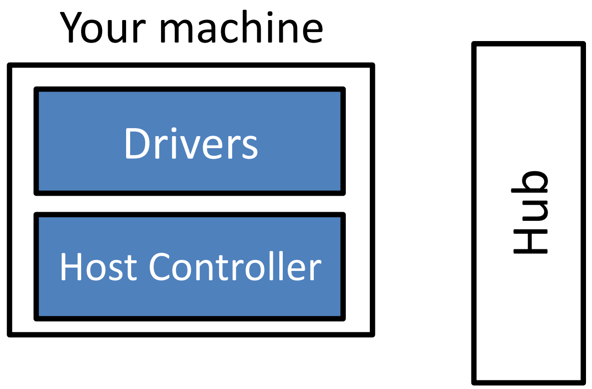
Defending against malicious peripherals with Cinch

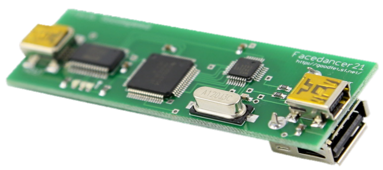
Presented by Avesta Hojjati
CS598
Computer Security in the Physical World
University of Illinois
Based on slides by Sebastian Angel

Citation

- S. Angel,R. Wahby, M. Howald, J. Leners, M. Spilo, Z. Sun, A. Blumberg, M. Walfish. "Defending against Malicious Peripherals with Cinch." **USENIX Security 2016**

USB architecture from 30,000 feet

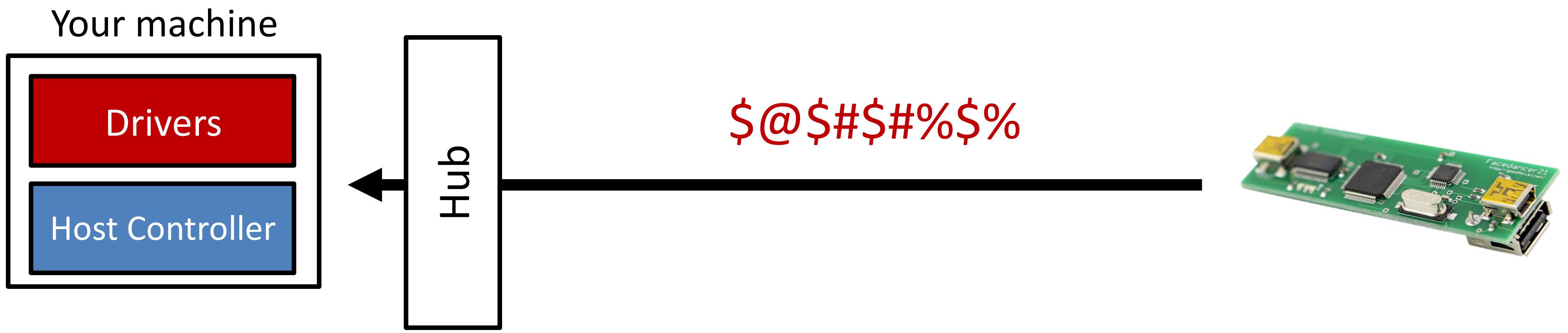




Peripherals’ firmware ca[illegible]with Bad[illegible]ohl and Lell, Black Hat 2014]

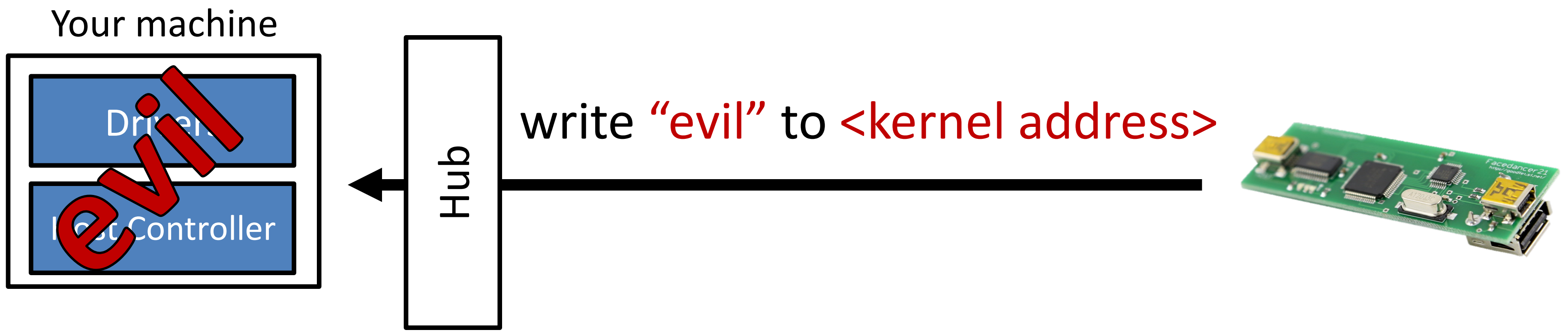
Government agencies intercept and [illegible]odify shipments [Glenn Greenwald, The Guardian 2014]

Peripherals can exploit driver vulnerabilities



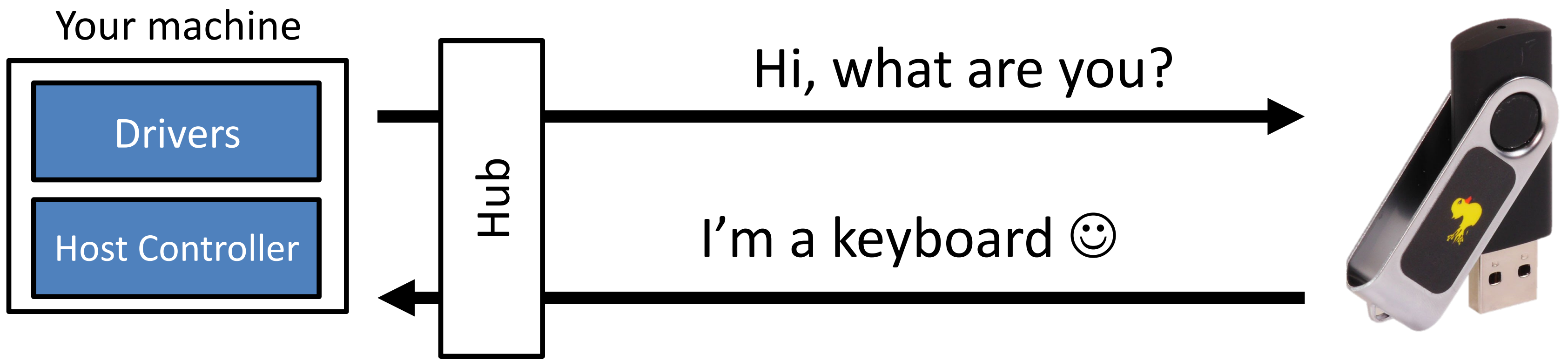
13 vulnerabilities in Linux’s USB stack reported in 2016 alone

