Los Alamos NATIONAL LABORATORY

Endpoint Detection: Rust & eBPF for HPC user activity monitoring

Skip McGee, HPC
Thomas Bertschinger, HPC

June 26, 2025

LA-UR-25-26155

Table of Contents

Background: the changing landscape of HPC

- The HPC security model has historically been a hard outer shell and a relatively gooey middle.
- HPC has been reluctant to field applications inside the cluster focused on monitoring and securing user workloads, due to performance impact considerations.
- LANL collaborating with OpenAI & implementing OpenAI models on Razorback (Testbed) and Venado (Production)
- New security paradigm: provide platform security while also providing cybersecurity as a service for externally managed OpenAI services and applications.
- The growth of EDR (Endpoint Detection and Response) products has been significant in the last 5-10 years.
- Growing emphasis on understanding the ability to apply Zero Trust principles to HPC.
- Anticipated workload changes to support GPU-intensive workloads like AI/ML training & inference.

Problem Definition:

Implement effective monitoring of user activity with minimal performance impact

Security End State:

A malicious user probably won't run `evil.sh`, but we CAN:

- ✓ *Audit to scope compromise (after the fact)*
- ✓ *Audit to detect exploitation (at/after the act)*
- ✓ *Pattern user behavior ****(before the act)****
- ✓ *Detect reconnaissance activity ****(before the act)****

We can NOT:

- ❑ *Significantly degrade node / cluster performance*

Identified Solutions

- Improve and maintain what we were already doing:
  - Custom system auditing (`auditd` rules)
  - Syslog forwarding audit and other events (`rsyslog`)
- Implement an existing EDR?
- Implement a terminal logger, needs to cover:
  - Bash
  - Zsh
  - Tcsh
  - other shells
- Develop our own solution?

EDR Alternatives

- CrowdStrike
- SentinelOne

What do these solutions have in common?

- Licensing Costs ($$$)
- eBPF
- Linux agents (cross platform, but we only run Linux)
- Monitoring platform (client / server linking architecture)
- Wide focus:
  - Desktop => Server
  - Lots of applications
  - Lots of host configurations

eBPF Review

- eBPF is old: formerly known as Extended Berkley Packet Filters, integrated into the Linux kernel in 2014, successor to BPF (implemented in the Linux kernel in 1993)
- BTF CO-RE (Compile Once - Run Everywhere)
  - "a **concept** to build cross-version kernel eBPF application by building in a single binary by bringing together the BTF type information, libbpf and the compiler."
- Languages: C, Go, Rust, Python
- Multiple different projects that are worth mentioning:
  - BCC Tools
  - Rust: libbpf-rs, aya
- eBPF is resource limited (especially memory)
- Exposed program types include tracepoints, user probes and kernel probes

The Aya Rust Project

- Pros:
  - Write Rust, compile into CO-RE compliant eBPF bytecode: “Support for the BPF Type Format (BTF), which is transparently enabled when supported by the target kernel. This allows eBPF programs compiled against one kernel version to run on different kernel versions without the need to recompile.” https://github.com/aya-rs/awesome-aya
  - Provide a single binary that contains the entire program (minimize host dependencies): “aya doesn't require a kernel build or compiled headers… or a C toolchain” https://github.com/aya-rs/awesome-aya
  - Leverage Rust workspaces for userspace, common and eBPF programs & exposes standard eBPF methods: “Support for function call relocation and global data maps, which allows eBPF programs to make function calls and use global variables and initializers” https://github.com/aya-rs/awesome-aya
  - There is significant development in this space: https://github.com/zoidyzoidzoid/awesome-ebpf
- Cons:
  - Currently in the unstable nightly build (as of rustc 1.90.0-nightly)
  - Limited documentation
  - Linux Kernel >= 4.3 for a majority of features

Panhandle

- Filter by uid
- Target shell and/or execve syscall
- Human or JSON results
- Console, file or http output

