Sandfly Security®

BPFDoor Detection and Analysis

2025-07-15

BPFDoor Introduction

BPFDoor is a simple but stealthy Linux backdoor linked to Chinese nation state threat actors. While often found targeting telecommunications infrastructure, it is likely used in other critical infrastructure breaches around the world. This document details detection of BPFDoor versions 1 and 2 by Sandfly Security's agentless intrusion detection and incident response platform.

We **strongly urge** customers to let Sandfly do agentless hunts for this threat to avoid missing any compromised systems. Sandfly works across nearly all Linux distributions including modern systems, 10+ year old legacy systems, and even embedded devices without deploying endpoint agents. Sandfly can find BPFDoor in seconds saving countless hours of time using risky scripts and manual work.

BPFDoor Technical Overview

The original BPFDoor has existed for some time and was able to remain unnoticed due to the low-key nature of how it worked and the lack of monitoring on most Linux systems. Sandfly wrote an in-depth technical overview of BPFDoor back in 2022. The complete technical breakdown of what it did is outlined in our original research below:

BPFDoor - An Evasive Linux Backdoor Technical Analysis

We also made a video presentation about the operation of this backdoor along with slides:

BPFDoor Presentation
BPFDoor Slides

The above analysis covered Version 1 of this backdoor. Version 2 of BPFDoor was recently discussed by Haxrob in his two part blog:

BPFDoor - Part 1 The Past
BPFDoor - Part 2 The Present

The two backdoors have largely similar functions with changes to evade some detection in the Version 2 variants. The main change for Version 2 is that the new versions incorporate much stronger public/private key encryption. The new version's stronger encryption prevents network monitoring and also unauthorized use by anyone except the operators that installed it.

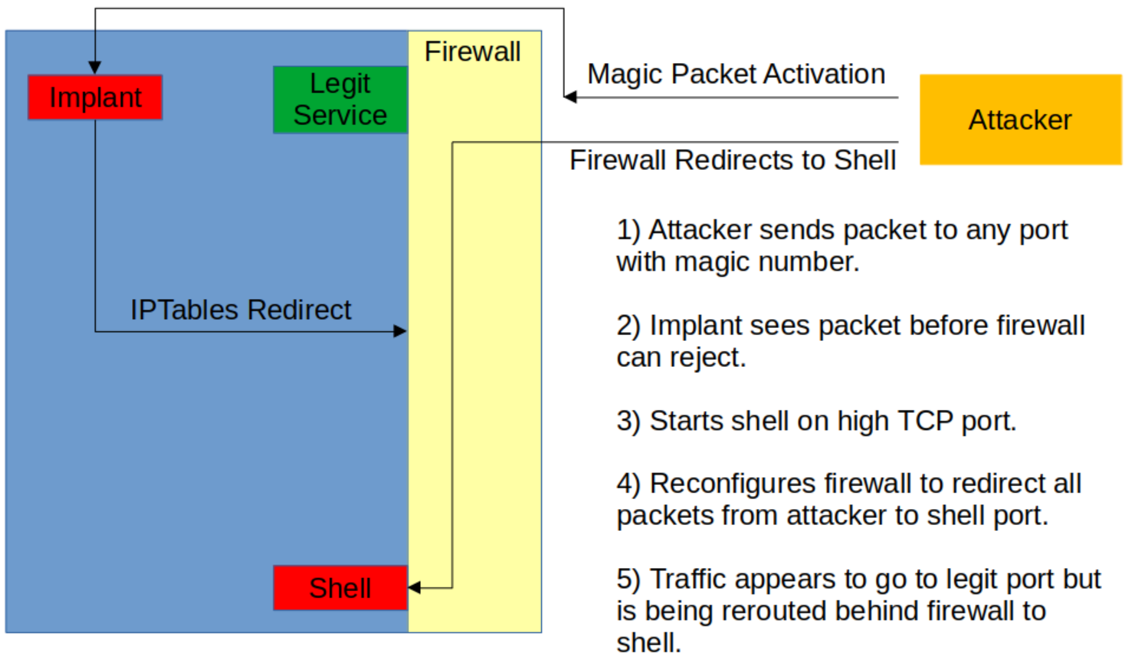
BPFDoor Basic Operation

BPFDoor waits for a "magic packet" to arrive on any port of the victim system once installed. The magic packet is a specially crafted network packet on the TCP, UDP or ICMP protocols that contains a string sequence and password to activate the backdoor. Without any magic packet received, the backdoor sits quietly using negligible system resources to not draw any attention to itself.

Once a magic packet is seen by the backdoor, it will reconfigure the local firewall to either start a bind shell backdoor that the attacker then connects to, or start a reverse shell back to the attacker. More critically, the attacker can communicate on the port it sent the original packet on. Meaning that if the victim is running a webserver on port 443 with encrypted traffic, the attacker can send a packet to port 443 and start an encrypted backdoor session to blend in with other traffic. Any port can be used in this way.

The important thing to understand is that it **does not matter** if the system has a local firewall configured to drop unauthorized packets. The backdoor will intercept the packets before the firewall has dropped them and activate. The firewall will not prevent the backdoor from activating once it sees a packet.

The above point is significant because a system that operators think is protected against unauthorized traffic by a firewall can in fact be accessed. The diagram below shows the basic operation of this mechanism for the bind shell backdoor from Version 1. If a reverse shell is requested instead of a bind shell, the backdoor will initiate a connection potentially bypassing packet filters that do not restrict traffic outbound.


Firewall
Implant
Legit Service
Magic Packet Activation
Attacker
Firewall Redirects to Shell
1) Attacker sends packet to any port with magic number.
IPTables Redirect
2) Implant sees packet before firewall can reject.
3) Starts shell on high TCP port.
4) Reconfigures firewall to redirect all packets from attacker to shell port.
Shell
5) Traffic appears to go to legit port but is being rerouted behind firewall to shell.

In both cases, the attacker can hit any open or closed port and cause the backdoor to activate. The attacker can hit an open port, such as a webserver, to blend into normal traffic. Or, they can hit a closed port that may not be monitored and get access that way.

Detecting BPFDoor

BPFDoor is a simple but effective backdoor because it limits its features to the bare minimum for the job, and incorporates simple hiding methods that are reliable across Linux distributions. However, it has several attributes that lend itself to reliable detection:

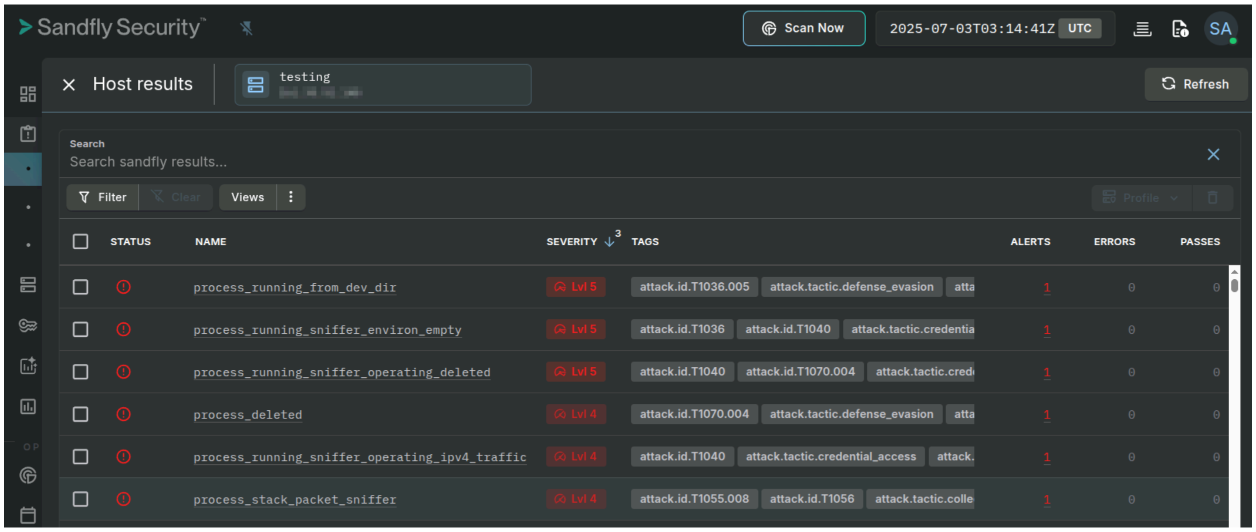
1) It is sniffing network traffic.
2) It masquerades the process name to hide.
3) It utilizes anti-forensics to conceal activity.
4) It launches command shells in suspicious or unusual ways.

We're going to now show you how Sandfly finds the above in all known variants of this backdoor.

Sandfly BPFDoor Alerts

Sandfly focuses on the tactics of compromise which makes our detection on Linux more versatile than standard malware signatures. In the case of BPFDoor, we do not try to identify it directly. Instead, we find the tactics of how it works which makes it obvious something serious is happening on the targeted system.

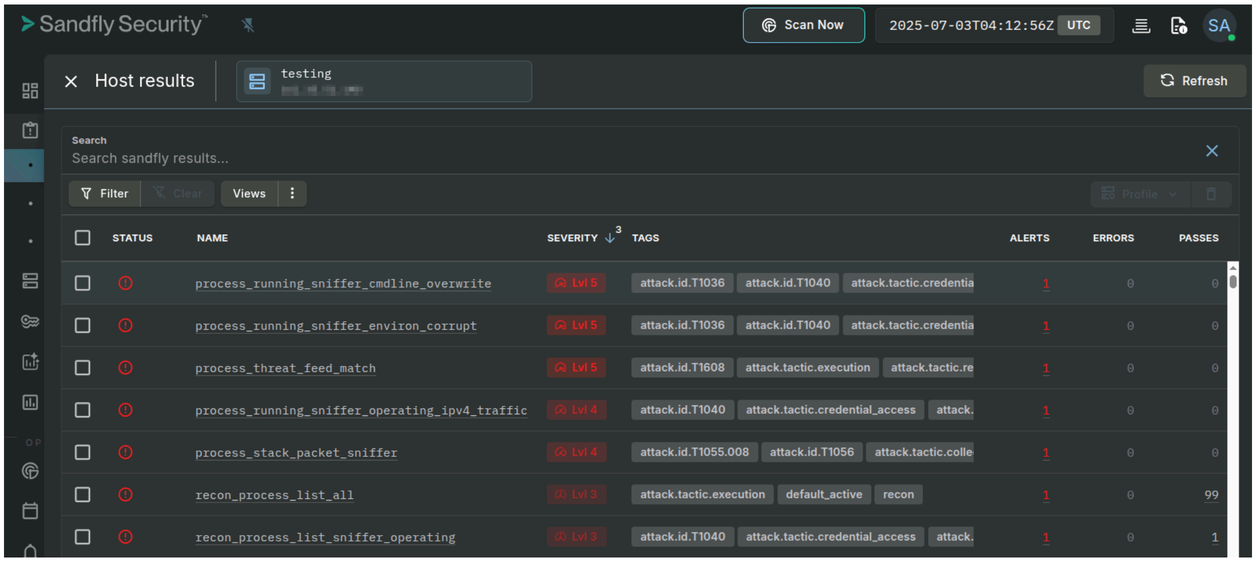
For example, we have Version 1 and 2 of BPFDoor operating on victim hosts. This is what Sandfly alerts on when the backdoor is running but idle (e.g. not running an active shell yet).



testing [redacted]

STATUS	NAME	SEVERITY	TAGS	ALERTS	ERRORS	PASSES
❗	process_running_from_dev_dir	Lvl 5	attack.id.T1036.005 attack.tactic.defense_evasion atta	1	0	0
❗	process_running_sniffer_environ_empty	Lvl 5	attack.id.T1036 attack.id.T1040 attack.tactic.credentia	1	0	0
❗	process_running_sniffer_operating_deleted	Lvl 5	attack.id.T1040 attack.id.T1070.004 attack.tactic.cred	1	0	0
❗	process_deleted	Lvl 4	attack.id.T1070.004 attack.tactic.defense_evasion atta	1	0	0
❗	process_running_sniffer_operating_ipv4_traffic	Lvl 4	attack.id.T1040 attack.tactic.credential_access attack.	1	0	0
❗	process_stack_packet_sniffer	Lvl 4	attack.id.T1055.008 attack.id.T1056 attack.tactic.colle	1	0	0

BPFDoor Version 1 Alerts



testing [redacted]

STATUS	NAME	SEVERITY	TAGS	ALERTS	ERRORS	PASSES
❗	process_running_sniffer_cmdline_overwrite	Lvl 5	attack.id.T1036 attack.id.T1040 attack.tactic.credentia	1	0	0
❗	process_running_sniffer_environ_corrupt	Lvl 5	attack.id.T1036 attack.id.T1040 attack.tactic.credentia	1	0	0
❗	process_threat_feed_match	Lvl 5	attack.id.T1608 attack.tactic.execution attack.tactic.re	1	0	0
❗	process_running_sniffer_operating_ipv4_traffic	Lvl 4	attack.id.T1040 attack.tactic.credential_access attack.	1	0	0
❗	process_stack_packet_sniffer	Lvl 4	attack.id.T1055.008 attack.id.T1056 attack.tactic.colle	1	0	0
❗	recon_process_list_all	Lvl 3	attack.tactic.execution default_active recon	1	0	99
❗	recon_process_list_sniffer_operating	Lvl 3	attack.id.T1040 attack.tactic.credential_access attack.	1	0	1

BPFDoor Version 2 Alerts

Sandfly Security®

BPFDoor Detection and Analysis

BPFDoor has many significant and serious alerts when idle:

process_running_from_dev_dir - A process running from */dev* directory.

process_deleted - A process with a deleted binary has been detected on the host.

process_running_sniffer_environ_empty - A network sniffing process is missing its environment.

process_running_sniffer_environ_corrupt - A network sniffing process has a corrupt environment.

process_running_sniffer_operating_deleted - A process binary that is sniffing traffic has been deleted from the disk.

process_running_sniffer_operating_ipv4_traffic - A process is grabbing all IPv4 traffic.

process_stack_packet_sniffer - A process with evidence it is operating as a sniffer in the stack has been detected.

process_running_sniffer_cmdline_overwrite - A process with a suspicious command line has been detected.

recon_process_list_all - Drift detection has found a new process that is not authorized to be running on this host.

recon_process_list_sniffer_operating - Drift detection has found a new process that is sniffing network traffic running on this host.

process_threat_feed_match - A known malware binary hash from a threat feed has been found operating on this host.

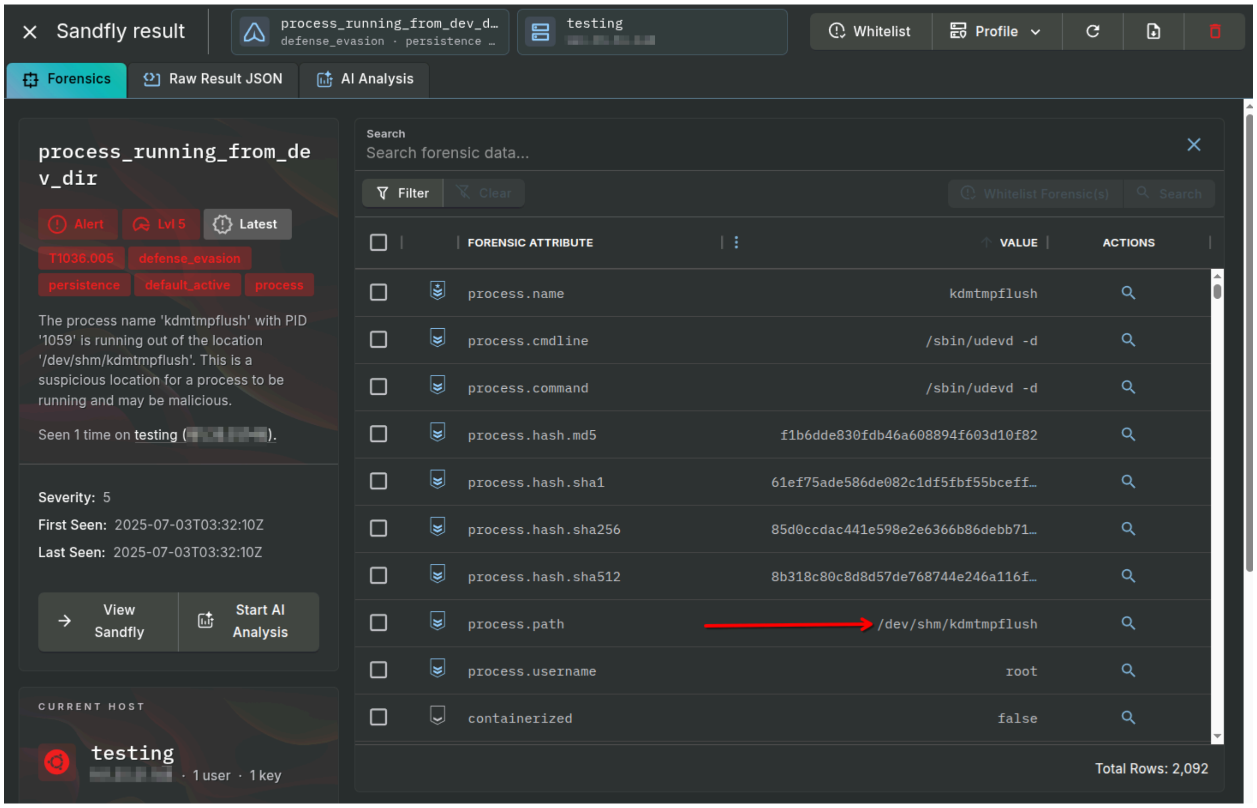
Suspicious Process Paths

The detection *process_running_from_dev_dir* is related to a series of checks Sandfly does to find processes that are running from suspicious locations on Linux. In general, Linux processes tend to run from system areas such as */bin*, */usr/bin*, or user installed software areas such as */opt* or similar.

