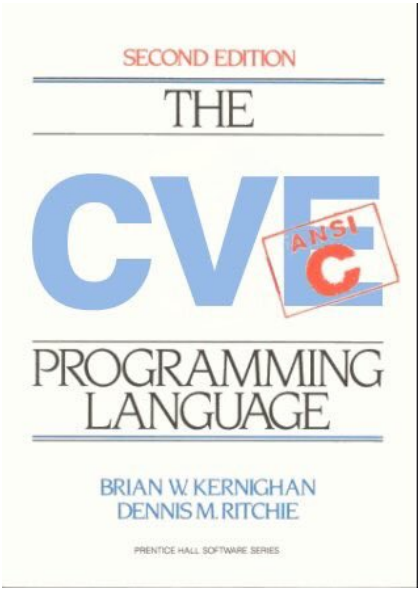
MirageOS: robust and secure services for the cloud

Hannes Mehnert, robur.io, hannes@mehnert.org

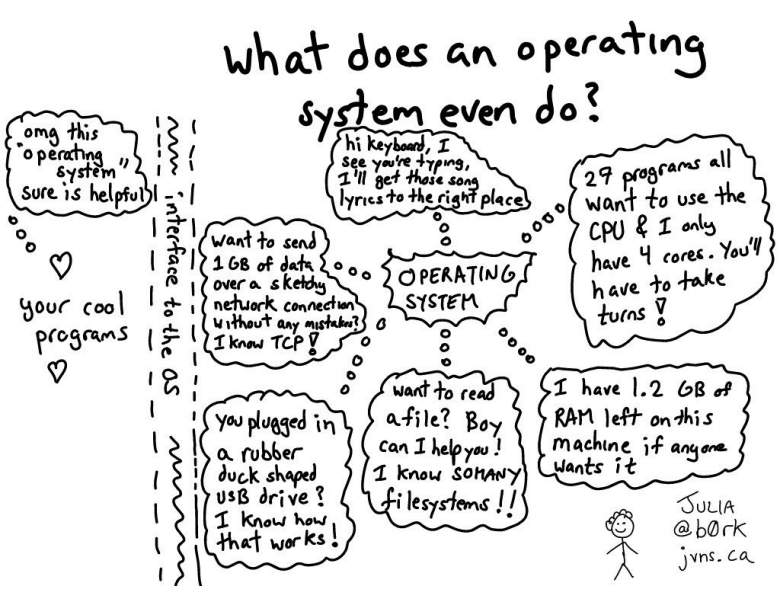
CERN computing seminar, 10th May 2019, Geneva

About me

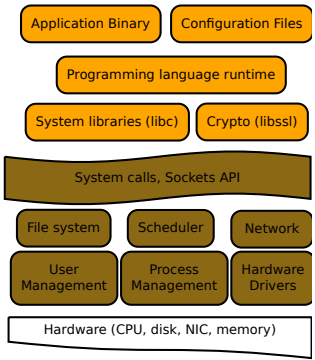
- Hacker interested in communication infrastructure, network and security protocols
- Since 2000 active in the Chaos Computer Club (CCC)
- Researching formal verification, programming language semantics, security
- PhD (2013, ITU Copenhagen) incremental verification of the correctness of object-oriented software using Coq and higher-order separation logic
- PostDoc (2014-2017) at University of Cambridge: MirageOS and formal model of TCP/IP and the Unix Sockets API in HOL4
- 2018 founded the non-profit robur.io in Berlin with the goal to deploy MirageOS
- Operating my mail server since 2000, and various other services



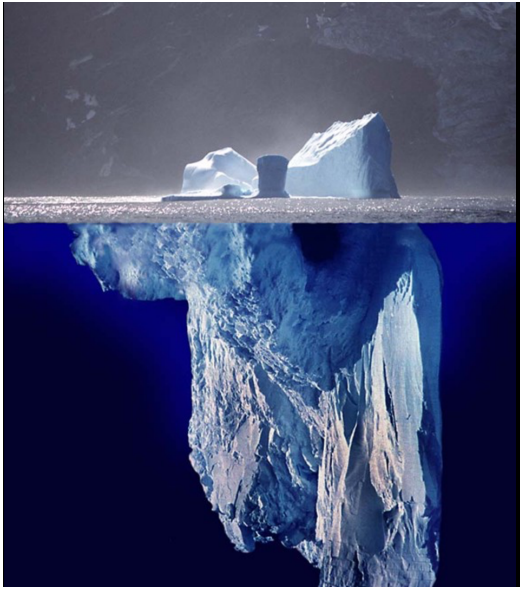
- Systems code usually written in C
- A low-level programming language
- Re-occuring memory safety issues (buffer overflow, ..)
- Not many guarantees from the compiler