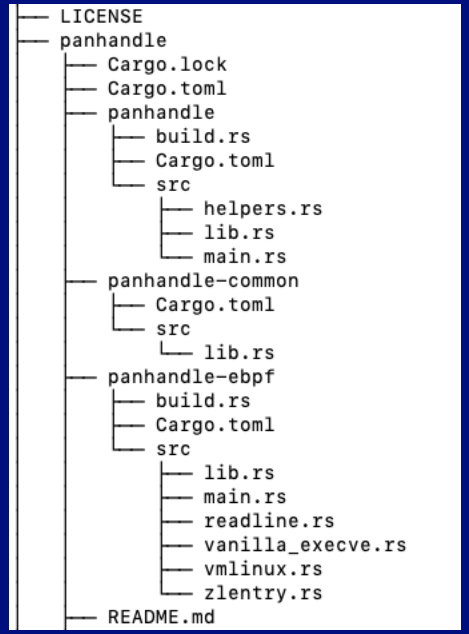
```
Panhandle provides the ability to monitor execve syscalls to identify specific interesting user behavior, as well as the ability to monitor specific s
hells (bash, zsh, and fmsh) on a linux host. Several optional filters enable an administrator to selectively apply criterion to examine desired user b
ehavior. These include UID filtering as well as filtering for the use of specific executables. Specified events are logged for further reporting and/o
r analysis. Logging options include file or terminal output for selected events.

Usage: panhandle [OPTIONS]

Options:
  -b, --bash
          Monitor events from the bash shell using the readline function
  -d, --debug
          Invalidates json option and returns output to help identify why the program might not pick up on desired events
      --exclude-min-uid <EXCLUDE_MIN_UID>
          Exclude a range of uids from monitoring, specify the MINIMUM uid of the range. Defaults to 1
      --exclude-max-uid <EXCLUDE_MAX_UID>
          Exclude a range of uids from monitoring, specify the MAXIMUM uid of the range. Defaults to 999
  -e, --executables <EXECUTABLES>...
          Specify a comma separated list of absolute executable paths to exclusively monitor. Maximum limit is 10
  -f, --fmsh
          Monitor events from the fmsh shell using the readline function. This also returns events from the bash shell as fmsh is a derivative of bash
      --file <FILE>
          Specify the absolute path to the desired log file
      --http <HTTP>
          Output events to the specified HTTP endpoint
      --include-uid <INCLUDE_UID>...
          Only include events from the specified uid(s), can be a comma separated list. Maximum limit is 10
  -j, --json
          Output events as valid json strings separated by newlines
  -s, --syscall-execve
          Monitor events from the execve syscall. This is the program default if no other arguments are selected
      --shells
          Output execve syscall events only from the shells: bash, tcsh, zsh & fmsh. This does not effect the output of the bash or zsh options
  -v, --verbose
          Verbose output
  -z, --zsh
          Monitor events from the zsh shell using the zlentry function
  -h, --help
          Print help
  -V, --version
          Print version
```

Panhandle Components

- Userland program (CLI, argument parsing, output)
- Common (shared) methods, structs, etc.
- eBPF Program(s)



RPM Contains:

- Binary
- Man page
- Systemd Service

```
[root@marcella2 ~]# rpm -ql panhandle
/usr/lib/systemd/system/panhandle.service
/usr/sbin/panhandle
/usr/share/man/man1/panhandle.1
```

Systemd Service:

```
● panhandle.service - Run the panhandle binary via systemd
     Loaded: loaded (/usr/lib/systemd/system/panhandle.service; enabled; preset: disabled)
     Active: active (running) since Tue 2025-06-17 06:50:46 MDT; 1 week 0 days ago
       Docs: https://lisdi-git.lanl.gov/hpc-cyber/panhandle
   Main PID: 213190 (panhandle)
      Tasks: 9 (limit: 202426)
     Memory: 24.8M
        CPU: 935ms
     CGroup: /system.slice/panhandle.service
             └─213190 /usr/sbin/panhandle --fmsh --shells --syscall-execve --exclude-min-uid

Jun 17 06:50:46 marcella2.lanl.gov systemd[1]: Started Run the panhandle binary via systemd.
```

Panhandle Output (Bash Shell Example)

Human readable, `panhandle --bash --file /var/log/panhandle/panhandle.log`:

17:08:44 [INFO] moniker: elvis, **entry: uname -a,** command: bash, uid: 77777, pid: 1604000, gid: 66666, tgid: 1604000, ts_utc: '2025-06-24_17:08:44'
17:08:46 [INFO] moniker: elvis, **entry: whoami,** command: bash, uid: 77777, pid: 1604000, gid: 66666, tgid: 1604000, ts_utc: '2025-06-24_17:08:46'
17:08:52 [INFO] moniker: elvis, **entry: bash evil.sh,** command: bash, uid: 77777, pid: 1604000, gid: 66666, tgid: 1604000, ts_utc: '2025-06-24_17:08:52'