The Version 1 of BPFDoor however chose to run from */dev/shm* which is a ramdisk area on Linux. This directory is used for temporary storage in memory for files and never for data like binaries that need to remain on the host. Typically the only binaries running from */dev/shm* will be malware. This is because the */dev/shm* directory is located in RAM so if the system reboots the malware is destroyed and makes recovery and analysis harder. Also, */dev/shm* is publicly writable on virtually every single Linux system on the planet. Attackers know a file dropped in the ramdisk have a high chance of not only running, but not being easily seen by casual observers.

For Version 1 of BPFDoor, Sandfly will show an alert as below for any kind of binary running from the */dev* directory and */dev/shm* in particular. About 99.9% of the time a binary running from this area is malicious whether it is BPFDoor or something else.

Below we see the alert along with the suspicious process path. Again, **any** process seen running from this path should be investigated as it is almost always malicious. Because running from */dev/shm* is such a huge red flag, Version 2 of BPFDoor did away with this tactic.



process_running_from_dev_dir

Alert · Lvl 5 · Latest

T1036.005 · defense_evasion · persistence · default_active · process

The process name 'kdmtmpflush' with PID '1059' is running out of the location '/dev/shm/kdmtmpflush'. This is a suspicious location for a process to be running and may be malicious.

Seen 1 time on testing ([redacted]).

Severity: 5
First Seen: 2025-07-03T03:32:10Z
Last Seen: 2025-07-03T03:32:10Z

testing
[redacted] · 1 user · 1 key

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	kdmtmpflush	
process.cmdline	/sbin/udevd -d	
process.command	/sbin/udevd -d	
process.hash.md5	f1b6dde830fdb46a608894f603d10f82	
process.hash.sha1	61ef75ade586de082c1df5fbf55bceff…	
process.hash.sha256	85d0ccdac441e598e2e6366b86debb71…	
process.hash.sha512	8b318c80c8d8d57de768744e246a116f…	
process.path	/dev/shm/kdmtmpflush	
process.username	root	
containerized	false	

Total Rows: 2,092

Version 1 of BPFDoor often named its process *kdmtmpflush*. It changed this name after execution (discussed below), but often will be seen in various process lists using this name. The arrow points to the full path used by the Version 1 binary before deleting itself.

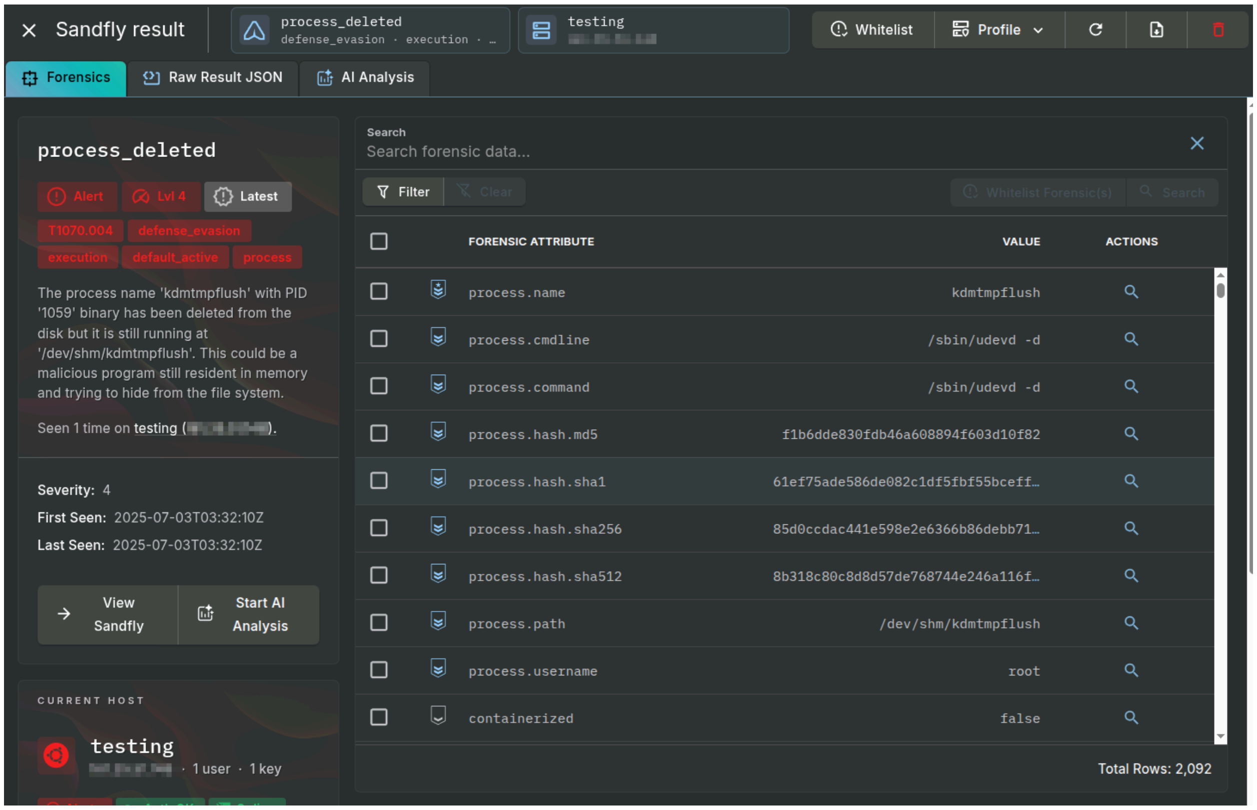
Deleted Binary

The next alerts we see with Version 1 of BPFDoor is *process_deleted*. This is a simple alert that means a process is running on the host, but the binary that started the process has been deleted from the disk. This is a simple way for malware to prevent its binary from being discovered by file scanning tools such as anti-virus or file integrity monitoring. Since the binary is not on the disk, traditional file scanning security tools will simply not see it is there even though it continues to run in memory. Sandfly considers this a serious alert, especially when combined with network operations, sniffing traffic, and other attributes.

As it turns out, it is very easy to recover a malicious process binary that has been deleted from the disk. We cover how to do this here:

[Recovering a Deleted Process Binary on Linux]()

Below we see the alert on the process *kdmtmpflush* along with the masquerading name */sbin/udev -d* which the malware selected from a random list of names when it started.



process_deleted

The process name 'kdmtmpflush' with PID '1059' binary has been deleted from the disk but it is still running at '/dev/shm/kdmtmpflush'. This could be a malicious program still resident in memory and trying to hide from the file system.

Seen 1 time on testing ([redacted]).

Severity: 4
First Seen: 2025-07-03T03:32:10Z
Last Seen: 2025-07-03T03:32:10Z

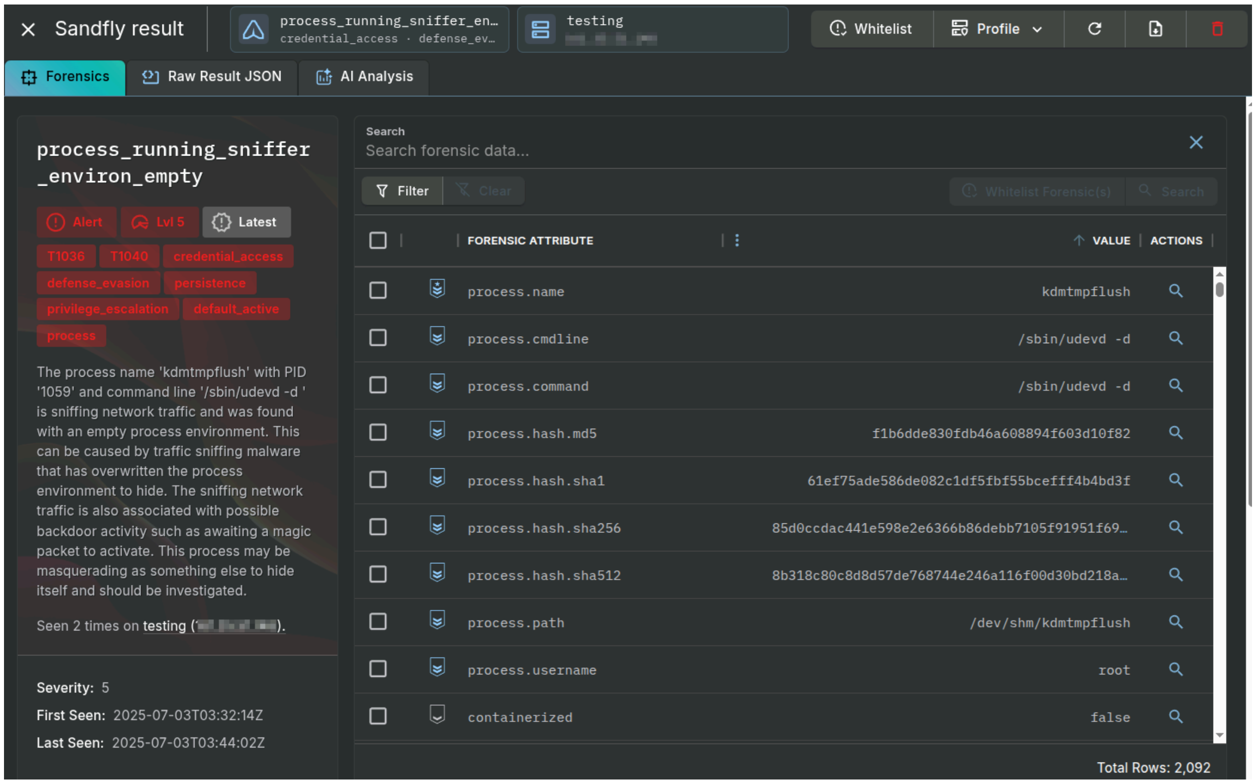
FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	kdmtmpflush	
process.cmdline	/sbin/udevd -d	
process.command	/sbin/udevd -d	
process.hash.md5	f1b6dde830fdb46a608894f603d10f82	
process.hash.sha1	61ef75ade586de082c1df5fbf55bceff…	
process.hash.sha256	85d0ccdac441e598e2e6366b86debb71…	
process.hash.sha512	8b318c80c8d8d57de768744e246a116f…	
process.path	/dev/shm/kdmtmpflush	
process.username	root	
containerized	false	

Total Rows: 2,092

Anti-Forensics Environment Wiping

Both versions of BPFDoor have various anti-forensics in the main binary and backdoor shells which wipe the process environment. By wiping the environment it makes root cause analysis of the intrusion more difficult for incident responders.

Below we see the alert *process_running_sniffer_environ_empty* which is present in Version 1 of the backdoor. We alert on this because while some processes on Linux may not have an environment set (such as a kernel thread), it is extremely unusual for a process sniffing network traffic to not have one.



process_running_sniffer_environ_empty

The process name 'kdmtmpflush' with PID '1059' and command line '/sbin/udevd -d ' is sniffing network traffic and was found with an empty process environment. This can be caused by traffic sniffing malware that has overwritten the process environment to hide. The sniffing network traffic is also associated with possible backdoor activity such as awaiting a magic packet to activate. This process may be masquerading as something else to hide itself and should be investigated.

Seen 2 times on testing ([redacted]).

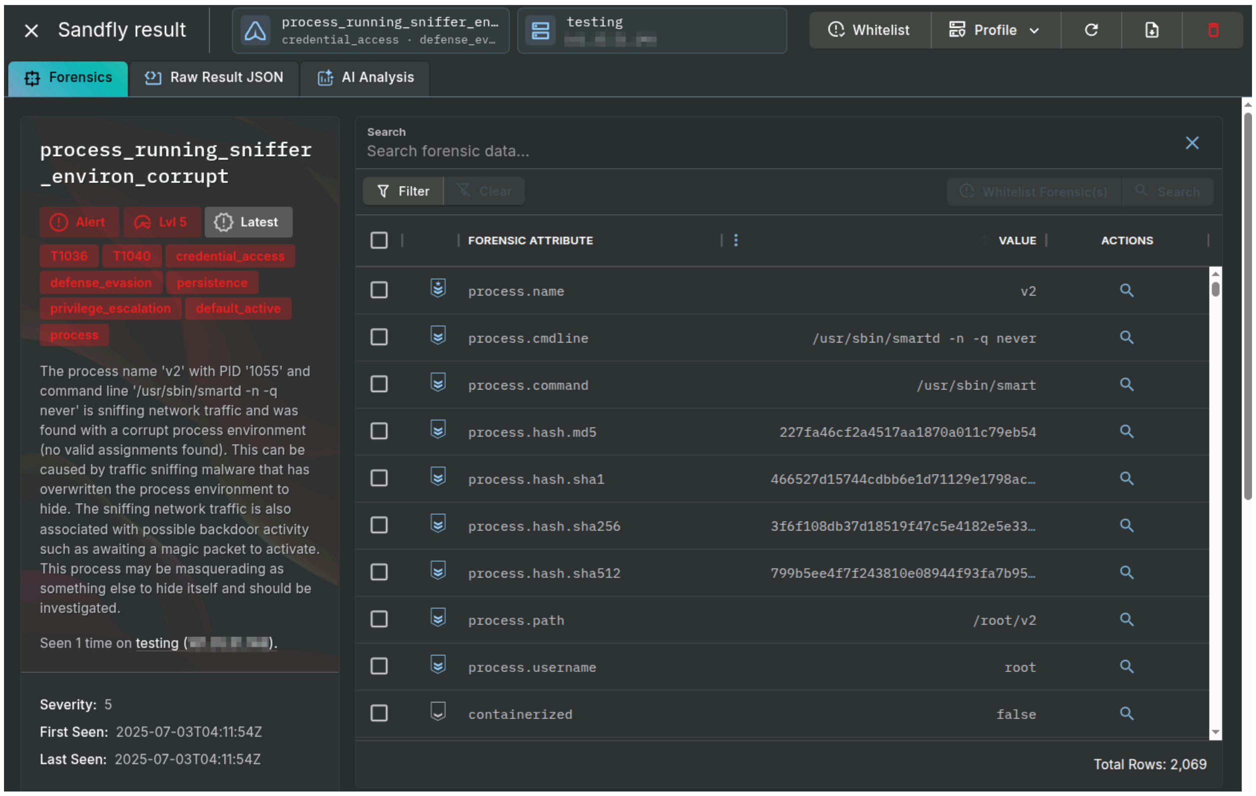
Severity: 5
First Seen: 2025-07-03T03:32:14Z
Last Seen: 2025-07-03T03:44:02Z

FORENSIC ATTRIBUTE	VALUE
process.name	kdmtmpflush
process.cmdline	/sbin/udevd -d
process.command	/sbin/udevd -d
process.hash.md5	f1b6dde830fdb46a608894f603d10f82
process.hash.sha1	61ef75ade586de082c1df5fbf55bcefff4b4bd3f
process.hash.sha256	85d0ccdac441e598e2e6366b86debb7105f91951f69…
process.hash.sha512	8b318c80c8d8d57de768744e246a116f00d30bd218a…
process.path	/dev/shm/kdmtmpflush
process.username	root
containerized	false

Total Rows: 2,092

Corrupt Environment

Version 2 of BPFDoor has a corrupt environment due to its process masquerading efforts. Meaning it does wipe its environment like Version 1, but when it hides its name with masquerading it corrupts it in a way that is unusual. We identify this problem as *process_running_sniffer_environ_corrupt*.



process_running_sniffer_environ_corrupt

Alert | Lvl 5 | Latest

T1036 T1040 credential_access defense_evasion persistence privilege_escalation default_active process

The process name 'v2' with PID '1055' and command line '/usr/sbin/smartd -n -q never' is sniffing network traffic and was found with a corrupt process environment (no valid assignments found). This can be caused by traffic sniffing malware that has overwritten the process environment to hide. The sniffing network traffic is also associated with possible backdoor activity such as awaiting a magic packet to activate. This process may be masquerading as something else to hide itself and should be investigated.

Seen 1 time on testing ([redacted]).

Severity: 5
First Seen: 2025-07-03T04:11:54Z
Last Seen: 2025-07-03T04:11:54Z

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	v2	
process.cmdline	/usr/sbin/smartd -n -q never	
process.command	/usr/sbin/smart	
process.hash.md5	227fa46cf2a4517aa1870a011c79eb54	
process.hash.sha1	466527d15744cdbb6e1d71129e1798ac…	
process.hash.sha256	3f6f108db37d18519f47c5e4182e5e33…	
process.hash.sha512	799b5ee4f7f243810e08944f93fa7b95…	
process.path	/root/v2	
process.username	root	
containerized	false	

Total Rows: 2,069

Packet Sniffing Evasion Detection

Packet sniffing on Linux can be legitimate for certain network services, but if something is grabbing network traffic for unknown reasons it often is trouble. For BPFDoor, it must grab network traffic in order to see magic packets to activate and this is a primary detection opportunity.

