Overview of Fuchsia, a new operating system

Mickaël Salaün

French National Cybersecurity Agency (ANSSI)

September 23, 2019

Introduction

Started in 2015 and open-sourced in 2016, Fuchsia is mainly developed by Google, with some private parts: roadmap and issue tracker (Atlassian).

What is Fuchsia?

- OS targeting end-user devices (e.g. smartphone, laptop, IoT, extended reality devices?)
- main goals: security, reliability and modularity
- future-proof: designed to be updatable for a long time

Introduction

Started in 2015 and open-sourced in 2016, Fuchsia is mainly developed by Google, with some private parts: roadmap and issue tracker (Atlassian).

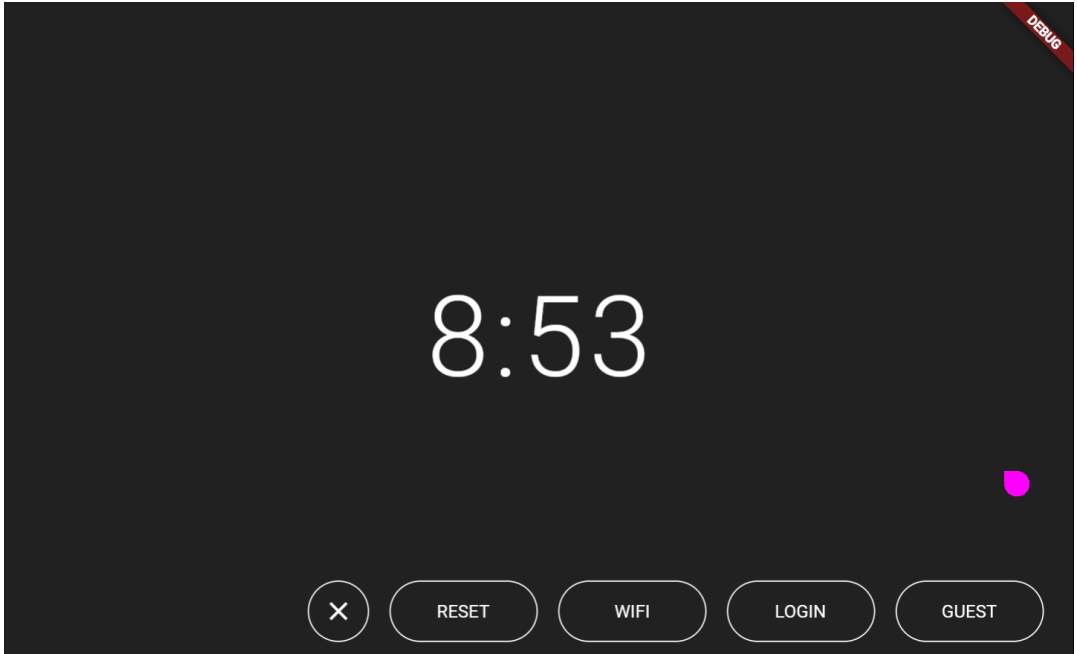
What is Fuchsia?

- OS targeting end-user devices (e.g. smartphone, laptop, IoT, extended reality devices?)
- main goals: security, reliability and modularity
- future-proof: designed to be updatable for a long time