JSON, `panhandle --bash –json --file /var/log/panhandle/panhandle.log`:

17:09:40 [INFO] {"moniker": ”elvis", **"entry": "whoami",** "command": "bash", "uid": " 77777 ", "pid": "1604000", "gid": " 66666 ", "tgid": "1604000", "ts_utc": "2025-06-24_17:09:40"}
17:09:43 [INFO] {"moniker": ”elvis", **"entry": "uname -a",** "command": "bash", "uid": " 77777 ", "pid": "1604000", "gid": " 66666 ", "tgid": "1604000", "ts_utc": "2025-06-24_17:09:43"}
17:09:45 [INFO] {"moniker": ”elvis", **"entry": "bash evil.sh",** "command": "bash", "uid": " 77777 ", "pid": "1604000", "gid": " 66666 ", "tgid": "1604000", "ts_utc": "2025-06-24_17:09:45"}

Panhandle Output (Execve Syscall Example)

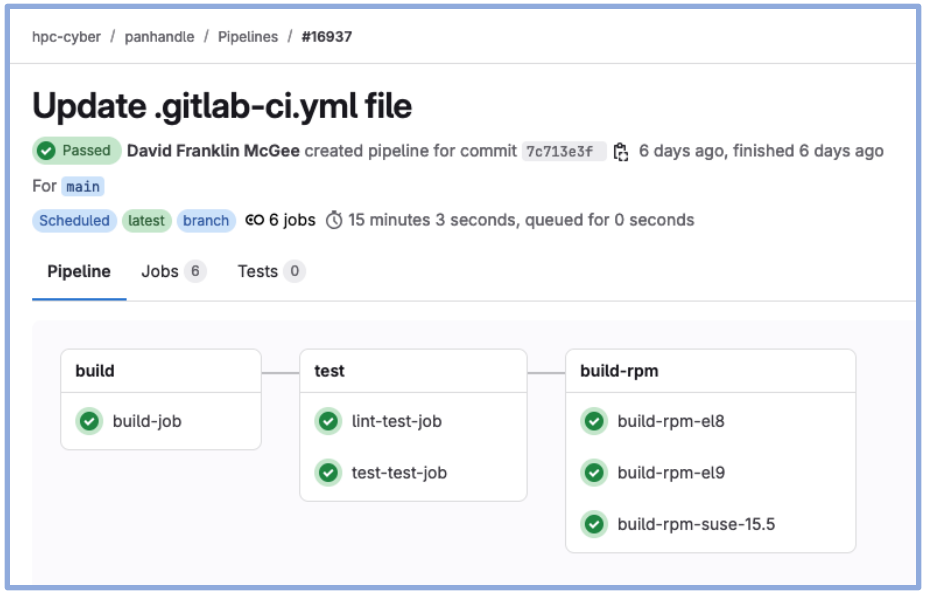
Human readable, `panhandle --syscall-execve –file /var/log/panhandle/panhandle.log`:

17:16:49 [INFO] moniker: elvis, **filename: /usr/bin/bash,** command: bash, uid: 77777, pid: 1604264, gid: 66666, tgid: 1604264, **args: [bash,evil.sh,],** envs: [SHELL=/bin/bash,HOSTNAME=vegas.lanl.gov,KRB5_PKCS11=PKCS11:/usr/lib64/pkcs11/libnetpkcs11.so,MODULES_VERBOSITY=concise, PWD=/home/elvis,KRB5CCNAME=FILE:/tmp/krb5cc_77777_TNRWne,LOGNAME= elvis,XDG_SESSION_TYPE=tty,HOME=/home/ elvis,LANG=en_US.UTF-8, SSH_CONNECTION=777.777.777.777 55122 666.666.666.666 22,XDG_SESSION_CLASS=user,TERM=xterm-256color,], ts_utc: '2025-06-24_17:16:49'

JSON, `panhandle --syscall-execve --file /var/log/panhandle/panhandle.log `:

17:20:44 [INFO] {"moniker": ”elvis", **"filename": "/usr/bin/bash",** "command": "bash", "uid": ”77777", "pid": "1604295", "gid": ”66666", "tgid": "1604295", **"args": ["bash", "evil.sh"],** "envs": ["SHELL=/bin/bash", "HOSTNAME=vegas.lanl.gov", "KRB5_PKCS11=PKCS11:/usr/lib64/pkcs11/libnetpkcs11.so", "MODULES_VERBOSITY=concise", "PWD=/home/elvis ", "KRB5CCNAME=FILE:/tmp/krb5cc_77777_TNRWne", "LOGNAME=elvis", "XDG_SESSION_TYPE=tty", "HOME=/home/elvis ", "LANG=en_US.UTF-8", "SSH_CONNECTION=777.777.777.777 55122 666.666.666.666 22", "XDG_SESSION_CLASS=user","TERM=xterm-256color"], "ts_utc": "2025-06-24_17:20:44" }

GitLab Project + CI



CI Lint, Test, build binary release, build rpms (ARM + AMD builds)

README.md

panhandle

Description

A project to provide user activity monitoring for HPC. This project takes a slightly different approach than the `crowdstrike` and `sentinel` products. The goal is to provide effective user activity monitoring with minimal impact on the host running this service. For development environment or source code questions, please reference the developer readme.

Tools

This project is written in Rust and uses the Aya library to reduce dependencies on LLVM and compilers and maximize the ability of this program to run on a variety of systems. The minimum kernel version supported is version 4.3.

Components

The RPMs in the build artifacts for RHEL8 & RHEL9 provide:

1. The panhandle binary `/usr/sbin/panhandle`
2. The default systemd service to run panhandle: `/usr/lib/systemd/system/panhandle.service`
3. The man page at `/usr/share/man/man1/panhandle.1`

Configuration & Implementation

1. Install the appropriate RPM for your RedHat version.
2. If log monitoring by a SEIM or Splunk server is desired, please add the logfile `/var/log/panhandle/panhandle.log` to your monitored rsyslog file inputs.
3. Enable and start the panhandle systemd service with: `systemctl enable --now panhandle` or your configuration manager of choice.