Peripherals can leverage DMA to attack OSes



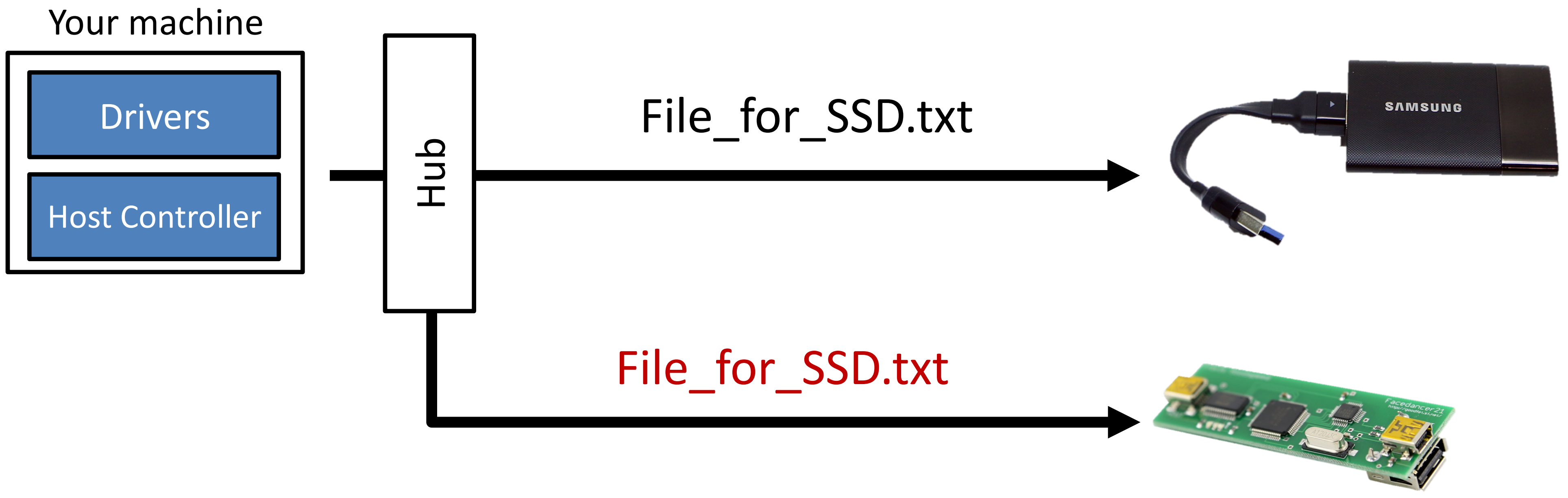
Inception [Maartmann-Moe 2014], Funderbolt [Black Hat 2013]

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!

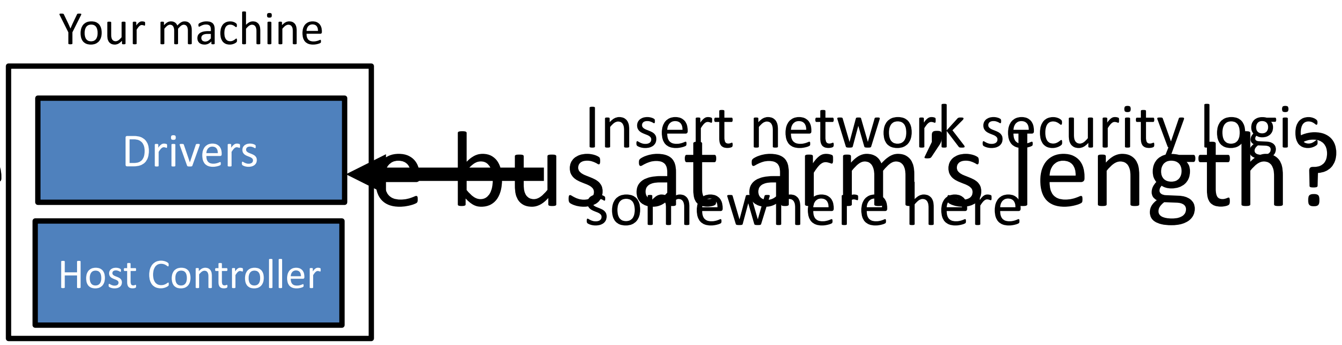
As part of this interaction, our machine routinely:

- 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…

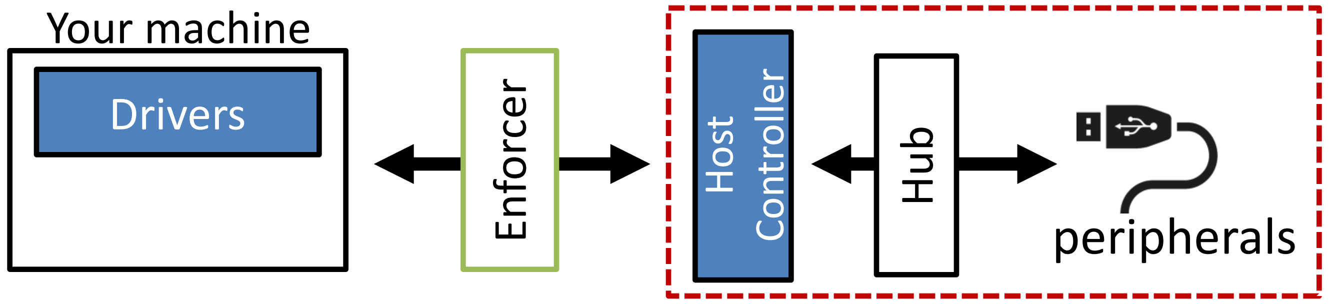
…while ke e bus at arm’s length?