What does an operating system even do?
omg this "operating system" sure is helpful
your cool programs
interface to the OS
hi keyboard, I see you're typing, I'll get those song lyrics to the right place!
OPERATING SYSTEM
29 programs all want to use the CPU & I only have 4 cores. You'll have to take turns
want to send 1 GB of data over a sketchy network connection without any mistakes? I know TCP
you plugged in a rubber duck shaped USB drive? I know how that works!
want to read a file? Boy can I help you! I know SOMANY filesystems!!
I have 1.2 GB of RAM left on this machine if anyone wants it
JULIA @b0rk jvns.ca

Stack



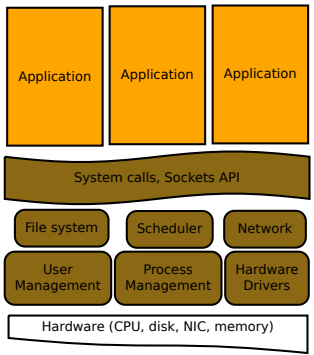
- Common Unix applications depend on shared libraries, their configuration files.
- Reproducing one application on a separate computer requires all these artifacts.



Code **you** care about

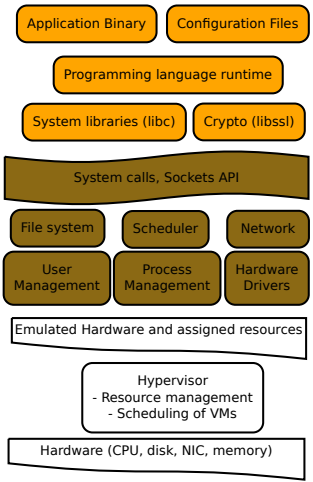
Code **the OS** insists you need

Multiple processes



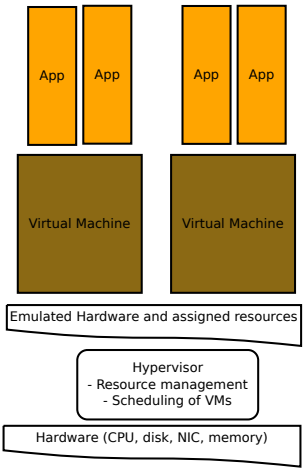
- The kernel isolates processes from each other, assigns resources, and controls access.
- An attack is contained to a single process (unless running under superuser privileges or privileges are escalated).

Hypervisor stack



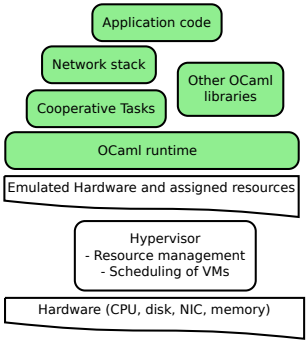
- The hypervisor manages the physical machine and assigns resources to virtual machines. Hardware is emulated.
- Leads to two layers of scheduling

Multiple Virtual Machines



- Hypervisor isolates virtual machines from each other
- An attack is contained to the VM

MirageOS unikernel - library operating system

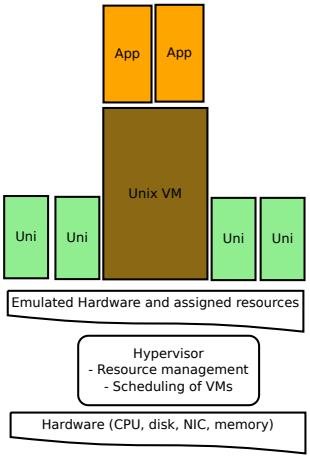


- Each MirageOS unikernel is a separate virtual machine
- Each is tailored for its single service at compilation time
- From a large set of reusable libraries
- Single address space
- Programming language guarantees isolation of libraries
- Cooperative tasks, no interrupts neither preemption
- Explicit resource dependencies: network devices, block device

Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.