Version 1 of BPFDoor used standard network sniffing sockets, but Version 2 used an unusual method detailed in the blog by HaxRob referenced in the links section. Basically, it took advantage of a poorly documented feature in Linux where a SOCK_DGRAM type socket can also intercept all packets just like traditional SOCK_RAW types. This seemingly minor change meant that conventional methods to spot processes accessing raw sockets that may be grabbing packets would not work.

Sandfly was always able to detect BPFDoor even before it became public in part due to the traffic sniffing anomalies. Version 2 allowed it to hide from some of these earlier detections, but still was being seen by the *process_stack_packet_sniffer* detection and others.

For Sandfly 5.5 we improved our detection by decoding open file descriptors of processes and what protocols and socket types they are. Specifically, we added the following modules to find these tactics:

process_running_sniffer_operating_ipv4_traffic - Flags any process that is specifically grabbing IPv4 traffic.

process_running_sniffer_operating_ipv6_traffic - Flags any process that is specifically grabbing IPv6 traffic.

process_running_sniffer_operating_unknown_protocol - Flags any process that is grabbing network traffic and an unknown protocol type.

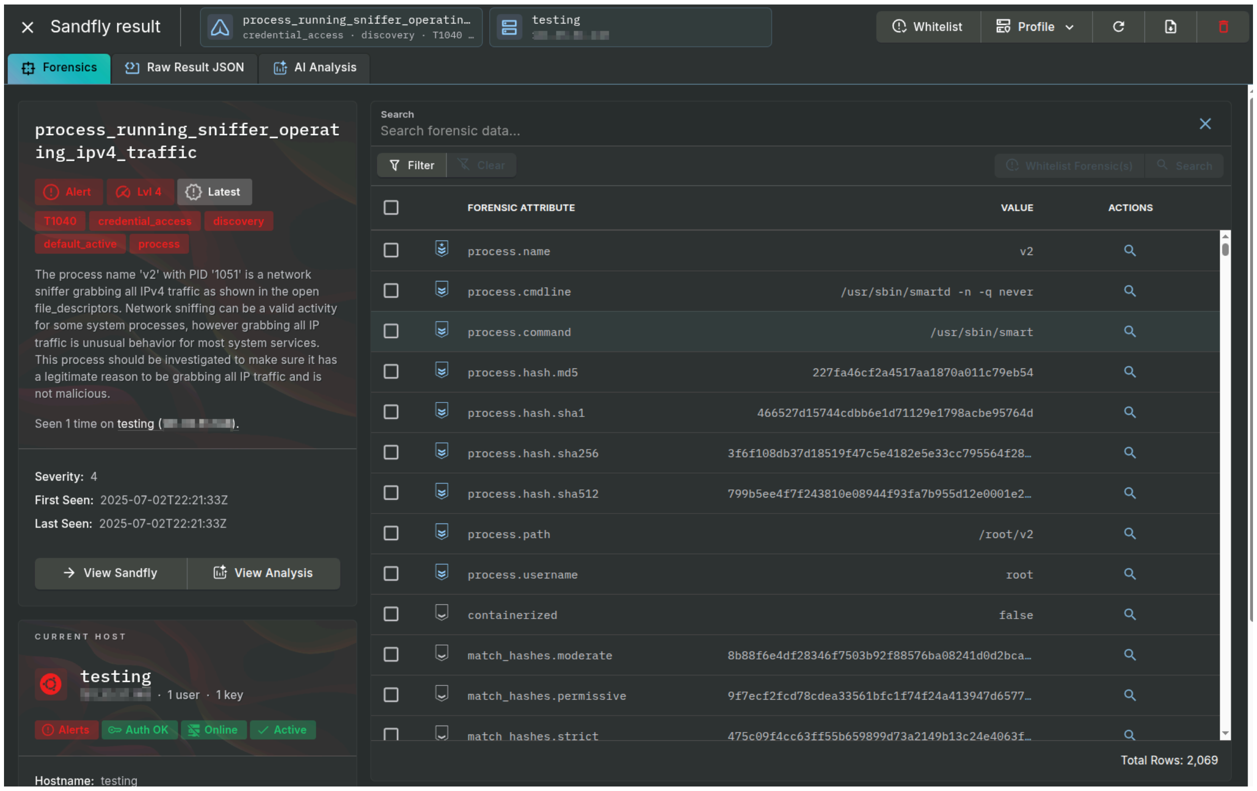
process_running_sniffer_operating_all_traffic - Flags any process that is grabbing all network traffic (disabled by default).

The IPv4 and IPv6 versions are enabled by default as normal system processes would rarely need to grab all IPv4 and IPv6 traffic.

The detection for unknown protocols is a catch-all for possible malware which may be operating with obscure settings to avoid network monitoring.

The detection for processes grabbing all traffic may cause false positives on certain default system services (e.g. DHCP servers), but can be useful for incident response teams once they whitelist the known good processes. They can in essence profile everything grabbing network traffic, sort out what they expect, and then investigate what is left. Again, this is disabled by default but can be enabled if teams wish to use it.

Below is a detection of the Version 2 of BPFDoor with these improved detections in Sandfly 5.5 and higher. BPFDoor is running under the false name */usr/sbin/smartd -n -q never*. Version 2 of BPFDoor may use different names depending on the build, but this name has been common.



process_running_sniffer_operating_ipv4_traffic

The process name 'v2' with PID '1051' is a network sniffer grabbing all IPv4 traffic as shown in the open file_descriptors. Network sniffing can be a valid activity for some system processes, however grabbing all IP traffic is unusual behavior for most system services. This process should be investigated to make sure it has a legitimate reason to be grabbing all IP traffic and is not malicious.

Seen 1 time on testing ([redacted]).

Severity: 4
First Seen: 2025-07-02T22:21:33Z
Last Seen: 2025-07-02T22:21:33Z

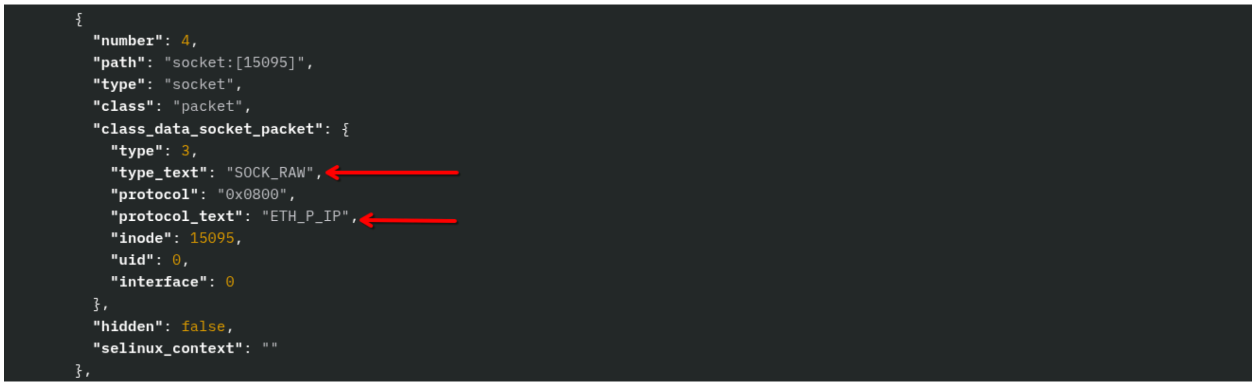
testing
[redacted] · 1 user · 1 key

Hostname: testing

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	v2	🔍
process.cmdline	/usr/sbin/smartd -n -q never	🔍
process.command	/usr/sbin/smart	🔍
process.hash.md5	227fa46cf2a4517aa1870a011c79eb54	🔍
process.hash.sha1	466527d15744cdbb6e1d71129e1798acbe95764d	🔍
process.hash.sha256	3f6f108db37d18519f47c5e4182e5e33cc795564f28…	🔍
process.hash.sha512	799b5ee4f7f243810e08944f93fa7b955d12e0001e2…	🔍
process.path	/root/v2	🔍
process.username	root	🔍
containerized	false	🔍
match_hashes.moderate	8b88f6e4df28346f7503b92f88576ba08241d0d2bca…	🔍
match_hashes.permissive	9f7ecf2fcd78cdea33561bfc1f74f24a413947d6577…	🔍
match_hashes.strict	475c09f4cc63ff55b659899d73a2149b13c24e4063f…	🔍

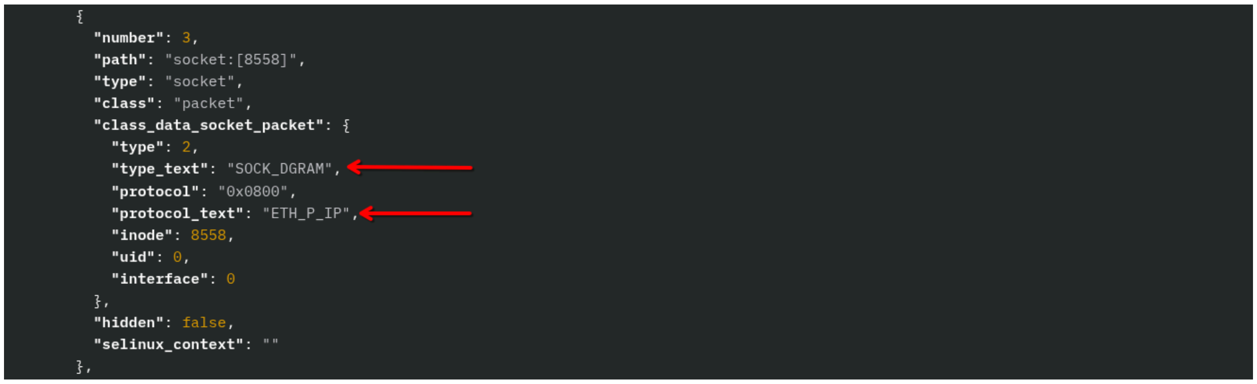
Total Rows: 2,069

For incident responders, the improved file descriptor forensics from Sandfly is shown below. Here is Version 1 of BPFDoor with open SOCK_RAW sockets grabbing all IPv4 traffic (ETH_P_IP).



```
{
  "number": 4,
  "path": "socket:[15095]",
  "type": "socket",
  "class": "packet",
  "class_data_socket_packet": {
    "type": 3,
    "type_text": "SOCK_RAW",
    "protocol": "0x0800",
    "protocol_text": "ETH_P_IP",
    "inode": 15095,
    "uid": 0,
    "interface": 0
  },
  "hidden": false,
  "selinux_context": ""
},
```

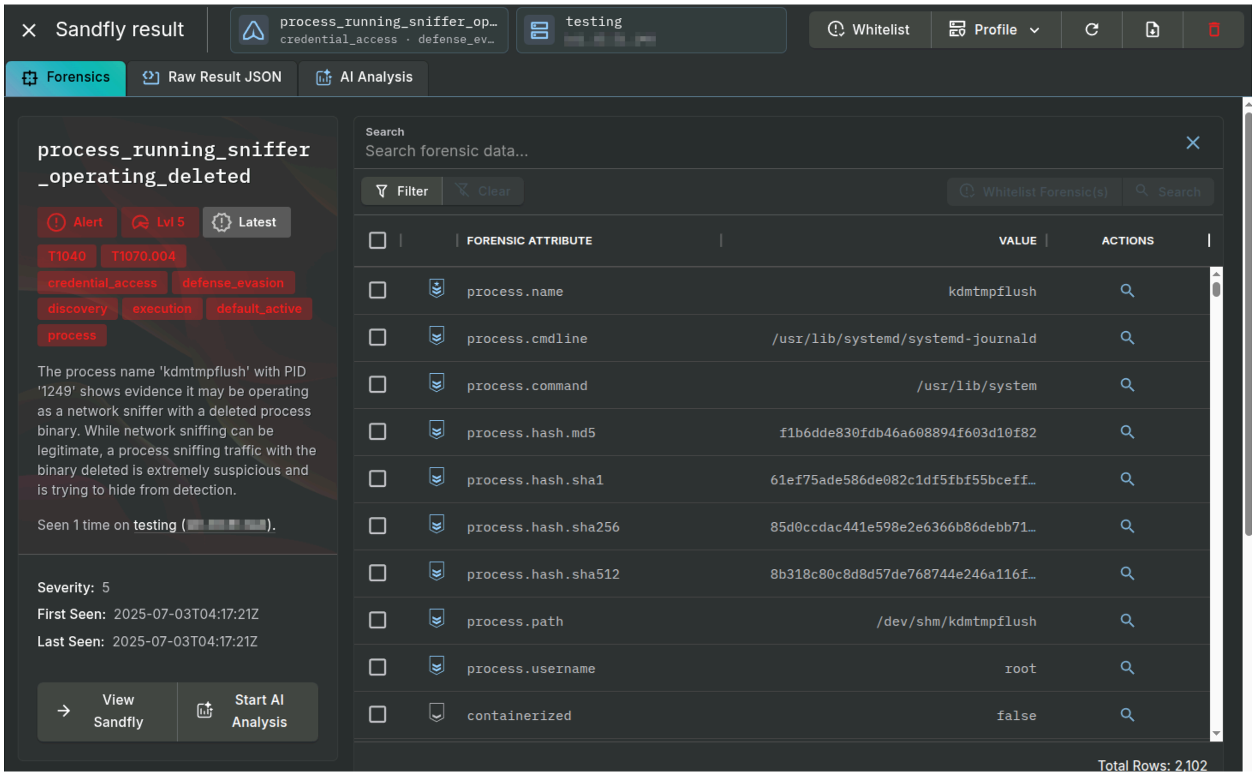
Version 2 will show something similar with SOCK_DGRAM sockets open and again all IPv4 traffic.



```
{
  "number": 3,
  "path": "socket:[8558]",
  "type": "socket",
  "class": "packet",
  "class_data_socket_packet": {
    "type": 2,
    "type_text": "SOCK_DGRAM",
    "protocol": "0x0800",
    "protocol_text": "ETH_P_IP",
    "inode": 8558,
    "uid": 0,
    "interface": 0
  },
  "hidden": false,
  "selinux_context": ""
},
```

Deleted Sniffer Process

Version 1 of BPFDoor presents another detection opportunity when it deletes itself after starting. This again is an extremely suspicious event to see on Linux. Version 2 stopped this behavior.



process_running_sniffer_operating_deleted

T1040 T1070.004 credential_access defense_evasion discovery execution default_active process

The process name 'kdmtmpflush' with PID '1249' shows evidence it may be operating as a network sniffer with a deleted process binary. While network sniffing can be legitimate, a process sniffing traffic with the binary deleted is extremely suspicious and is trying to hide from detection.

Seen 1 time on testing ([redacted]).

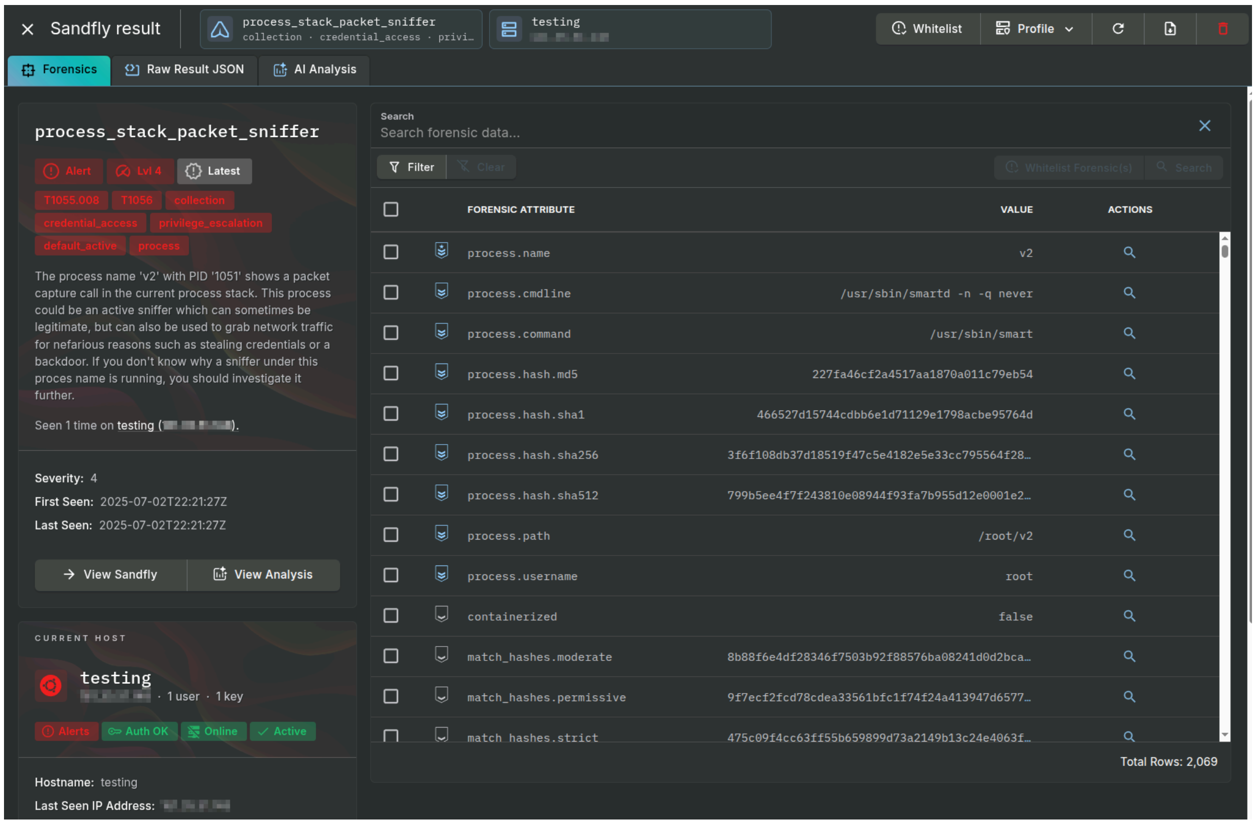
Severity: 5
First Seen: 2025-07-03T04:17:21Z
Last Seen: 2025-07-03T04:17:21Z

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	kdmtmpflush	
process.cmdline	/usr/lib/systemd/systemd-journald	
process.command	/usr/lib/system	
process.hash.md5	f1b6dde830fdb46a608894f603d10f82	
process.hash.sha1	61ef75ade586de082c1df5fbf55bceff…	
process.hash.sha256	85d0ccdac441e598e2e6366b86debb71…	
process.hash.sha512	8b318c80c8d8d57de768744e246a116f…	
process.path	/dev/shm/kdmtmpflush	
process.username	root	
containerized	false	

Total Rows: 2,102

Process Stack Sniffer Activity

Both versions of BPFDoor show clear network sniffing activity in their process stack data under */proc/PID/stack*. Sandfly identifies both versions with an alert like below:



process_stack_packet_sniffer

The process name 'v2' with PID '1051' shows a packet capture call in the current process stack. This process could be an active sniffer which can sometimes be legitimate, but can also be used to grab network traffic for nefarious reasons such as stealing credentials or a backdoor. If you don't know why a sniffer under this proces name is running, you should investigate it further.

