# Detecting Attacks, Part 1

## *CS 161: Computer Security*


**Prof. Vern Paxson**

TAs: Paul Bramsen, Apoorva Dornadula,
David Fifield, Mia Gil Epner, David Hahn, Warren He,
Grant Ho, Frank Li, Nathan Malkin, Mitar Milutinovic,
Rishabh Poddar, Rebecca Portnoff, Nate Wang


*https://inst.eecs.berkeley.edu/~cs161/*

**April 13, 2017**

# Summary of TLS & DNSSEC Technologies

- TLS: provides channel security for communication over TCP (confidentiality, integrity, authentication)
  - Client & server agree on crypto, session keys
  - Underlying security dependent on trust in Certificate Authorities (as well as implementors)
- DNSSEC: provides object security for DNS results
  - Just integrity & authentication, not confidentiality
  - No client/server setup "dialog"
  - Tailored to be caching-friendly
  - Underlying security dependent on trust in Root Name Server's key …
  - … plus support provided by every level of DNS hierarchy from Root to final name server… and local resolver!

# TaoSecurity

Richard Bejtlich's blog on digital security, strategic thought, and military history.

Tuesday, September 25, 2012

## Unrealistic "Security Advice"

I just read a blog post (no need to direct traffic there with a link) that included the following content:

*This week, I had the opportunity to interview the hacking teams that used zero-day vulnerabilities and clever exploitation techniques to compromise fully patched iPhone 4S and Android 4.0.4 (Samsung S3) and the big message from these hackers was simple:* ***Do not use your mobile device for *anything* of value, especially for work e-mail or the transfer of sensitive business documents.***

*For many, this is not practical advice. After all, your mobile device is seen as an extension of the*

We would be much better served if we accepted that **prevention eventually fails**, so we need **detection, response, and containment** for the incidents that will occur.

