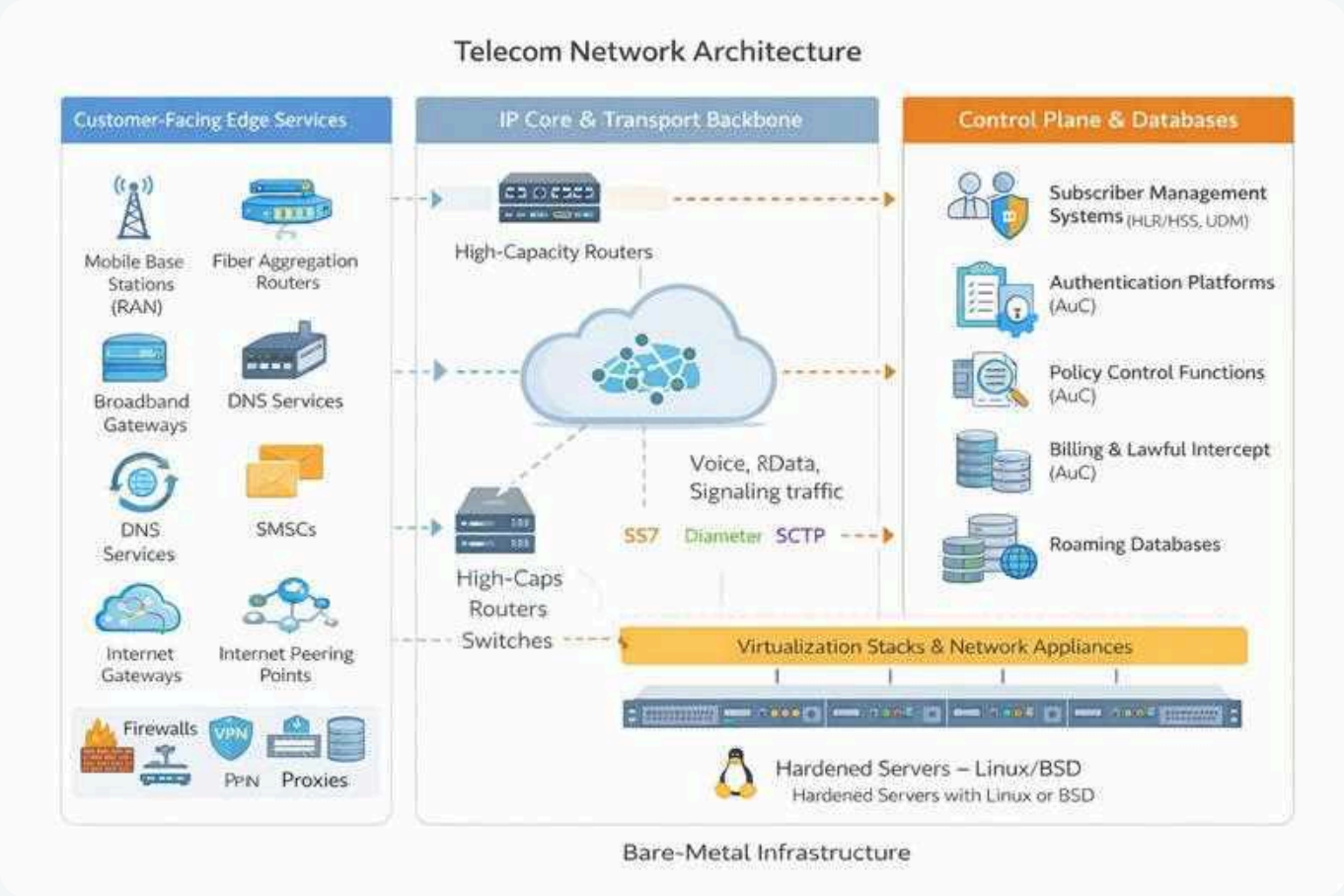

Figure 2: Simplified version of a telecom provider's network

Deeper inside lies the control plane, the heart of the telecom network, built around subscriber management systems such as HLR/HSS or UDM, authentication platforms (AuC), policy control functions, billing systems, lawful intercept platforms, and roaming databases. These systems communicate using specialized telecom signaling protocols such as SS7, Diameter, and increasingly SCTP-based signaling for LTE and 5G core components. At the foundation, much of this infrastructure ultimately runs on hardened, but often standard, Linux or BSD-based bare-metal servers, virtualization stacks, and high-performance network appliances. When an adversary implants a persistent backdoor at the kernel level within these environments, they are not simply compromising a server, they are positioning themselves adjacent to subscriber data, signaling flows, and the mechanisms that authenticate and route national and international communications.

