

# Linux From Scratch in Rust

**Mingshen Sun**

**2019.04.20**

**RustCon Asia, Beijing**

# Intro

- **Linux From Scratch (LFS)** is a project that provides you with step-by-step instructions for building you own custom Linux system, entirely from source code.
- *Why/Can/How we do LFS entirely in Rust?*

# whoami

- Senior Security Research in **Baidu X-Lab**, Baidu USA
- System security, mobile security, IoT security
- MesaLock Linux, MesaPy, Rust OP-TEE TrustZone SDK, TaintART, Pass for iOS, etc.
- `mssun @ GitHub` | <https://mssun.me>

# Why LFS in Rust?

- **Cause we are in a Rust conference! Yes, it's fun!**

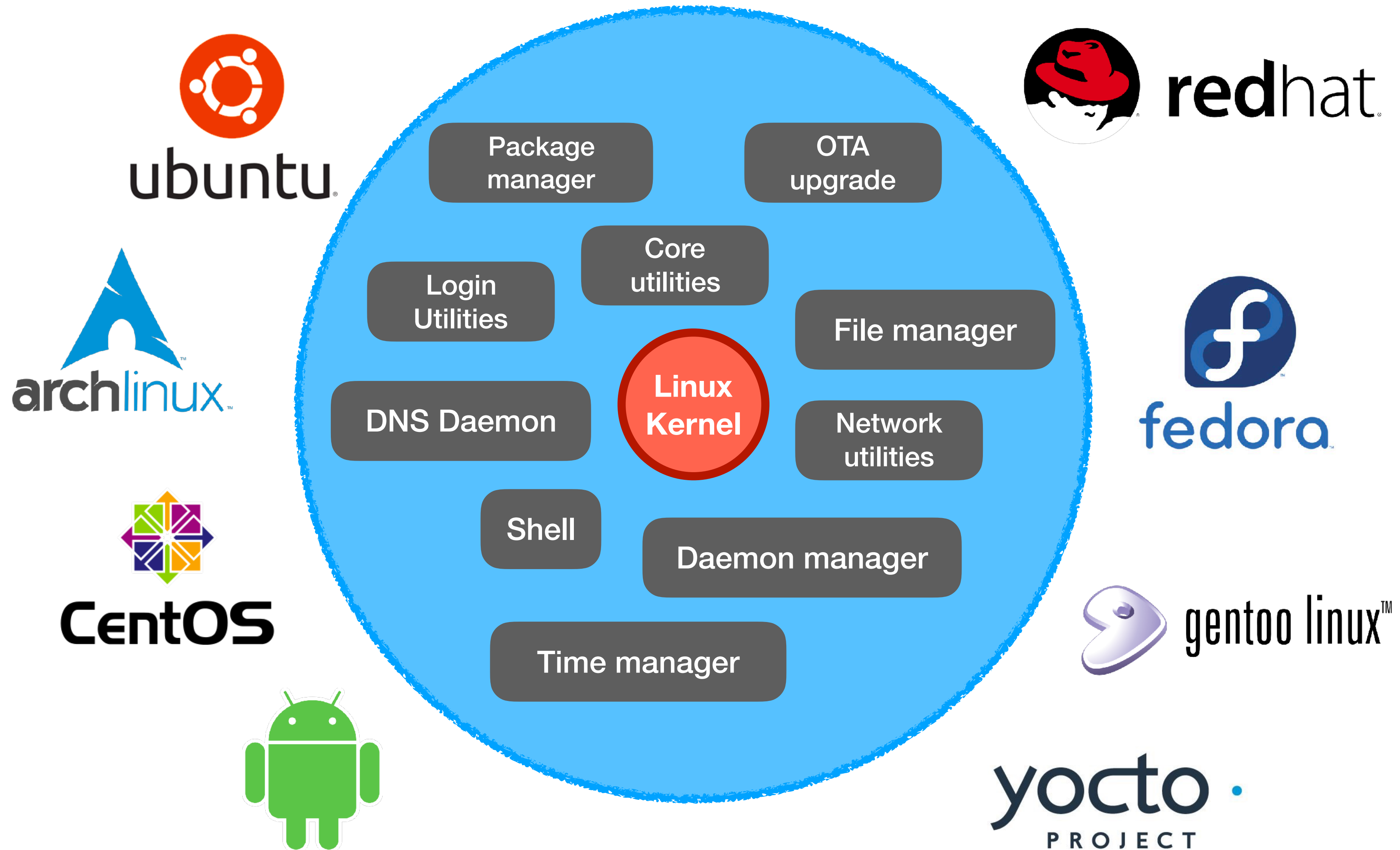
# Why LFS in Rust?