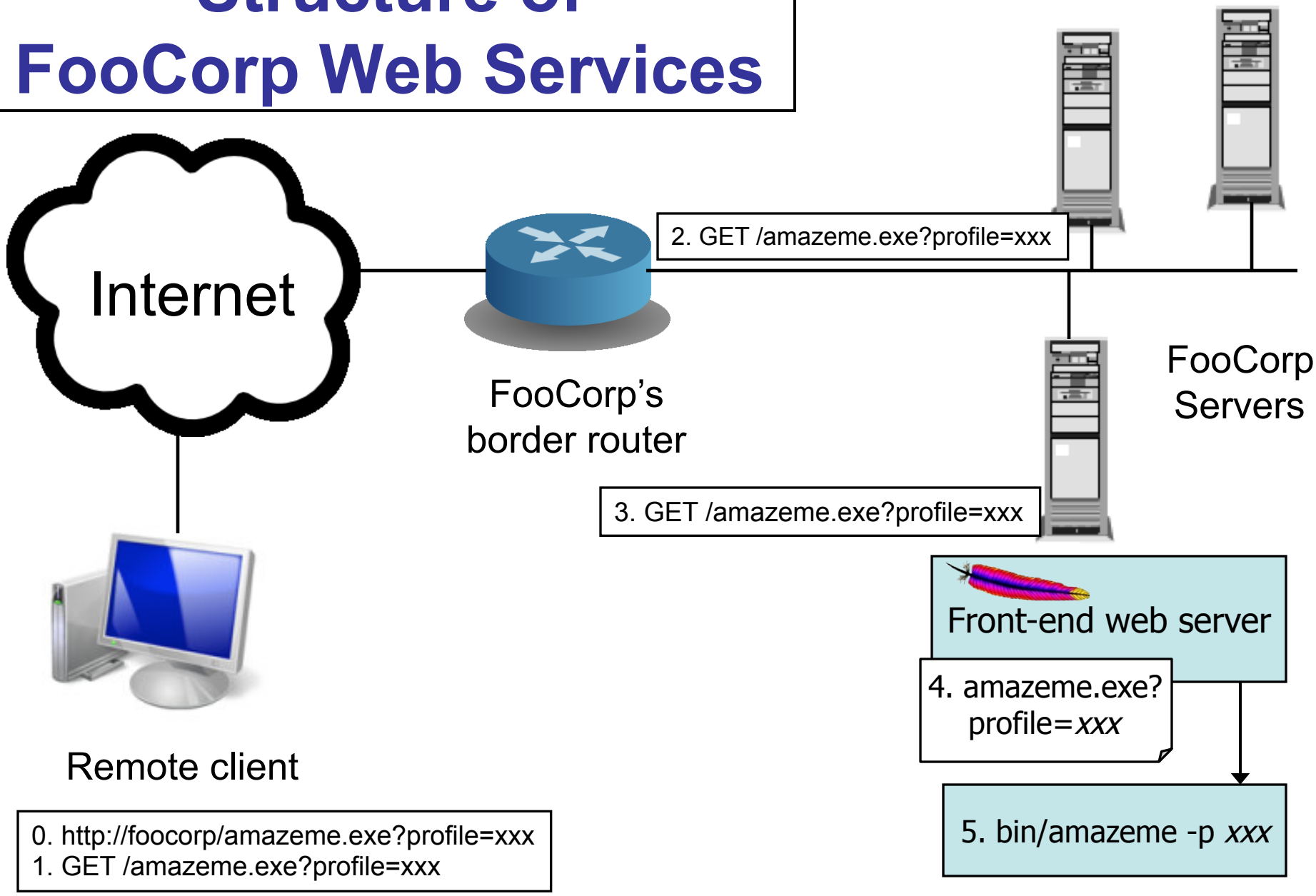
# The Problem of Detecting Attacks

- Given a choice, we'd like our systems to be airtight-secure
- But often we don't have that choice
  - #1 reason why not: **cost** (in different dimensions)
- A (messy) alternative: detect misuse rather than build a system that can't be misused
  - Upon detection: clean up damage, maybe block incipient "*intrusion*"
  - Note: prudent for us to do this even if we think system is solid - defense in depth
  - Note: "misuse" might be about policy rather than security
    - E.g. your own employees shouldn't be using file-sharing apps
- Problem space:
  - *Lacks principles*
  - Has many dimensions (where to monitor, how to look for problems, how much accuracy required, what can attackers due to elude us)
  - Is messy and in practice also very useful