Seen 1 time on testing ([redacted]).

Severity: 4
First Seen: 2025-07-02T22:21:27Z
Last Seen: 2025-07-02T22:21:27Z

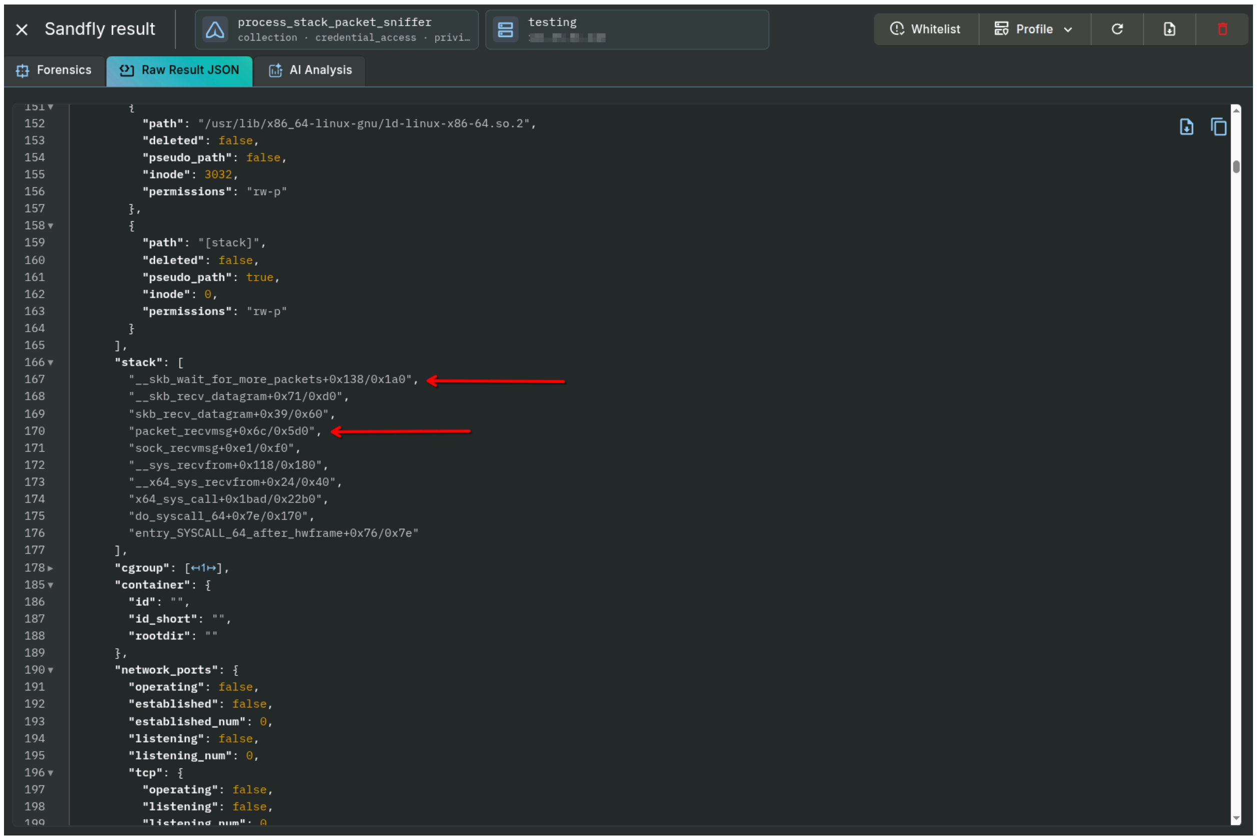
testing
[redacted] · 1 user · 1 key

Hostname: testing
Last Seen IP Address: [redacted]

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	v2	🔍
process.cmdline	/usr/sbin/smartd -n -q never	🔍
process.command	/usr/sbin/smart	🔍
process.hash.md5	227fa46cf2a4517aa1870a011c79eb54	🔍
process.hash.sha1	466527d15744cdbb6e1d71129e1798acbe95764d	🔍
process.hash.sha256	3f6f108db37d18519f47c5e4182e5e33cc795564f28…	🔍
process.hash.sha512	799b5ee4f7f243810e08944f93fa7b955d12e0001e2…	🔍
process.path	/root/v2	🔍
process.username	root	🔍
containerized	false	🔍
match_hashes.moderate	8b88f6e4df28346f7503b92f88576ba08241d0d2bca…	🔍
match_hashes.permissive	9f7ecf2fcd78cdea33561bfc1f74f24a413947d6577…	🔍
match_hashes.strict	475c09f4cc63ff55b659899d73a2149b13c24e4063f…	🔍

Total Rows: 2,069

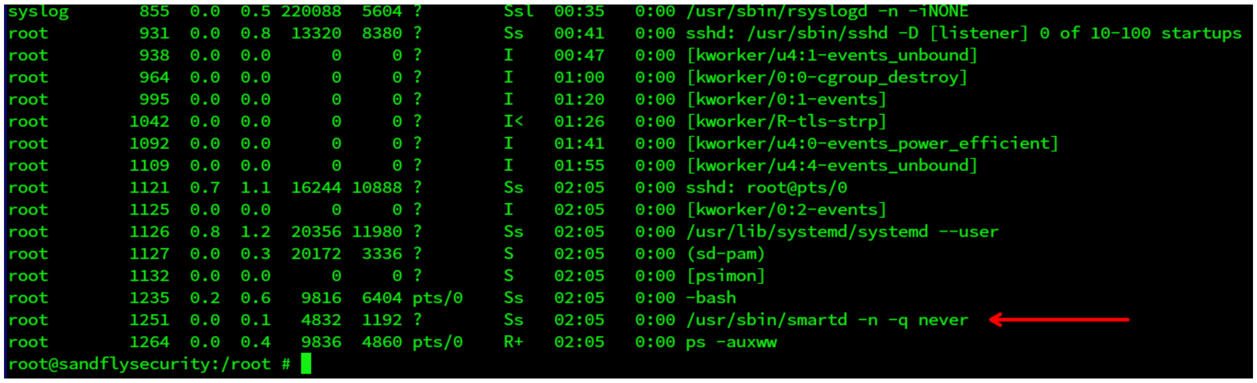
The forensic data shows clear packet sniffing functions present in the process stack of both versions.



```
        {
          "path": "/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2",
          "deleted": false,
          "pseudo_path": false,
          "inode": 3032,
          "permissions": "rw-p"
        },
        {
          "path": "[stack]",
          "deleted": false,
          "pseudo_path": true,
          "inode": 0,
          "permissions": "rw-p"
        }
      ],
      "stack": [
        "__skb_wait_for_more_packets+0x138/0x1a0",
        "__skb_recv_datagram+0x71/0xd0",
        "skb_recv_datagram+0x39/0x60",
        "packet_recvmsg+0x6c/0x5d0",
        "sock_recvmsg+0xe1/0xf0",
        "__sys_recvfrom+0x118/0x180",
        "__x64_sys_recvfrom+0x24/0x40",
        "x64_sys_call+0x1bad/0x22b0",
        "do_syscall_64+0x7e/0x170",
        "entry_SYSCALL_64_after_hwframe+0x76/0x7e"
      ],
      "cgroup": [↤1↦],
      "container": {
        "id": "",
        "id_short": "",
        "rootdir": ""
      },
      "network_ports": {
        "operating": false,
        "established": false,
        "established_num": 0,
        "listening": false,
        "listening_num": 0,
        "tcp": {
          "operating": false,
          "listening": false,
          "listening_num": 0
```

Process Masquerading

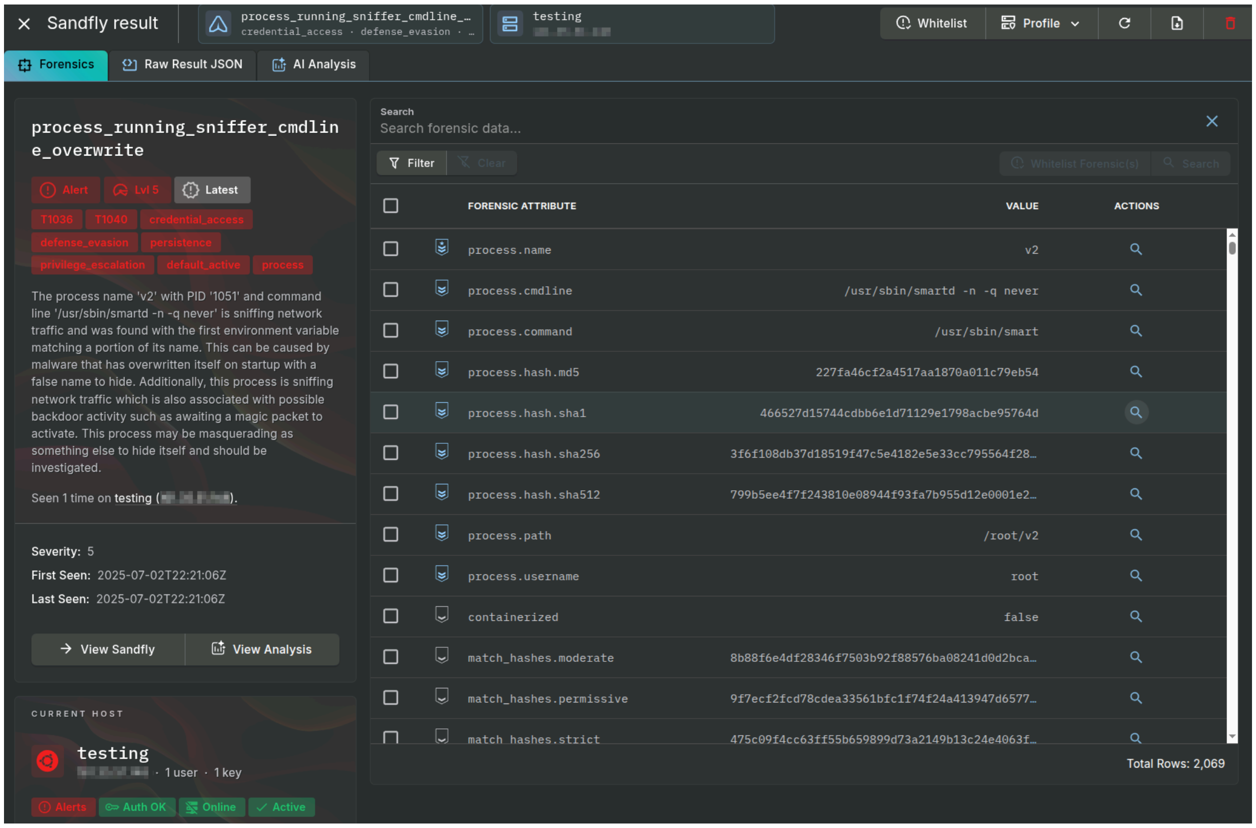
Both versions of BPFDoor incorporate process masquerading. Meaning, they will set their process to appear to be something else when run. Version 1 selected from a list of possible names to use at random. Version 2 appears to just use one name which can vary. In the samples we have it would use */usr/sbin/smartd -n -q never.* Below we show what Version 2 looks like with a simple *ps* process list on Linux.



```
syslog       855  0.0  0.5 220088  5604 ?        Ssl  00:35   0:00 /usr/sbin/rsyslogd -n -iNONE
root         931  0.0  0.8  13320  8380 ?        Ss   00:41   0:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
root         938  0.0  0.0      0     0 ?        I    00:47   0:00 [kworker/u4:1-events_unbound]
root         964  0.0  0.0      0     0 ?        I    01:00   0:00 [kworker/0:0-cgroup_destroy]
root         995  0.0  0.0      0     0 ?        I    01:20   0:00 [kworker/0:1-events]
root        1042  0.0  0.0      0     0 ?        I<   01:26   0:00 [kworker/R-tls-strp]
root        1092  0.0  0.0      0     0 ?        I    01:41   0:00 [kworker/u4:0-events_power_efficient]
root        1109  0.0  0.0      0     0 ?        I    01:55   0:00 [kworker/u4:4-events_unbound]
root        1121  0.7  1.1  16244 10888 ?        Ss   02:05   0:00 sshd: root@pts/0
root        1125  0.0  0.0      0     0 ?        I    02:05   0:00 [kworker/0:2-events]
root        1126  0.8  1.2  20356 11980 ?        Ss   02:05   0:00 /usr/lib/systemd/systemd --user
root        1127  0.0  0.3  20172  3336 ?        S    02:05   0:00 (sd-pam)
root        1132  0.0  0.0      0     0 ?        S    02:05   0:00 [psimon]
root        1235  0.2  0.6   9816  6404 pts/0    Ss   02:05   0:00 -bash
root        1251  0.0  0.1   4832  1192 ?        Ss   02:05   0:00 /usr/sbin/smartd -n -q never
root        1264  0.0  0.4   9836  4860 pts/0    R+   02:05   0:00 ps -auxww
root@sandflysecurity:/root # 
```

When Version 2 of BPFDoor starts, it overwrites its name with the bogus value and in doing so stomps on its process environment. We detect this problem with the corrupt environment detection seen above, but also a new *process_running_sniffer_cmdline_overwrite* detection.

This detection specifically looks for sniffing processes that have overwritten their environment with their masqueraded name to hide.



Sandfly result
process_running_sniffer_cmdline_…
credential_access · defense_evasion · …
testing
Whitelist
Profile
Forensics
Raw Result JSON
AI Analysis

process_running_sniffer_cmdline_overwrite

Alert · Lvl 5 · Latest

T1036 · T1040 · credential_access · defense_evasion · persistence · privilege_escalation · default_active · process

The process name 'v2' with PID '1051' and command line '/usr/sbin/smartd -n -q never' is sniffing network traffic and was found with the first environment variable matching a portion of its name. This can be caused by malware that has overwritten itself on startup with a false name to hide. Additionally, this process is sniffing network traffic which is also associated with possible backdoor activity such as awaiting a magic packet to activate. This process may be masquerading as something else to hide itself and should be investigated.

Seen 1 time on testing ([redacted]).

Severity: 5
First Seen: 2025-07-02T22:21:06Z
Last Seen: 2025-07-02T22:21:06Z

View Sandfly · View Analysis

CURRENT HOST

testing
[redacted] · 1 user · 1 key

Alerts · Auth OK · Online · Active

Search
Search forensic data...