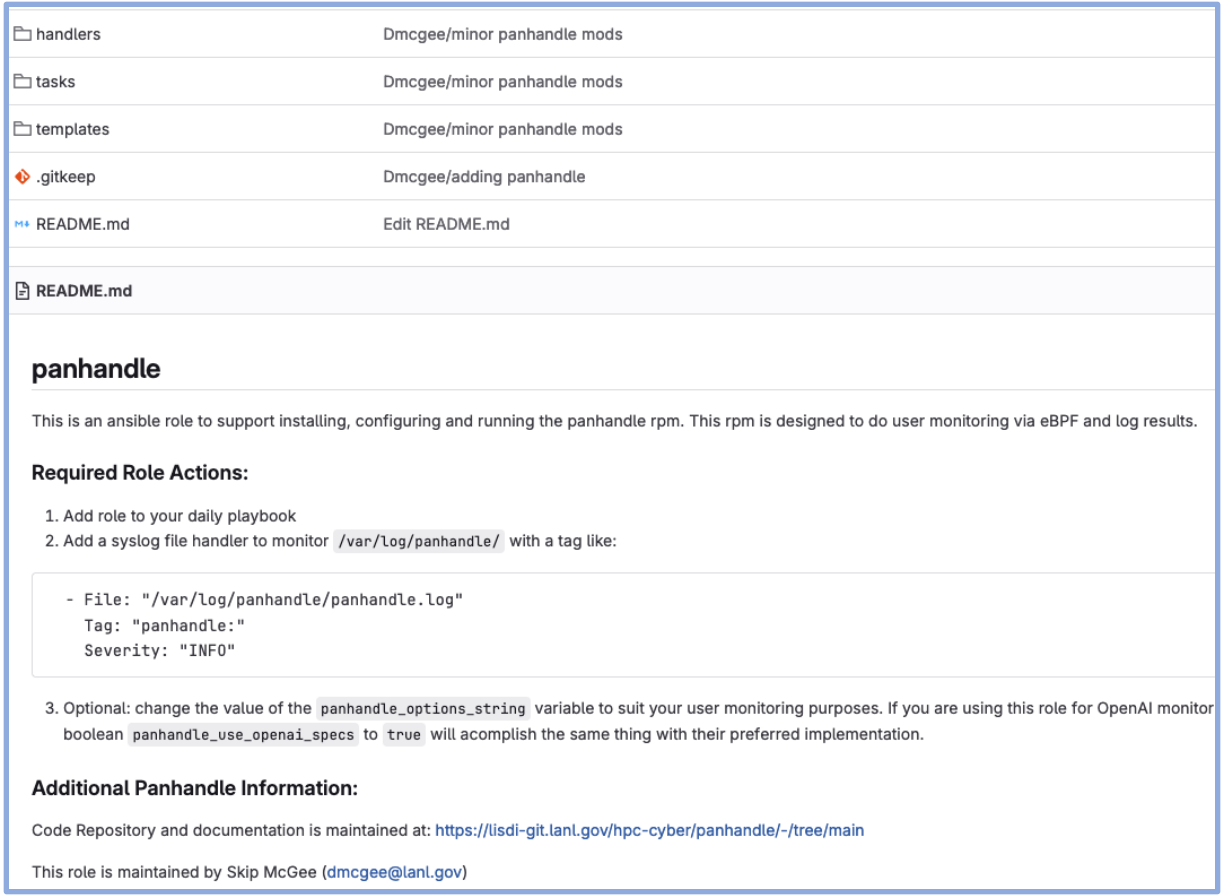
Additional Resources

1. an Ansible role for installing and managing `panhandle` options at scale.
2. a Splunk App for displaying and searching `panhandle` data.

GitLab project for collaboration

Ansible Role

- Install rpm
- Set up custom system configs
- Expose OpenAI defaults as a boolean
- Enable system service file customization
- Manage service
- Rsyslog file handler or http logger



handlers	Dmcgee/minor panhandle mods
tasks	Dmcgee/minor panhandle mods
templates	Dmcgee/minor panhandle mods
.gitkeep	Dmcgee/adding panhandle
README.md	Edit README.md

README.md

panhandle

This is an ansible role to support installing, configuring and running the panhandle rpm. This rpm is designed to do user monitoring via eBPF and log results.

Required Role Actions:

1. Add role to your daily playbook
2. Add a syslog file handler to monitor `/var/log/panhandle/` with a tag like:

```
- File: "/var/log/panhandle/panhandle.log"
  Tag: "panhandle:"
  Severity: "INFO"
```

3. Optional: change the value of the `panhandle_options_string` variable to suit your user monitoring purposes. If you are using this role for OpenAI monitor boolean `panhandle_use_openai_specs` to `true` will acomplish the same thing with their preferred implementation.

Additional Panhandle Information:

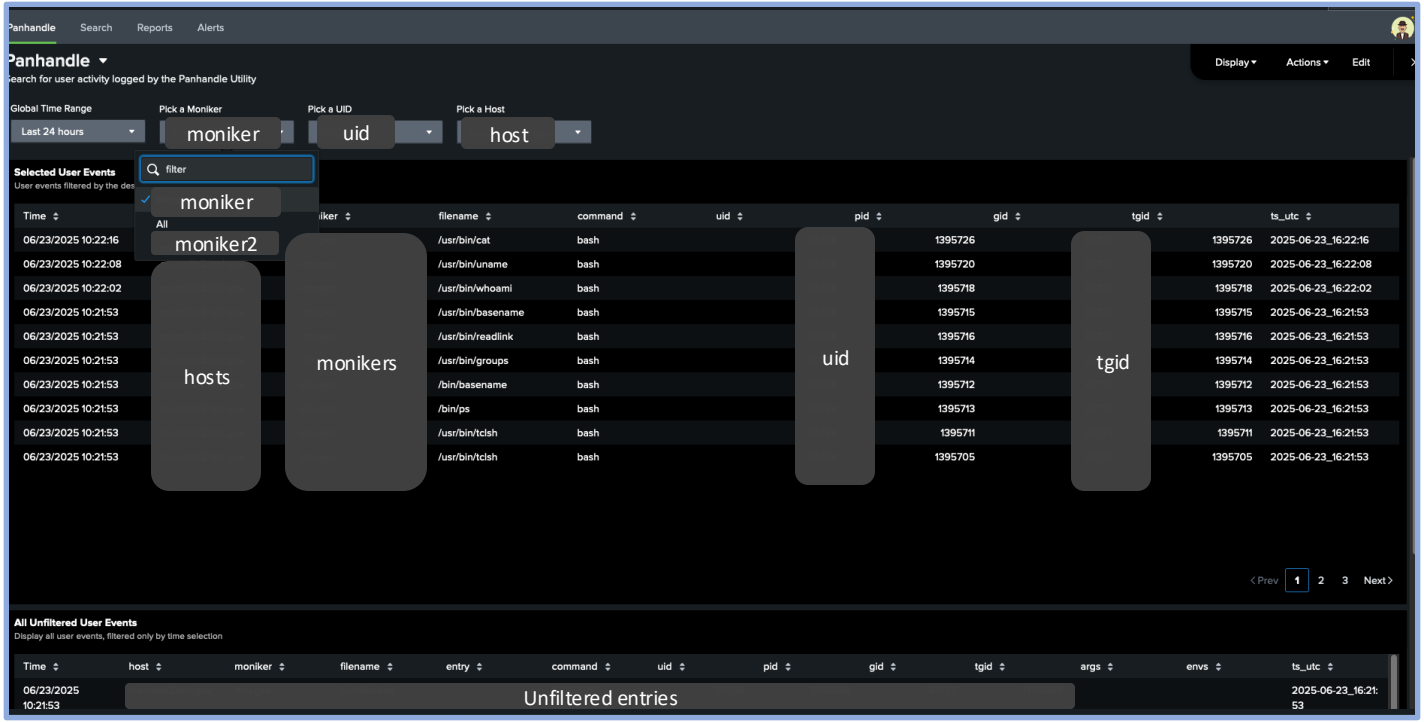
Code Repository and documentation is maintained at: https://lisdi-git.lanl.gov/hpc-cyber/panhandle/-/tree/main