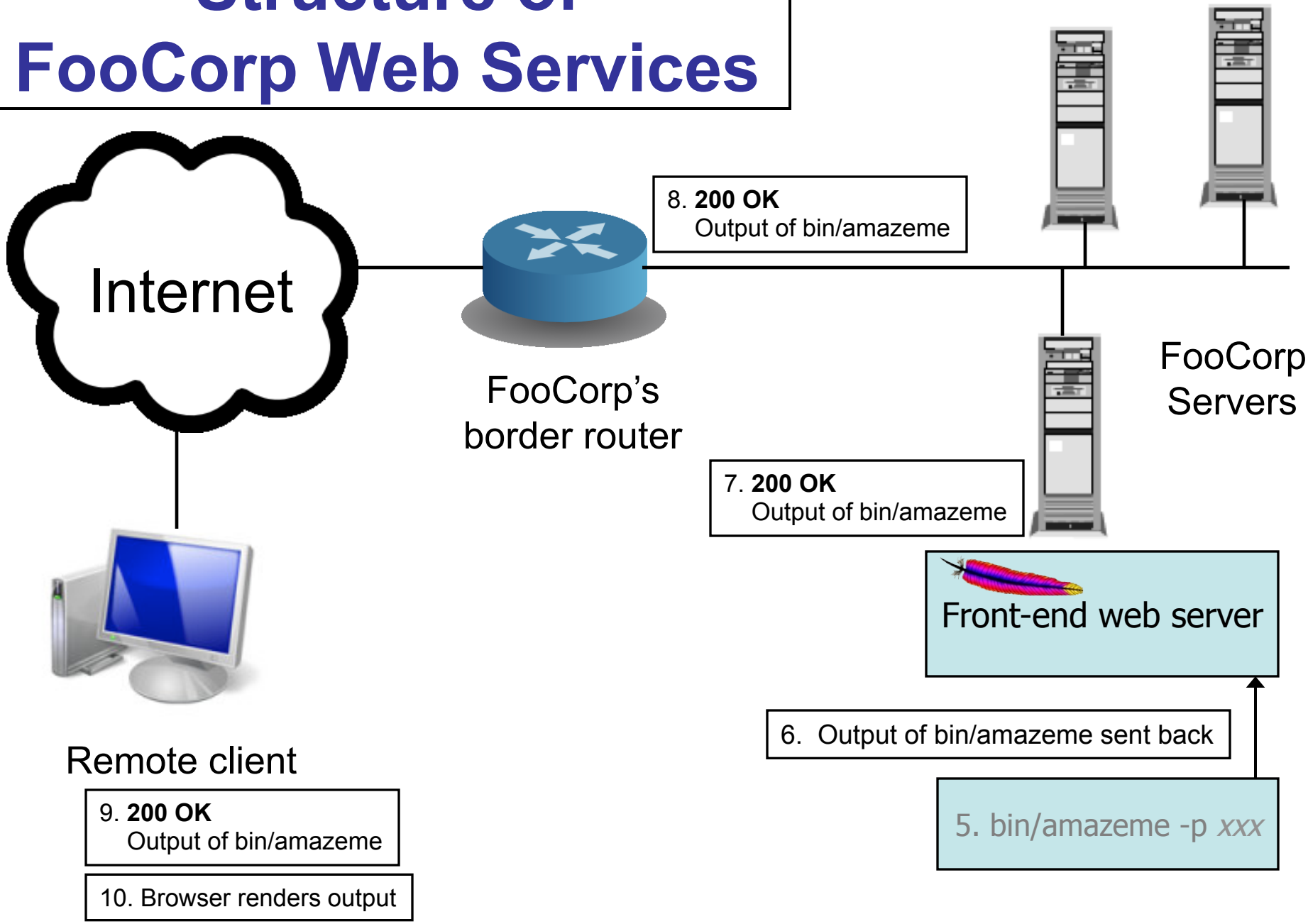
# Example Scenario

- Suppose you've been hired to provide computer security for FooCorp. They offer web-based services via backend programs invoked via URLs:
  - http://foocorp.com/amazeme.exe?profile=info/luser.txt
  - Script makes sure that "profile" arg. is a relative filename

# Structure of FooCorp Web Services



# Structure of FooCorp Web Services



# Example Scenario

- Suppose you've been hired to provide computer security for FooCorp. They offer web-based services via backend programs invoked via URLs:
  - http://foocorp.com/amazeme.exe?profile=info/luser.txt
  - Script makes sure that "profile" arg. is a relative filename
- Due to installed base issues, you can't alter backend components like `amazeme.exe`
- One of the zillion of attacks you're worried about is information leakage via *directory traversal*:
  - E.g. GET /amazeme.exe?profile=../../../../../etc/passwd

# Problem with accessing the AmazeMe Foocorp service

*Error parsing profile: ../../../../../etc/passwd*
*Can't find foreground/background color preferences in:*

---

root:fo8bXK3L6xI:0:0:Administrator:/:/bin/sh
flash:pR.33HwJa2c:51:51:Flash User:/flash:/bin/false
nobody:*:99:99:Nobody:/:
jluser:lT9q23cjwVs:500:503:Jerome L. User:/home/jlusr:/bin/tcsh
hefalump:bKKdz92sk1b:501:503:Mr. Hef:/home/hef:/bin/bash
backdoor:9aBz331dDe1:0:0:Emergency Access:/:/bin/sh
ncsd:$1GnYOsA552:505:505:NSCD Daemon:/ncsd:/sbin/nologin