- Cause we are in a Rust conference! Yes, it's fun!
  - **Memory-safety in userspace**
- CVE-2017-13089 **wget**: Stack-based buffer overflow in HTTP protocol handling
  - A **stack-based buffer overflow** when processing chunked, encoded HTTP responses was found in wget. By tricking an unsuspecting user into connecting to a malicious HTTP server, an attacker could exploit this flaw to potentially **execute arbitrary code**.
  - [https://bugzilla.redhat.com/show\\_bug.cgi?id=1505444](https://bugzilla.redhat.com/show_bug.cgi?id=1505444)
  - Proof-of-concept: <https://github.com/r1b/CVE-2017-13089>

# Why LFS in Rust?

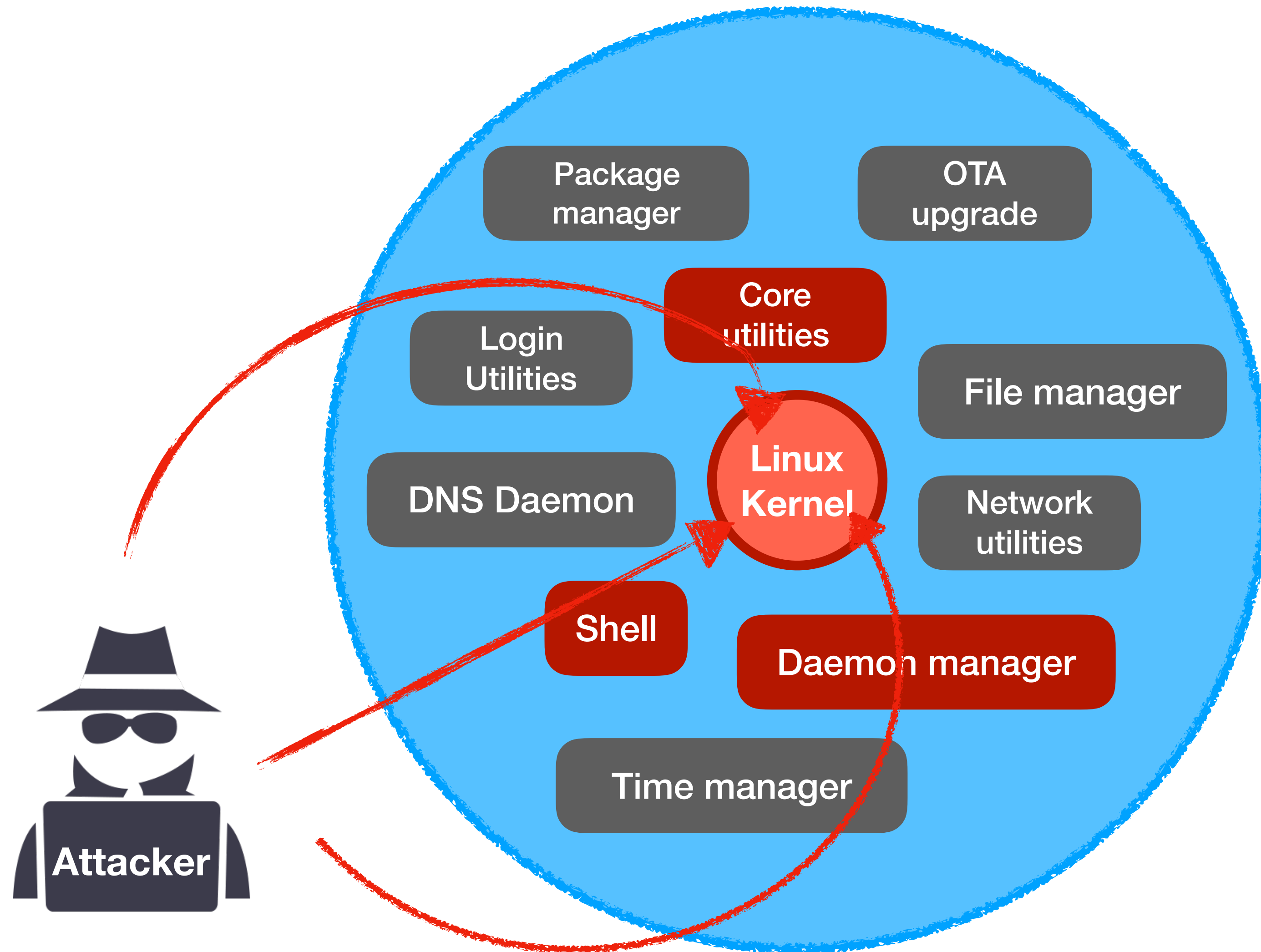
- **Cause we are in a Rust conference!**
- Memory-safety in userspace
- **Capabilities and "pitfall" of system programming in Rust**