This role is maintained by Skip McGee (dmcgee@lanl.gov)

Splunk Monitoring App

Filter by:

- Moniker
- UID
- Host
- Time
- Future:
  - Top/Tail
  - Additional analytics

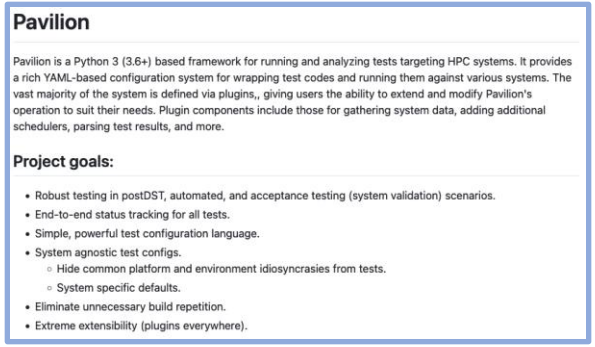


Panhandle Splunk App

Performance Testing

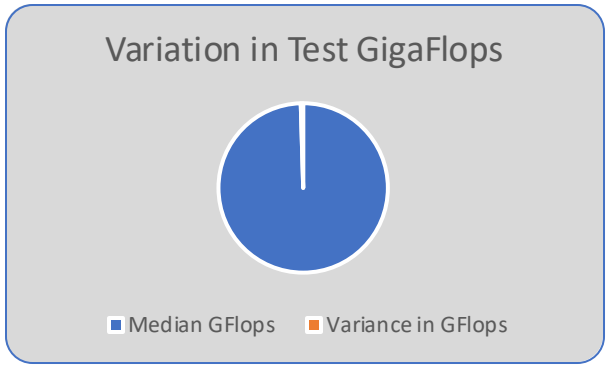
- Dynamic service: workload increases with user activity
- Razorback (Testbed for OpenAI, SLES)
- Selene (Testbed for OpenCHAMI, RHEL8)
- Test suite: Pavilion because that is what our PRE team uses (detail in next slide)
- Anticipated performance impacts:
  - ***gigaflops (computing speed impacted by background system processes)***
  - ***memory (memory will be used by the process)***
  - filesystem: write to a single log file when configured (negligible file size, <1MB when run for 7 days)
  - network: only uses network directly when the http option is selected to send the same data/size as the log file
  - other process impacts: rsyslogd file handler addition (trivial impact for a single file addition, this service is already used for monitoring other log files)

Performance Test Suite (Pavilion)



