SELinux policy for Slurm

Gilles WIBER - CEA
Mathieu BLANC - CEA
M’hamed BOUAZIZ - Atos
Liana BOZGA - Atos

26-09-2017
SELinux policy for Slurm

- Cyber security in HPC
- SE Linux presentation
  - SE Linux basics and benefits
  - How it works
  - Challenges
- SE Linux Performance Results
- SE Linux for Slurm
  - Slurm architecture
  - Confined processes
  - Confined features
- Future work
Cyber Security in HPC
Securing an HPC?

- HPC have become increasingly desirable targets to attackers

- HPC protection includes:
  - Protecting the set of distributed resources (network access, compute nodes, storage...)
  - Ensure infrastructures, users, data, and jobs are running securely

- Standard security must be enhanced to address issues of HPC security
Encountered challenges

The issues related to HPC security are not exactly like general computer security

- Addressing and implementing traditional security solutions is the base for HPC (large and heterogeneous environment)
  - Maintaining and monitoring cluster security is a challenge due to large-scale skill requirements and production constraints
- Keeping performance (or very low impact) is mandatory
Mitigating HPC threats

- Identity and authorization management must be put in place / managed (Kerberos, LDAP, etc)
  - the solution must scale

- Confine and monitor network traffic
  - To maximize computing resources availability

- The HPC should be perceived as one system, not as a set of systems
  - Multi-level security must be put in place (in-deep security)
  - Component security in addition to global security
    => Securing HPC services using SELinux
SELinux Presentation
General introduction

- Security-Enhanced Linux released by the NSA
  - integrated into the Linux Security Modules (LSM) framework standard kernel
  - implements MAC (Mandatory Access Control) based security policies
  - provides service and user confinement

- Policy is the heart of SELinux
  - A set of rules determines security and access permissions for everything in the system
  - Defined by Types, Domains, Identities, Roles and Access with associated transitions
  - Expertise is required to write/adapt policies (SELinux, service behavior, system calls, etc)

- Using a policy is simple and doesn’t need an expertise in SELinux
Benefits of running SELinux

- Reduce vulnerability against privilege escalation attacks
- Can be used to enforce data confidentiality and integrity control
- Provide fine-grained access control

To reach that goal all processes and files are labeled with a specific type
Using SELinux in a HPC?

- Red Hat 7.x provides SELinux targeted policies for standard UNIX services

![Diagram of SELinux components]

- In a HPC context, resources are distributed
  - SELinux protects local resources for each node
  - A global policy will be loaded even if all services are not installed
3
SELinux Performance Results
## Suite tests description

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parallelism</th>
<th>IO</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Init</td>
<td>MPI</td>
<td>MPI</td>
<td>Aucun</td>
</tr>
<tr>
<td>MPI_STREAM</td>
<td>Memory</td>
<td>MPI/OpenMP</td>
<td>Aucun</td>
</tr>
<tr>
<td>IMB</td>
<td>MPI</td>
<td>MPI</td>
<td>Aucun</td>
</tr>
<tr>
<td>HPCC</td>
<td>MPI</td>
<td>MPI</td>
<td>Aucun</td>
</tr>
<tr>
<td>OpenMPBench</td>
<td>OpenMP</td>
<td>OpenMP</td>
<td>Aucun</td>
</tr>
<tr>
<td>SHOC</td>
<td>GPU</td>
<td>CUDA/OpenCL/MPI</td>
<td>Aucun</td>
</tr>
<tr>
<td>IOR</td>
<td>I/O bandwidth</td>
<td>MPI</td>
<td>POSIX/MPIIO/HDF5</td>
</tr>
<tr>
<td>b_eff_io</td>
<td>I/O bandwidth</td>
<td>MPI</td>
<td>MPHO</td>
</tr>
<tr>
<td>metarates</td>
<td>I/O bandwidth</td>
<td>MPI</td>
<td>POSIX</td>
</tr>
<tr>
<td>mdtest</td>
<td>I/O bandwidth</td>
<td>MPI</td>
<td>POSIX</td>
</tr>
<tr>
<td>AbinitCompilation</td>
<td>I/O bandwidth</td>
<td>Aucun</td>
<td>POSIX</td>
</tr>
<tr>
<td>SPECViewPerf</td>
<td>Visualization</td>
<td>Aucun</td>
<td>Aucun</td>
</tr>
<tr>
<td>Abinit</td>
<td>Compute code</td>
<td>MPI</td>
<td>POSIX/MPIIO</td>
</tr>
<tr>
<td>WRF</td>
<td>Compute code</td>
<td>MPI</td>
<td>NETCDF/POSIX</td>
</tr>
<tr>
<td>NAMD</td>
<td>Compute code</td>
<td>MPI</td>
<td>POSIX</td>
</tr>
</tbody>
</table>
Compute codes
Lustre IOR tests (write bandwidth)
mdtest: files creation
mdtest: files stat

![Graphs showing stats for mdtest_shm, mdtest_scratch, and mdtest_tmp](image)
HPCC: latency and bandwidth
Performance and results

▶ SELinux impact:

- No impact on pure compute code (even with GPU)
- Average 3% to 4% degradation on latency

- I/O:
  - No impact on I/O bandwidth
  - More than 100% degradation on metadata management

Notice:

Perhaps a good way to limit metadata access !!!
4 SELinux for Slurm
Security vision on Slurm

Management Nodes

- Slurmctld
  - Slurm user privileges
  - Slurm scripts
  - Sensitive data

- Slurmdbd
  - Slurm user privileges
  - Slurm scripts
  - Sensitive data

Login nodes

- Slurm cmd
  - User domain
  - Open to all users
  - Job submission

Compute nodes

- Slurmd
  - Root privileges
  - Root Epilog/Prolog scripts
  - User step jobs
  - Sensitive data
Securing Slurm Using SELinux requires:
- Confining Slurmd
- Confining Slurmctld
- Confining user commands
- Confining Slurmdbd
- Confining Slurm scripts

Confining services => control accesses to local resources such as:
- network ports, files, directories...

