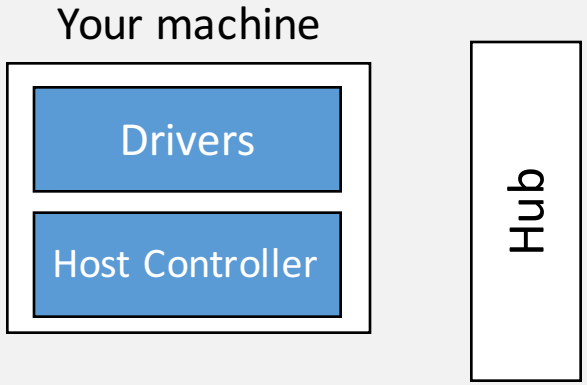
Defending against malicious peripherals with Cinch

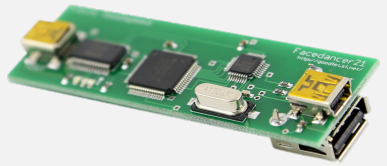
Sebastian Angel[1,2]

Riad S. Wahby[3], Max Howald[2,4], Joshua B. Leners[5],
Michael Spilo[2], Zhen Sun[2], Andrew J. Blumberg[1], and Michael Walfish[2]

[1]UT Austin [2]NYU [3]Stanford [4]The Cooper Union [5]Two Sigma

USB architecture from 30,000 feet





Peripherals' firmware ca ith Bad[illegible] [Nohl and Lell, Black Hat 2014]





Government agencies intercept and [illegible] shipments [Glenn Greenwald, The Guardian 2014]

Peripherals can exploit driver vulnerabilities



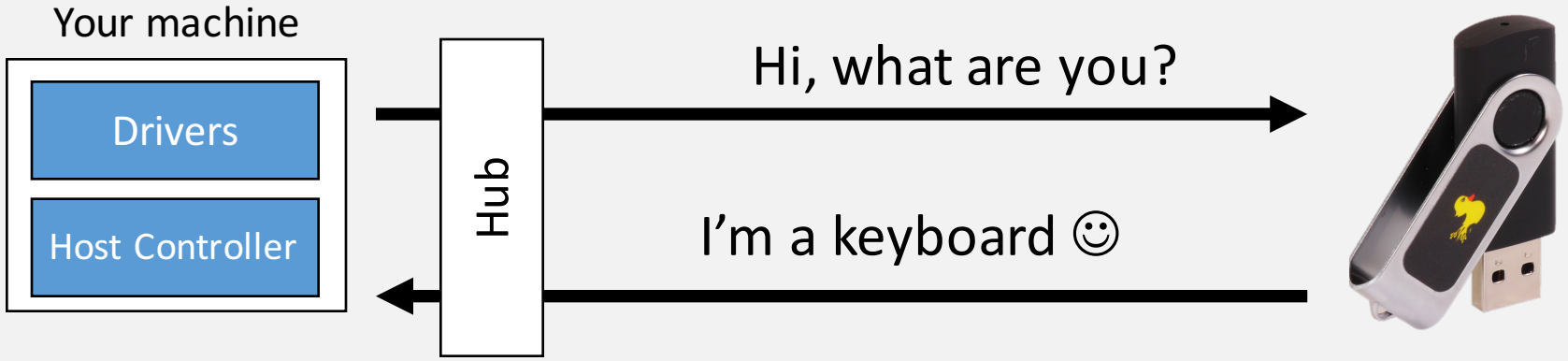
13 vulnerabilities in Linux's USB stack reported in 2016 alone

Peripherals can leverage DMA to attack OSes



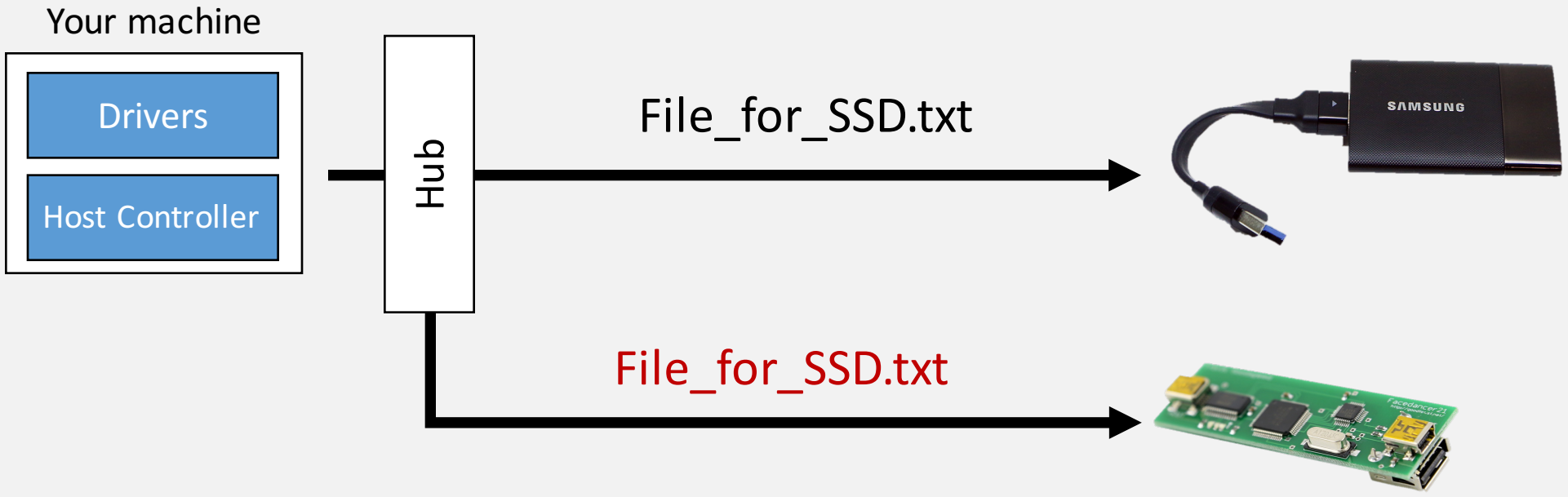
Inception [Maartmann-Moe 2014], Funderbolt [Black Hat 2013]

Peripherals can lie about their identity



Users Really Do Plug in USB Drives They Find
[Tischer et al., S&P 2016]

Hubs broadcast messages downstream



Compromised hubs can eavesdrop and modify all traffic

Okay, so what can we do?