Initial access

Telecom intrusions rarely begin deep inside the core. Instead, attackers focus on exposed edge services and internet-facing infrastructure. Techniques such as exploitation of public-facing applications (T1190) and abuse of valid accounts (T1078) are repeatedly observed. Devices commonly targeted include: Ivanti Connect Secure VPN appliances, Cisco IOS and JunOS network devices, Fortinet firewalls, VMware ESXi hosts, Palo Alto appliances, and even web-facing platforms like Apache Struts. These systems sit at the boundary between external traffic and internal telecom environments, making them high-value entry points. Once compromised, they provide authenticated

pathways into the provider's network, often without triggering traditional endpoint detection mechanisms.

Let's highlight some of the tools we observed during initial access and attempt to get more credentials for lateral movement.

CrossC2

Once initial access is secured, the operators frequently deploy Linux-compatible beacon frameworks such as CrossC2. This Cobalt Strike-derived loader enables beacon functionality on Linux hosts and has been repeatedly observed in PRC-aligned intrusion campaigns. It provides the same post-exploitation capabilities traditionally seen in Windows environments, command execution, pivoting, staging, but tailored for Linux-heavy telecom infrastructure. CrossC2 allows operators to blend into server environments that form the backbone of telecom operations, particularly edge devices and core routing systems. Just as with the Cross C2 configuration, investing reveals the C2 server. For example:

```
# Key C2 parameters (confirmed via live strace execution:

  c2_server_ip:  27.102.113.190
  c2_port:  8843
  c2_uri:   submit.php   // default URI for C2 traffic
  user_agent: "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.1
               (KHTML, like Gecko) Chrome/14.0.835.163"
  authentication:
    protocol: cookie
    header_key: beacon ID

  # fallback/test C2:
  backup_c2:  "0.0.0.0:8843
    backup_c2:  "0.0.0:8843 → 127.0.0.1
```

Figure 3: CrossC2 configuration

TinyShell

For long-term persistence, actors often rely on TinyShell, an open-source passive backdoor framework repurposed and customized by multiple APT groups. TinyShell is frequently observed on boundary devices such as firewalls, VPN appliances, and virtualization hosts. Compiled for Linux and FreeBSD, it is designed with stealth in mind: minimal network footprint, passive communication model, and reliable remote command execution capabilities.

Keyloggers and bruteforcers

After foothold establishment, attackers focus on persistence and lateral movement. Tooling such as Sliver, CrossC2, and TinyShell are complemented by SSH brute forcers and custom ELF-based keyloggers. In some cases, operators deploy brute-force utilities containing pre-populated credential lists tailored for telecom environments, even including specific usernames like “imsi,” referencing subscriber identity systems. This level of contextual awareness indicates reconnaissance and targeting aligned with telecom operational terminology. The goal is clear: move laterally, harvest credentials, and reach control-plane systems where subscriber data and signaling infrastructure reside.

BPFdoor

BPFdoor first came to broader public attention around 2021, when researchers uncovered a stealthy Linux backdoor used in long-running espionage campaigns targeting telecommunications and government networks. The BPFDoor source code reportedly leaked online in 2022, making the previously specialized Linux backdoor more accessible to other threat actors. Normally, BPF is used by tools like tcpdump or libpcap to capture specific network traffic, such as filtering for TCP port 443. It operates partly in kernel space, meaning it processes packets before they reach user-space applications.

BPFdoor abuses this capability. Rather than binding to a visible listening port, the implant installs a custom BPF filter inside the kernel that inspects incoming packets for a specific pattern, a predefined sequence of bytes often referred to as a “magic packet” or “magic byte.” If the pattern does not match, nothing happens. The traffic continues as normal. No open port or obvious process-accepting connections. But when the correct sequence is delivered to the correct destination port, the behavior changes instantly.