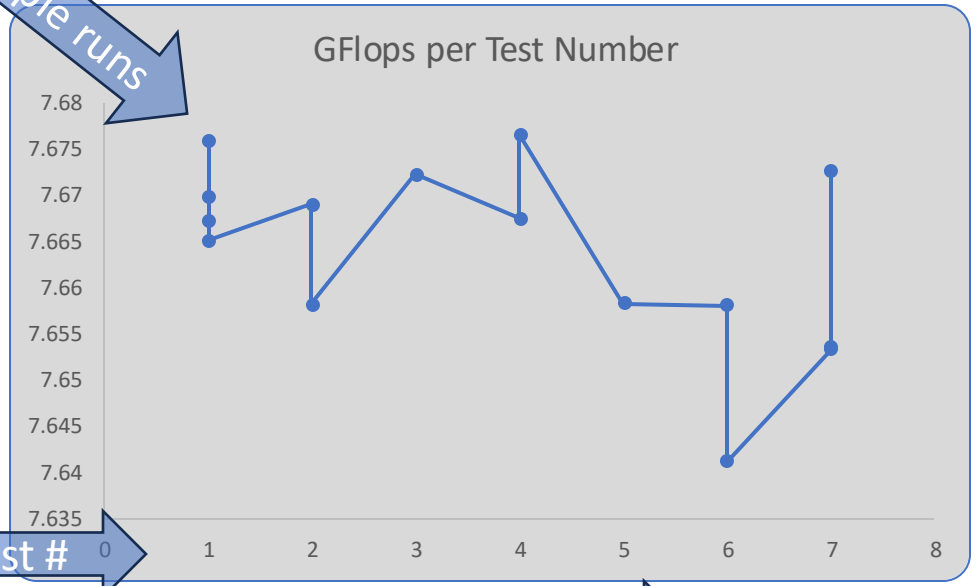
- What is Pavilion?
  - Used to configure test inputs and problem sizes
- The High Performance Conjugate Gradients (HPCG) Benchmark:
  - “uses a preconditioned conjugate gradient (PCG) algorithm to measure the performance of HPC platforms with respect to frequently observed, yet challenging, patterns of execution, memory access, and global communication.” https://icl.utk.edu/files/print/2021/hpcg-sc21.pdf
- How did we set it up?
  - Razorback: `pav run hpcg -c schedule.include_nodes=nid001030 -c schedule.slurm.sbatch_extra='--mem=20G --ntasks=12’`
  - Selene: `pav run hpcg -c schedule.slurm.sbatch_extra='--ntasks=2 --mem-per-gpu=40G --gpus-per-task=1' -c schedule.include_nodes=se003`

GigaFlops Performance Test Results (Razorback)



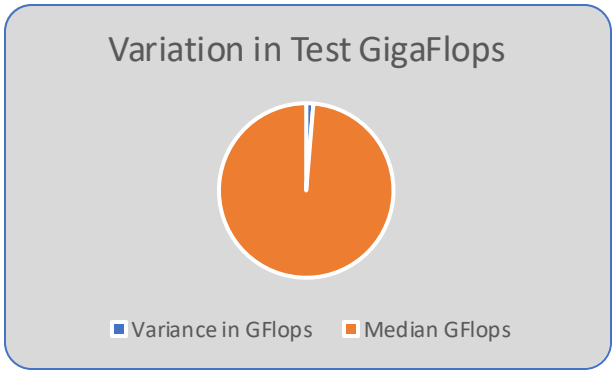
Max/Min Difference (GFlops):	0.03538
Variation from Median Performance (%)	0.00461443
Average Node Performance (GFlops)	7.6639
Median Node Performance (GFlops)	7.66726