Writing the policy mustn’t affect the work of Slurm  ->  The policy must ensure that all features of Slurm are preserved
Confining Slurmd
A view on used resources

slurmd

Network
- slurmd.pid
- /etc/passwd
- /etc/resolv.conf
- Epilog/Prolog

Files
- slurmd.pid
- /etc/passwd
- /etc/resolv.conf

Directories
- port
- slurmctld
- slurmd
- ldap
- ephemeral
- dns

process
- /var/spool/slurmd/
- /proc/cpuinfo
- /var/log/slurm
- slurmstepd
### Slurmd Domain

#### Creating Slurmd Policy

<table>
<thead>
<tr>
<th>Linux</th>
<th>SELinux</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ports</strong></td>
<td></td>
</tr>
<tr>
<td>slurmd</td>
<td>slurmd_t</td>
</tr>
<tr>
<td><strong>Directories</strong></td>
<td></td>
</tr>
<tr>
<td>/var/spool/slurmd</td>
<td>slurmd_var_spool_t</td>
</tr>
<tr>
<td>/var/log/slurm</td>
<td>slurmd_log_t</td>
</tr>
<tr>
<td>slurmd.pid</td>
<td>slurmd_run_t</td>
</tr>
<tr>
<td><strong>Files</strong></td>
<td></td>
</tr>
<tr>
<td>slurmd.epilog</td>
<td>slurmd_epilog_t</td>
</tr>
<tr>
<td>/etc/passwd</td>
<td>auth_read_passwd(slurmd_t)</td>
</tr>
</tbody>
</table>

**Epilog/Prolog**
Transition from slurmd_t to slurmstepd_t

- slurmd_t: Parent process
  - fork(): system_u:system_r:slurmd_t
    - slurmd_t: child process
      - execve(): system_u:system_r:slurmstepd_t
        - slurmstepd_t: New program
          - domain_auto_trans(slurmd_t, slurmstepd_exec_t, slurmstepd_t)
Prolog/Epilog requires various privileges depending on job requirement
=> to ease implementation, transitions has been implemented (epilog_t)
=> “open” environment to execute specific actions outside Slurm policy
Slurm controller Domain
A view on used resources

- slurmd
- slurmd
- slurmctld
- ephemeral
- ldap
- directories
- files
- port
- /var/spool/slurm
- /var/log/slurm
- slurmd.pid
- /etc/passwd
# Slurm controller Domain

## Creating SELinux Policies

<table>
<thead>
<tr>
<th>Linux</th>
<th>SELinux</th>
</tr>
</thead>
<tbody>
<tr>
<td>slurmctld</td>
<td>slurmctld_t</td>
</tr>
<tr>
<td>slurmd</td>
<td>slurmctld_port_t</td>
</tr>
<tr>
<td>slurmdbd</td>
<td>slurmd_port_t</td>
</tr>
<tr>
<td>ephemeral</td>
<td>ephemeral_t</td>
</tr>
<tr>
<td>/var/spool/slurm</td>
<td>slurmctld_var_spool_t</td>
</tr>
<tr>
<td>/var/log/slurm</td>
<td>slurmdbd_port_t</td>
</tr>
<tr>
<td>/proc/self/fd/3</td>
<td>ephemeral_t</td>
</tr>
<tr>
<td>slurmd.pid</td>
<td>slurmd_log_t</td>
</tr>
<tr>
<td>/etc/passwd</td>
<td>slurmctld_run_t</td>
</tr>
<tr>
<td></td>
<td>auth_read_passwd(slurmctld_t)</td>
</tr>
</tbody>
</table>
Confining slurmdbd

- slurmdbd
- Network
  - port
  - slurmctld
  - slurmdbd
  - mysql
- Files
  - slurmdbd.pid
  - /var/log/slurm
- process
  - slurmdbd
Confining Slurm user commands

- We defined a policy to confine:
  - `srun`, `sinfo`, `sacct`, `sbatch`, `scancel`...

- Each command runs in a `slurm_t` domain

- Malicious users can’t use compiled commands
  - copied or hacked commandes without label

- It allows user cmd to access only authorized Slurm ports
Conclusion
To conclude...

- SELinux Slurm policy can be used to enforce security without additional complexity (pre-defined for Red Hat Linux)
  - Policy also supports some features and plugins such as: X11 spank plugin, interactive jobs, etc
  - Additional work has to be done to extend coverage

- SELinux security provides strong protection against Slurm processes threats (privilege escalation, etc) and also on data integrity (database, accounting, etc) without any additional impact on performance

- But keep in mind that:
  - It is not an all-in-one security solution (part of the a global security design)
  - Policy development and update requires tough expertise
Thanks

For more information please contact:
m-hamed.bouaziz@atos.net