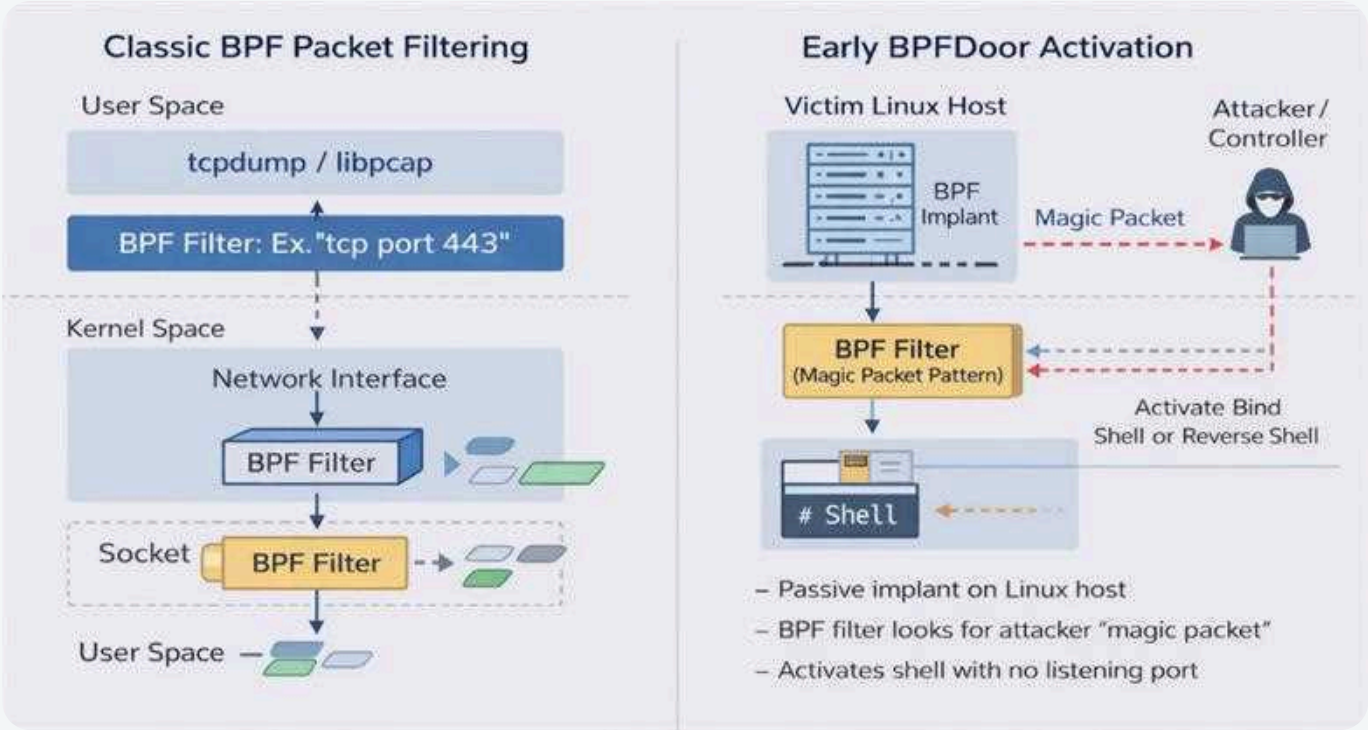

Figure 4: Overview of BPF and how early BPFdoor variants are operating

Imagine retrieving a parcel from a secure pickup locker. The locker sits quietly in public view, no alarms, no obvious signs of activity. It only opens when the correct code is entered.

BPFdoor behaves the same way.

The implant remains dormant inside the Linux kernel, passively inspecting network traffic. It does not advertise itself. It does not respond to scans. But when an operator sends the correct “code”, the specific magic byte sequence embedded in a crafted packet, the BPF filter recognizes the pattern and triggers the next stage.

Instead of opening a physical door, it spawns a bind shell or reverse shell. Importantly, this activation can occur without a traditional listening service ever being visible in netstat or ss. To a defender, the system appears clean; there is no persistent open port to detect.

Before we showcase this, something important to note is that BPFdoor operations consist of two distinct components: the implant and the controller.

The implant is the passive backdoor deployed on the compromised Linux system, where it installs a malicious BPF filter and silently inspects incoming traffic for a predefined “magic” packet. It does not continuously beacon or expose a listening port, making it extremely stealthy.

The controller, on the other hand, is operated by the attacker and is responsible for crafting and sending the specially formatted packets that activate the backdoor and establish a remote shell. While it can be run from attacker-controlled infrastructure such as compromised routers or external systems, the controller is also designed to operate within the victim’s environment itself. In this mode it can masquerade as legitimate system processes and trigger additional implants across internal hosts by sending activation packets or by opening a local listener to receive shell connections, effectively enabling controlled lateral movement between compromised systems. In essence, the implant acts as the hidden lock embedded within the system, while the controller functions as the key that can activate it. A deeper technical analysis of the controller architecture and its role in lateral movement will be covered in a forthcoming technical blog.

To demonstrate how these first backdoors work, we created the video below, in which we are running a BPFdoor made visible. Next, we send the magic packet and instructions to the IP address and port we are listening on. Then the BPFdoor opens up the “safe” and creates the tunnel. In the final part of the demo, we see that on our Netcat listener, we have a remote shell and can query the system.

Next, we will highlight how we started to hunt for BPFdoor.

Hunting for BPFdoor variants

Since we were aware of several BPFdoor attacks and samples circulating, we started hunting for more samples and developed internal tools to extract, compare, and detect early indicators of new features. One threat hunting angle Rapid7 Labs really loves to focus on is code similarity of samples. Code similarity of malware samples can result in clusters of samples with similar activity, but most importantly, also demonstrate outliers that are potential candidates for research since they do not share commodity with the other samples.