Antoine de Saint-Exupery (1900 - 1944)

Unikernels and traditional OS



- Small memory footprint, can run thousands on my laptop
- Drastically reduced attack surface
- Boots in milliseconds (no hardware probing)

- Multi-paradigm memory-managed programming language
- Expressive type system with type inference
- Strong type system, no downcasts or pointer arithmetics
- Unique module system
- Compiles to native machine code, types are erased during compilation
- Used by industry and academia for proof assistants, compilers
- Developed since more than 25 years at INRIA

MirageOS OCaml style

- No objects
- Immutable data as much as sensible
- Errors are explicitly declared in the API, no exceptions
- Concurrent programming with promises using the `lwt` library
- Declarative code - easy to understand and reason about
- Expressing invariants (e.g. read-only buffer) in the type system

Example: DNS resource records

- The Domain Name System (DNS) defines several resource record types (address (A), start of authority (SOA), ..)
- Their value shapes depend on the type: an A record may only contain a set of IPv4 addresses
- The OCaml type system is sufficiently expressive to encode these invariants

```
type ttl = int32
type _ rr =
  | Soa : Soa.t rr
  | A : (ttl * Ipv4_set.t) rr
  | Aaaa : (ttl * Ipv6_set.t) rr
  | Ns : (ttl * Domain_name.Set.t) rr
  | Cname : (ttl * Domain_name.t) rr
  | Unknown : int -> (ttl * String.Set.t) rr
```

Example: MirageOS CLOCK interface

```
module type PCLOCK = sig
  val now_d_ps : unit -> int * int64
  (** [now_d_ps ()] is [(d, ps)] representing the POSIX time occuring
      at [d] * 86'400e12 + [ps] POSIX picoseconds from the epoch
      1970-01-01 00:00:00 UTC. [ps] is in the range
      [0;86_399_999_999_999_999L]. *)
end
```

- Implementation depends on target
- On Unix, `gettimeofday` is used
- On KVM and Xen, the CPU instruction `RDTSC` is used

Example: Interfaces and implementation

- As MirageOS unikernel developer, you program against the interface!
- Write once, run anywhere ;)

```
module Main (T : TIME) (P : PCLOCK) = struct
 let start _time _pclock =
  let rec speak () =
   let now = Ptime.v (P.now_d_ps ()) in
   let pp_ts = Ptime.pp_rfc3339 () in
   Format.printf "It is %a\n%!" pp_ts now;
   T.sleep_ns (Duration.of_sec 1) >>= fun () ->
   speak ()
  in
  speak ()
end
```

Example: Running a unikernel

```
$ mirage configure #configures (defaults to Unix target)
$ make depend #installs necessary dependencies
$ mirage build #compiles the unikernel, producing an ELF executable
$ ./speaking_clock #executes it

$ mirage configure -t hvt #configures for Hardware Virtualized Tender (KVM)
$ make depend #installs necessary dependencies
$ mirage build #compiles, output: monitor (solo5-hvt) and VM image (.hvt)
$ ./solo5-hvt speaking_clock.hvt  #executes it
```

Supported MirageOS targets

- Unix binary (development and testing)
- Architectures: x86-64, arm64, ESP32, soon RISC-V
- Hypervisor: Xen, KVM, FreeBSD BHyve, OpenBSD VMM, Virtio
- Muen separation kernel (Ada/SPARK)
- Genode operating system framework (Nova microkernel, L4)
- Linux binary with strict seccomp filters

You have reached the BTC Piñata.

BTC Piñata knows the private key to the bitcoin address **183XuXTTgnfYfKcHbJ4sZeF46a49Fnihdh**. If you break the Piñata, you get to keep what's inside.

Here are the rules of the game:

- You can connect to port 10000 using TLS. Piñata will send the key and hang up.
- You can connect to port 10001 using TCP. Piñata will immediately close the connection and connect back over TLS to port 40001 on the initiating host, send the key, and hang up.
- You can connect to port 10002 using TCP. Piñata will initiate a TLS handshake over that channel serving as a client, send the key over TLS, and hang up.

And here's the kicker: in both the client and server roles, Piñata requires the other end to present a certificate. Authentication is performed using standard **path validation** with a single certificate as the trust anchor. And no, you can't have the certificate key.

It follows that it should be impossible to successfully establish a TLS connection as long as Piñata is working properly. To get the spoils, you have to smash it.

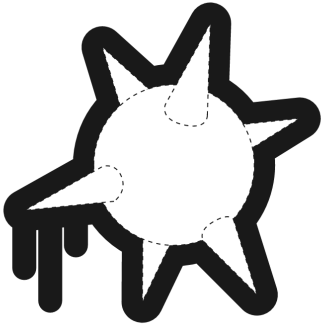
Before you ask: yes, Piñata will talk to itself and you can enjoy watching it do so.

BTC Piñata is a **MirageOS** unikernel using **not quite so broken** software. It is written in OCaml, runs directly on Xen, and is using native OCaml **TLS** and **X.509** implementations.