Design requirements

- Making peripheral buses look “remote”, preventing direct action with the rest of the computer
- Traffic between the “remote” devices and rest of the computer should travel through a “narrow choke point”, this is essential to apply defense
- The solution should NOT require modification of the *bus*
- Portability, no re-design, or re-implementation for different OSes
- Flexibility and extensibility
- Imposing reasonable overhead

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…

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

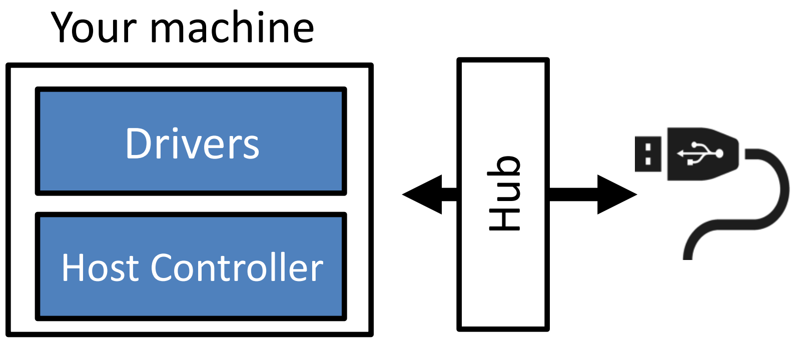
In the rest of this talk…

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

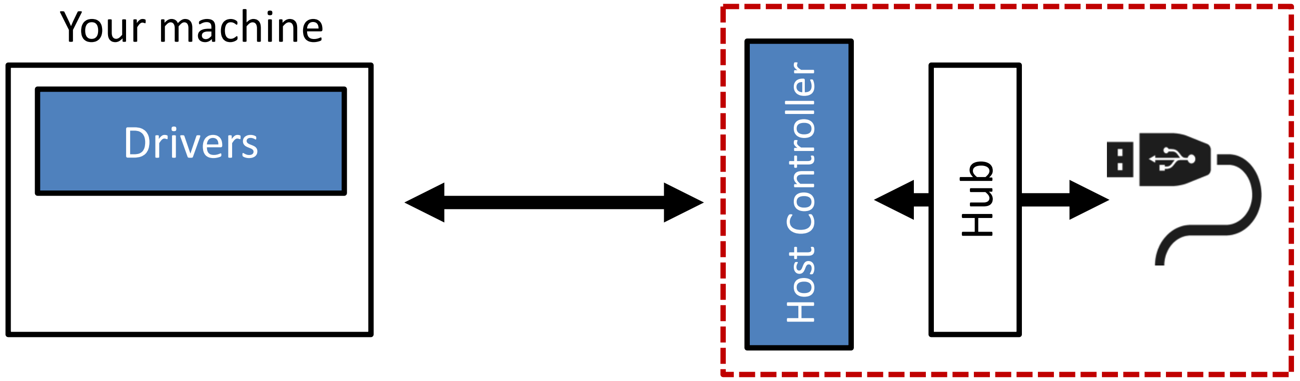
What do we need to answer?

- Where and how can one create a logical separation between the bus and the host, while arranging for an explicit communication channel that a policy enforcement mechanism can interpose on?

- How can one instantiate this separation and channel with no modifications to bus standards, OSes, or driver stacks?