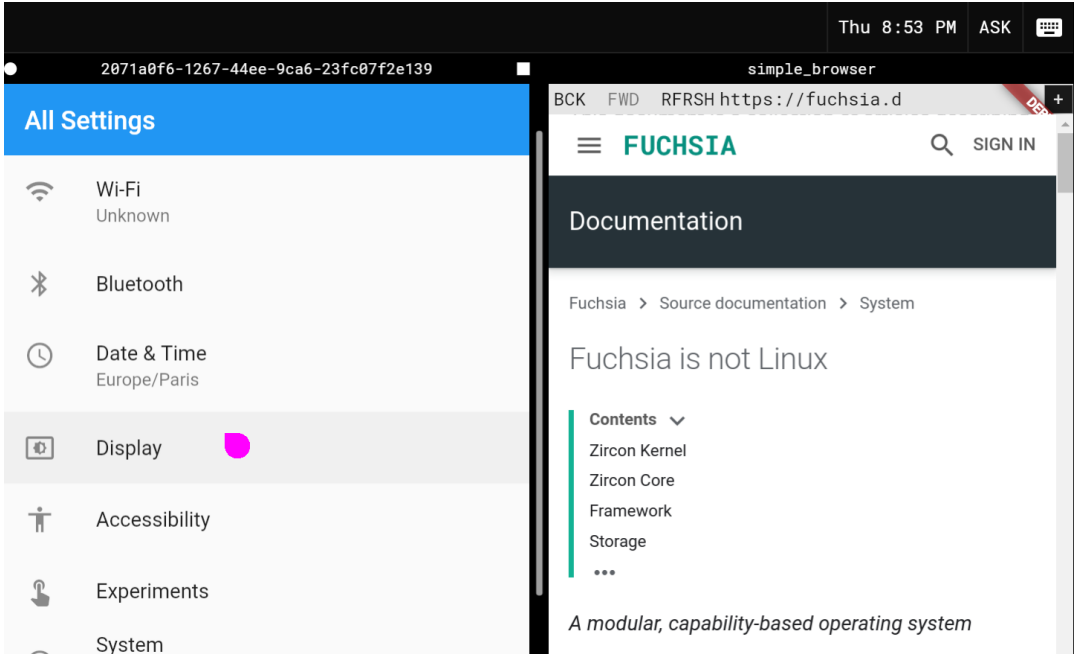
Warning

- Fuchsia is a moving target right now, this talk might be partially outdated soon.
- A lot of new things are developed for this OS, but I don't have time to cover all aspecs of them in this talk.

Screenshot of Fuchsia: greeter



Screenshot of Fuchsia: apps



Major properties

Open and supported

- mainly open-source
- supported by a big company, already developing Android and Chrome OS

Major properties

Open and supported

- mainly open-source
- supported by a big company, already developing Android and Chrome OS

Support big existing ecosystems (WIP)

- Flutter apps, Android apps, Chromecast apps
- Linux VM (file transferts, Wayland bridge)

Major properties

Open and supported

- mainly open-source
- supported by a big company, already developing Android and Chrome OS

Support big existing ecosystems (WIP)

- Flutter apps, Android apps, Chromecast apps
- Linux VM (file transferts, Wayland bridge)

Connected

- composable apps and task-centric
- distributed data storage: Ledger
- bridges with other systems (overnet/gRPC): iOS and Android

Implementation

IPC

Capability

- reference (handle) to a kernel object (e.g. memory, interrupt, process)
- associated with a set of access rights
- unforgeable
- communicable (e.g. through channels)

IPC

Capability

- reference (handle) to a kernel object (e.g. memory, interrupt, process)
- associated with a set of access rights
- unforgeable
- communicable (e.g. through channels)

FIDL

- static definition of protocols (inspired by Chromium's Mojo)
- enables to transfert (typed) data and handles (including other protocols)
- generates serialization and deserialization libraries

$\Rightarrow$ agnostic to the underlying languages

$\Rightarrow$ consistent and unique entry point to audit and test services

$\Rightarrow$ defines semantics (e.g. user-defined capability, revocation, state machine)

Zircon

Microkernel

- well suited for security: small TCB (running in ring 0)
- origin: Little Kernel (32-bits, no syscall, no MMU...)
- 64-bits only
- 150+ syscalls, mainly called with handles, mostly asynchronous
- vDSO: mandatory entry point to the kernel
- partial POSIX compatibility (e.g. no UID, no `fork()`)
- hypervisor, realtime
- 98K+ SLOC: subset of C++ 17 (e.g. no exception, casting and inheritance restrictions)

