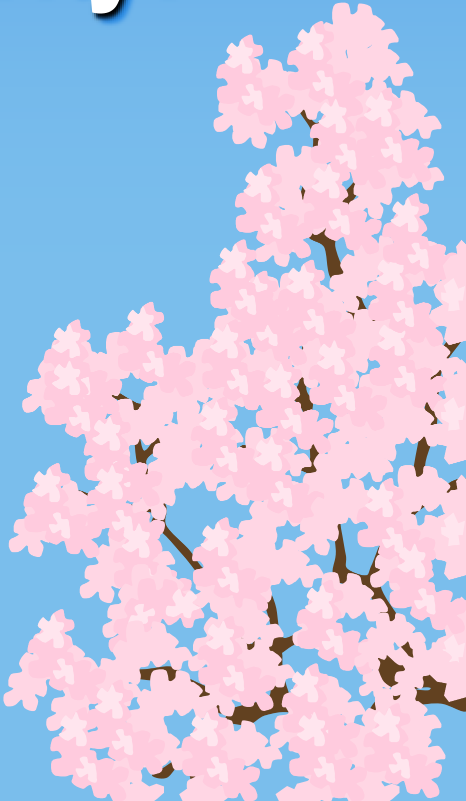
2008.07.10　LFJ Symposium BoF

The role of "pathname based access control" in security.



Tetsuo Handa

<penguin-kernel@I-love.SAKURA.ne.jp>

Two types of access control

- Label (i.e. attribute) based
  - SELinux
  - SMACK
- Pathname (i.e. name) based
  - AppArmor
  - TOMOYO Linux

Pathname based access control depends on the location of a file.

What happens if /etc/shadow is linked to /tmp/shadow ?

- Pathname based
  - An attacker can access password information via /tmp/shadow if the access control allows "ln /etc/shadow /tmp/shadow" and "cat /tmp/shadow".
  - So, we need to restrict pathname changes.

Pathname based access control depends on the location of a file.

What happens if /etc/shadow is linked to /tmp/shadow ?

- Label based
  - An attacker can't access password information via /tmp/shadow even if the access control allows "ln /etc/shadow /tmp/shadow" as long as the access control forbids "cat /etc/shadow" since /tmp/shadow preserves the same attribute with /etc/shadow .
  - So, we needn't to care about pathname changes.

Pathname based access control depends on the location of a file.

What happens if /etc/ is bind mounted to /tmp/ ?

- Pathname based
  - An attacker can access files under /etc/ via /tmp/ if the access control allows "mount --bind /etc/ /tmp/" and "cat /tmp/*".
  - So, we need to restrict namespace manipulation.

Pathname based access control depends on the location of a file.

What happens if /etc/ is bind mounted to /tmp/ ?

- Label based
  - An attacker can't access files under /etc/ via /tmp/ even if the access control allows "mount --bind /etc/ /tmp/" as long as the access control forbids to access files under /etc/ since /tmp/ preserves the same attribute with /etc/ .
  - So, we needn't to care about namespace manipulation.

That's right. But...

- The label based access control is indeed robust against change of pathnames and namespaces.
  - But that does not mean label based access control can allow changing pathnames and namespaces freely.
- It is not appropriate to say "We don't need to care about the location of a file if we use label based access control."
  - The location of a file has a meaning.

What happens if /etc/ is bind mounted to /tmp/ ?

- It is true that the content of /etc/shadow will not be read by "cat /tmp/shadow" if we use label based access control.
- But, since /etc/ and /tmp/ have the same attribute, writing to /tmp/ is denied if writing to /etc/ is not permitted.
  - Unwritable /tmp/ causes trouble with the applications. Can you tolerate it? (I can't.)

What happens if /etc/ is bind mounted to /tmp/ ?

- The matter is no longer "whether the content of /etc/shadow can be protected or not", but now "whether the system can work properly or not".
- To keep the system workable, you had better not to allow "mount --bind /etc/ /tmp/" from the beginning, even if you use label based access control.

What happens if /etc/shadow is linked to /etc/nologin ?

- It is true that the content of /etc/shadow will not be read by "cat /etc/nologin" if we use label based access control.
- But /etc/nologin has special meaning, it prevents unprivileged users from logging into the system. Can you tolerate it? (I can't.)

What happens if /etc/shadow is linked to /etc/nologin ?

- The matter is no longer "whether the content of /etc/shadow can be protected or not", but now "whether the specific pathname is allowed to be created or not".
- To keep the system usable, you had better not to allow "ln /etc/shadow /etc/nologin" from the beginning, even if you use label based access control.

What happens if /var/www/html/.htpasswd is renamed to /var/www/html/index.html ?

- We have to allow Apache to read both files.
  - Apache will send the content of index.html to clients.
  - Apache will not send the content of .htpasswd to clients.
  - Of course, we don't want Apache to leak password information, do we?

What happens if /var/www/html/.htpasswd is renamed to /var/www/html/index.html ?

- The matter is no longer "whether these files are accessible or not", but now "how these files are processed".
- To keep /var/www/html/.htpasswd secret, you had better not to allow "mv /var/www/html/.htpasswd /var/www/html/index.html" from the beginning, even if you use label based access control.

What happens if /usr/sbin/httpd and /usr/sbin/sshd are exchanged?