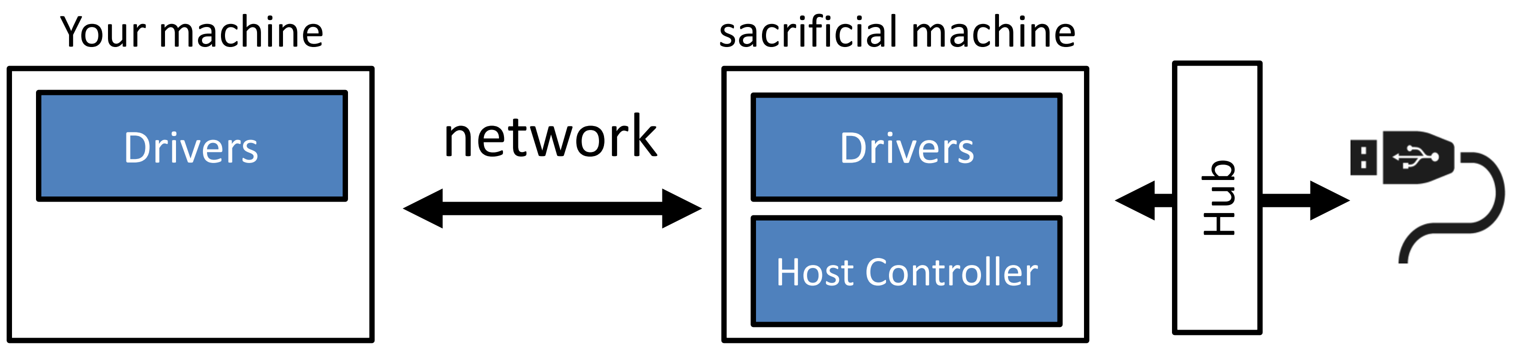


What we have today



What we want

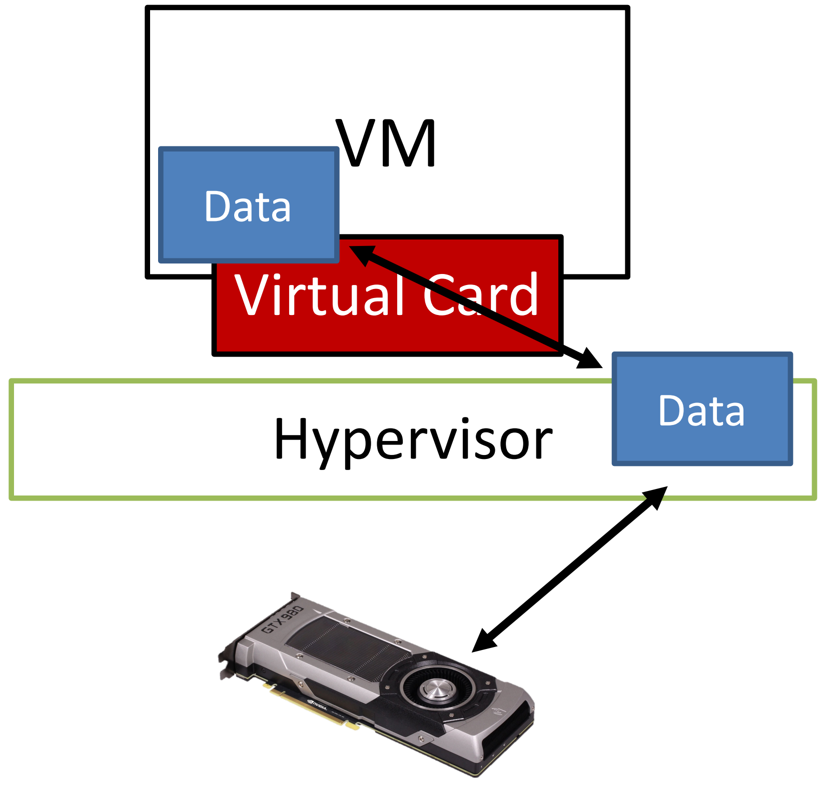
Devices can be attached to another machine

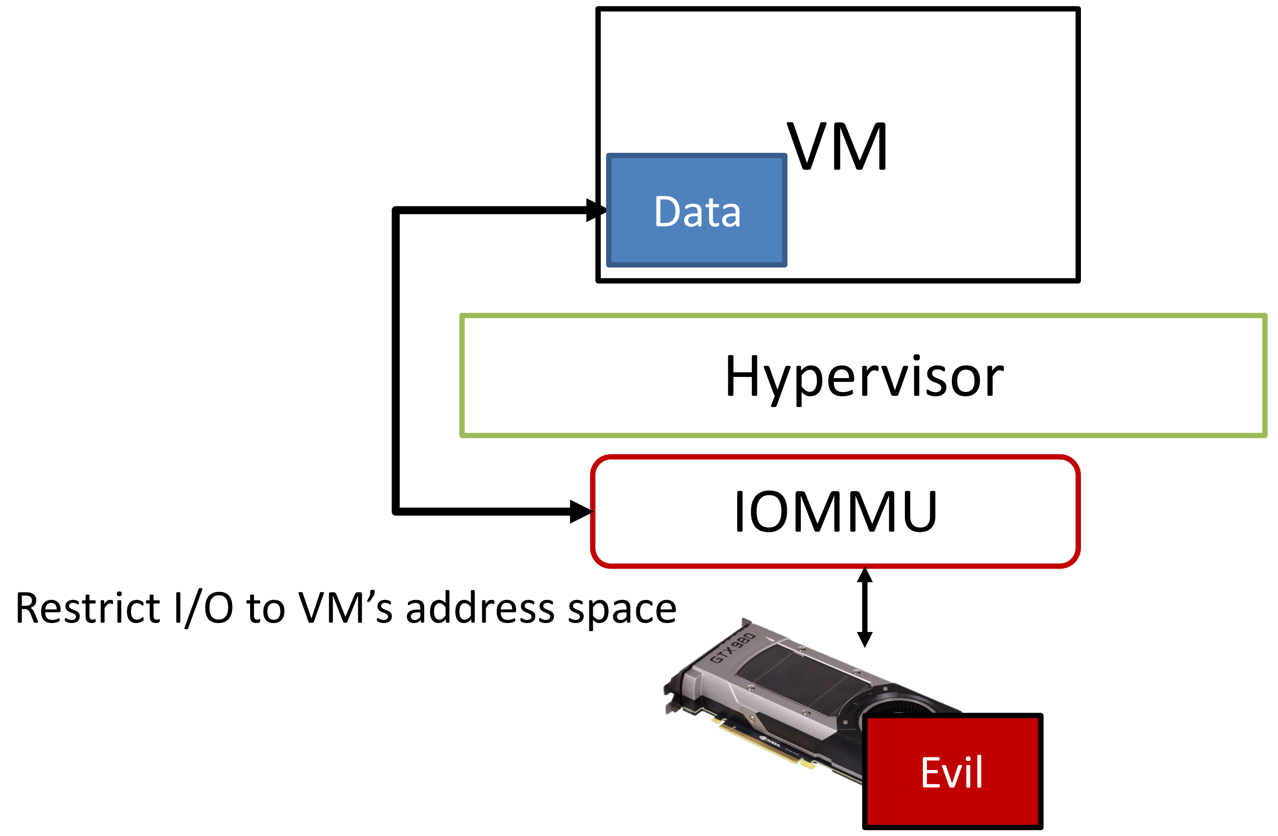


But this requires an additional machine…

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

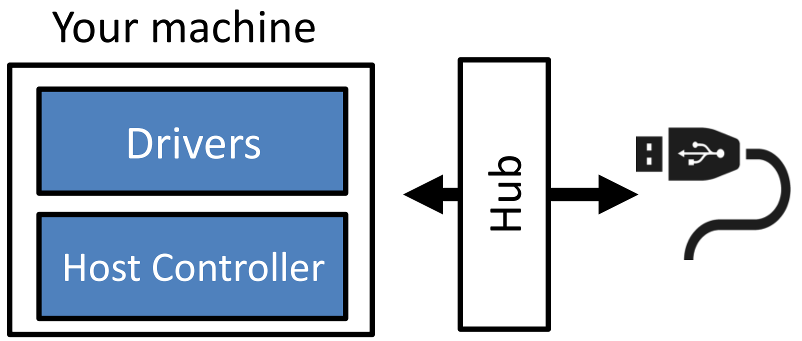
An IOMMU can be used to restrict where in memory a device may write