- Don’t use a computer



- Close all the ports



Our machine interacts with untrusted devices every day… on the Internet!

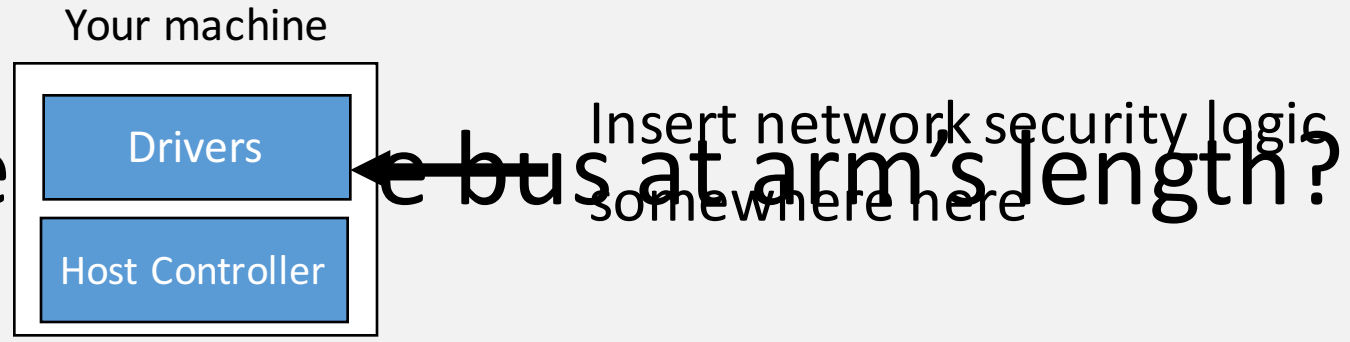
As part of this interaction, our machine routinely:

- Determines to whom it is talking
- Prevents eavesdropping and data tampering
- Defends against malicious traffic

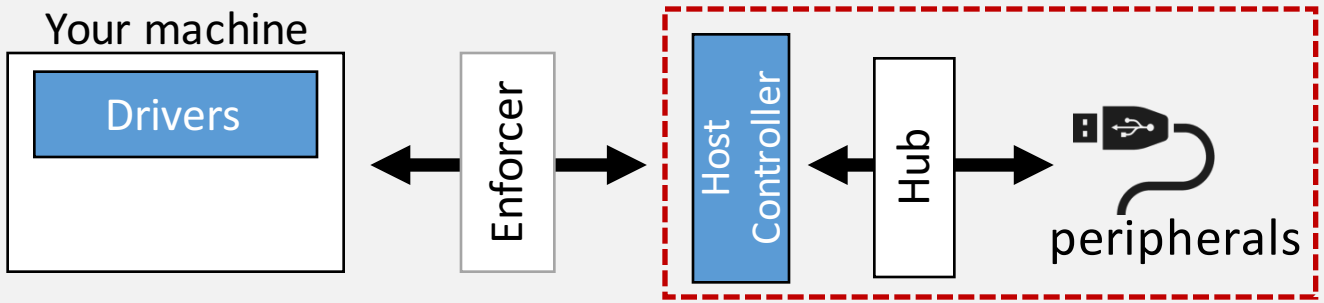
How do we apply the arsenal of network security tools to peripheral buses?

And how can this be done with minor or no modifications to OSes and existing devices…

…while keeping the bus at arm's length?



Cinch brings network defenses to USB



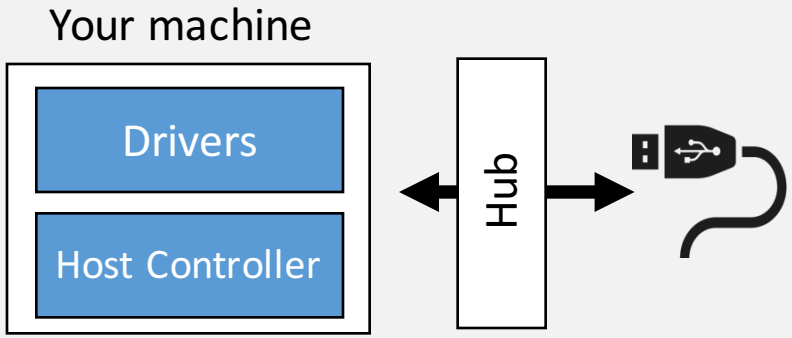
- Cinch is effective (but not perfect!) against the threats described
- Cinch is portable and backwards-compatible
  - Works transparently across OSes
  - Requires no driver or USB protocol modifications
- Cinch separates the bus from your machine, creating an enforcement point