Zircon

Microkernel

- well suited for security: small TCB (running in ring 0)
- origin: Little Kernel (32-bits, no syscall, no MMU...)
- 64-bits only
- 150+ syscalls, mainly called with handles, mostly asynchronous
- vDSO: mandatory entry point to the kernel
- partial POSIX compatibility (e.g. no UID, no `fork()`)
- hypervisor, realtime
- 98K+ SLOC: subset of C++ 17 (e.g. no exception, casting and inheritance restrictions)

Drivers

- shared libraries (ELF)
- API/ABI defined with Banjo and Binding Instructions
- composables (relative addresses/routes)

Modularity

Components

- basic unit of executable software (e.g. app)
- sandboxed: principle of least privilege

Modularity

Components

- basic unit of executable software (e.g. app)
- sandboxed: principle of least privilege

Packages

- set of files, including component(s)
- integrity checked with Merkle tree hash for each file, thanks to a content-addressed FS: blobfs
- OTA updates: TUF and Omaha

Modularity

Components

- basic unit of executable software (e.g. app)
- sandboxed: principle of least privilege

Packages

- set of files, including component(s)
- integrity checked with Merkle tree hash for each file, thanks to a content-addressed FS: blobfs
- OTA updates: TUF and Omaha

Customization and derivability

- derivable board and product definitions with GN and Jiri
- stable system ABI with FIDL
- permissive licences (BSD-like)

Security mitigations

Good development practices

- strict language guidelines, sane/safe API, tests, doc., code review
- fuzzing (libraries, drivers, services): libFuzzer, syzkaller
- sanitizers: Address (ASAN), Undefined Behavior, Coverage...

Security mitigations

Good development practices

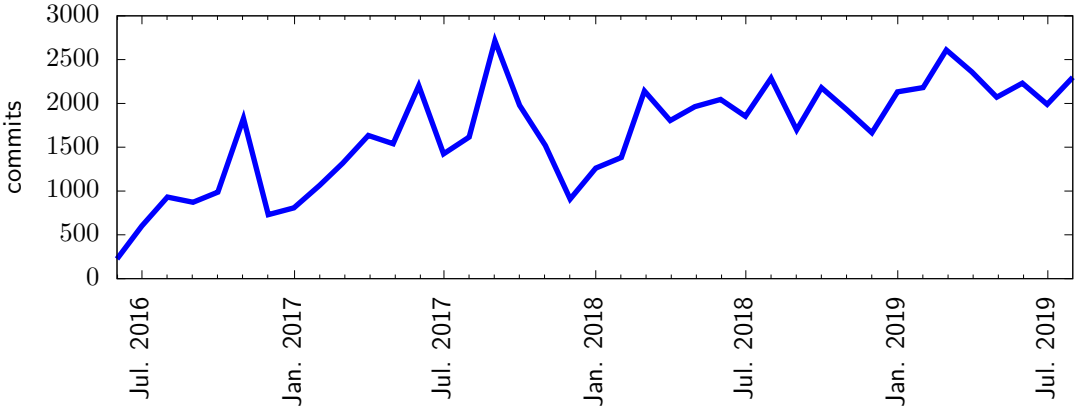
- strict language guidelines, sane/safe API, tests, doc., code review
- fuzzing (libraries, drivers, services): libFuzzer, syzkaller
- sanitizers: Address (ASAN), Undefined Behavior, Coverage...