Filter · Clear · Whitelist Forensic(s) · Search

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	v2	🔍
process.cmdline	/usr/sbin/smartd -n -q never	🔍
process.command	/usr/sbin/smart	🔍
process.hash.md5	227fa46cf2a4517aa1870a011c79eb54	🔍
process.hash.sha1	466527d15744cdbb6e1d71129e1798acbe95764d	🔍
process.hash.sha256	3f6f108db37d18519f47c5e4182e5e33cc795564f28…	🔍
process.hash.sha512	799b5ee4f7f243810e08944f93fa7b955d12e0001e2…	🔍
process.path	/root/v2	🔍
process.username	root	🔍
containerized	false	🔍
match_hashes.moderate	8b88f6e4df28346f7503b92f88576ba08241d0d2bca…	🔍
match_hashes.permissive	9f7ecf2fcd78cdea33561bfc1f74f24a413947d6577…	🔍
match_hashes.strict	475c09f4cc63ff55b659899d73a2149b13c24e4063f…	🔍

Total Rows: 2,069

Backdoor in Operation Detection

BPFDoor has two modes to activate a backdoor:

1) Bind Shell
2) Reverse Shell

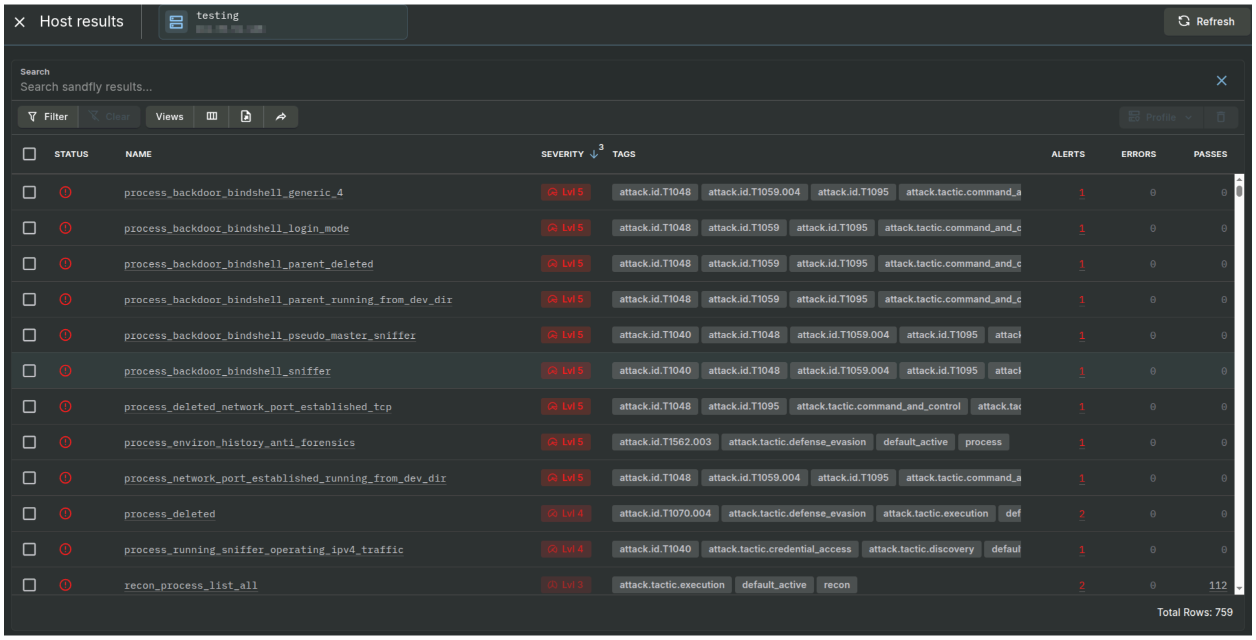
The bind shell attaches a command shell to a port on the host and directs the firewall to forward traffic from the attacker to this port after the magic packet is received.

The reverse shell will instruct the host to connect *back* to the attacker on a listening port once the magic packet is received. This can cause a host to hop a firewall back to the attacker if the firewall allows outbound connections from protected hosts.

In both cases, the shell will give access to the host with elevated privileges (root) as BPFDoor will typically need privileged access to sniff traffic with raw sockets on Linux.

Backdoor Shell Detection

Below we see many alerts generated once the backdoor is activated. In our example, we activated the reverse shell backdoor but detections for the standard bind shell are similar.



[redacted]

STATUS	NAME	SEVERITY	TAGS	ALERTS	ERRORS	PASSES
❗	process_backdoor_bindshell_generic_4	Lvl 5	attack.id.T1048 attack.id.T1059.004 attack.id.T1095 attack.tactic.command_a	1	0	0
❗	process_backdoor_bindshell_login_mode	Lvl 5	attack.id.T1048 attack.id.T1059 attack.id.T1095 attack.tactic.command_and_c	1	0	0
❗	process_backdoor_bindshell_parent_deleted	Lvl 5	attack.id.T1048 attack.id.T1059 attack.id.T1095 attack.tactic.command_and_c	1	0	0
❗	process_backdoor_bindshell_parent_running_from_dev_dir	Lvl 5	attack.id.T1048 attack.id.T1059 attack.id.T1095 attack.tactic.command_and_c	1	0	0
❗	process_backdoor_bindshell_pseudo_master_sniffer	Lvl 5	attack.id.T1040 attack.id.T1048 attack.id.T1059.004 attack.id.T1095 attack	1	0	0
❗	process_backdoor_bindshell_sniffer	Lvl 5	attack.id.T1040 attack.id.T1048 attack.id.T1059.004 attack.id.T1095 attack	1	0	0
❗	process_deleted_network_port_established_tcp	Lvl 5	attack.id.T1048 attack.id.T1095 attack.tactic.command_and_control attack.ta	1	0	0
❗	process_environ_history_anti_forensics	Lvl 5	attack.id.T1562.003 attack.tactic.defense_evasion default_active process	1	0	0
❗	process_network_port_established_running_from_dev_dir	Lvl 5	attack.id.T1048 attack.id.T1059.004 attack.id.T1095 attack.tactic.command_a	1	0	0
❗	process_deleted	Lvl 4	attack.id.T1070.004 attack.tactic.defense_evasion attack.tactic.execution def	2	0	0
❗	process_running_sniffer_operating_ipv4_traffic	Lvl 4	attack.id.T1040 attack.tactic.credential_access attack.tactic.discovery defaul	1	0	0
❗	recon_process_list_all	Lvl 3	attack.tactic.execution default_active recon	2	0	112

Total Rows: 759

Many alerts are generated once the shell is activated and attached to the network:

process_backdoor_bindshell_generic_4 - A generic bind shell backdoor has been detected (multiple variants possible).

process_backdoor_bindshell_login_mode - A bind shell with login mode has been detected.

process_backdoor_bindshell_parent_deleted - A bind shell spawned by a deleted parent process has been detected.

process_backdoor_bindshell_parent_running_from_dev_dir - A bind shell where the parent process is running from */dev* (e.g. */dev/shm*) has been detected.

process_backdoor_bindshell_pseudo_master_sniffer - A bind shell using a master terminal */dev/ptmx* along with sniffer activity has been detected.

process_backdoor_bindshell_sniffer - A bind shell that is sniffing traffic has been detected.

process_environ_history_anti_forensics - A process was started with anti-forensics detected in the process environment.

process_deleted_network_port_established_tcp - A process running from */dev* directory that is deleted with open network ports has been detected.

process_network_port_established_running_from_dev_dir - A process running from */dev* directory with established network ports has been detected.

process_deleted - A deleted process binary has been detected.

process_running_sniffer_operating_ipv4_traffic - A process sniffing IPv4 network traffic has been detected.

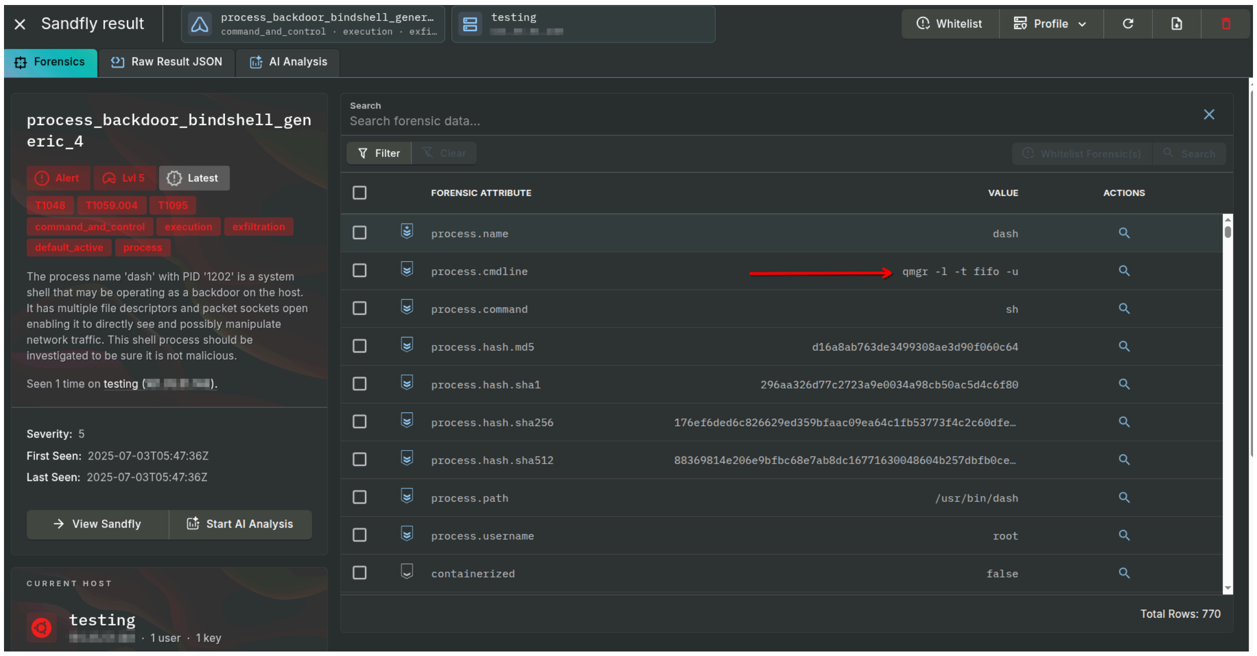
recon_process_list_all - Drift detection has found a new process that is not authorized to be running on this host.

recon_process_list_sniffer_operating - Drift detection has found a new process that is sniffing network traffic running on this host.

Process Bind Shell Backdoor Alerts

Sandfly generates many alerts about an active bind shell backdoor on hosts. Some of these alerts are below indicating extremely suspicious command shell use. There are many variants of bind shell backdoors and each has unique attributes detected by Sandfly.

The red arrow shows the bogus process name the malware is trying to hide under along with the real name of the shell (*dash*).



process_backdoor_bindshell_generic_4

The process name 'dash' with PID '1202' is a system shell that may be operating as a backdoor on the host. It has multiple file descriptors and packet sockets open enabling it to directly see and possibly manipulate network traffic. This shell process should be investigated to be sure it is not malicious.

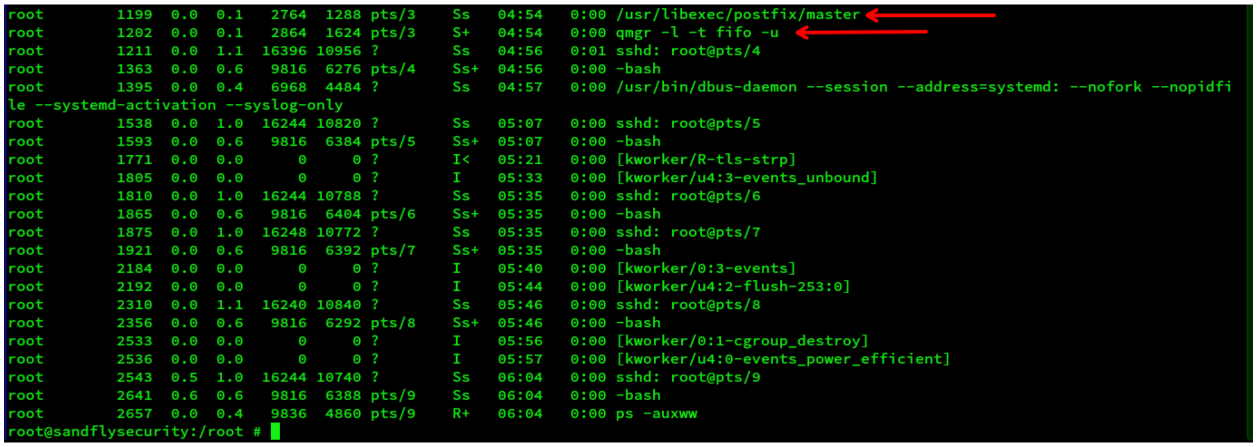
Seen 1 time on testing ([redacted]).

Severity: 5
First Seen: 2025-07-03T05:47:36Z
Last Seen: 2025-07-03T05:47:36Z

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	dash	🔍
process.cmdline	qmgr -l -t fifo -u	🔍
process.command	sh	🔍
process.hash.md5	d16a8ab763de3499308ae3d90f060c64	🔍
process.hash.sha1	296aa326d77c2723a9e0034a98cb50ac5d4c6f80	🔍
process.hash.sha256	176ef6ded6c826629ed359bfaac09ea64c1fb53773f4c2c60dfe…	🔍
process.hash.sha512	88369814e206e9bfbc68e7ab8dc16771630048604b257dbfb0ce…	🔍
process.path	/usr/bin/dash	🔍
process.username	root	🔍
containerized	false	🔍

Total Rows: 770

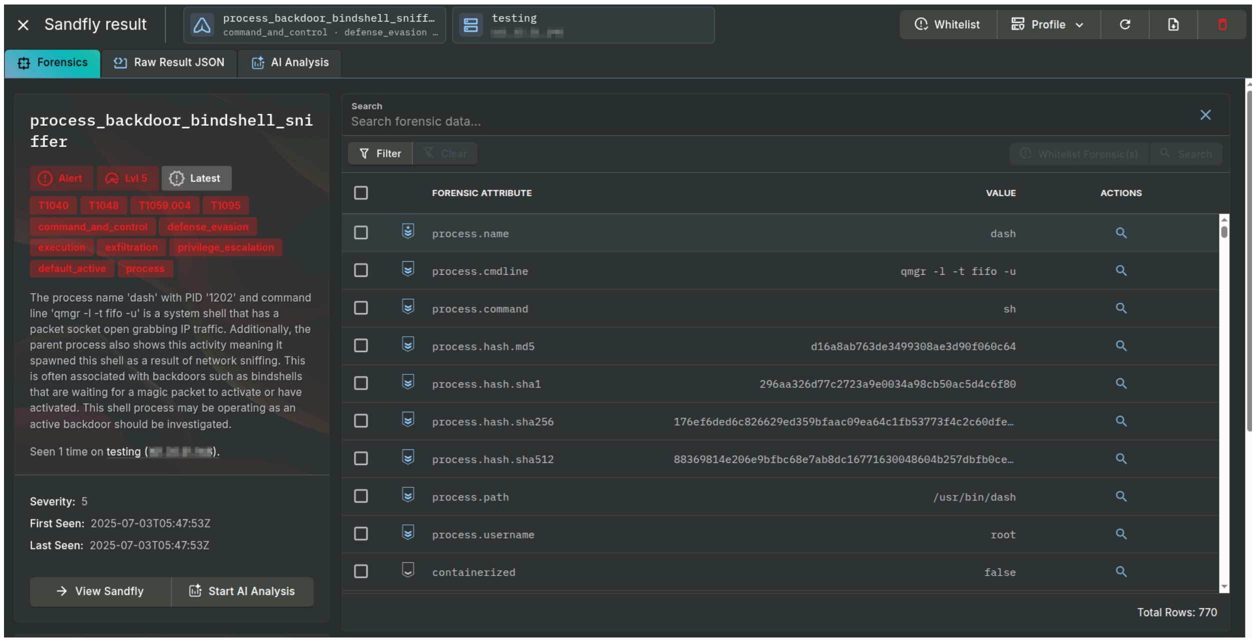
On Version 1, when the shell executes, it hides itself by re-spawning the parent process as */usr/libexec/postfix/master.* The shell itself is run under the bogus name *qmg -l -t fifo -u*. Other names could be used in other variants. The process list below shows what BPFDoor Version 1 looks like from the command line when running the bind shell.



```
root        1199  0.0  0.1   2764  1288 pts/3    Ss   04:54   0:00 /usr/libexec/postfix/master
root        1202  0.0  0.1   2864  1624 pts/3    S+   04:54   0:00 qmgr -l -t fifo -u
root        1211  0.0  1.1  16396 10956 ?        Ss   04:56   0:01 sshd: root@pts/4
root        1363  0.0  0.6   9816  6276 pts/4    Ss+  04:56   0:00 -bash
root        1395  0.0  0.4   6968  4484 ?        Ss   04:57   0:00 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfi
le --systemd-activation --syslog-only
root        1538  0.0  1.0  16244 10820 ?        Ss   05:07   0:00 sshd: root@pts/5
root        1593  0.0  0.6   9816  6384 pts/5    Ss+  05:07   0:00 -bash
root        1771  0.0  0.0      0     0 ?        I<   05:21   0:00 [kworker/R-tls-strp]
root        1805  0.0  0.0      0     0 ?        I    05:33   0:00 [kworker/u4:3-events_unbound]
root        1810  0.0  1.0  16244 10788 ?        Ss   05:35   0:00 sshd: root@pts/6
root        1865  0.0  0.6   9816  6404 pts/6    Ss+  05:35   0:00 -bash
root        1875  0.0  1.0  16248 10772 ?        Ss   05:35   0:00 sshd: root@pts/7
root        1921  0.0  0.6   9816  6392 pts/7    Ss+  05:35   0:00 -bash
root        2184  0.0  0.0      0     0 ?        I    05:40   0:00 [kworker/0:3-events]
root        2192  0.0  0.0      0     0 ?        I    05:44   0:00 [kworker/u4:2-flush-253:0]
root        2310  0.0  1.1  16240 10840 ?        Ss   05:46   0:00 sshd: root@pts/8
root        2356  0.0  0.6   9816  6292 pts/8    Ss+  05:46   0:00 -bash
root        2533  0.0  0.0      0     0 ?        I    05:56   0:00 [kworker/0:1-cgroup_destroy]
root        2536  0.0  0.0      0     0 ?        I    05:57   0:00 [kworker/u4:0-events_power_efficient]
root        2543  0.5  1.0  16244 10740 ?        Ss   06:04   0:00 sshd: root@pts/9
root        2641  0.6  0.6   9816  6388 pts/9    Ss   06:04   0:00 -bash
root        2657  0.0  0.4   9836  4860 pts/9    R+   06:04   0:00 ps -auxww
root@sandflysecurity:/root #
```

Many more alerts are generated from the BPFDoor bind shell. There will be various bind shell detection alerts shown above with similar forensic attributes.