# Linux Userspace





# Security of Linux Userspace





# Linux Distros

- A Linux distribution (often abbreviated as distro) is an operating system made from a **software collection**, which is based upon the **Linux kernel** and, often, a **package management system**.

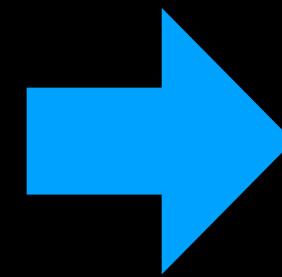
- **Server:** CentOS, Fedora, RedHat, Debian
- **Desktop:** Ubuntu
- **Mobile:** Android
- **Embedded:** OpenWRT, Yocto
- **Enthusiast:** Arch Linux, Gentoo
- **Misc:** ChromeOS, Alpine Linux

## Security and Safety?

- Gentoo Hardened: enables several risk-mitigating options in the toolchain, supports PaX, grSecurity, SELinux, TPE and more.
- Kernel hardening patches
- Safety? No.
- User space? GNU.

# Basic Components of LFS in Rust

- bootloader
- Linux kernel
- init
- getty
- login
- iproute2
- coreutils



- syslinux
- Linux 4.9.58
- minit
- mgetty
- mlogin
- giproute2
- utils-coreutils

MesaLock  
Linux

MesaBox

# MesaBox

- MesaBox is a collection of core system utilities written in Rust for Unix-like systems.
- Like the well-known BusyBox and Toybox sets of utilities popular on embedded devices, MesaBox seeks to provide a fully functioning command-line environment.

# Status

Utility	Type	Status
arch	GNU	Complete
base32	GNU	Complete
base64	GNU	Complete
yes	GNU	Complete
getty	Login	Simple Version
tar	LSB	Beginning Stages
ping	Networking	Simple Version
cat	POSIX/GNU	Complete
chmod	POSIX/GNU	Mostly Complete (missing --reference)
head	POSIX/GNU	Complete
echo	POSIX	Complete
init	POSIX	Simple Version
sh	POSIX	Significant Progress
sleep	POSIX	Complete

# Add New Tools?

Name and description

```
const NAME: &str = "dummy";
pub const DESCRIPTION: &str = "A dummy utility to demonstrate the framework";

type DummyResult<T> = ::std::result::Result<T, DummyError>;

#[derive(Fail, Debug)]
enum DummyError {
    #[fail(display = "oh no, something wrong")]
    SomethingWrong
}
```