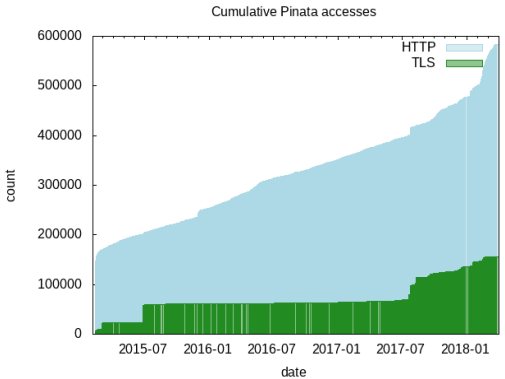
The **full list** of installed software and a **toy unikernel** without secrets are available. There is no need to use the old automated tools on Piñata - roll your own instead. This challenge runs until the above address no longer contains the 10 bitcoins it started with, or until we lose interest.

Why are we doing this? At the beginning of 2014 we started to develop a **not quite so broken** TLS implementation from scratch. You can read more about it on **https://nqsb.io** or watch our **31c3 talk** about it. Now, we want to boost our confidence in the TLS implementation we've developed and show that robust systems software can be written in a functional language. We recapitulated the **first five months of the Piñata**.

We are well aware that **bounties** can only disprove the security of a system, and never prove it. We won't take home the message that we are 'unbreakable', 'correct', and especially not



- Marketing of our from-scratch TLS implementation
- Transparent and self-serving security bait
- Web server which contains a private key for a Bitcoin wallet
- If a peer authenticates (using TLS and client certificates), it sends the private key
- Online since February 2015
- Contained 10 BTC until March 2018



@amirmc

SMALL!

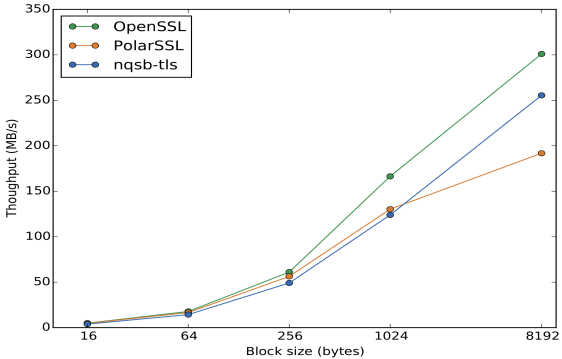
8.2MB
102 kloc

No extra stuff!

~200MB
2560 kloc

Performance of nqsb-TLS (2015, on a laptop)

- Throughput



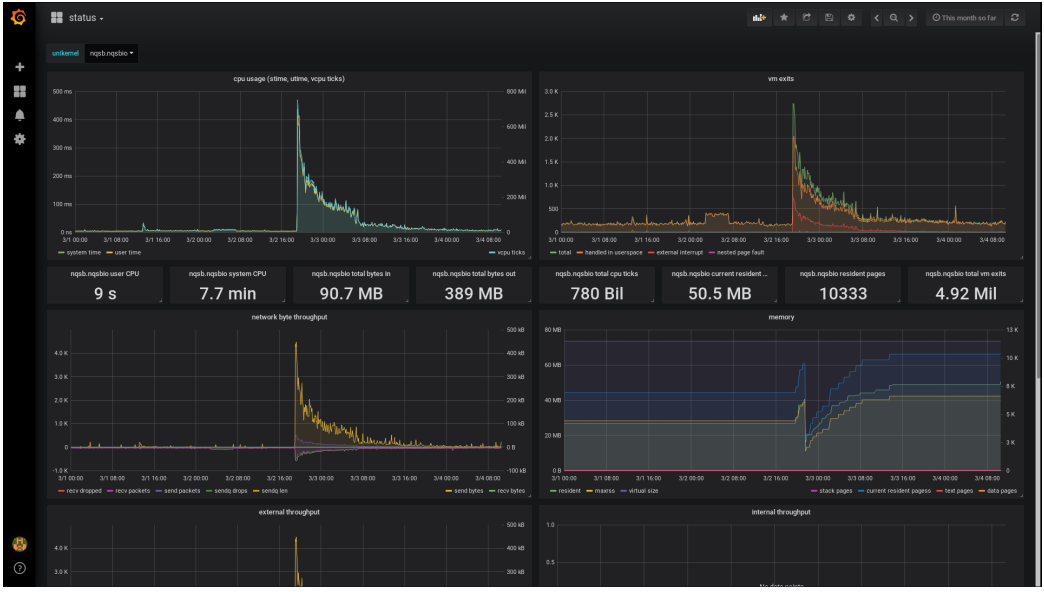
- Handshakes (number per second)

	nqsb	OpenSSL	Polar
RSA	698	723	672
DHE-RSA	601	515	367

Unikernel: CalDAV server

- Implements iCalendar and basic CalDAV features (no scheduling)
- Interoperable with macOS, iOS, Thunderbird, Android DAVdroid, and more
- Storage of data can be in a remote git repository (any key-value store)
- Funded by German ministry for research and education in 2018
- Under the prototypefund open source funding scheme (6 months)