- Label based access control would block execution if /usr/sbin/httpd got the label 'sshd_exec_t' and /usr/sbin/sshd got the label 'httpd_exec_t'.
  - But, are you happy to have a server which doesn't provide services? (I'm not.)
- The matter is no longer "whether these programs can preserve appropriate attributes", but now "whether the system can continue providing services".

What happens if /usr/sbin/httpd and /usr/sbin/sshd are exchanged?

- To keep the system providing services, you had better not to allow "mv /usr/sbin/httpd /usr/sbin/httpd.tmp; mv /usr/sbin/sshd /usr/sbin/httpd; mv /usr/sbin/httpd.tmp /usr/sbin/sshd" from the beginning, even if you use label based access control.

More and more examples...

What happens if /bin/cat and /usr/bin/md5sum are exchanged?

- You don't care because both files have the label 'bin_t'?
- I do care because shell scripts will stop working properly.

More and more examples...

What happens if a symbolic link /bin/md5sum to /usr/bin/sha1sum is created?

- Applications want to execute md5sum, but they actually execute sha1sum if environment variable PATH is something like PATH=/bin:/usr/bin .

More and more examples...

What happens if /etc/shadow is renamed to /etc/my_shadow?

- Nobody will be able to login to the system.

System's availability depends on the location of a file.

- It is the file's *attribute* that decides "whether the file is readable and/or writable and/or executable or not", but it is the file's *name* that decides "how the file's content is processed" and "how the system behaves".
- Pathname is the basis of system's availability.

Need for protecting pathnames.

- To prevent the system from triggering contingency, it is quite natural thing to restrict changing pathnames.
- It is an indispensable prerequisite for the system to work properly that necessary files are in place with appropriate names.
  - Almost all files' pathnames needn't to be changed, and the range of pathname changes is not infinite.

Need for protecting pathnames.

- It is important to enforce the rule
  "Deny name changes by default.
  Allow name changes only by specific names."
  AS MUCH AS POSSIBLE.

Need for protecting pathnames.

The maximal granularity of restricting name changes.

- Label based
  - Per a directory (when each directory is assigned a different label).
- Pathname based
  - Per a filename (when wild card is not used).

Need for protecting pathnames.

- Label based access control can't restrict names within a directory.
  - It is impossible for label based access control to handle cases where the names have meaning.
- Pathname based access control can restrict names within a directory.
  - It is possible for pathname based access control to handle cases where the names have meaning.
  - This sometimes helps.

Current Problem.

- To make it possible to restrict pathname changes, I want to calculate the requested file's pathname from the LSM.
- Miklos has developed the patch to pass information needed for calculating the requested file's pathname from the LSM.
  - I want you to understand the meaning of the patch and send Acked-by: response.

Goal for the future.

- I agree that use of pathname based access control alone is not sufficient.
- I hope you understand that use of label based access control alone is not sufficient neither.
- Thus, I want the LSM coexist pathname based access control which is good at restricting names and label based access control which is good at restricting attributes.

2008.07.10 LFJ Symposium BoF

What does TOMOYO Linux provide?

Tetsuo Handa

<penguin-kernel@I-love.SAKURA.ne.jp>

Ability to minimize pathname changes.

...because unexpected pathnames leads to unexpected results.

- You can check old/new pathnames together for rename()/link().
- You can restrict namespace manipulation such as mount()/umount()/chroot()/pivot_root().

Ability to minimize accessible pathnames.

...because you want to allow programs to open only essential files.

- You can use realpath derived by traversing up to the process's namespace's root directory.

Ability to minimize program's invocation names.

...because multi-call binaries behave differently depending on argv[0].

- You can check the combination of realpath and argv[0].

Ability to validate parameters for program's execution.

...because argv[] and envp[] can lead to unexpected behavior.

- You can check argv[] and envp[] passed to execve().

Ability to insert a setup program before executing the requested program.

...because you want to manipulate parameters and environments.

- You can insert a program for validating/modifying argv[]/envp[] and setting up environments (e.g. private namespace), at the price of ability to return to the caller when the requested program could not be executed.

Ability to minimize reachable IP addresses and port numbers.

...because you want to use per-a-program iptables.

- You can check peer's IP address and port number of socket operation.

Ability to minimize allowed system calls.

...because it is impossible to predict what system calls a program will issue.

- You can control system calls which individual program can call.
- Though, current granularity is far from sufficient.

Ability to ...

- Oops, I have no more time...
- See online documentation for other abilities.

Summary

- TOMOYO Linux is a tool for reinforcing access control which is supposed to be performed by the userland process.
  - It performs access control from the perspective of subjects (i.e. processes) rather than the perspective of objects (i.e. files).
- Why not do it in the userland?
  - Access control performed in the userland is easily bypassed by errors and improper configurations (e.g. buffer overflow, statically linked applications, environment variables like LD_PRELOAD). To make access control inevitable, it is essential to do it in the kernel.

Summary

- Processes are born to achieve something, and they die after they achieved the purpose.
  - TOMOYO Linux tracks behavior of each process and restricts requests of each process in accordance with the purpose of each process.
  - It can permit necessity minimum requests in each context.

- TOMOYO Linux is a parameter checking tool like Web Application Firewall which is embedded into the kernel.

Conclusion

- Both SELinux and TOMOYO Linux perform policy based access restrictions.
- But, what TOMOYO Linux is doing is different from what SELinux is doing.
- I believe both restrictions are important.
- TOMOYO Linux is ready to coexist with SELinux, SMACK, AppArmor, LIDS etc.