Use failure for error handling

# Add New Tools?

clap for command line  
argument parsing

```
fn create_app() -> App<'static, 'static> {
    util_app!(NAME)
        .arg(Arg::with_name("verbose")
            .short("v")
            .long("verbose")
            .help("Say hello in verbose mode"))
}

pub fn execute<S, T>(setup: &mut S, args: T) -> Result<()>
where
    S: UtilSetup,
    T: ArgsIter,
{
    let app = create_app();
    let matches = app.get_matches_from_safe(args)?;
    let options = DummyOptions::from_matches(&matches);
    let output = setup.output();
    let mut output = output.lock()?;
    let mut dummyer = Dummyer::new(output);
    dummyer.dummy(&options)?;
    Ok(())
}
```

start the tool with input  
arguments

# One more interesting feature in MesaBox

- It can be used as a library.

```
9  fn main() {
10      let listener = TcpListener::bind("127.0.0.1:9876").unwrap();
11
12      for stream in listener.incoming() {
13          let mut stream = stream.unwrap();
14
15          let mut stdout = stream.try_clone().unwrap();
16          let mut stderr = stream.try_clone().unwrap();
17
18          let res = {
19              let mut setup =
20                  UtilData::new(&mut stream, &mut stdout, &mut stderr, iter::empty(), None);
21
22              mesabox::execute(&mut setup, &mut ["head", "-n", "4"].into_iter())
23          };
24          if let Err(f) = res {
25              let _ = writeln!(stderr, "{}", f);
26          }
27      }
28  }
```

Use "head" to handle TcpStream.



# Packages of MesaLock Linux

- `brothli`: compression tool written in Rust
- `busybox`: busybox tool set for testing only
- `exa`: replacement for `ls` written in Rust
- `fd-find`: simple, fast and user-friendly alternative to `find`
- `filesystem`: base filesystem layout
- `gcc-libs`: GCC library, only `libgcc_s.so` is used
- `giproute2`: ip tool written in Go
- `glibc`: glibc library
- `init`: init script
- `ion-shell`: shell written in Rust
- `linux`: Linux kernel

# Packages of MesaLock Linux

- `mesalock-demo`: some demo projects
- `mgetty`: getty written in Rust
- `micro`: modern and intuitive terminal-based text editor in written Go
- `minit`: init written in Rust
- `ripgrep`: ripgrep combines the usability of The Silver Searcher with the raw speed of grep, written in Rust
- `syslinux`: bootloader
- `tokei`: count your code, quickly, in Rust
- `tzdata`: timezone data
- `utils-coreutils`: cross-platform Rust rewrite of the GNU coreutils
- `utils-findutils`: rust implementation of findutils
- `xi-core`: a modern editor with a backend written in Rust
- `xi-tui`: a tui frontend for Xi

# Add New Packages?

```
1 package:
2   name: ripgrep
3   version: 0.8.0
4   description: ripgrep combines the usability of The Silver Searcher with the raw speed of grep
5   license: [MIT, Unlicense]
6   url: https://github.com/BurntSushi/ripgrep
7   skip_check: true
8
9   source:
10    - git+https://github.com/BurntSushi/$name.git
11
12   prepare:
13    - cd "$name".git && git checkout -B 0.8.0
14
15   build:
16    - cd "$name".git && cargo build --release
17    - cd "$name".git && cargo test
18
19   install:
20    - cd "$name".git && install -D -m744 target/release/rg -t "$pkgdir"/bin/
```

- build.yml
- name, version, description, license, url, skip\_check
- source, prepare, build, install
- mkpkg will automatically build and package tools

# Rust in System Programming

- The `nix` library is very useful.
  - `unistd.h`, `mount.h`, `fcntl.h`, `stdlib.h`
  - handle PTY, network interface, users/groups, `ioctl`, `mount`, `kmod`
- Signal in Rust is unsafe.
  - `SIGUSR1`, `SIGUSR2`, `SIGTERM`, `SIGQUIT`, `SIGINT`, `SIGHUP`, `SIGTSTP`, `SIGSTOP`...

# Rust in System Programming

- Rust standard library only provides APIs with high-level abstraction.
  - `std::net` v.s. `socket2/net2` v.s. `libpnet` v.s. `libc`
- You have to use `libc` in the end.

## Structs

<code>AddrParseError</code>	An error which can be returned when parsing an IP address or a socket address.
<code>Incoming</code>	An iterator that infinitely <code>accept</code> s connections on a <code>TcpListener</code> .
<code>Ipv4Addr</code>	An IPv4 address.
<code>Ipv6Addr</code>	An IPv6 address.
<code>SocketAddrV4</code>	An IPv4 socket address.
<code>SocketAddrV6</code>	An IPv6 socket address.
<code>TcpListener</code>	A TCP socket server, listening for connections.
<code>TcpStream</code>	A TCP stream between a local and a remote socket.
<code>UdpSocket</code>	A UDP socket.