The BPFdoor samples we collected and hunted for are all Executable and Linkable Format (ELF) files, but we are aware of samples compiled for running on Solaris. ELF is the standard binary file format for executables, object code, shared libraries, and core dumps on Linux and Unix-like operating systems. For the ELF files, we wrote a custom tool for clustering ELF/BPFdoor. By extracting .text section byte code blocks, generating MinHash signatures, and completing a few other steps, it will then compute exact Jaccard similarity and export the resulting similarity graph for visual cluster analysis.

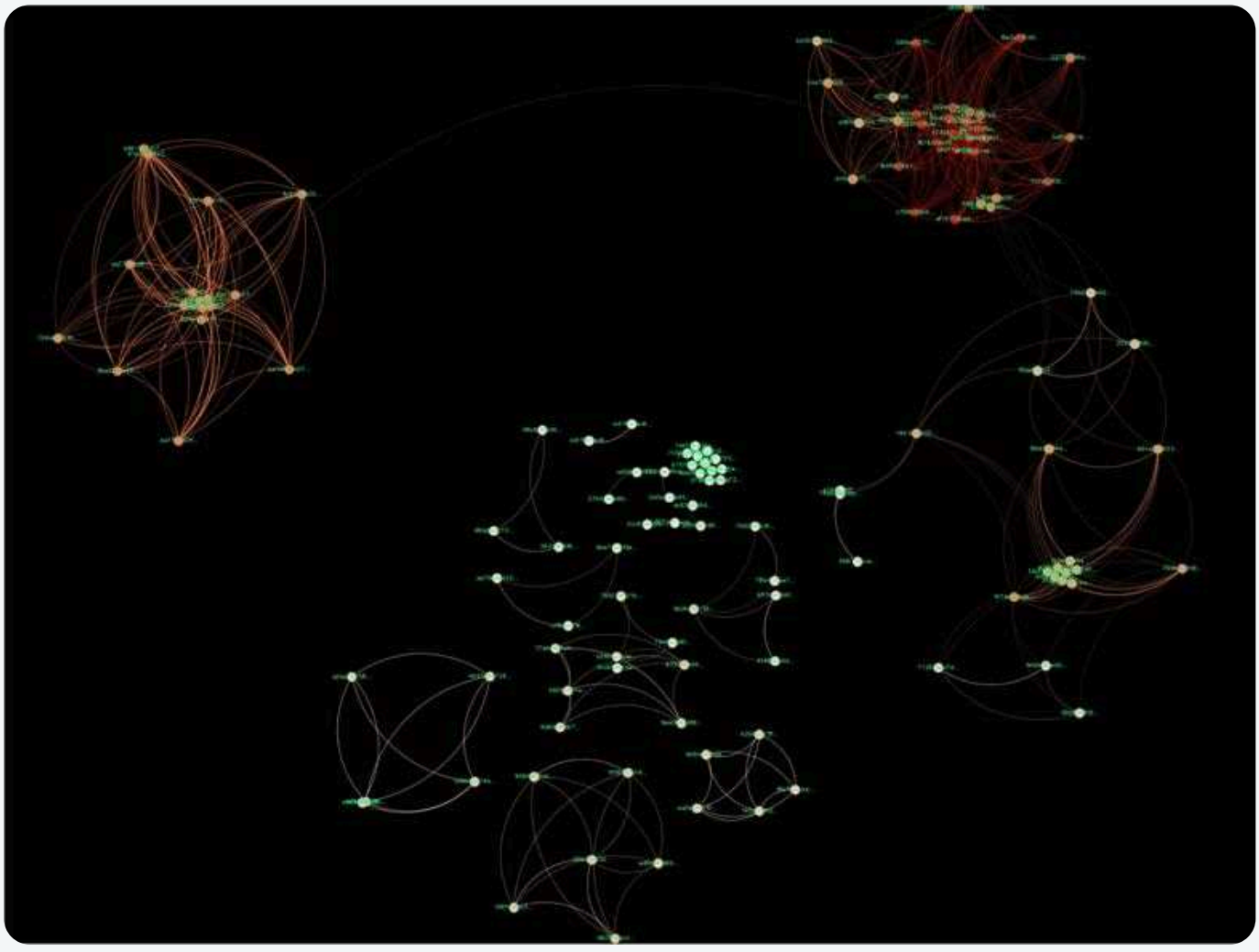

Figure 5: Code Similarity clustering of BPFdoor samples

In our visualization, we clearly observe certain clusters of BPFdoor, but also outliers and smaller clusters that were up for investigation. The thicker the line, the more similar the code is to the samples it is attached to. By creating a feature comparison/extraction tool, we started to discover interesting features in the samples, which led us to a new controller discovery and security bypass feature. For example, we discovered a variant we dubbed “F” that uses a 26 BPF instruction filter with new magic packets.