In the rest of this talk we answer

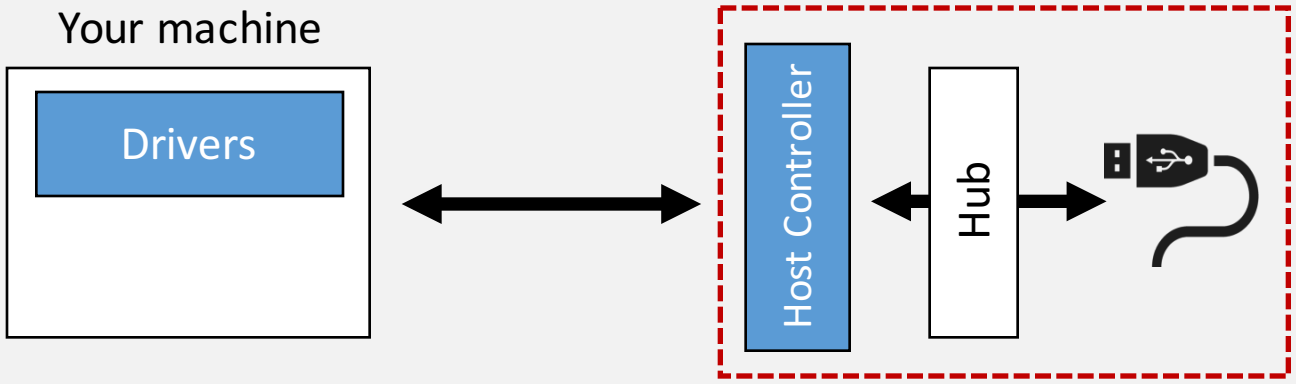
- How do we build Cinch?
- What defenses can be built on Cinch?
- How well do defenses work and what is their cost?

In the rest of this talk we answer

- How do we build Cinch?
- What defenses can be built on Cinch?
- How well do defenses work and what is their cost?

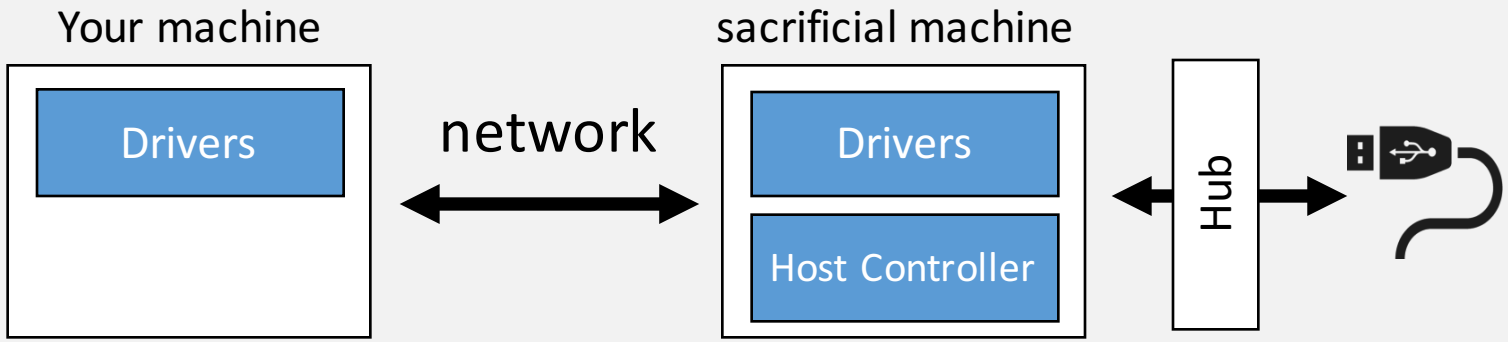


What we have today



What we want

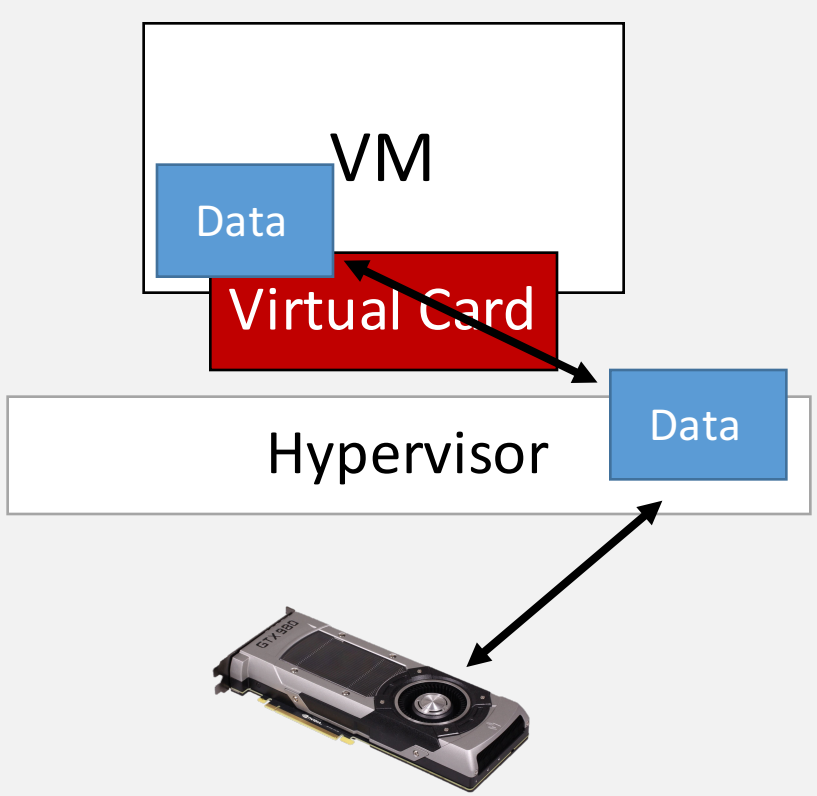
Devices can be attached to another machine