A common trait with the bind shells is they all inherit the same suspicious sniffer activity at least in Version 1.



process_backdoor_bindshell_sniffer

The process name 'dash' with PID '1202' and command line 'qmgr -l -t fifo -u' is a system shell that has a packet socket open grabbing IP traffic. Additionally, the parent process also shows this activity meaning it spawned this shell as a result of network sniffing. This is often associated with backdoors such as bindshells that are waiting for a magic packet to activate or have activated. This shell process may be operating as an active backdoor should be investigated.

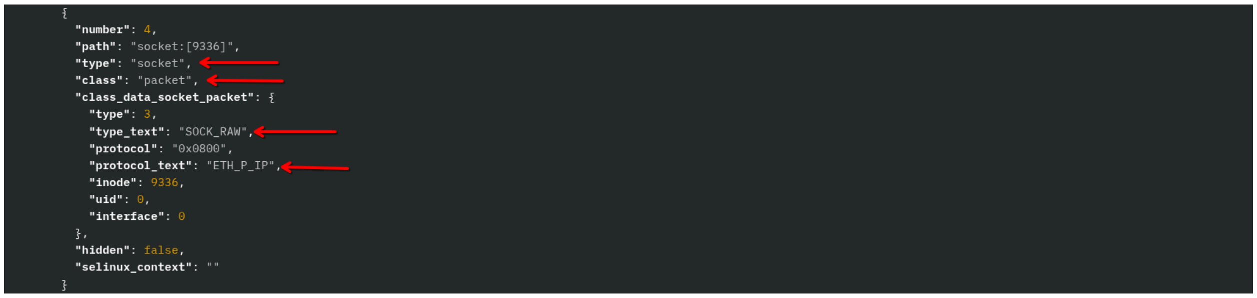
Seen 1 time on testing ([redacted]).

Severity: 5
First Seen: 2025-07-03T05:47:53Z
Last Seen: 2025-07-03T05:47:53Z

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	dash	
process.cmdline	qmgr -l -t fifo -u	
process.command	sh	
process.hash.md5	d16a8ab763de3499308ae3d90f060c64	
process.hash.sha1	296aa326d77c2723a9e0034a98cb50ac5d4c6f80	
process.hash.sha256	176ef6ded6c826629ed359bfaac09ea64c1fb53773f4c2c60dfe…	
process.hash.sha512	88369814e206e9bfbc68e7ab8dc16771630048604b257dbfb0ce…	
process.path	/usr/bin/dash	
process.username	root	
containerized	false	

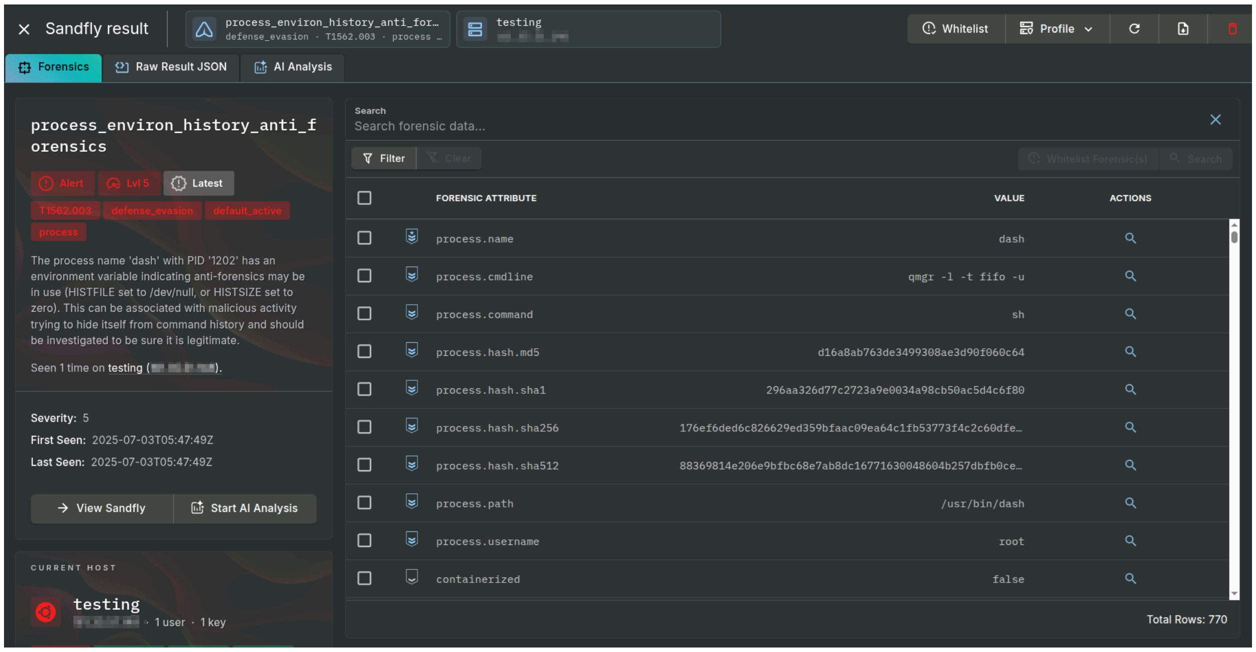
Total Rows: 770

Below we see the open file descriptors from the command shell when running. The red arrows indicate again the suspicious areas that this shell is involved in grabbing network traffic either directly or as part of the parent process that spawned it. This is all available in the Sandfly raw forensics data tab on each alert for closer investigation if an alert is generated.


```
  {
    "number": 4,
    "path": "socket:[9336]",
    "type": "socket",
    "class": "packet",
    "class_data_socket_packet": {
      "type": 3,
      "type_text": "SOCK_RAW",
      "protocol": "0x0800",
      "protocol_text": "ETH_P_IP",
      "inode": 9336,
      "uid": 0,
      "interface": 0
    },
    "hidden": false,
    "selinux_context": ""
  }
```

Backdoor Shell Anti-Forensics

When the backdoor shell is spawned, it incorporates anti-forensics to prevent writing to the history file and other system artifacts. This is identified in the alert below.



process_environ_history_anti_forensics

Alert · Lvl 5 · Latest

T1562.003 · defense_evasion · default_active · process

The process name 'dash' with PID '1202' has an environment variable indicating anti-forensics may be in use (HISTFILE set to /dev/null, or HISTSIZE set to zero). This can be associated with malicious activity trying to hide itself from command history and should be investigated to be sure it is legitimate.

Seen 1 time on testing ([redacted]).

Severity: 5
First Seen: 2025-07-03T05:47:49Z
Last Seen: 2025-07-03T05:47:49Z

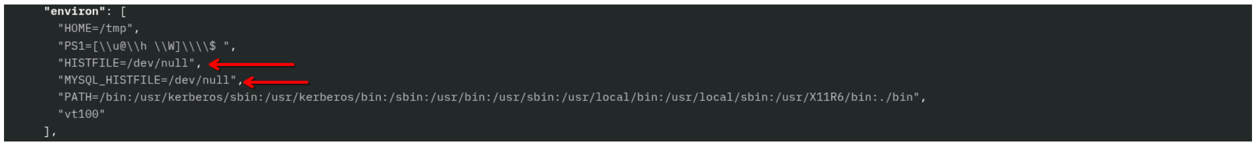
View Sandfly · Start AI Analysis

CURRENT HOST
testing
[redacted] · 1 user · 1 key

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	dash	🔍
process.cmdline	qmgr -l -t fifo -u	🔍
process.command	sh	🔍
process.hash.md5	d16a8ab763de3499308ae3d90f060c64	🔍
process.hash.sha1	296aa326d77c2723a9e0034a98cb50ac5d4c6f80	🔍
process.hash.sha256	176ef6ded6c826629ed359bfaac09ea64c1fb53773f4c2c60dfe…	🔍
process.hash.sha512	88369814e206e9bfbc68e7ab8dc16771630048604b257dbfb0ce…	🔍
process.path	/usr/bin/dash	🔍
process.username	root	🔍
containerized	false	🔍

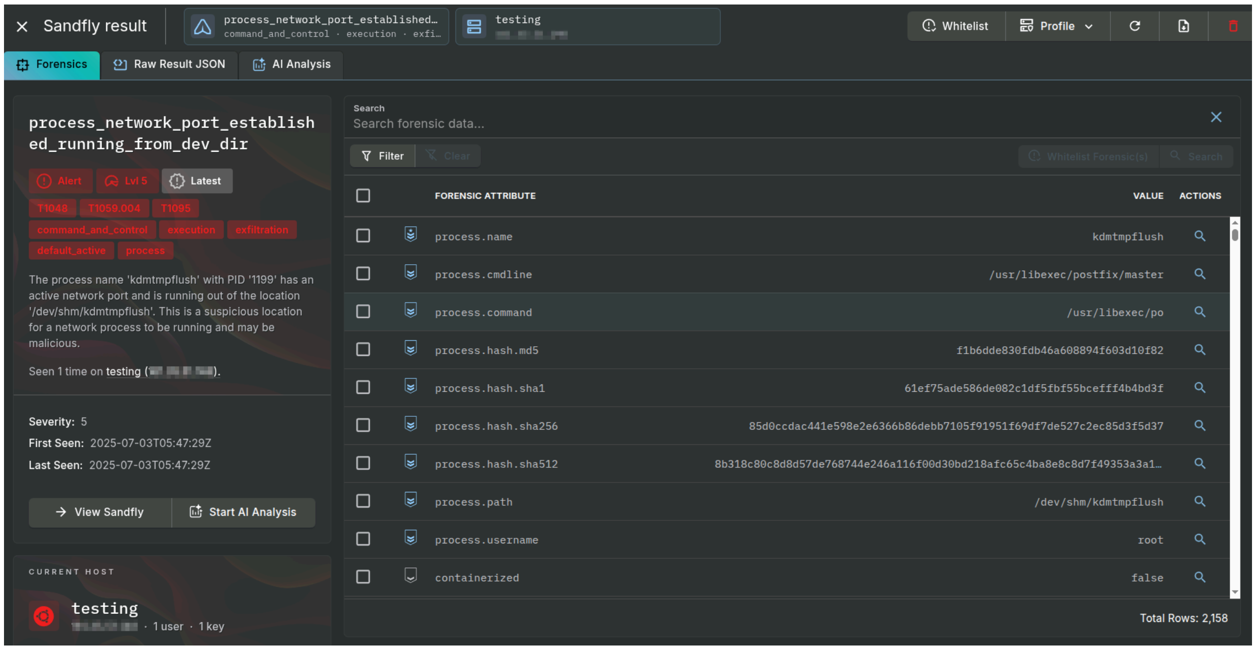
Total Rows: 770

Further, we see examples of the anti-forensic values in the process environment forensic data Sandfly collected. Specifically, we see attempts to make HISTFILE not write command shell history by directing it to */dev/null*, and something similar for MySQL.


```
    "environ": [
      "HOME=/tmp",
      "PS1=[\\u@\\h \\W]\\\\$ ",
      "HISTFILE=/dev/null",
      "MYSQL_HISTFILE=/dev/null",
      "PATH=/bin:/usr/kerberos/sbin:/usr/kerberos/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/usr/X11R6/bin:./bin",
      "vt100"
    ],
```

Backdoor Shell Network Connections

As BPFDoor must communicate with the network for the shell to work, we get additional alerts around network activity. For Version 1, the operation from the */dev/shm* directory generates an alert, but we get other alerts as well.


Sandfly result
process_network_port_established…
command_and_control · execution · exfi…
testing
Whitelist
Profile
Forensics
Raw Result JSON
AI Analysis

process_network_port_established_running_from_dev_dir

Alert · Lvl 5 · Latest

T1048 · T1059.004 · T1095

command_and_control · execution · exfiltration · default_active · process

The process name 'kdmtmpflush' with PID '1199' has an active network port and is running out of the location '/dev/shm/kdmtmpflush'. This is a suspicious location for a network process to be running and may be malicious.

Seen 1 time on testing ([redacted]).

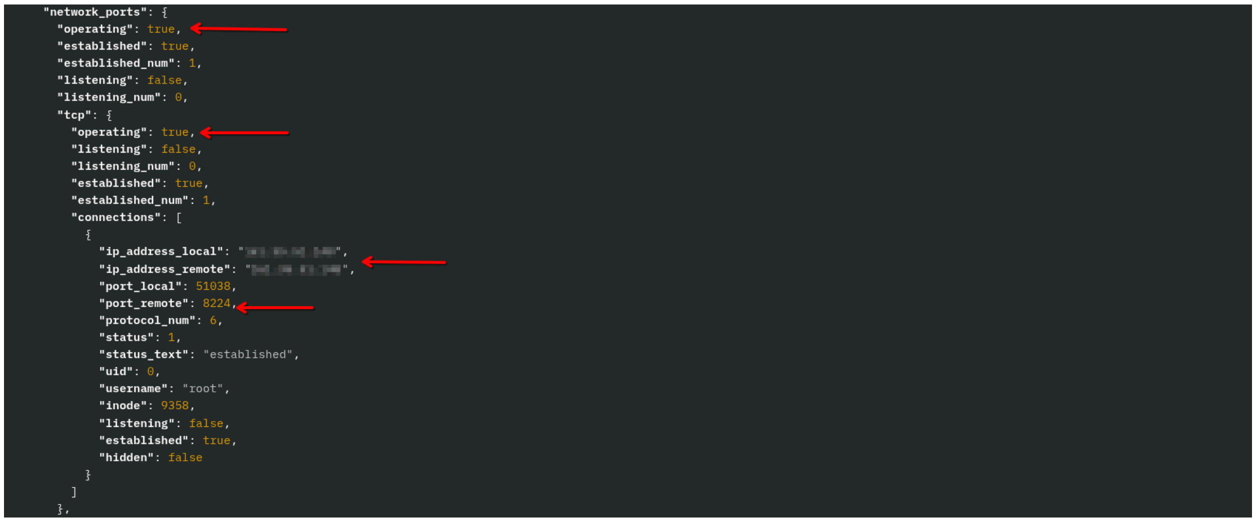
Severity: 5
First Seen: 2025-07-03T05:47:29Z
Last Seen: 2025-07-03T05:47:29Z

testing
[redacted] · 1 user · 1 key

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	kdmtmpflush	🔍
process.cmdline	/usr/libexec/postfix/master	🔍
process.command	/usr/libexec/po	🔍
process.hash.md5	f1b6dde830fdb46a608894f603d10f82	🔍
process.hash.sha1	61ef75ade586de082c1df5fbf55bcefff4b4bd3f	🔍
process.hash.sha256	85d0ccdac441e598e2e6366b86debb7105f91951f69df7de527c2ec85d3f5d37	🔍
process.hash.sha512	8b318c80c8d8d57de768744e246a116f00d30bd218afc65c4ba8e8c8d7f49353a3a1…	🔍
process.path	/dev/shm/kdmtmpflush	🔍
process.username	root	🔍
containerized	false	🔍

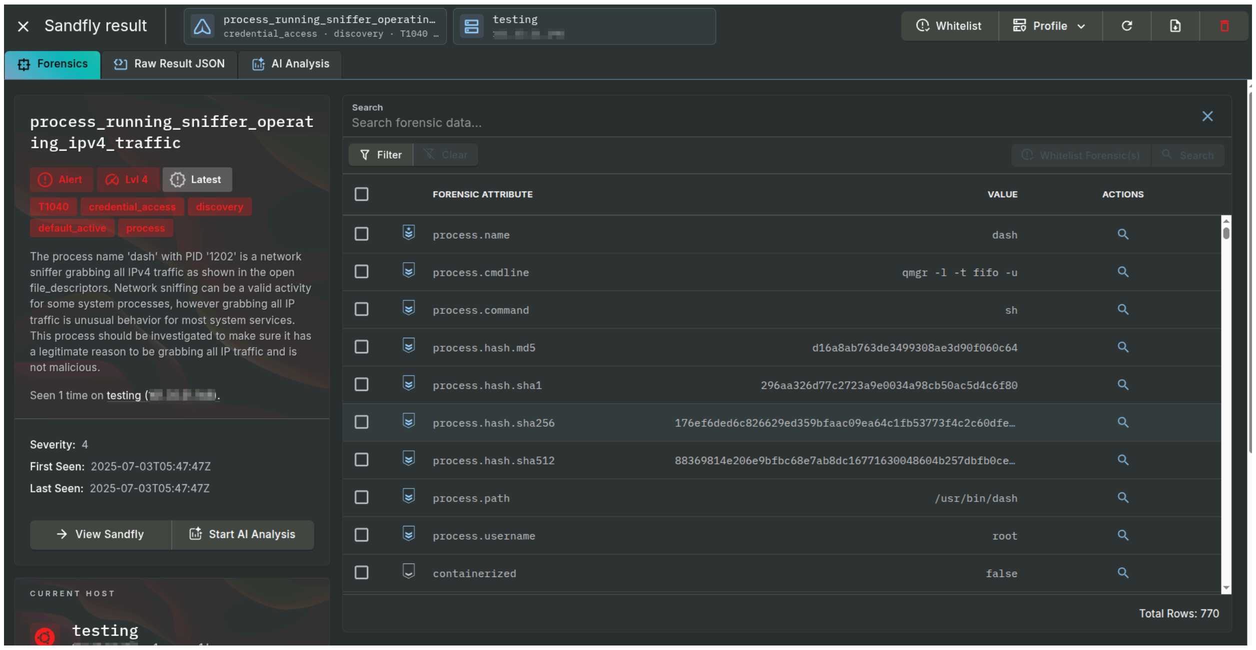
Total Rows: 2,158

Here Sandfly's forensic data shows all network activity for the suspicious process. For the bind shell, we see an established TCP connection to a remote system on an unusual port in the reverse shell variant.



```
"network_ports": {
  "operating": true,
  "established": true,
  "established_num": 1,
  "listening": false,
  "listening_num": 0,
  "tcp": {
    "operating": true,
    "listening": false,
    "listening_num": 0,
    "established": true,
    "established_num": 1,
    "connections": [
      {
        "ip_address_local": "[redacted]",
        "ip_address_remote": "[redacted]",
        "port_local": 51038,
        "port_remote": 8224,
        "protocol_num": 6,
        "status": 1,
        "status_text": "established",
        "uid": 0,
        "username": "root",
        "inode": 9358,
        "listening": false,
        "established": true,
        "hidden": false
      }
    ]
  },
```

Finally, we get the shell itself flagged also as a process with unusual activity concerning sniffing IPv4 traffic. Shell processes should not be involved in sniffing network traffic, so again this is a high risk alert.



process_running_sniffer_operating_ipv4_traffic

The process name 'dash' with PID '1202' is a network sniffer grabbing all IPv4 traffic as shown in the open file_descriptors. Network sniffing can be a valid activity for some system processes, however grabbing all IP traffic is unusual behavior for most system services. This process should be investigated to make sure it has a legitimate reason to be grabbing all IP traffic and is not malicious.