Although it was previously reported that some samples support the Stream Control Transmission Protocol (SCTP), there is a tendency to read over it and not put it into the right context of what the consequences are. SCTP is not typical enterprise traffic; it underpins Public Switch Telephone Network (PSTN) signaling and real-time communication between core 4G and 5G network elements. By configuring BPF filters to inspect SCTP traffic directly, operators are no longer just maintaining server access, they are embedding themselves into the signaling plane of the telecom network. This is a fundamentally different level of positioning. Instead of sitting at the IT perimeter, the implant resides adjacent to the mechanisms that route calls, authenticate devices, and manage subscriber mobility.

```
0x00435760:  ldh  [0xc]              ; Load EtherType at offset 12
             jeq  0x86dd, +9         ; Check if IPv6 (0x86dd)

0x00435770:  ldb  [0x14]             ; Load Next Header at offset 20
             jeq  0x6, +2            ; Check if TCP (6)

0x00435780:  ldh  [0x38]             ; Load port at offset 56
             jeq  0x50, +13          ; Check port 80
             jeq  0x2c, +1          ; Check port 44

0x00435798:  jeq  0x84, +1           ; CHECK IF SCTP (132) <- KEY FINDING
             jeq  0x11, +0x14       ; Check if UDP (17)

0x004357a8:  ldh  [0x38]             ; Load port at offset 56
             jeq  0x1bb, +0x10      ; Check port 443
             jeq  0x800, +0x11      ; Check if IPv4 (0x800)

0x004357c0:  ldb  [0x17]             ; Load byte at offset 23
             jeq  0x6, +6           ; Check if TCP (6)

0x004357d0:  ldh  [0x14]             ; Load port at offset 20
             jset 0x1fff, +0xd      ; Check if high port

0x004357e0:  (invalid)
             ldh  [r0+0x10]         ; Load half-word

0x004357f0:  jeq  0x50, +9           ; Check port 80
             jeq  0x1bb, +7         ; Check port 443

0x00435800:  jeq  0x84, +1           ; CHECK IF SCTP (132) <- SECOND SCTP CHECK
             jeq  0x11, +7          ; Check if UDP (17)
```

Figure 6: Example of SCTP route extracted from the BPF code

Access to SCTP traffic opens powerful intelligence collection opportunities. In legacy and transitional environments, improperly secured signaling can expose SMS message contents, IMSI identifiers, and source/destination metadata. By observing or manipulating traffic over SCTP commands such as ProvideSubscriberLocation or UpdateLocation, an adversary can track a device's real-world movement. In 5G environments, traffic over SCTP carries registration requests and Subscription Concealed Identifiers (SUCI), allowing identity probing at scale. At this point, the compromise is no longer about server persistence; it becomes population-level visibility into subscriber behavior and location. Translated, you could track individuals of interest.

Interesting observations

The bare-metal to telecom equipment link

During the code investigations, we discovered that some BPFdoor samples are using code to mimic the bare-metal infrastructure, particularly enterprise-grade hardware platforms commonly deployed in telecom environments. By masquerading as legitimate system services that run only on bare metal, the implant blends into operational noise. This is especially relevant in environments leveraging HPE ProLiant and similar high-performance compute systems used for 5G core and edge deployments.

```
strcpy(v7, "73b9989bb8dd522b8e172f2e985810eb");
strcpy(v6, "d46bf5d43cffd7793665d40fc767ed86");
src = "hpasmlited -f /dev/hpilo";
file = "/var/run/hpasmlited.pid";
if ( !access("/var/run/hpasmlited.pid", 4) )
  exit(0);
if ( !getuid() )
{
  bzero(&unk_78ABE0, 0x24Au);
  v4 = time(0);
  srand(v4);
  strcpy(byte_78ABE8, src);
  strcpy(s1, v7);
  strcpy(s, v6);
  set_proc_name(a1, a2, byte_78ABE8);
  daemon(0, 0);
  chdir("/");
  init_signal();
  signal(17, (__sighandler_t)b_sub_4028A6_sinal_child);
  dword_78ABB0 = getpid();
  v5 = open(file, 65, 420);
  close(v5);
  signal(17, (__sighandler_t)1);
  pthread_create(&th, 0, (void *(*)(void *))start_routine, 0);
  pthread_create(&qword_78ABC0, 0, (void *(*)(void *))sub_4089BB, 0);
  pthread_create(&qword_78ABC8, 0, (void *(*)(void *))sub_4084F7, 0);
  pthread_join(th, 0);
  pthread_join(qword_78ABC0, 0);
  pthread_join(qword_78ABC8, 0);
}
```

Figure 7: Example of code mimicking HP Proliant servers

In the above screenshot of one of the BPFdoor samples, we observed the processname *"hpaslimited".*

By mimicking legitimate service names and process behavior of HPE ProLiant servers, attackers ensure the implant appears native to the hardware environment, a tactic that significantly complicates detection. Several of these service names have been observed in BPFdoor samples, but this name stood out. The *hpasmlited.pid* creates process threads, and mimics daemon-style behavior consistent with hardware monitoring services. The real *hpasmlited* process belongs to HPE's Agentless Management Service, which runs on bare-metal ProLiant servers to expose hardware telemetry and system health data.