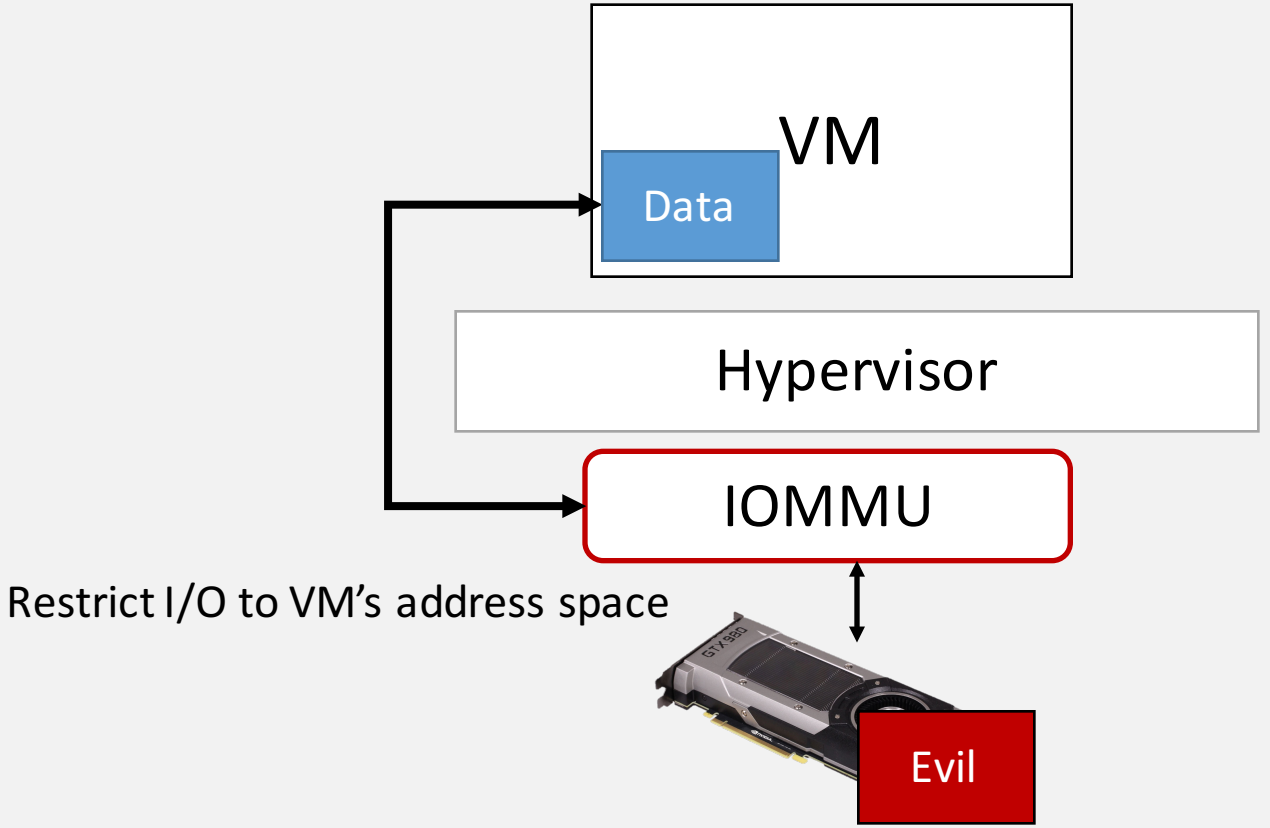


But this requires an additional machine…

Pragmatic choice: leverage virtualization technology to instantiate the (sacrificial) machine on the same hardware

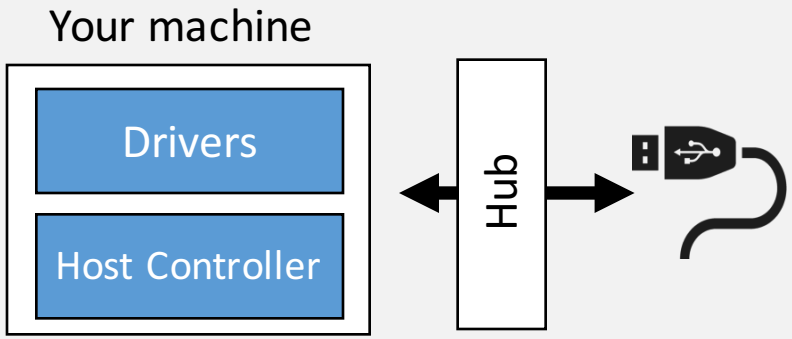
An IOMMU can be used to restrict where in memory a device may write