Hardening

- ASLR with PIC/PIE, full RELRO, stack protector (strong), SafeStack, W$\oplus$X memory
- strong typing (e.g. user space vs. kernel space addresses)
- use of object destructors for security: auto-closing (pointer, handle, FD, lock), memory zeroing, type-confusion checks (debug-only)

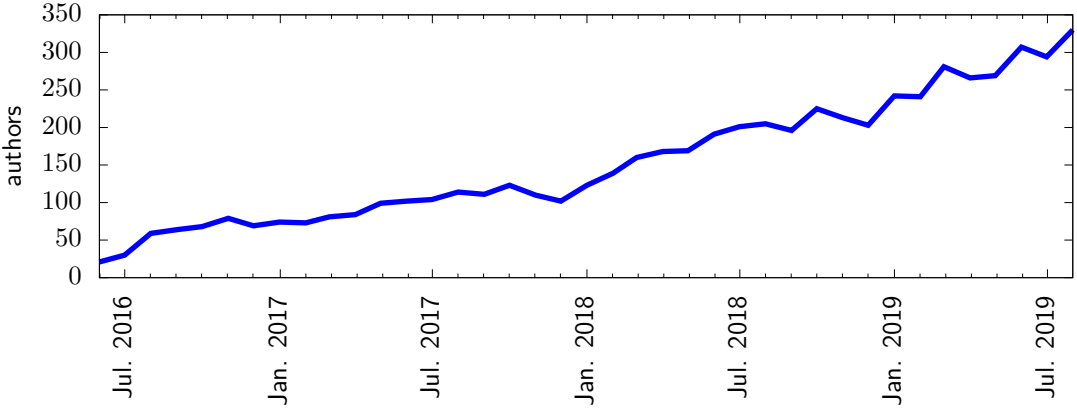
Development

Commits per month[1]



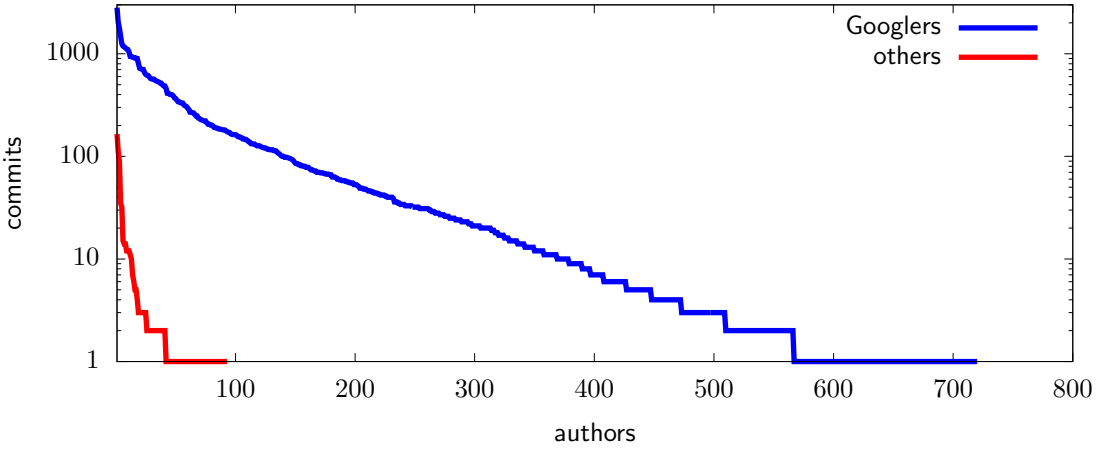
[1]Generated from the public repositories (excluding bots) the 19th September, 2019.

Authors per month[2]



[2]Generated from the public repositories (excluding bots) the 19th September, 2019.

Commits per author[3]



[3]Generated from the public repositories (excluding bots) the 19th September, 2019.