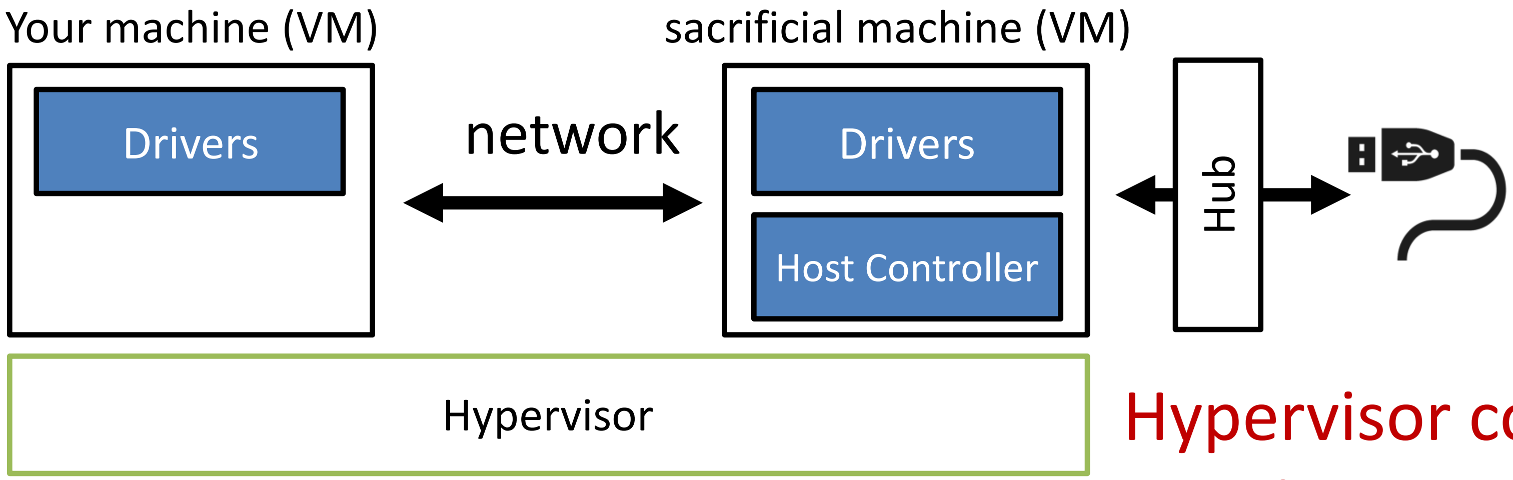


Device can only write to configured addresses

Devices are attached to a sacrificial VM



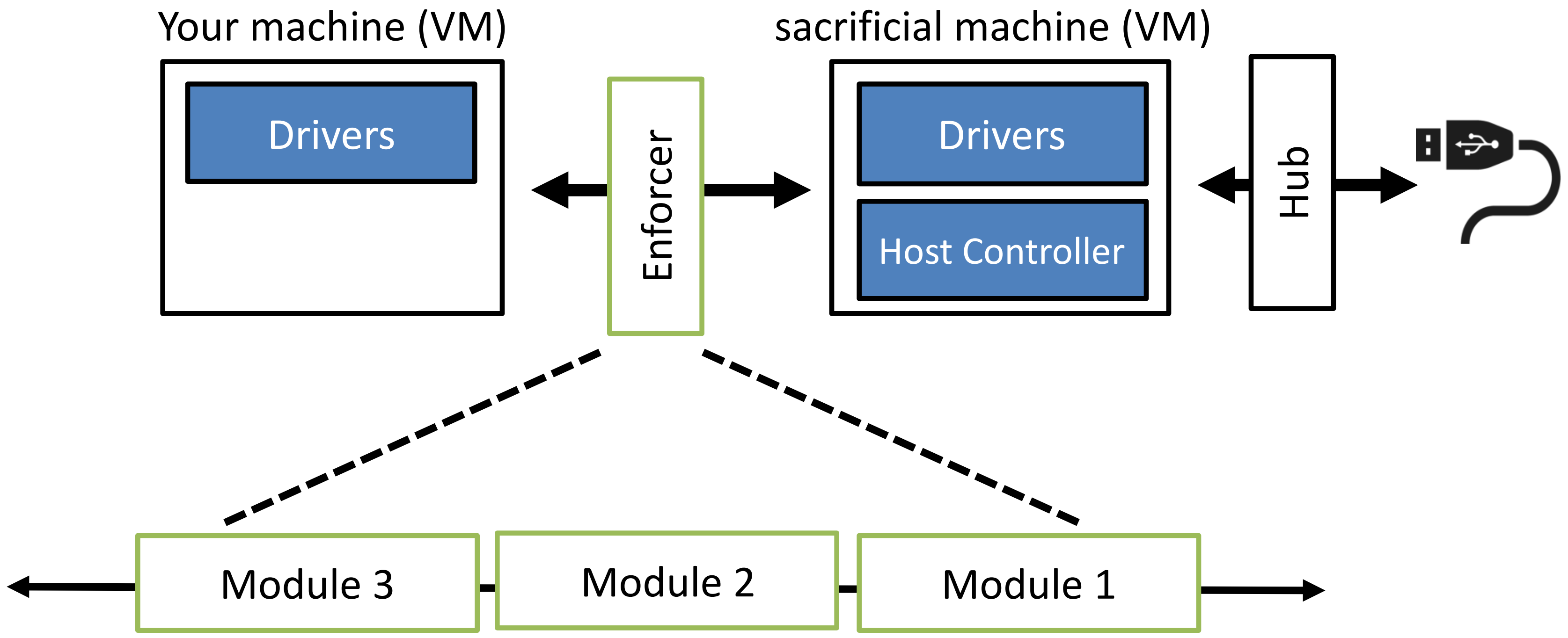
What we have today



Under Cinch

Hypervisor configures IOMMU to map bus to sacrificial machine

Interposing on VM-VM communication



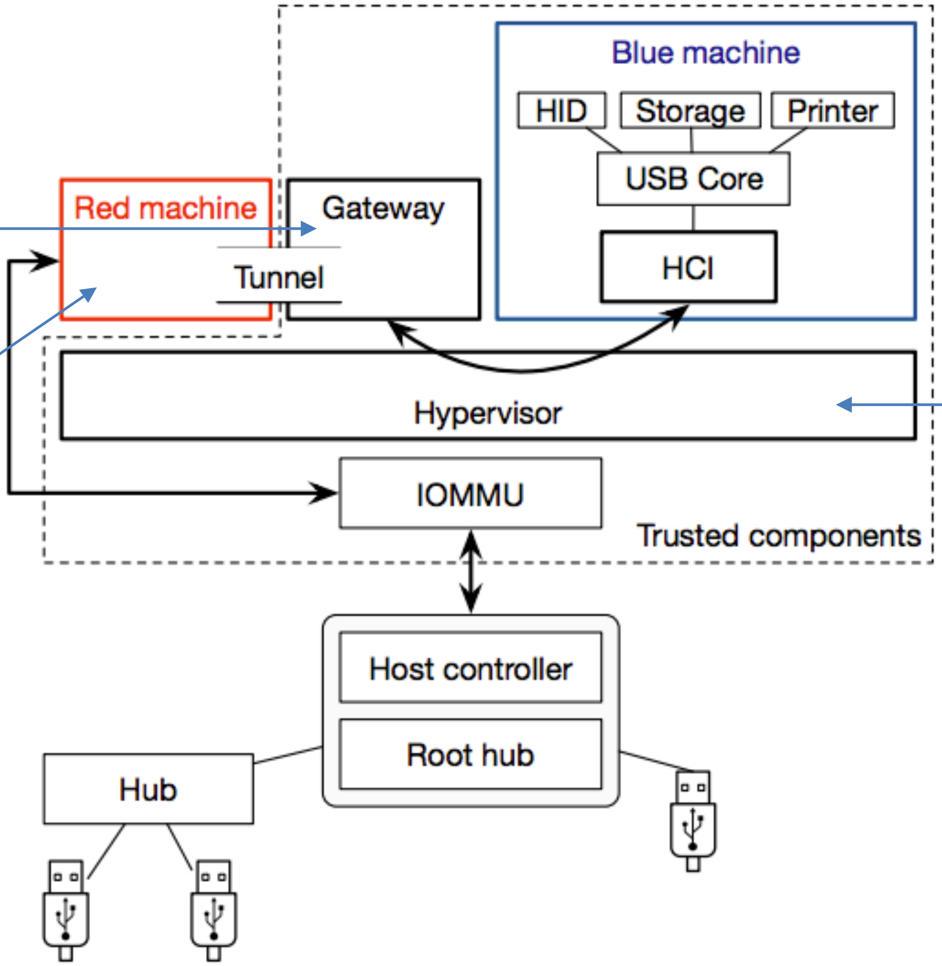
Enforcer's design is inspired by the Click modular router [Kohler et al., ACM TOCS 2000]

The architecture of Cinch

Enforces security policy

Normal OS with stripped down USB STACK