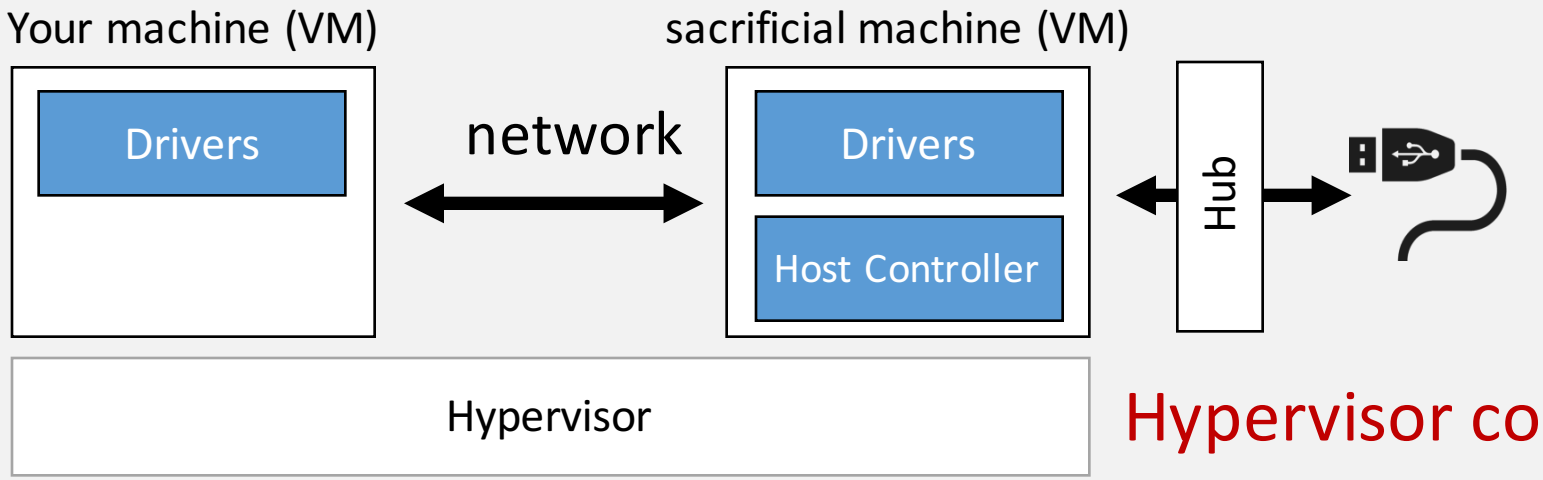


Device can only write to configured addresses

Devices are attached to a sacrificial VM



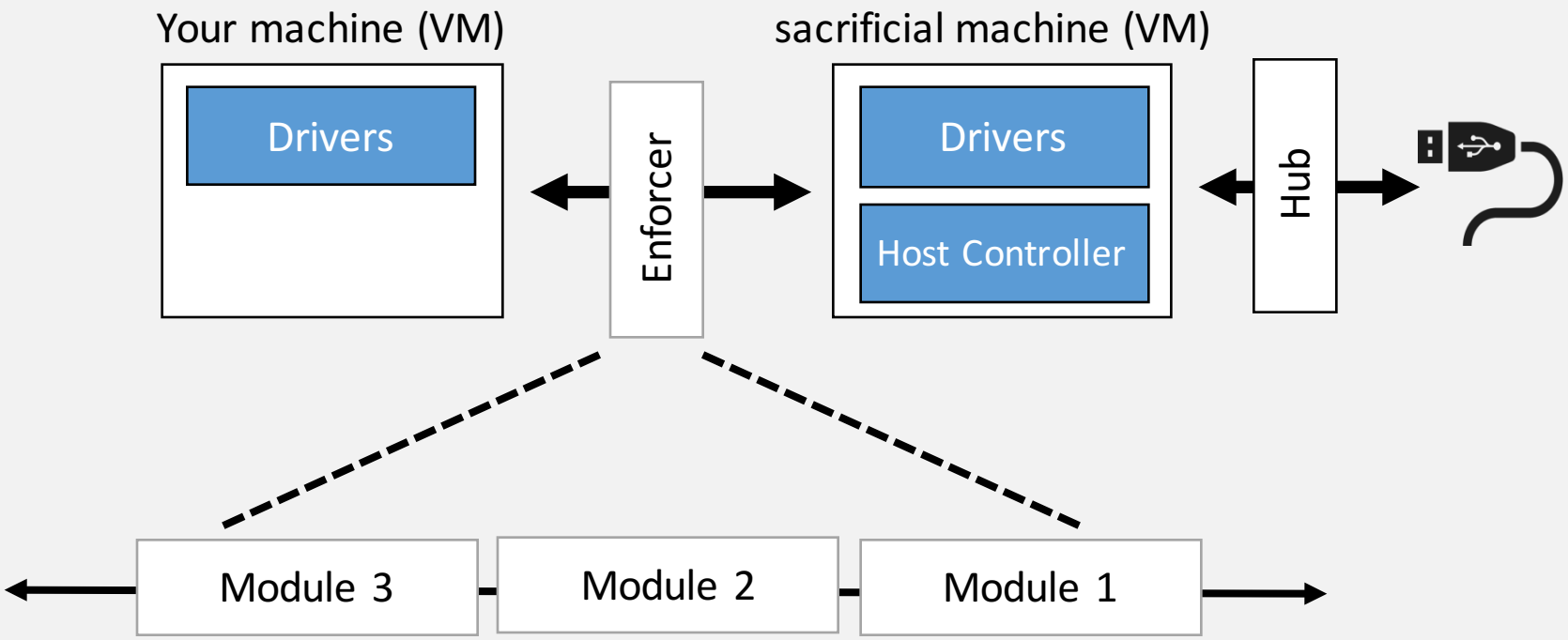
What we have today



Under Cinch

Hypervisor configures IOMMU to map bus to sacrificial machine

Interposing on VM-VM communication

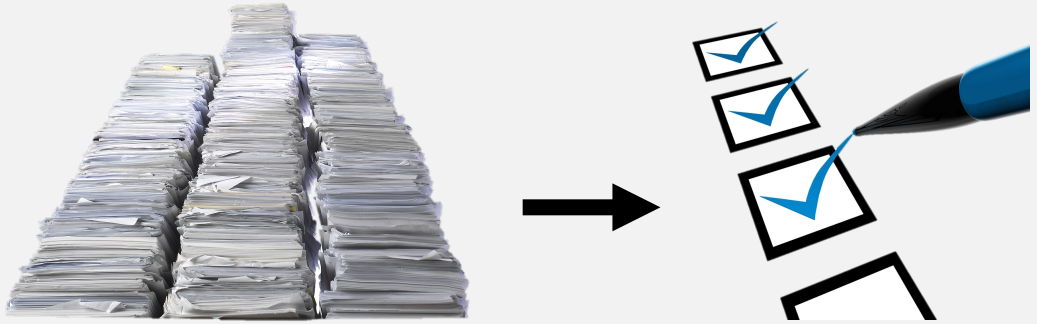


Enforcer's design is inspired by the Click modular router [Kohler et al., ACM TOCS 2000]

In the rest of this talk we answer

- How do we build Cinch?
- What defenses can be built on Cinch?
- How well do defenses work and what is their cost?