Seen 1 time on testing ([redacted]).

Severity: 4
First Seen: 2025-07-03T05:47:47Z
Last Seen: 2025-07-03T05:47:47Z

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	dash	
process.cmdline	qmgr -l -t fifo -u	
process.command	sh	
process.hash.md5	d16a8ab763de3499308ae3d90f060c64	
process.hash.sha1	296aa326d77c2723a9e0034a98cb50ac5d4c6f80	
process.hash.sha256	176ef6ded6c826629ed359bfaac09ea64c1fb53773f4c2c60dfe…	
process.hash.sha512	88369814e206e9bfbc68e7ab8dc16771630048604b257dbfb0ce…	
process.path	/usr/bin/dash	
process.username	root	
containerized	false	

Total Rows: 770

Drift Detection

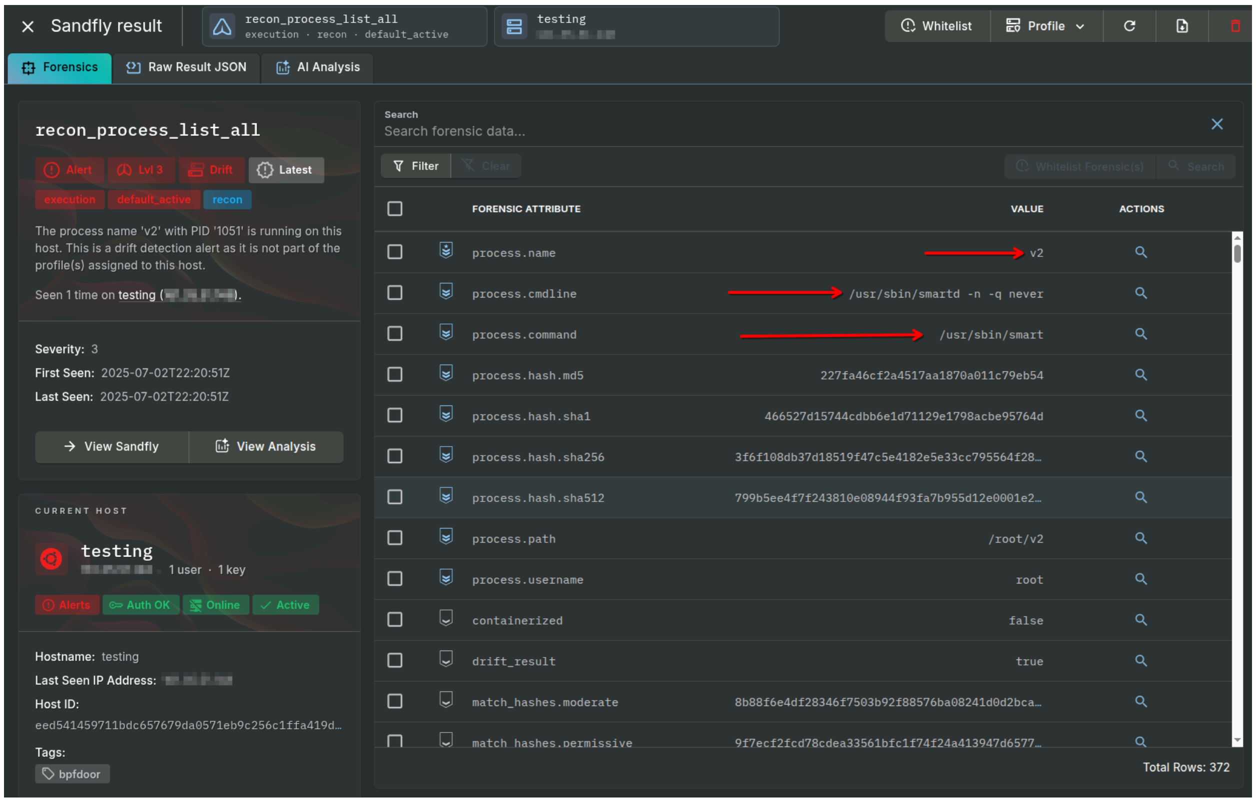
A powerful feature in Sandfly is our ability to do drift detection on a system. Sandfly allows customers to profile a system and track what authorized processes, users, files, directories, systemd services, cron jobs, kernel modules, and other critical system areas are running. Users can decide to watch only for new processes, new users, etc. as they deem necessary. There are many uses for this feature described here:

[Sandfly Security Agentless Drift Detection]()

In the basic form, customers can profile a known-good host and then apply that profile to similar systems. For instance, a customer may be running identical Linux Virtual Machine (VM) images that should not change. If a new process, user, etc. were to show up on the monitored hosts, then alerts are generated as a drift from what is expected.

Alternatively, it can be used for incident response by profiling a known-good system, then taking that profile to scan similar hosts and see what is different. For instance, customers can profile a known-good server or network device in their lab. Then, security teams can take that profile to sweep their network of similar systems. Any process or other difference on the scanned systems immediately becomes visible.

For purposes of BPFDoor, drift detection provides an excellent detection mechanism. If customers know what a system should be running, any new process such as BPFDoor sticks out immediately. Below we see a process drift alert for BPFDoor as an example. The process binary name and masquerading data is immediately shown.



recon_process_list_all

The process name 'v2' with PID '1051' is running on this host. This is a drift detection alert as it is not part of the profile(s) assigned to this host.

Seen 1 time on testing ([redacted]).

Severity: 3
First Seen: 2025-07-02T22:20:51Z
Last Seen: 2025-07-02T22:20:51Z

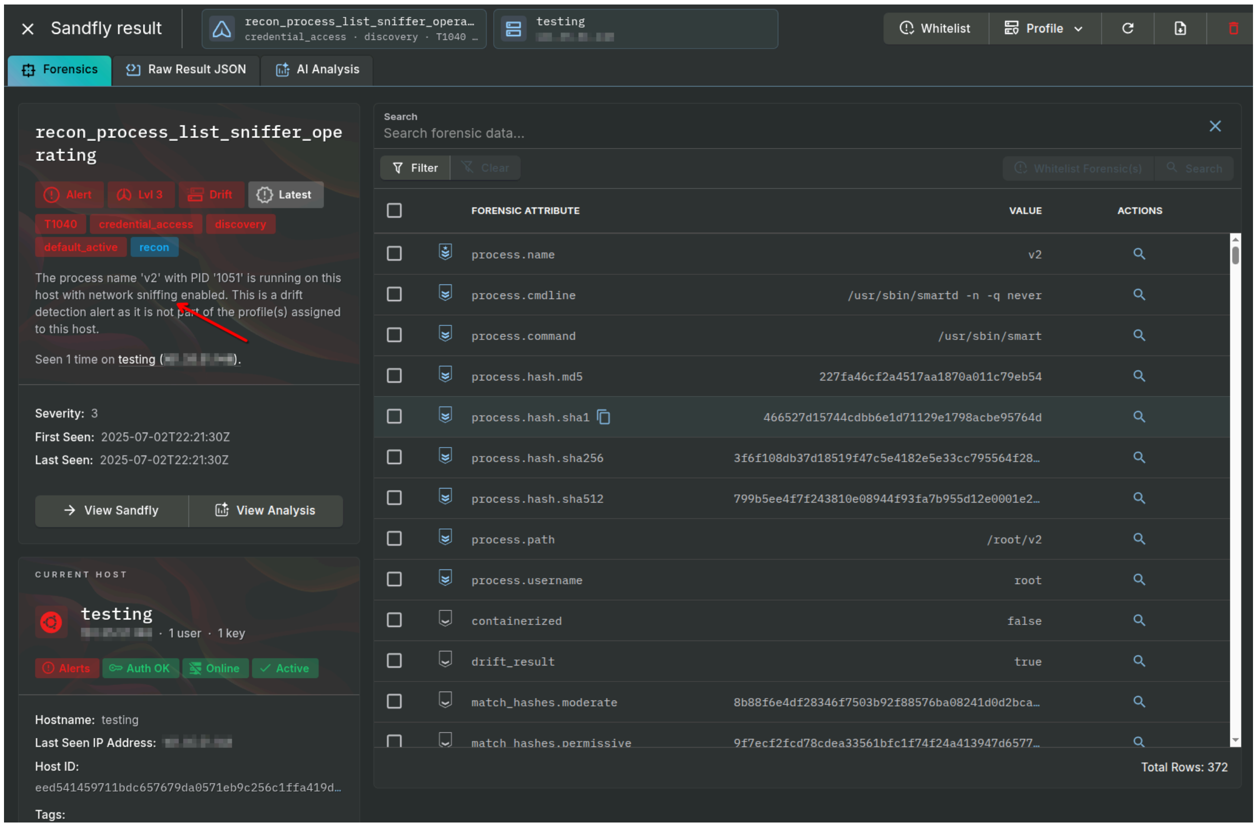
testing
[redacted] 1 user · 1 key

Hostname: testing
Last Seen IP Address: [redacted]
Host ID:
eed541459711bdc657679da0571eb9c256c1ffa419d…
Tags:
bpfdoor

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	v2	
process.cmdline	/usr/sbin/smartd -n -q never	
process.command	/usr/sbin/smart	
process.hash.md5	227fa46cf2a4517aa1870a011c79eb54	
process.hash.sha1	466527d15744cdbb6e1d71129e1798acbe95764d	
process.hash.sha256	3f6f108db37d18519f47c5e4182e5e33cc795564f28…	
process.hash.sha512	799b5ee4f7f243810e08944f93fa7b955d12e0001e2…	
process.path	/root/v2	
process.username	root	
containerized	false	
drift_result	true	
match_hashes.moderate	8b88f6e4df28346f7503b92f88576ba08241d0d2bca…	
match_hashes.permissive	9f7ecf2fcd78cdea33561bfc1f74f24a413947d6577…	

Total Rows: 372

Additionally, customers can use the built in *recon_process_list_sniffer_operating* to build a list of all processes sniffing traffic. Incident response teams can add known good processes to a result profile or whitelist, and then quickly find sniffing processes which are not authorized on any host. This is an excellent way to identify potential BPFDoor victims.



recon_process_list_sniffer_operating

The process name 'v2' with PID '1051' is running on this host with network sniffing enabled. This is a drift detection alert as it is not part of the profile(s) assigned to this host.

Seen 1 time on testing ([redacted]).

Severity: 3
First Seen: 2025-07-02T22:21:30Z
Last Seen: 2025-07-02T22:21:30Z

testing
[redacted] · 1 user · 1 key

Hostname: testing
Last Seen IP Address: [redacted]
Host ID:
eed541459711bdc657679da0571eb9c256c1ffa419d…
Tags:

FORENSIC ATTRIBUTE	VALUE	ACTIONS
process.name	v2	🔍
process.cmdline	/usr/sbin/smartd -n -q never	🔍
process.command	/usr/sbin/smart	🔍
process.hash.md5	227fa46cf2a4517aa1870a011c79eb54	🔍
process.hash.sha1	466527d15744cdbb6e1d71129e1798acbe95764d	🔍
process.hash.sha256	3f6f108db37d18519f47c5e4182e5e33cc795564f28…	🔍
process.hash.sha512	799b5ee4f7f243810e08944f93fa7b955d12e0001e2…	🔍
process.path	/root/v2	🔍
process.username	root	🔍
containerized	false	🔍
drift_result	true	🔍
match_hashes.moderate	8b88f6e4df28346f7503b92f88576ba08241d0d2bca…	🔍
match_hashes.permissive	9f7ecf2fcd78cdea33561bfc1f74f24a413947d6577…	🔍

Total Rows: 372

Threat Feed and Cryptographic Hash Searches

Many times cryptographic hashes are supplied to help find malware. While this can sometimes work on Windows systems, on Linux it is extremely unreliable. **We do not recommend using cryptographic hashes as a primary detection strategy for any Linux malware.** It is trivial on Linux to alter a binary hash and have it completely evade detection using cryptographic hashes. We even show how to do it in this video:

Stop Using Cryptographic Hashes to Find Linux Malware

With that said, sometimes using hashes can be helpful for malware that has not changed yet. In this case, customers can use Sandfly's built in Threat Feed feature. This feature allows you to tie into public threat intel sources or use your own list of hashes at a dedicated URL.

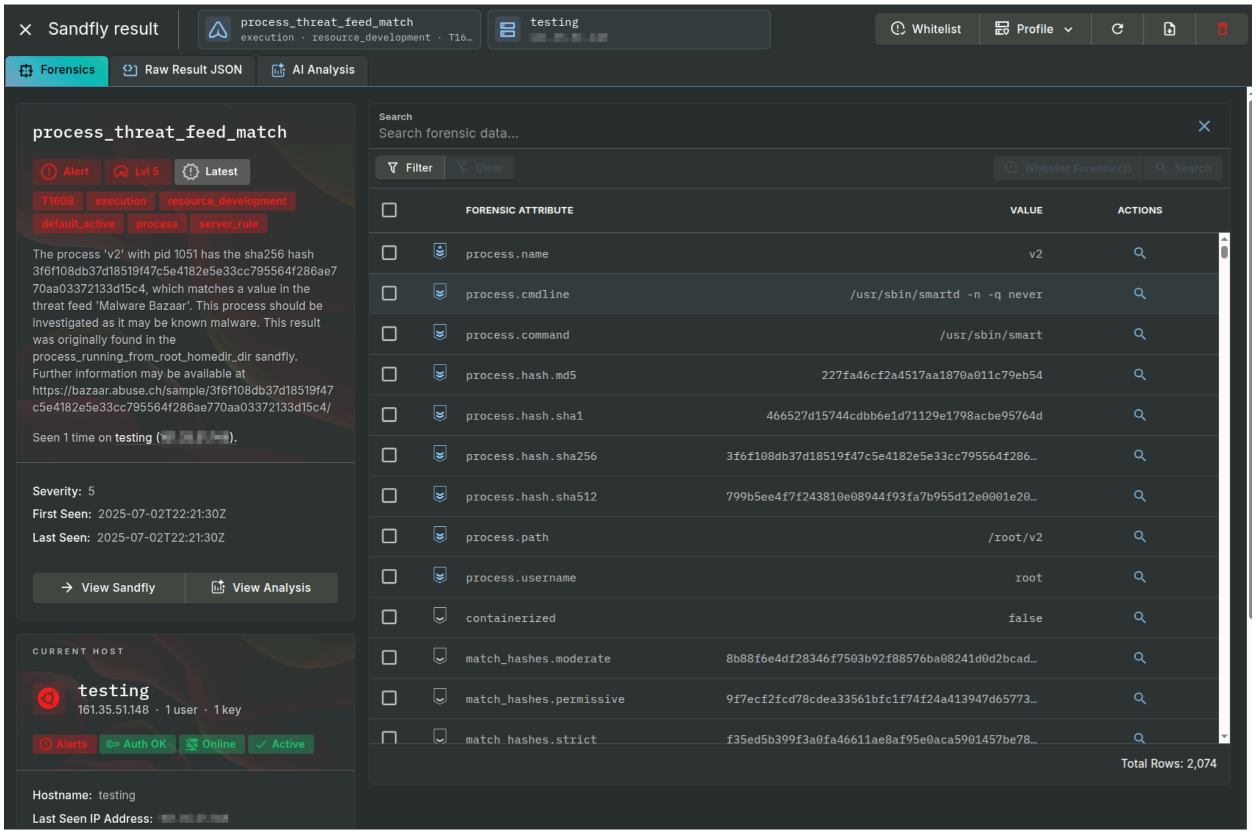
Configuring Threat Feeds

Configuring threat feeds on Sandfly is very easy and documented below. Customers can use commercial feeds, free feeds, or even internal feeds they maintain on their own.