By adopting this name and writing a corresponding PID file, the malware blends into expected operational noise on telecom-grade ProLiant infrastructure. Of course this is not accidental naming, it demonstrates environment awareness and targeting intent. The operators appear to know they are running on physical HPE hardware commonly deployed in 4G/5G core and edge systems. By impersonating a trusted hardware management daemon that administrators expect to see, the implant reduces suspicion during forensic review while embedding itself directly into the physical backbone layer of telecom infrastructure. This tactic reflects a broader strategy: hide not just in Linux, but in the hardware identity of the telecom environment itself.

Mimicking containers

A second strategy involves spoofing core containerization components. Critical 5G core components such as the Access and Mobility Management Function (AMF), Session Management Function (SMF), and User Data Management (UDM) run as cloud native network functions inside Kubernetes pods. The following code excerpt demonstrates that the implant is aware of it.

```
v10 = __readfsqword(0x28u);
strcpy(v8, "berkeley");
strcpy(v9, "packet filter");
src[0] = "/usr/lib/systemd/systemd-journald";
src[1] = "/usr/lib/systemd/systemd-logind";
src[2] = "dbus-daemon --system";
src[3] = "/sbin/agetty -o -p --noclear tty1 linux";
src[4] = "/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock";
strcpy(&pid_path, "/var/run/haldrund.pid");
if ( !access(&pid_path, 4) )
  exit(0);
if ( getuid() )
  return 0;
if ( argc == 1 )
{
  if ( !(unsigned int)to_open(*argv, "containerd") )
    _exit(0);
   exit(-1);
```

Figure 8: Code showing the mimicking of container/docker service

Docker Daemon (/usr/bin/dockerd) and containerd: The malware is executed with root privileges and adopts the exact command-line arguments of a legitimate Docker daemon (e.g., -H fd:// --containerd=/run/containerd/containerd.sock).

Recap for a moment

Up to this point, what we've described in our technical analysis has, more or less, been publicly available information; however, these pieces have not been assembled in a way that provides the context Rapid7 Labs has discovered through its in-depth investigation. Therefore, before we deep dive into some of the new technical findings that completes the picture of what is truly happening here, let's pause for a moment to sync up on what we've just described.

So far, our findings illustrate that BPFdoor is far more than a stealthy Linux backdoor. The kernel-level packet filtering, passive activation through magic packets, masquerading as legitimate hardware management services, awareness of container runtimes, and the ability to monitor telecom-native protocols such as SCTP, point to a tool designed for deep infrastructure positioning. Rather than targeting individual servers, the operators appear to focus on the underlying platforms that power modern telecommunications networks: bare-metal systems running telecom workloads, cloud-native Kubernetes environments hosting Containerized Network Functions, and the signaling protocols that coordinate subscriber identity, mobility, and communication flows. In this context, BPFdoor functions as an access layer embedded within the telecom backbone, providing long-term, low-noise visibility into critical network operations.

What Rapid7 found in newer BPFdoor variants

The following sections provide a high-level overview of several newly observed capabilities and behavioral patterns in recent BPFdoor samples. While these findings highlight important technical developments, this blog intentionally focuses on the architectural implications and operational context rather than a full reverse-engineering deep dive. Detailed technical analyses, including code-level breakdowns, will be published in upcoming research posts.

During our investigation, we identified a previously undocumented variant of BPFdoor that introduces several architectural changes designed to improve stealth and survivability in modern enterprise and telecom environments. We will highlight these features and illustrate how the malware continues to evolve beyond the earlier "magic packet" activation model.

Network-level invisibility: The BPF trapdoor

As we described before, the early BPFdoor installed a Berkeley Packet Filter inside the Linux kernel that inspected incoming network traffic. When a specially crafted "magic packet" containing a predefined byte sequence arrived at the correct port, the backdoor would activate and spawn a shell. Because the system never actually opened a port, tools such as netstat, ss, or nmap saw nothing unusual.

The newly observed variant evolves this concept. Instead of relying on a simple magic packet that could potentially be detected by intrusion detection signatures, the trigger is now embedded within seemingly legitimate HTTPS traffic. The attacker sends a carefully crafted request that travels through standard network infrastructure such as reverse proxies, load balancers, or web application firewalls.

Once the traffic reaches the compromised host and is decrypted as part of normal SSL termination, the hidden command sequence can be extracted and used to activate the backdoor. In essence, in our previously mentioned analogy explaining the magic packet mechanism, the safe still requires a code, but now the code is concealed inside normal, encrypted web traffic, allowing it to pass through modern security controls before unlocking the trapdoor.