Defense 1: Enforcing allowed device behavior



USB specifications

Constraints on:

- Packet formats
- Individual fields
- Packet sequences

- Restricted field values
- Sizes within allowed range
- Proper encoding (e.g. UTF-16)

Defense 1: Enforcing allowed device behavior

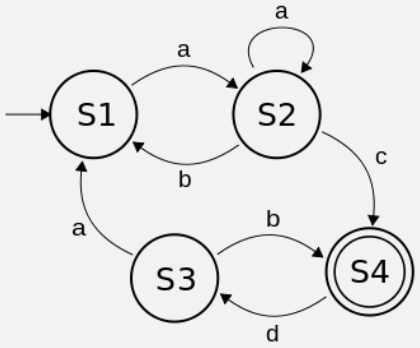


USB specifications



Constraints on:

- Packet formats
- Individual fields
- Packet sequences



- States based on history
- Transitions based on incoming packets



Allow / Drop packet

Defense 2: Filtering known exploits



Download / populate database with known malicious signatures



Inspect incoming traffic for matches



Allow / Drop packet

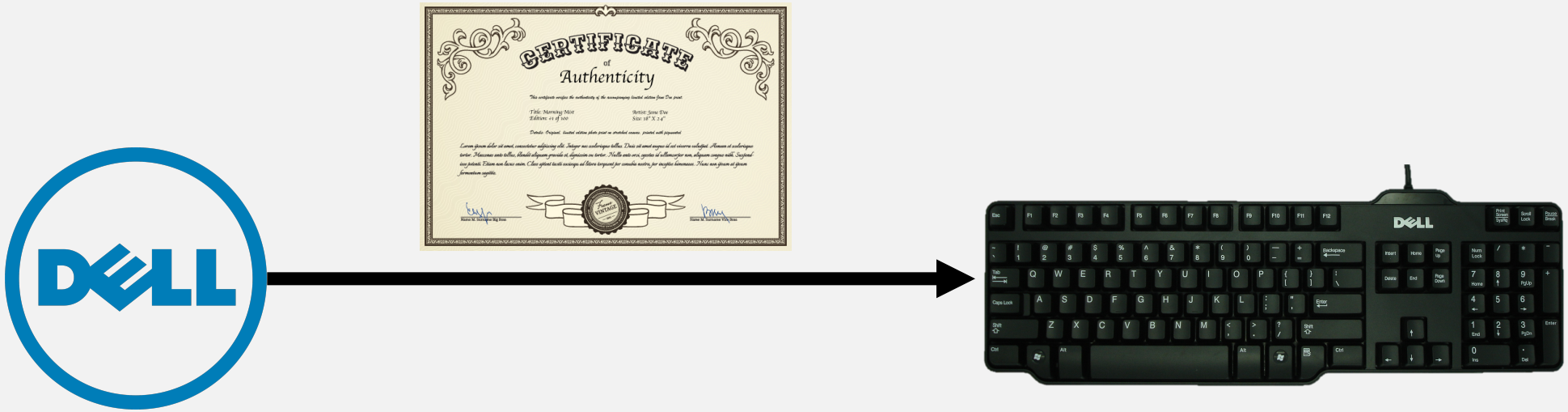
Benefits of signature-based defenses

- Quick response to an attack
  - Deriving a signature is usually faster than understanding the exploit and finding the root cause
- Useful for closed-source OSes
  - No need to wait for OS vendor patch vulnerability

