# What's up in the land of Linux kernel security!

Vaishali Thakkar [@kernel_girl]

FrosCon 2017

# Hi! I’m Vaishali Thakkar.

# What this talk is about?

# History behind Linux as a OS

- Clone of a UNIX operating system [early 1990s]
- Core security model - Discretionary Access Control (DAC)

# UNIX DAC Security Model

- DAC = Restricting access to objects based on the identity of subjects and/or groups to which they belong
- UNIX DAC = Allows the owner of an object (such as a file) to set the security policy for that object
- Superuser— an entity which bypasses Unix DAC policy for the purpose of managing the system.

# Problems with UNIX DAC

- Originally aimed at protection [rather than security] in multiuser systems
- DAC does not protect against flawed or malicious code
- Superuser == compromise on user's security policy
- Cannot express modern security requirements as lots of rights accessible by default

# Problems with UNIX DAC

- Users can invoke system services by switching to root user (setuid)
- 9 bits model (rwx per owner, group and others)
- No protection against malicious code

# Problems with UNIX DAC

- Users can invoke system services by switching to root user (setuid)
- 9 bits model (rwx per owner, group and others)
- No protection against malicious code

# Extension of UNIX DAC Features

# POSIX ACCESS CONTROL LISTS

- Extension of abbreviated UNIX DAC ACLs, allows separate permissions for individual users and different groups

| **Entry Type** | **Text form** |
|---|---|
| owner | user : : rwx |
| Named user | user : name : rwx |
| Owning group | group : : rwx |
| Named group | group : name : rwx |
| Mask | mask :: rwx |
| Others | Other :: rwx |

# POSIX Capabilities

- Solution of a problem with superuser
- An application requiring some privilege do not get all privileges
- A process has three sets of bitmaps called the inheritable(I), permitted(P), and effective(E) capabilities.  Each capability is implemented as a bit in each of these bitmaps which is either set or unset.

# Namespaces

- Derived from the plan 9 operating system
- Partitioning resources as seen by the process
- Not a security feature but helps with implementing it

# Namespaces

- Have been used to help implement multi-level security, where files are labeled with security classifications, and potentially entirely hidden from users without an appropriate security clearance

- Not a security feature but helps with implementing it

# Cryptography API

- Used by kernel subsystems
- Provides support for a wide range of cryptographic algorithms and operating modes, including commonly deployed ciphers, hash functions, and limited support for asymmetric cryptography
- Key management subsystem for managing cryptographic keys within the kernel.

# Cryptography API

- Who uses cryptography API : IPSec code, Disk encryption schemes, Kernel module signature verification

- Support for hardware-based cryptographic features is growing too.

# Network Security

- Netfilter : An IP network layer framework which hooks packets which pass into, through and from the system.

- Iptables : A module which implements an IPv4 firewalling scheme, managed via the userland iptables tool.

- Ebtables : Provides filtering at the link layer, and is used to implement access control for Linux bridges

# Network Security

- Arptables: provides filtering of ARP packets

- IPSec : A network protocol suite which authenticates and encrypts the packets of data sent over a network

# Linux Security Modules [LSMs]

- Linux Security Modules (LSM) API implements hooks at all security-critical points within the kernel

- A user of the framework (LSM) can register with the API and receive callbacks from these hooks

- Was designed to provide the specific needs of everything needed to successfully implement MAC [Mandatory Access control]

# SELinux

- A LSM which provides a mechanism for supporting access control policies

- In SELinux, all objects on the system, are assigned security labels.  All security-relevant interactions between entities on the system are hooked by LSM and passed to the SELinux module, which consults its security policy to determine whether the operation should continue.

- Security policy is loaded from userland, and can be modified

# Smack

- A LSM which was designed to provide a simple form of MAC security, in response to the relative complexity of SELinux

- Works best with file-systems which supports extended attributes

- It’s a part of the Tizen security architecture

# AppArmor

- Fundamentally different MAC scheme to SELinux and Smack, no direct labeling and security policy is applied to pathnames
- Allows the system administrator to restrict programs' capabilities with per-program profiles
- Also features a learning mode, where the security behavior of the application is observed and converted automatically into a security profile

# TOMOYO

- Another MAC scheme, which implements path based security
- Utilizes the learning mode similar to AppArmor where behavior of the system is observed to enhance the security policy
- It records the trees of process invocation, described as domains

# YAMA

- Collection of DAC security enhancements from projects like Grsecurity

- Enhanced restrictions on ptrace are implemented in YAMA

# LoadPin

- Fairly new LSM, which ensures that all kernel loaded files are loaded from trusted device [dm-verity or CDROM]

- Allows systems that have a verified and/or unchangeable filesystem to enforce module and firmware loading restrictions without needing to sign the files individually.