multiple runs

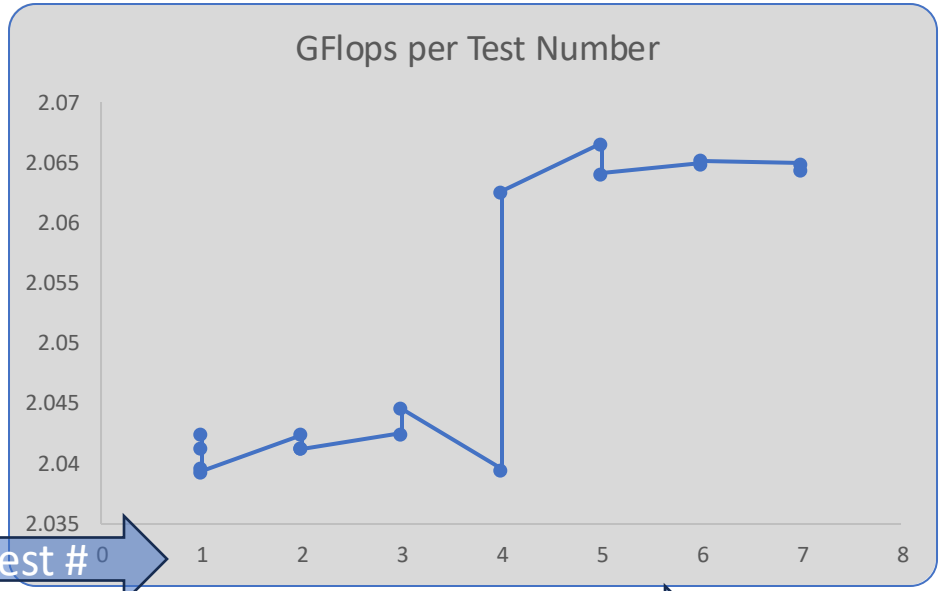


Test Number	Panhandle Status
1	*Control (Not Running)*
2	*OpenAI Options*
3	*Control (Not Running)*
4	*Gateway Options*
5	*Control (Not Running)*
6	*FMSH monitoring*
7	*Control (Not Running)*

GigaFlops Performance Test Results (Selene)



Max/Min Difference (GFlops):	0.0273
Variation from Median Performance (%)	0.01336597
Average Node Performance (GFlops)	2.05098235
Median Node Performance (GFlops)	2.0425



Test Number	Panhandle Status
1	*Control (Not Running)*
2	*OpenAI Options*
3	*Control (Not Running)*
4	*Gateway Options*
5	*Control (Not Running)*
6	*FMSH monitoring*
7	*Control (Not Running)*

Sequential

Memory Useage Results (RHEL8 Gateway Node)

- The running Panhandle service memory Resident Set Size:
  - 113.9 MB:

```
[root@turq-gate3 ~]# pmap -x 1715639 | grep total | cut -d ' ' -f 13
113912
```

- The running CrowdStrike service memory Resident Set Size :
  - 1.1697 GB

```
[root@turq-gate3 ~]# pmap -x 1763 | grep total | cut -d ' ' -f 12
1169700
```

- Top from running process:
  - Panhandle:

```
top - 09:14:19 up 7 days,  5:40, 18 users,  load average: 1.04, 1.01, 0.65
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.3 sy,  0.0 ni, 99.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  63731.1 total,  16076.5 free,   9035.4 used,  38619.2 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.  52330.7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
1715639 root      20   0 3378732 108528   9648 S   0.0   0.2   0:02.52 panhandle
```

  - CrowdStrike:

```
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  63731.1 total,  16065.3 free,   9046.9 used,  38618.8 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.  52319.4 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1763 root      20   0 1935996   1.1g 800516 S   0.0   1.8 175:33.28 falcon-sensor-b
```

Performance Assessment

- Both the best and worst runs were made while `panhandle` was running
- ***The impact of `panhandle` to user job compute performance is within the system noise***
- `panhandle` consumes more memory than any other system resource
- ***`panhandle` improvements should focus on reducing memory useage***
- The `panhandle` service used between 1/9-1/10 the memory of the running CrowdStrike service
- ***`panhandle` has significantly less system impact than running CrowdStrike***

Caveat: this is system integration testing so the service impact may vary depending on system configurations

Future

- Monitoring: Splunk application refinement
- Behavioral patterning
  - This can be as obvious as anything !=`ssh frontend` from a gateway
- Reconnaissance identification?
- Exploit attempt identifications? Perhaps based on Proof of Concept Exploits?
- Potential `panhandle` feature additions
  - Machine Learning model integration?
  - Response capability (initially to meet OpenAI requests)
  - What is the impact of additional features?

Review

- **Labor Costs:**
  - ✓ *10 hours research and design*
  - ✓ *200 hours programming*
  - ✓ *10 hours testing*
  - ✓ *5 hours integration*
  - ❖ *Total: 225 hours*
- **Hardware Costs:** *N/A*
- **Performance Impact:**
  - ✓ *No significant node performance impact*
- **Results:**
  - ✓ *Increased security monitoring*
  - ✓ *Established a platform for future features*

Questions