Driver



In the rest of this talk…

- How did they 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

$\longrightarrow$

USB specifications

Constraints on:

- Packet formats
- Individual fields
- Packet sequences

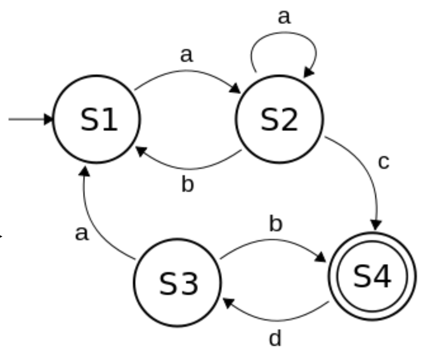
- Restricted field values
- Sizes within allowed range
- Proper encoding (e.g. UTF-16)

Defense 1: Enforcing allowed device behavior

USB specifications

Constraints on:

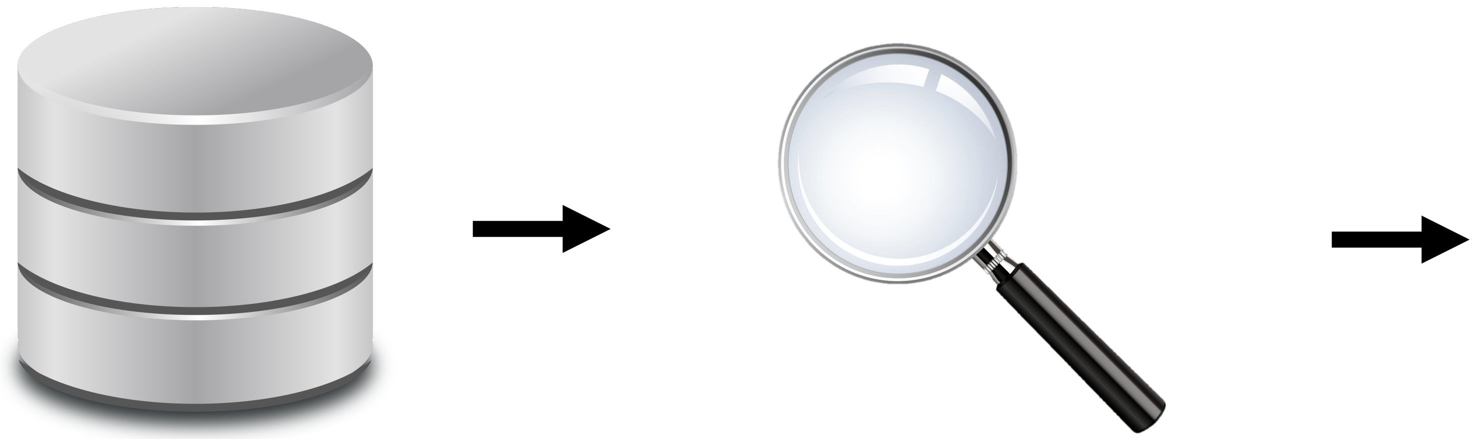
- Packet formats
- Individual fields
- Packet sequences