## Structs

<code>Domain</code>	Specification of the communication domain for a socket.
<code>Protocol</code>	Protocol specification used for creating sockets via <code>Socket</code> .
<code>SockAddr</code>	The address of a socket.
<code>Socket</code>	Newtype, owned, wrapper around a system socket.
<code>Type</code>	Specification of communication semantics on a socket.

`bind`  
`listen`  
`accept`  
`local_addr`  
`peer_addr`  
`try_clone`  
`take_error`  
`set_nonblocking`  
`shutdown`  
`recv`  
`peek`  
`recv_from`  
`peek_from`  
`send`  
`send_to`  
`ttl`  
`set_ttl`  
`unicast_hops_v6`  
`set_unicast_hops_v6`  
`only_v6`  
`set_only_v6`  
`read_timeout`  
`set_read_timeout`  
`write_timeout`  
`set_write_timeout`  
`nodelay`  
`set_nodelay`  
`broadcast`  
`set_broadcast`  
`multicast_loop_v4`  
`set_multicast_loop_v4`  
`multicast_ttl_v4`

# Rust in System Programming

- Handling string for CLI in Rust is very difficult.
  - String, &str
  - CString, &CStr
  - OsString, &OsStr
- Low-level system operation in Rust is very difficult.
  - E.g., netlink: used to transfer information between the kernel and user-space processes.

# Rust in System Programming

- Testing and code coverage in Rust are non-trivial tasks.
  - Integration test framework for CLI: `assert_cmd`, `assert_fs`
  - **tarpaulin**: a code coverage reporting tool for the Cargo build system
  - **gcov**: a source code coverage analysis and statement-by-statement profiling tool

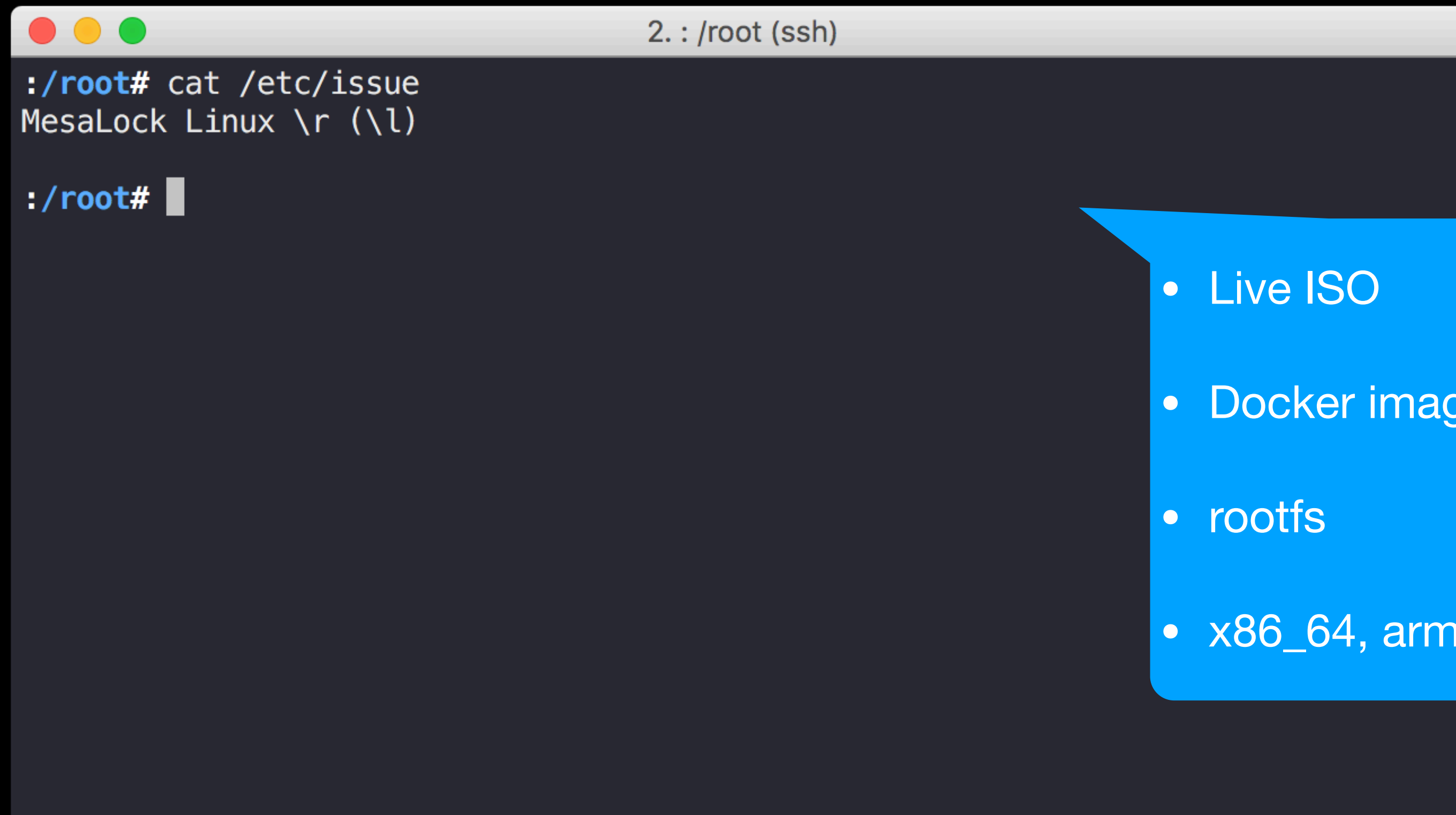
ptrace-based code  
coverage profiling tool