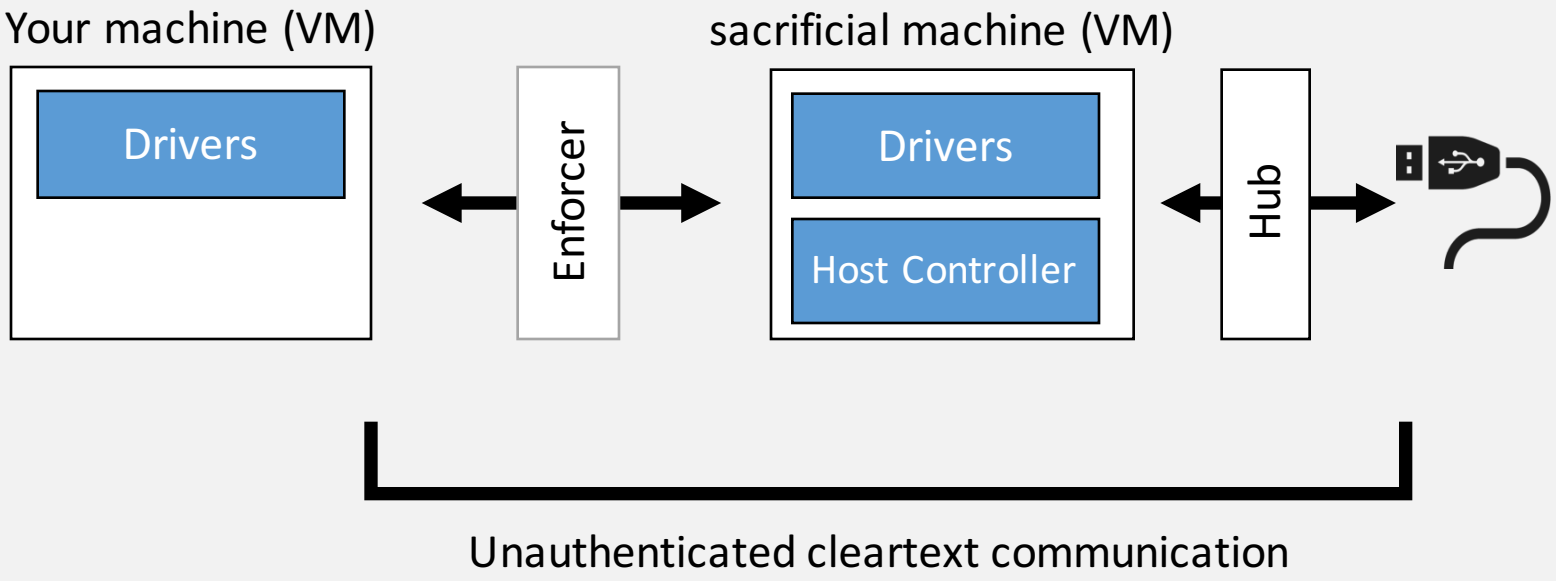
Limitations of signature-based defenses

- Cannot prevent zero-day attacks
- Tension between protection and compatibility
  - Exact signatures are not very effective
  - Very general signatures (e.g. wildcard / regex) can prevent benign traffic
- Signatures do not fix the underlying problem

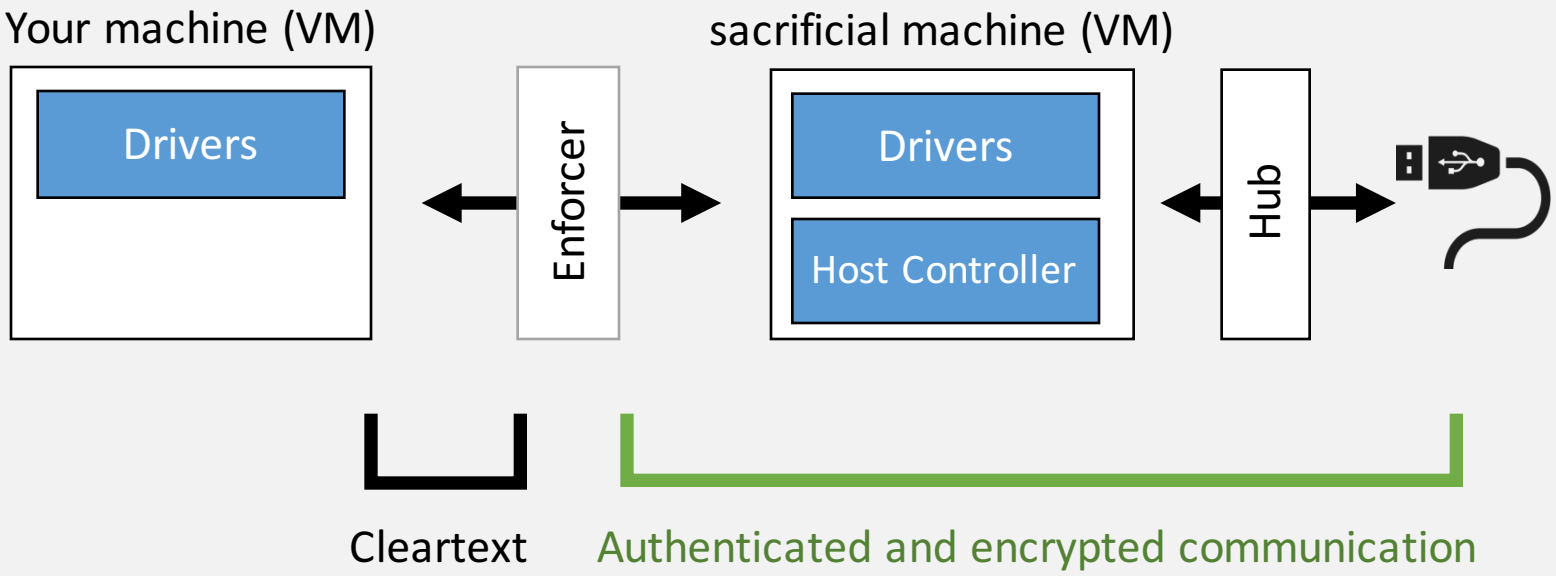
Defense 3: authentication and encryption