- 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

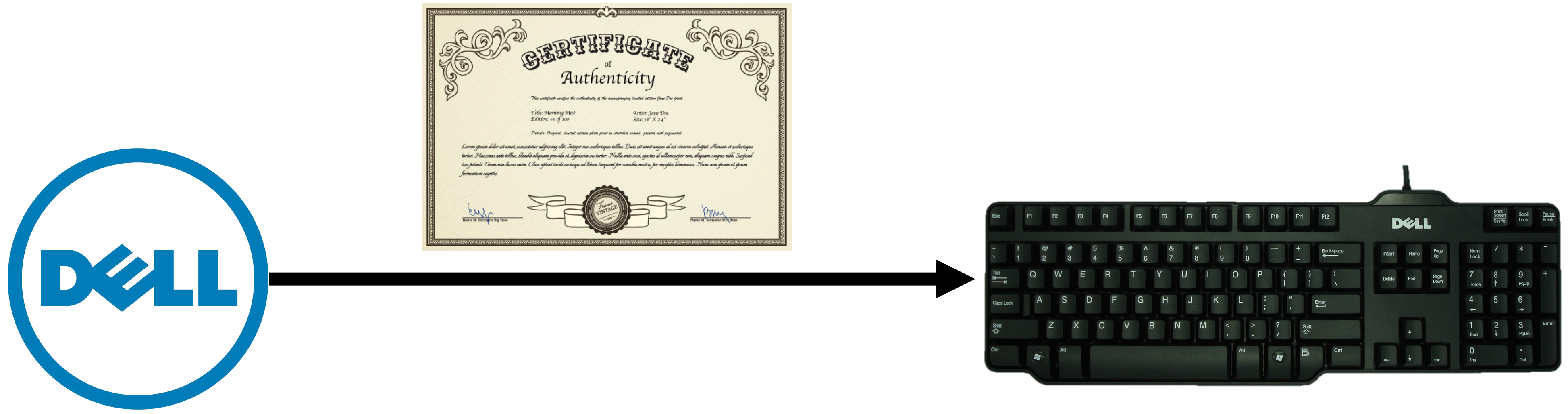
Limitations of signature-based defenses

- 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



CERTIFICATE
of
Authenticity

This certificate verifies the authenticity of the accompanying limited edition Jane Doe print.

Title: Morning Mist
Edition: #1 of 100

Artist: Jane Doe
Size: 18" X 24"

Details: Original, limited edition photo print on stretched canvas, printed with pigmented

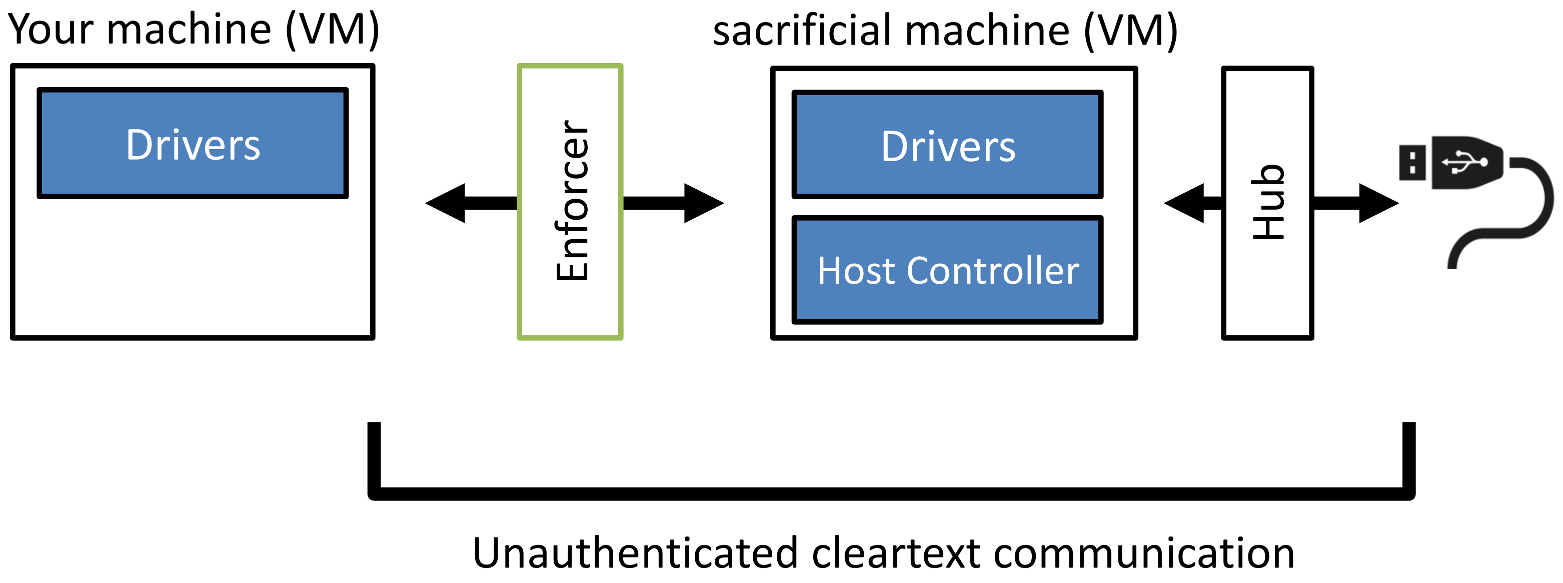
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec scelerisque tellus. Duis sit amet augue id est viverra volutpat. Aenean et scelerisque tortor. Maecenas ante tellus, blandit aliquam gravida et, dignissim eu tortor. Nulla ante orci, egestas id ullamcorper non, aliquam congue nibh. Suspendisse potenti. Etiam non lacus enim. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nunc non ipsum at ipsum fermentum sagittis.