# Audit Subsystem

- Was first designed to meet government certification requirements, now used by LSMs and other security components

- Helps to track security relevant information

# Seccomp

- A mechanism which restricts access to system calls by processes

- Reduce the attack surface of the kernel by preventing applications from entering system calls they don't need

- The original seccomp code, also known as “mode 1”, provided access to only four system calls: read, write, exit, and sigreturn

# Seccomp - bpf

- An  arbitrary specification of which system calls are permitted for a process, and integration with audit logging

- Was developed for use as part of the Google Chrome OS.

# Integrity management subsystem

- Used to maintain the integrity of files on the system

- Integrity Measurement Architecture component performs runtime integrity measurements of files using cryptographic hashes, comparing them with a list of valid hashes

- Dm-verity module: Device mapper target which manages file integrity at block level

## Is this level of security sufficient?

# What are the possible solutions?

# Bug Fixing Using Tools

- Wide scope of research projects and static/dynamic analysis tools
- Useful only if it is used regularly to detect the security issues
- Automatic testing helps, but have limitations

# Widely used tools in Linux kernel

- **Sparse**:
    - Written by Linus Torvalds, provides a set of annotations designed to convey semantic information about types
    - Warns about unsupported operations or type mismatches with restricted integer types
    - Warns about any non-static variable or function definition that has no previous declaration.

# Widely used tools in Linux kernel

- **Smatch:**
  - Written by Dan Carpenter, more than 3000 bug fixes so far
  - Warns about issues like null pointer dereference, error pointer dereference, uninitialized data, information leak, some cases of use after free, double free, unnecessary/missing null check etc
- **Coccinelle:**
  - Pattern matching and transformation tool, developed by Julia Lawall, more than 4000 bug fixes so far
  - Handles few security issues like null pointer dereference, use of sleeping functions under locks, use after free, few locking related bugs etc

# Widely used tools in Linux kernel

- GCC and GCC plugins:
  - New GCC versions [6 and 7] has added many new warning options, though sometimes they are added on case basis and might not be able to handle all kind of cases for a particular bug classes
  - GCC plugins helps with handling specific kind of bug classes at compiler level, without adding code in compiler itself
  - As of now, there are 5 gcc plugins added in the Linux kernel
  - Plugins like randomizing structures layout at compile time or detecting any structures that contain __user attributes and makes sure it is being fulling initialized
  - helps with making the attack surface harder

# Widely used tools in Linux kernel

- **Fuzzers:**
  - Trinity: Developed by dave Jones, helps with OOPS, locking related bugs and memory leaks etc
  - Syzkaller: Developed by Dmitry Vyukov and a team [Google], helps with {resource, memory, information} leaks, deadlocks etc
  - Some others: AFL [American Fuzzy Loop], Address sanitizer, Thread sanitizer etc

# Is Fixing bugs sufficient?

# Kernel Self Protection [KSPP]

- Idea : Bugs have longer lifetime, kernel should be able to protect itself
- Kill classes of bugs instead of individual bug
- Current focus on upstreaming grsecurity/PAX features
- More information on Kees cook's blog: https://kernsec.org/wiki/index.php/Feature_List

# Conclusion

- We have come far from UNIX security but there is always a scope of more research and improvement at the kernel level.

- With the advancement of technology and wide variety of requirements, security is no longer a buzzword.

# Resources

- LWN: http://lwn.net/Security
- Kernel Self Protection project: https://kernsec.org/wiki/index.php/Kernel_Self_Protection_Project
- Bug finding/fixing tools: http://events.linuxfoundation.org/sites/events/files/slides/Using%20static%20checking%20to%20find%20security%20vulnerabilities%20in%20the%20Linux%20Kernel.pdf
- LSM mailing list and kernel-hardening mailing list
- Kernel security summit [2016]: https://www.youtube.com/playlist?list=PLbzoR-pLrL6pq6qCHZUuhbXsTsyz1N1c0