Source codes

Fuchsia's own code

Language	Files	Share
C++/C	11.7k+	80%
Rust	1.5k+	10%
Dart/Flutter	1.0k+	7%
Go	400+	3%

Source codes

Fuchsia's own code

Language	Files	Share
C++/C	11.7k+	80%
Rust	1.5k+	10%
Dart/Flutter	1.0k+	7%
Go	400+	3%

Third parties

- musl libc (stripped), jemalloc, scudo
- e1000, iwlwifi, brcm80211, ath10k...
- BoringSSL, Cairo, FreeType, ICU, Mesa, Roughtime, OpenSSH, Dash...
- Chromium

Conclusion

Takeaway

Great properties

- capability-based security OS with nice IPC specifications
- very modular architecture
- microkernel with stable device driver ABI

Takeaway

Great properties

- capability-based security OS with nice IPC specifications
- very modular architecture
- microkernel with stable device driver ABI

Limitations

- performance: balance with security, safety and patents (e.g. RCU?)
- hardware (in)security can still undermine software (e.g. side channels, Spectre)
- some coarse-grained rights (e.g. ZX_KIND_RSRC_ROOT)
- no security proof of critical components

$\Rightarrow$ not stable yet: opportunity for experiments, feedbacks and improvements

Takeaway

Great properties

- capability-based security OS with nice IPC specifications
- very modular architecture
- microkernel with stable device driver ABI

Limitations

- performance: balance with security, safety and patents (e.g. RCU?)
- hardware (in)security can still undermine software (e.g. side channels, Spectre)
- some coarse-grained rights (e.g. ZX_KIND_RSRC_ROOT)
- no security proof of critical components

$\Rightarrow$ not stable yet: opportunity for experiments, feedbacks and improvements

https://fuchsia.dev

Misc

Processes (partial)

- bootsvc
- component_manager
  - devcoordinator
  - zircon-drivers
    - devhost:sys - hid, rtc, ps2...
    - devhost:root - null, zero
    - devhost:misc - console, dmctl, ptmx sysinfo, acpi, pci...
    - devhost:pci#1 - display
    - devhost:pci#2 - block/fvm
    - devhost:pci#3 - ethernet
  - zircon-services
    - svchost
    - fshost
    - netsvc
    - virtual-console
    - blobfs:/blob
    - pkgfs
    - minfs:/data
  - fuchsia
    - appmgr (from /pkgfs) ... *.cmx

File System(s)

Full root directories

- bin
- cache
- config
- blob
- boot
- bootsvc
- data
- dev
- hub
- install
- pkg
- pkgfs
- svc
- system
- tmp
- volume

Some syscalls

```
bti_create
cache_flush
channel_call
channel_create
channel_read
channel_read_etc
channel_write
clock_adjust
clock_get
cprng_add_entropy
cprng_draw
eventpair_create
fifo_create
futex_wait
guest_create
handle_close
```

```
interrupt_bind
iommu_create
ioports_request
job_create
job_set_policy
nanosleep
object_get_info
process_create
socket_create
system_mexec
task_kill
thread_create
ticks_get
vcpu_create
vmar_allocate
vmo_create
```

FIDL example (partial)

```
library fuchsia.overnet;

using fuchsia.overnet.protocol;

[Discoverable]
protocol Overnet {
  ListPeers(uint64 last_seen_version) -> (uint64 version, vector<Peer> peers);
  RegisterService(string service_name, ServiceProvider provider);
  ConnectToService(fuchsia.overnet.protocol.NodeId node, string service_name,
                   handle<channel> chan);
};

struct Peer {
  fuchsia.overnet.protocol.NodeId id;
  bool is_self;
  fuchsia.overnet.protocol.PeerDescription description;
};
```

Package: *.cmx

1.1k+ packages

```
    "program": {
        "data": "data/ermine"
    },
    "sandbox": {
        "pkgfs": [ "packages" ],
        "services": [
            "fuchsia.bluetooth.control.Control",
            "fuchsia.cobalt.LoggerFactory",
            "fuchsia.fonts.Provider",
            "fuchsia.logger.LogSink",
            ...
            "fuchsia.modular.Clipboard",
            ...
            "fuchsia.power.BatteryManager",
            "fuchsia.sys.Environment",
            "fuchsia.sys.Launcher",
            ...
        ],
        "system": [ "data/sysui" ]
    }
```