[illegible]
Name M. Surname Big Boss

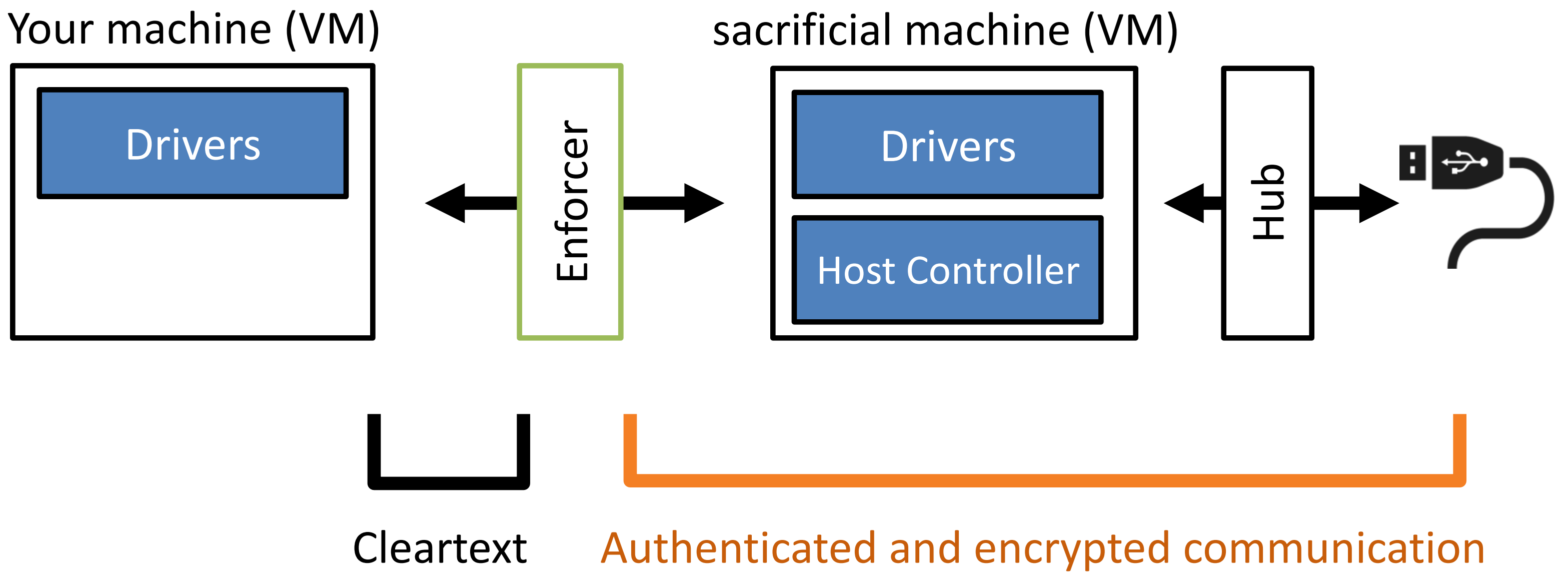
Frame VINTAGE

[illegible]
Name M. Surname Vice Boss

Defense 3: authentication and encryption

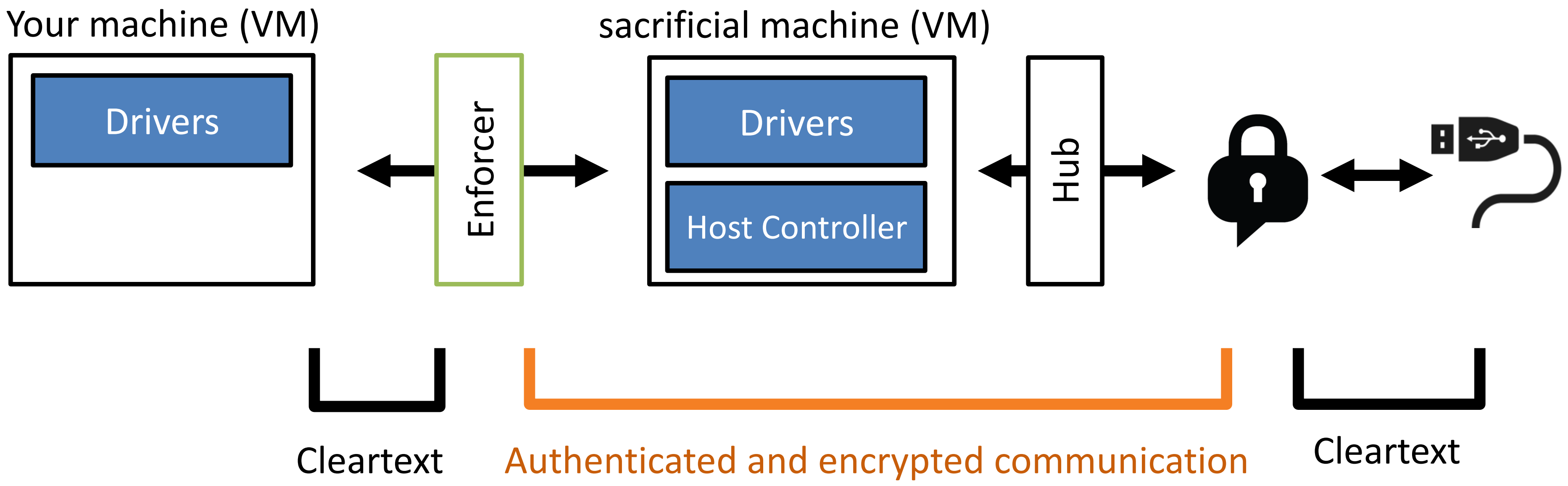


Defense 3: authentication and encryption



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

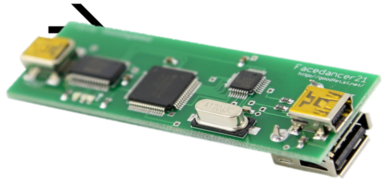
- Other: Log and replay, remote auditing, exporting functionality via higher-layer protocols (e.g., access flash drives via NFS)

In the rest of this talk…

- How did they 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
- They built the TLS adapter on a Beaglebone Black (arm-based computer)
- They implemented exploits using a facedancer21 ->



How well do defenses work?

Evaluation of Cinch's effectiveness happens in 3 ways

- They implemented exploits for existing USB driver vulnerabilities
- They carried out a 3-phase penetration testing exercise
- They used 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

They implemented 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]

Their findings:

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

They carried 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

Their 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)

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

Discussion

- What do you think about their work compare to GoodUSB & USBFILTER?
- Is the 38% throughput reduction worth it?
- Any fundamental issues with QEMU and KVM model?
- USBee
- Can GoodUSB, USBFILTER, Cinch; protect us against USBee?