Defense 3: authentication and encryption

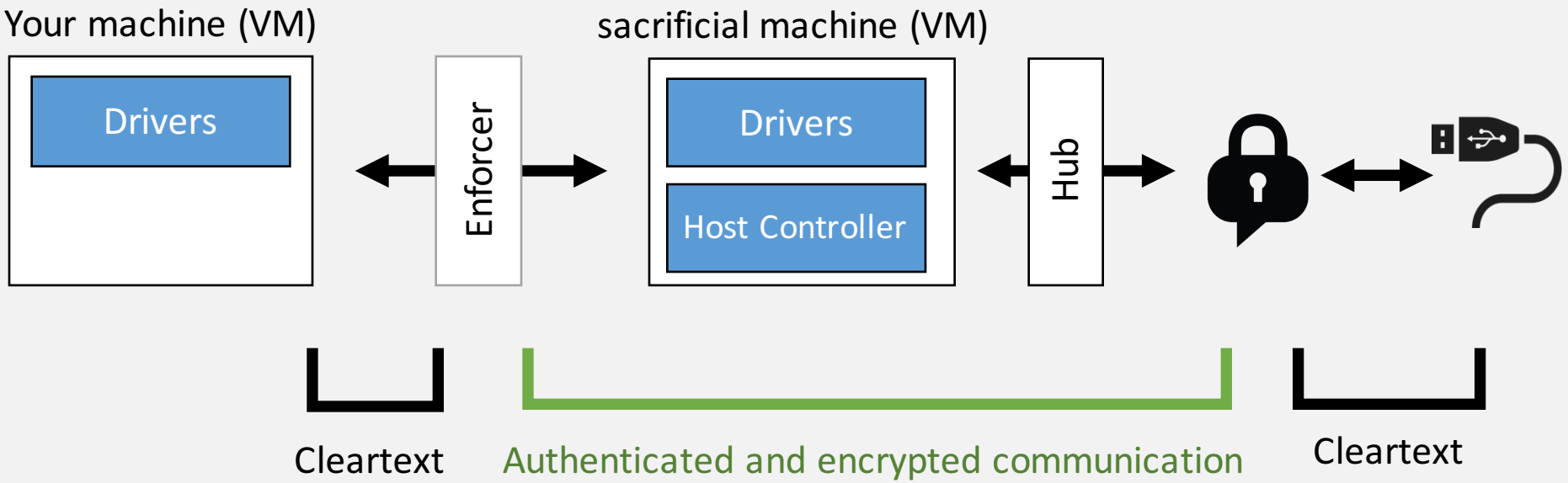


Defense 3: authentication and encryption



Install TLS endpoint at device and enforcer

Defense 3: authentication and encryption



Existing devices can be retrofitted with an adapter

Summary of defenses

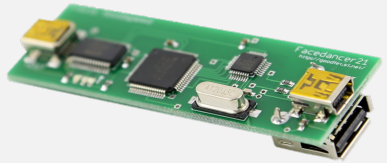
- Compliance with the USB specification
  - Prevents certain types of driver bugs from being exploited
- Signature matching
  - Prevents known exploits and can be used as a quick response
- Authentication and encryption
  - Prevent masquerading and eavesdropping on the bus
- Other: Log and replay, remote auditing, exporting functionality via higher-layer protocols (e.g., access flash drives via NFS)

In the rest of this talk we answer

- How do we build Cinch?
- What defenses can be built on Cinch?
- How well do defenses work and what is their cost?

Implementation details

- Hypervisor is Linux running QEMU/KVM
- Enforcer is a Linux user-level process and it is written in Rust
- USB transfers are encapsulated/decapsulated in TCP/IP
- We build the TLS adapter on a Beaglebone Black (arm-based computer)
- We implement exploits using a facedancer21 →



How well do defenses work?

We evaluate Cinch's effectiveness in 3 ways

- We implement exploits for existing USB driver vulnerabilities
- We carry out a 3-phase penetration testing exercise
- We use a fuzzing tool to test 10,000 invalid devices
  - Summary: Cinch's enforcer prevents all 10,000
  - Subtlety: None of the tests affected a machine without Cinch either

We implement exploits for existing USB driver vulnerabilities

- Linux CVEs reported from Jan to June 2016. They affect Linux 4.5.1
- 5 exploits that work on Windows 8.1

[Boteanu and Fowler, Black Hat Europe 2015]

Our findings:

- 16 out of 18 exploits were prevented immediately
- 2 exploits succeeded, but can be prevented with a signature

We carry out a 3-phase penetration testing exercise

- Phase 1: Red team has vague knowledge of Cinch
- Phase 2: Red team has access to a pre-configured Cinch binary
- Phase 3: Red team has Cinch's source code

Our findings:

- Increased knowledge of Cinch's functionality resulted in more intricate exploits

- Cinch is not able to prevent polymorphic attacks

What is the cost of these defenses?

Performance evaluation highlights

Baseline: connecting devices directly to your machine

Experiment 1: transferring 1 GB file to a USB 3.0 SSD

- Throughput reduction: 38% (due to memory copies)
- Memory overhead: 200 MB (due to sacrificial VM)
- CPU overhead: 8X (due to virtualization and enforcer)

Experiment 2: ping from a remote machine using USB Ethernet adapter

- Round-trip time increase: ~2 ms

Cinch brings network defenses to USB…
… but it also inherits their limitations

- Weak against polymorphic attacks on vulnerable drivers
- Requires identifying trusted manufacturers
- Requires device support (or an adapter) for TLS
- Requires hardware support for virtualizing IO (IOMMU)

Related work

- Alternate OS designs
  - Separation kernels [Rushby, SOSP ’81] [Muen, 2013] [seL4, S&P 2013] [Qubes]
  - Deprivileging drivers [Microkernel, CACM ’70, SOSP ‘95], [Exokernel, SOSP ’95]

- Driver isolation and reliability
  - Correct driver synthesis [Termite, SOSP’09]
  - Driver behavior monitoring [Nexus, OSDI ‘08], [SUD, ATC’10]
  - Driver isolation [Nooks, SOSP ‘03]

- USB-specific approaches
  - Hotplug and device containment frameworks [GoodUSB, ACSAC ‘15] [USBFILTER, next talk!]
  - Bus encryption [UScramBle, EUROSEC ‘16] [USBSec, SERE ‘12]

Summary

- Cinch provides a backward-compatible and portable way of enhancing peripheral buses with tools from network security
- Cinch's enforcer is modular and defenses are natural and easy to implement
- Cinch is not perfect, but eliminates some attack classes and increases the barrier for others