Figure 9: Overview of how the new sample communicates

Layer 7 camouflage and the “magic ruler”

To remain reliable across proxy layers, the attackers introduced a clever parsing mechanism. HTTP proxies often modify headers by inserting additional fields such as client IP addresses, timestamps, or routing metadata. These changes can shift the position of data within the request and break traditional signature-based triggers. To solve this problem, the attackers designed a mathematical padding scheme that ensures a specific marker, in the observed samples the string *“9999”*, always appears at a fixed byte offset within the request.

This is where the 26-byte or 40-byte “magic ruler” comes into play. Rather than parsing the entire HTTP header, which can vary depending on proxy behavior, the malware treats the request body as a predictable coordinate space. By carefully padding the HTTP request with filler bytes, the attacker ensures that the marker always lands exactly at the 26th byte offset of the inspected data structure. The implant simply checks this fixed position; if the marker appears at that byte location, it interprets the surrounding data as the activation command.

Because the header itself can fluctuate while the padded payload remains predictable, the malware does not need to understand or parse the full HTTP structure. Instead, it relies on this fixed “measurement point”, effectively using the 26-byte offset as a ruler inside the packet. This technique allows the trigger to survive proxy rewriting and header injection while still remaining hidden inside otherwise normal HTTPS traffic. The 26-byte rule is used in case of a socket creation with the “SOCK_DGRAM” flags, but in case of a “SOCK_RAW” flag, it will use a 40-byte ruler.

In practice, this turns the messy, variable HTTP protocol into something the malware can treat like a fixed coordinate system, enabling what could be described as dynamic Layer-7 camouflage, a surprisingly simple but effective technique for hiding command triggers inside legitimate encrypted web traffic.

The RC4-MD5 paradox

Another interesting feature of the new controller is its continued use of the legacy RC4-MD5 encryption routine. While this combination is considered deprecated in modern cryptographic standards, it still appears in several malware samples. In this case, the RC4-MD5 implementation is not part of TLS, but rather a lightweight encryption layer applied to the interactive command-and-control channel after the backdoor is activated. RC4 provides extremely fast stream encryption suitable for interactive shells, introducing minimal latency during command execution. In addition, the use of older or non-standard encryption routines can sometimes confuse inspection systems, particularly when traffic does not follow typical protocol expectations. Finally, reuse of older cryptographic modules often reflects code lineage and operational efficiency, adversaries frequently recycle proven components across campaigns. In this case, code comparison revealed similarities with routines that have circulated in Chinese-nexus malware families such as RedXOR and PWNIX for several years.

ICMP control channel: “phone home”

While earlier BPFdoor variants focused primarily on covert activation, the new sample also introduces a lightweight communication mechanism built around Internet Control Message Protocol (ICMP). The code excerpt shows the malware preparing an ICMP payload and inserting a specific value *“0xFFFFFFFF”* into a field before transmitting the packet using a dedicated routine (*send_ICMP_data*). At first glance this appears trivial, but the logic reveals something more interesting: The ICMP packet is not just a signal back to the operator, it is also used as a control mechanism between compromised systems.