Supported Protocols


status
This month so far
unikernel
nqsb.nqsbio
cpu usage (stime, utime, vcpu ticks)
system time
user time
vcpu ticks
vm exits
total
handled in userspace
external interrupt
nested page fault
nqsb.nqsbio user CPU
9 s
nqsb.nqsbio system CPU
7.7 min
nqsb.nqsbio total bytes in
90.7 MB
nqsb.nqsbio total bytes out
389 MB
nqsb.nqsbio total cpu ticks
780 Bil
nqsb.nqsbio current resident ...
50.5 MB
nqsb.nqsbio resident pages
10333
nqsb.nqsbio total vm exits
4.92 Mil
network byte throughput
recv dropped
recv packets
send packets
sendq drops
sendq len
send bytes
recv bytes
memory
resident
maxrss
virtual size
stack pages
current resident pages
text pages
data pages
external throughput
internal throughput

Existing MirageOS unikernels

- Various web sites including mirage.io and nqsb.io
- Content management system `Canopy` that serves Markdown files from a git remote
- OpenVPN client (work in progress)
- QubesOS firewall (run-time rule changes under development)
- CalDAV server (test deployment since November 2018)
- DNS resolver
- DNS server including let's encrypt certificate issuance
- DHCP server
- SMTP server (work in progress)
- Albatross orchestration system, deploying unikernels via TLS client certificates

- Minimized attack surface
- Avoiding common attack vectors, such as memory corruptions
- Defense in depth (in progress): R ^ X mapping, stack guard, ASLR
- Formal verification on the horizon
- Supply chain: signed library releases, reproducible builds

Signed library releases (wip)

- Library authors sign their releases
- A quorum of repository maintainers delegates packages to authors
- Impact of a private key compromise or loss is contained to their packages
- If a quorum of maintainers is compromised, game over
- Rollback, mix-and-match attacks mitigated by snapshot service
- Freeze, slow retrieval attacks mitigated by timestamp service
- Using `update framework` (Cappos NYU) with augmentation proposal TAP8

- Simple idea: compiling the source should produce identical output
- Temporary files names, timestamps, build path are problematic
- Environment (C compiler, libc, ..) needs to be specified as input
- The OCaml compiler and runtime are reproducible, MirageOS unikernels mostly
- reproducible-builds.org

Testing and formal verification

- Unit testing, quickcheck, fuzz-based testing
- Model checking with TLA+ was used for libraries
- Interactive proof assistant Coq generates OCaml code
- Dependently typed Agda generates OCaml code
- CFML is a proof system specifically for OCaml

- Research at University of Cambridge since 2008 (ongoing student projects, etc.)
- Bi-annual hack retreats since 2016 (7 days, ~35 participants)
- Using MirageOS unikernels at the retreats (DNS resolver, DHCP server, ...)
- Open source contributors worldwide, more than 100 met at retreats
- Docker for Mac and Docker for Windows was developed by MirageOS team members, and uses the MirageOS libraries
- I'm running 15 unikernels on my servers since a year (DNS, Webserver, CalDAV)

Rome ne s'est pas faite en un jour

Li Proverbe au Vilain, around 1190

Conclusion

- Security from the ground up
- Reduced complexity
- Reasonable performance
- Boots in milliseconds
- Tiny memory footprint
- Permissively licensed (BSD/MIT)
- More information at mirage.io
- We at robur.io provide commercial MirageOS development as non-profit company
- Sometimes I blog at hannes.nqsb.io
- If you're interested in further discussions, contact me via eMail hannes@mehnert.org

- At FOSDEM 2019 about Solo5 by Martin Lucina
  `https://fosdem.org/2019/schedule/event/solo5_unikernels/`
- At 34C3 (2017) by Mindy Preston
  `https://media.ccc.de/v/34c3-8949-library_operating_systems`
- At Lambda World 2018 by Romain Calascibetta
  `https://www.youtube.com/watch?v=urG5BjvjW18`
- At Esper (2015) by Anil Madhavapeddy
  `https://www.youtube.com/watch?v=bC7rTUEZfmI`