Microsoft

Closing the script execution control gap

Linux Plumbers Conference

Mickaël Salaün

2024-09-19

Context

A secure Linux system on which we should know every executable code: **trusted code**

Trust requires **integrity**: measurement + code authentication (e.g., secure boot, IMA/EVM, IPE)

Once **attackers** get a foot on the system, we don’t want them to **execute their code**.

What is missing?

Linux already has several access control systems: DAC, mount points, SELinux, Landlock…

We can configure a set of **access** rights, including **execution**… except for scripts.

Main use case and **prerequisite**: systems with a well-configured and enforced access control, including **code integrity**.

Issue

./script.sh

vs.

sh script.sh

What is execution?

Subjective idea:

- Data interpreted by the CPU: code
- Data interpreted by executable code

binfmt_misc can make any “data” executable.

Goal

Protect the system from **untrusted** instructions that could do malicious things: explicitly do **syscalls**, modify **user's data**, leak data…

Do not rely on variants of script interpreters (hardcoded with or without restriction).

Properly handle stdin, command arguments, environment variables…

Legitimate calls

Same security restrictions with these commands:

- ./script.py
- python script.py
- python < script.py
- python -m script.py

Untrusted calls

Too difficult to reliably identify the origin of the script with these commands:

- xargs -a script.py -d '\r' -- python –c
- cat script.py | python
- python

Security policy

Different use cases:

- Developers or sysadmins may need to write and execute their own scripts
- System services may not be required to execute scripts

Script control could be only implemented by interpreters, but it does not make much sense without **consistent** system **policy**:

- **Define** access rights
- **Enforce** restrictions

Define access rights

Let user space check if the kernel’s policy would allow execution of a file.

A simple policy can be defined with mount point’s **noexec**

Check is done with a new **execveat**(2)’s flag: AT_CHECK

Enforce restrictions

Compatibility challenge: because user space is involved, we need a way to smoothly migrate (or not) to a more tighten access control.

New securebits for enlightened script interpreters:

- SECBIT_EXEC_RESTRICT_FILE
- SECBIT_EXEC_DENY_INTERACTIVE (REPL)

Extendable security policy

Mount points and process hierarchies might not be enough for more complex use cases. Leverage LSM security policies to get a more fine-grained control over restrictions: e.g.,

- Only for a set of users/services
- Always enforce for a set of script interpreters…

Consistent protection

The execution context (e.g., environment variables, command arguments) might be malicious, but not the **executable** files.

libc needs to properly check executable **libraries** e.g., because of LD_PRELOAD or LD_LIBRARY_PATH.

Potential drawbacks and limitations

- execveat(2) accepts both a file descriptor (good) or a path (may be bad)
- execveat(2) only handles regular files
- securebits were only used for root-related restrictions
- Mark all (script) libraries as executable
- Executable scripts need to safely deal with untrusted inputs (e.g., dangerous “eval” functions)

Previous proposals

1. open(2) + O_MAYEXEC, with dedicated sysctl. First implemented with Yama, then with a dedicated LSM, and finally without LSM.
2. faccessat2(2) + AT_INTERPRETED flag
3. New dedicated trusted_for(2)
4. access(2) + OK_EXECVE mode

Current approach: v19+

Two complementary kernel changes:

- execveat(2): check for executability of a file **according to the kernel** (not only file permission)
- securebits: configuration flags for **user space's interpreters** (e.g., containers, user sessions, system services)

User space changes:

- Scripts interpreters: Python, Perl, Bash…
- libc

Next steps

New patch series with:

- Simplified implementation
- A toy interpreter to showcase the required changes
- Extended tests

Enlighten script interpreters and libc.

References

- [RFC PATCH v19 0/5] Script execution control (was O_MAYEXEC)
- Restricting execution of scripts — the third approach [LWN.net]
- Initial execveat sample - PR #12 - zooba/spython
- Windows Defender Application Control - script enforcement
- CLIP OS's O_MAYEXEC patches