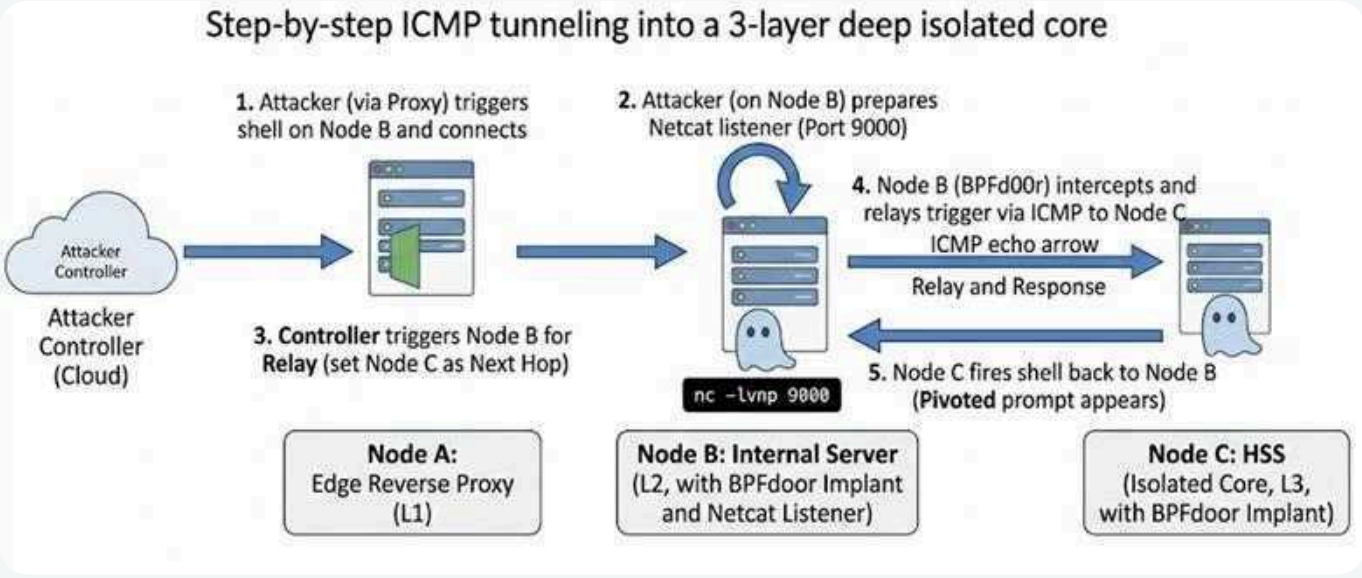

Figure 10: ICMP Tunneling

In this model, ICMP functions as a minimal command channel between infected hosts. One compromised server can forward specially crafted ICMP packets to another, effectively passing along execution instructions without requiring traditional command-and-control traffic. The key marker in this mechanism is the value 0xFFFFFFFF (signed as -1), which acts as a destination signal embedded inside the packet structure. When a receiving host detects this value, it interprets the packet as a terminal instruction rather than something to be forwarded further.

In practical terms, *Server A is telling Server B: "You are the final destination."* Instead of relaying the signal onward, the receiving system executes the next stage, typically triggering the reverse shell or command handler. This simple signaling mechanism allows the operators to control how far a command propagates through compromised infrastructure without introducing additional protocol complexity.

What makes this mechanism notable is its simplicity. Rather than expanding the structure of the activation packet or introducing additional fields, the attackers reuse an existing value within the packet structure to signal the end of the chain. By setting this field to 0xFFFFFFFF, they effectively create a "do not forward" flag inside their communication channel. This allows them to manage hop behavior across compromised nodes while keeping the packet format compact and consistent.

Key takeaways

Taken together, the newly observed capabilities demonstrate how BPFdoor has evolved beyond a stealth backdoor into a layered access framework. The updated variant combines encrypted HTTPS triggers, proxy-aware command delivery, application-layer camouflage techniques, ICMP-based control signals, and kernel-level packet filtering to bypass multiple layers of modern network defenses. Each technique targets a different security boundary, from TLS inspection at the edge, to IDS detection in transit, and endpoint monitoring on the host, illustrating a deliberate effort to operate across the full defensive stack.

Kernel-level backdoors are redefining stealth.
Tools like BPFdoor operate below traditional visibility layers, abusing Berkeley Packet Filter mechanisms to create network listeners that do not expose ports, processes, or conventional command-and-control indicators.

Telecommunications infrastructure is a prime espionage target.
Modern 4G and 5G networks rely on complex stacks of signaling systems, Containerized Network Functions, and high-performance infrastructure. Access to these environments can enable long-term intelligence collection, subscriber monitoring, and deep visibility into national communications infrastructure.

Security controls can be turned into delivery mechanisms.
In the latest BPFdoor variant, attackers weaponize normal security workflows. Traffic that passes through TLS termination and deep packet inspection can deliver malicious commands once it reaches the decrypted internal zone.

BPF-based implants are likely the beginning of a larger trend.
BPFdoor and new eBPF malware families like Symbiote demonstrate how kernel packet filtering can

be abused for stealth persistence. As defenders improve visibility at higher layers, adversaries are increasingly shifting implants deeper into the operating system.

How defenders can detect BPFdoor activity

Detecting these threats requires shifting visibility deeper into the operating system and network stack, focusing on indicators such as unusual raw socket usage, anomalous packet filtering behavior, and unexpected service masquerading on critical infrastructure hosts.

To support defenders in identifying potential BPFdoor activity, we developed a scanning script designed to detect both previously documented variants and the newer samples discussed in this research. The script focuses on identifying indicators associated with the stealth activation mechanism, kernel-level packet filtering behavior, and process masquerading techniques used by BPFdoor implants. By combining checks for known artifacts and behavioral patterns, the scanner helps security teams quickly assess whether systems may be impacted.

We are making this tool available to the community to assist organizations in proactively identifying potential compromises. The scanner can be used across Linux environments to search for artifacts linked to BPFdoor activity, including indicators observed in both historical samples and the latest variant analyzed during this research. Our goal is to help defenders rapidly validate exposure and begin incident response investigations where necessary.

In the video below, Rapid7 Labs demonstrates how our detection script would be run within the system of an infected victim organization. The video starts with the right window, showing that the BPFdoor backdoor is running and the particular services that relate are highlighted. Then, in the bottom left screen, the BPFdoor is activated by sending the right packet sequence and password, whereby a remote control shell is established. The attacker is running some commands on the victim machine and shows it can execute remote commands. Finally, in the top window, we run our developed detection script that will show the detected processes, and the alerts are showcased.