Sandfly Threat Feed Configuration

Threat Feed Alerts

When any process or file is collected by Sandfly, it generates SHA512, SHA256, SHA1 and MD5 hashes. These are compared against threat feeds lists and if there is a match an alert is generated. Below we see an alert for a BPFDoor Version 2 binary from a threat feed.



process_threat_feed_match

The process 'v2' with pid 1051 has the sha256 hash 3f6f108db37d18519f47c5e4182e5e33cc795564f286ae770aa03372133d15c4, which matches a value in the threat feed 'Malware Bazaar'. This process should be investigated as it may be known malware. This result was originally found in the process_running_from_root_homedir_dir sandfly. Further information may be available at https://bazaar.abuse.ch/sample/3f6f108db37d18519f47c5e4182e5e33cc795564f286ae770aa03372133d15c4/

Seen 1 time on testing ([redacted]).

Severity: 5
First Seen: 2025-07-02T22:21:30Z
Last Seen: 2025-07-02T22:21:30Z

CURRENT HOST
testing
161.35.51.148 · 1 user · 1 key

Hostname: testing
Last Seen IP Address: [redacted]

FORENSIC ATTRIBUTE	VALUE
process.name	v2
process.cmdline	/usr/sbin/smartd -n -q never
process.command	/usr/sbin/smart
process.hash.md5	227fa46cf2a4517aa1870a011c79eb54
process.hash.sha1	466527d15744cdbb6e1d71129e1798acbe95764d
process.hash.sha256	3f6f108db37d18519f47c5e4182e5e33cc795564f286…
process.hash.sha512	799b5ee4f7f243810e08944f93fa7b955d12e0001e20…
process.path	/root/v2
process.username	root
containerized	false
match_hashes.moderate	8b88f6e4df28346f7503b92f88576ba08241d0d2bcad…
match_hashes.permissive	9f7ecf2fcd78cdea33561bfc1f74f24a413947d65773…
match_hashes.strict	f35ed5b399f3a0fa46611ae8af95e0aca5901457be78…

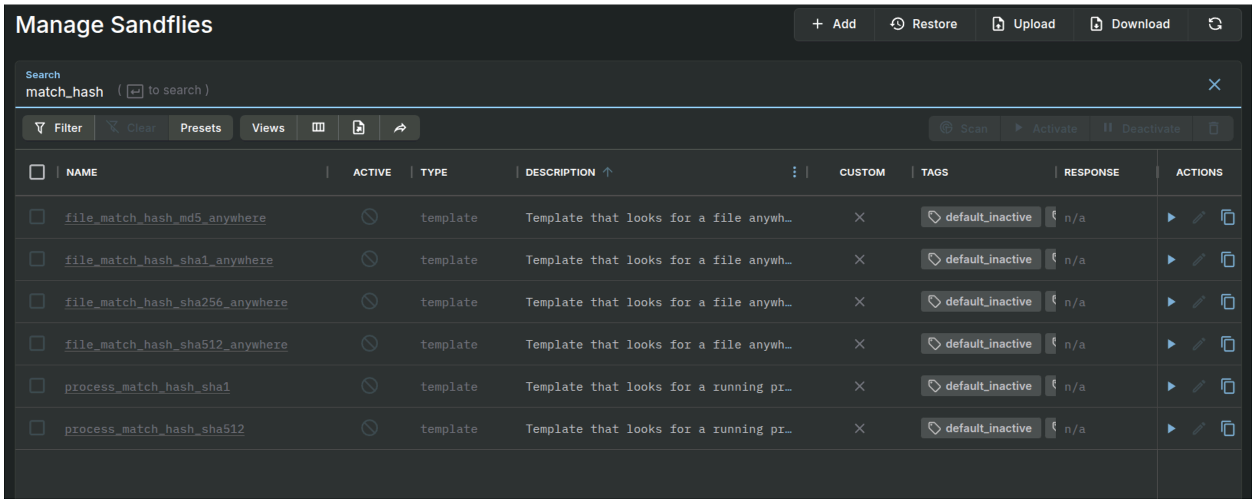
Total Rows: 2,074

Custom Hash Threat Hunting

Customers can also create on-demand hash searches. For instance, searching for "*match_hash*" under the Sandflies area will list several templates that can be cloned and converted into hash search modules. If you need assistance in doing this, please reach out to our support team for assistance.

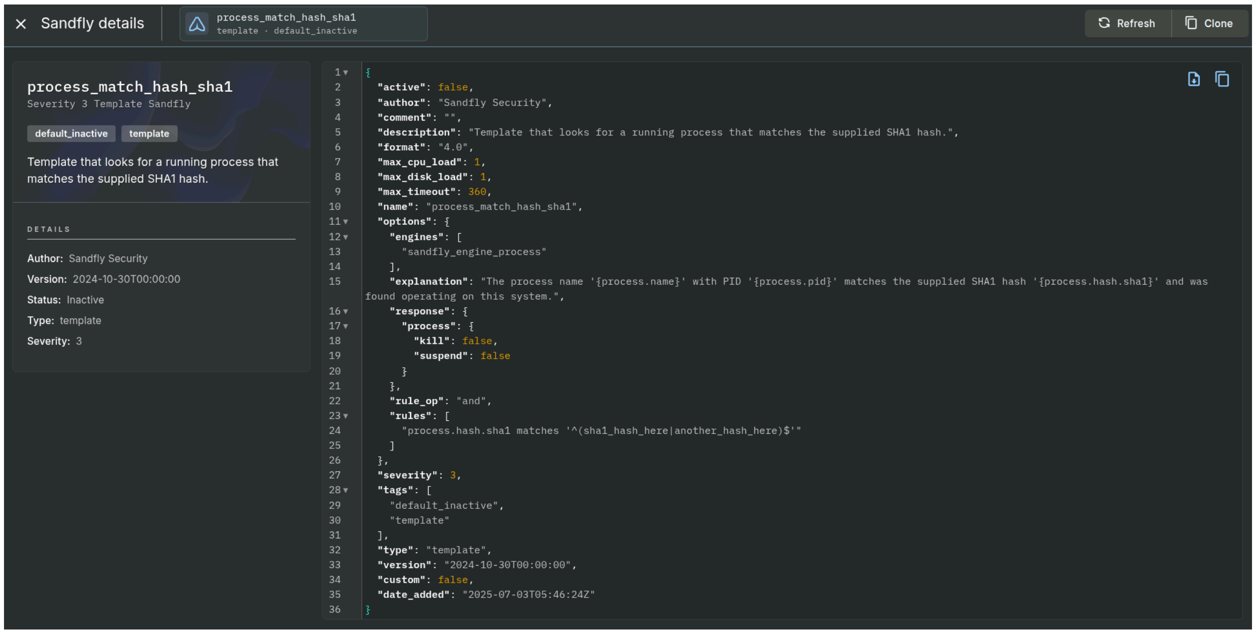
In most cases, if you wish to search for process hashes it will be easier to make your own threat feed hash list and use the Threat Feed feature. Again, please reach out to support if you need help with this.

Below we show a list of the hash match templates, and the template for process hash matching. Sandfly can search for process and file hashes, but it can also search inside files, systemd services, cron entries, at jobs, user entries, and more.



Search: match_hash

NAME	ACTIVE	TYPE	DESCRIPTION	CUSTOM	TAGS	RESPONSE	ACTIONS
file_match_hash_md5_anywhere		template	Template that looks for a file anywh…	✕	default_inactive	n/a	
file_match_hash_sha1_anywhere		template	Template that looks for a file anywh…	✕	default_inactive	n/a	
file_match_hash_sha256_anywhere		template	Template that looks for a file anywh…	✕	default_inactive	n/a	
file_match_hash_sha512_anywhere		template	Template that looks for a file anywh…	✕	default_inactive	n/a	
process_match_hash_sha1		template	Template that looks for a running pr…	✕	default_inactive	n/a	
process_match_hash_sha512		template	Template that looks for a running pr…	✕	default_inactive	n/a	



process_match_hash_sha1
Severity 3 Template Sandfly

Template that looks for a running process that matches the supplied SHA1 hash.

Author: Sandfly Security
Version: 2024-10-30T00:00:00
Status: Inactive
Type: template
Severity: 3

```
{
  "active": false,
  "author": "Sandfly Security",
  "comment": "",
  "description": "Template that looks for a running process that matches the supplied SHA1 hash.",
  "format": "4.0",
  "max_cpu_load": 1,
  "max_disk_load": 1,
  "max_timeout": 360,
  "name": "process_match_hash_sha1",
  "options": {
    "engines": [
      "sandfly_engine_process"
    ],
    "explanation": "The process name '{process.name}' with PID '{process.pid}' matches the supplied SHA1 hash '{process.hash.sha1}' and was
found operating on this system.",
    "response": {
      "process": {
        "kill": false,
        "suspend": false
      }
    },
    "rule_op": "and",
    "rules": [
      "process.hash.sha1 matches '^(sha1_hash_here|another_hash_here)$'"
    ]
  },
  "severity": 3,
  "tags": [
    "default_inactive",
    "template"
  ],
  "type": "template",
  "version": "2024-10-30T00:00:00",
  "custom": false,
  "date_added": "2025-07-03T05:46:24Z"
}
```

Optional Sandfly Modules

The detection modules discussed so far are all enabled by default. They have low false alarm risk and provide exceptional coverage for BPFDoor variants we've seen. However, we have some additional modules that incident responders can enable if they want to widen the net at the risk of higher false positive risks.

process_running_sniffer_libraries_libpcap - Show all sniffers using *libpcap* libraries for packet capture.

process_running_sniffer_operating - Show any process sniffing any kind of traffic.

process_running_sniffer_operating_tuned - Show any process sniffing traffic, but ignore some common system services that do it.

process_running_sniffer_operating_all_traffic - Show any process sniffing traffic set to grab all traffic regardless of protocol.

process_masquerade_shell - Show any shells that may be masquerading under a false name.

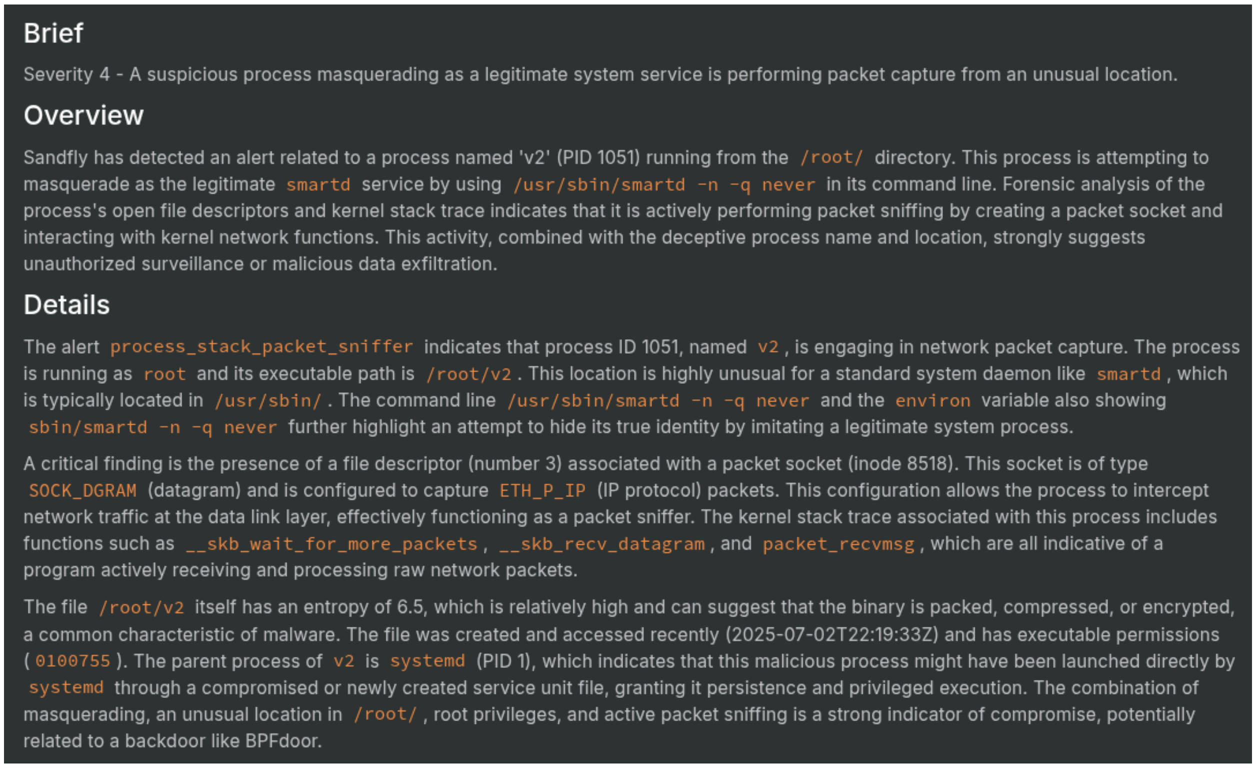
process_masquerade_check - Show any process that may be masquerading under a false name.

process_masquerade_binary_renamed - Show any binary that has been renamed to masquerade what it is.

These modules may provide additional detection coverage, but may not be worth the possible extra false positive risks. Security teams can try them in their environment to see if they are useful however.

AI Analysis

Sandfly provides excellent and very reliable Linux forensics data. As a result, our AI integration feature starting in Sandfly 5.5 and higher provides extremely good analysis and investigation advice. Below we see an analysis of an alert from BPFDoor Version 2 when found on a system.



Brief

Severity 4 - A suspicious process masquerading as a legitimate system service is performing packet capture from an unusual location.

Overview

Sandfly has detected an alert related to a process named 'v2' (PID 1051) running from the `/root/` directory. This process is attempting to masquerade as the legitimate `smartd` service by using `/usr/sbin/smartd -n -q never` in its command line. Forensic analysis of the process's open file descriptors and kernel stack trace indicates that it is actively performing packet sniffing by creating a packet socket and interacting with kernel network functions. This activity, combined with the deceptive process name and location, strongly suggests unauthorized surveillance or malicious data exfiltration.

Details

The alert `process_stack_packet_sniffer` indicates that process ID 1051, named `v2`, is engaging in network packet capture. The process is running as `root` and its executable path is `/root/v2`. This location is highly unusual for a standard system daemon like `smartd`, which is typically located in `/usr/sbin/`. The command line `/usr/sbin/smartd -n -q never` and the `environ` variable also showing `sbin/smartd -n -q never` further highlight an attempt to hide its true identity by imitating a legitimate system process.

A critical finding is the presence of a file descriptor (number 3) associated with a packet socket (inode 8518). This socket is of type `SOCK_DGRAM` (datagram) and is configured to capture `ETH_P_IP` (IP protocol) packets. This configuration allows the process to intercept network traffic at the data link layer, effectively functioning as a packet sniffer. The kernel stack trace associated with this process includes functions such as `__skb_wait_for_more_packets`, `__skb_recv_datagram`, and `packet_recvmsg`, which are all indicative of a program actively receiving and processing raw network packets.

The file `/root/v2` itself has an entropy of 6.5, which is relatively high and can suggest that the binary is packed, compressed, or encrypted, a common characteristic of malware. The file was created and accessed recently (2025-07-02T22:19:33Z) and has executable permissions (`0100755`). The parent process of `v2` is `systemd` (PID 1), which indicates that this malicious process might have been launched directly by `systemd` through a compromised or newly created service unit file, granting it persistence and privileged execution. The combination of masquerading, an unusual location in `/root/`, root privileges, and active packet sniffing is a strong indicator of compromise, potentially related to a backdoor like BPFdoor.

Conclusions

BPFDoor is a simple but effective backdoor that has been widely used to attack telecommunications and other critical infrastructure providers globally. While it contains many evasive features, it can be easily found by Sandfly in seconds without needing to deploy endpoint agents. If you are experiencing a breach, please reach out to our team and we can supply an Incident Response license to help.

Sandfly is compatible with Linux systems over a decade old to modern cloud deployments. We also work on air-gap and mixed architecture environments using Intel, AMD, ARM, MIPS, and IBM CPUs. Sandfly can work not just on traditional servers, but embedded systems and IoT devices with high compatibility and very low risk of downtime. With no endpoint agent, Sandfly deploys instantly and can give answers in seconds about the security state of Linux systems.

Please visit our website for more information:

www.sandflysecurity.com

Links

Sandfly Security BPFDoor Research

BPFDoor - An Evasive Linux Backdoor Technical Analysis
BPFDoor Presentation
BPFDoor Slides

HaxRob BPFDoor Research

BPFDoor - Part 1 The Past
BPFDoor - Part 2 The Present