llvm-gcov-based  
code coverage profiling  
tool



# Quick Start

```
$ docker run -it mesalocklinux/mesalock-linux
```



A terminal window titled "2. : /root (ssh)" with standard macOS window controls (red, yellow, green buttons). The terminal shows the command `cat /etc/issue` being executed, resulting in the output `MesaLock Linux \r (\l)`. The prompt `:/root#` is visible at the bottom of the terminal.

```
2. : /root (ssh)
:/root# cat /etc/issue
MesaLock Linux \r (\l)
:/root#
```

- Live ISO
- Docker image
- rootfs
- x86\_64, arm in the near future

# Contributing

- <https://github.com/mesalock-linux/mesalock-distro>
- You can get involved in various forms:
  - Try to **use** MesaLock Linux, report issue, enhancement suggestions, etc
  - **Contribute**: optimize development process, improve documents, closing issues, etc
  - **Contribute to core packages**: improving minit, mgetty, giproute2, etc
  - **Writing applications** using memory safe programming languages like Rust/Go, and joining the packages
  - **Auditing source code** of the projects and related packages
- You are welcome to send **pull requests** and report **issues** on GitHub.



# Thank you!

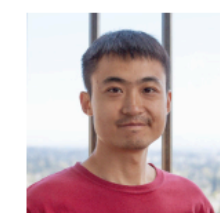
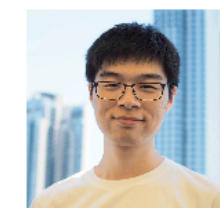
## Day 4: April 23

09:00 12:00 **Build a Secure and Trusted Framework in Rust**

This talk intends to be a long talk (~4 hrs) introducing the Rust TEE ecosystem.

**Rust SGX SDK** has become the most popular SGX development environment. Previous talks at [RustFest](#), [QConSF](#) and [QConBJ](#) are all brief talks within 30 minutes and limited Q&As. This time I'd like to present its details and current ecosystems. In addition, we'll include the Rust-trustzone part.

The talk would guide the audience to play Rust on two TEEs: SGX and Trustzone. In the beginning, we'll talk about the trusted computing theory and hardware-assisted trust execution engines. Then we assist the audience with hands-on experiments on Rust+SGX and Rust+Trustzone platforms. At last, we'll discuss about the internals and ecosystems.



**Yu Ding &  
Mingshen Sun**