---

*Please correct the profile entries and resubmit.*

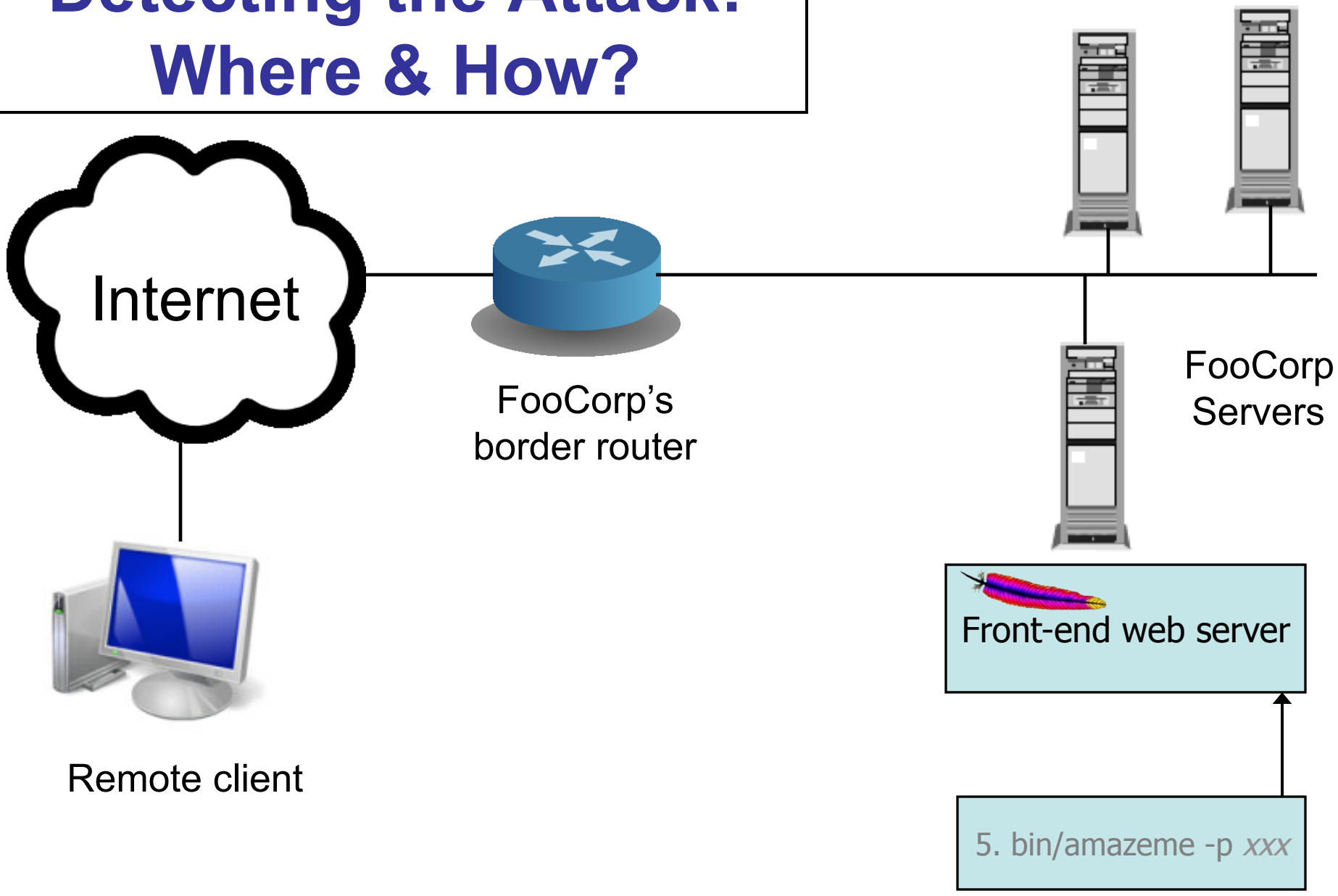
**Thank you for using FooCorp.**

Helpful error message returns contents of profile that appeared mis-formed, revealing the raw password file

# Example Scenario

- Suppose you've been hired to provide computer security for FooCorp. They offer web-based services via backend programs invoked via URLs:
  - http://foocorp.com/amazeme.exe?profile=info/luser.txt
  - Script makes sure that "profile" arg. is a relative filename
- Due to installed base issues, you can't alter backend components like `amazeme.exe`
- One of the zillion of attacks you're worried about is information leakage via *directory traversal*:
  - E.g. GET /amazeme.exe?profile=../../../../../etc/passwd
- What different approaches could detect this attack?

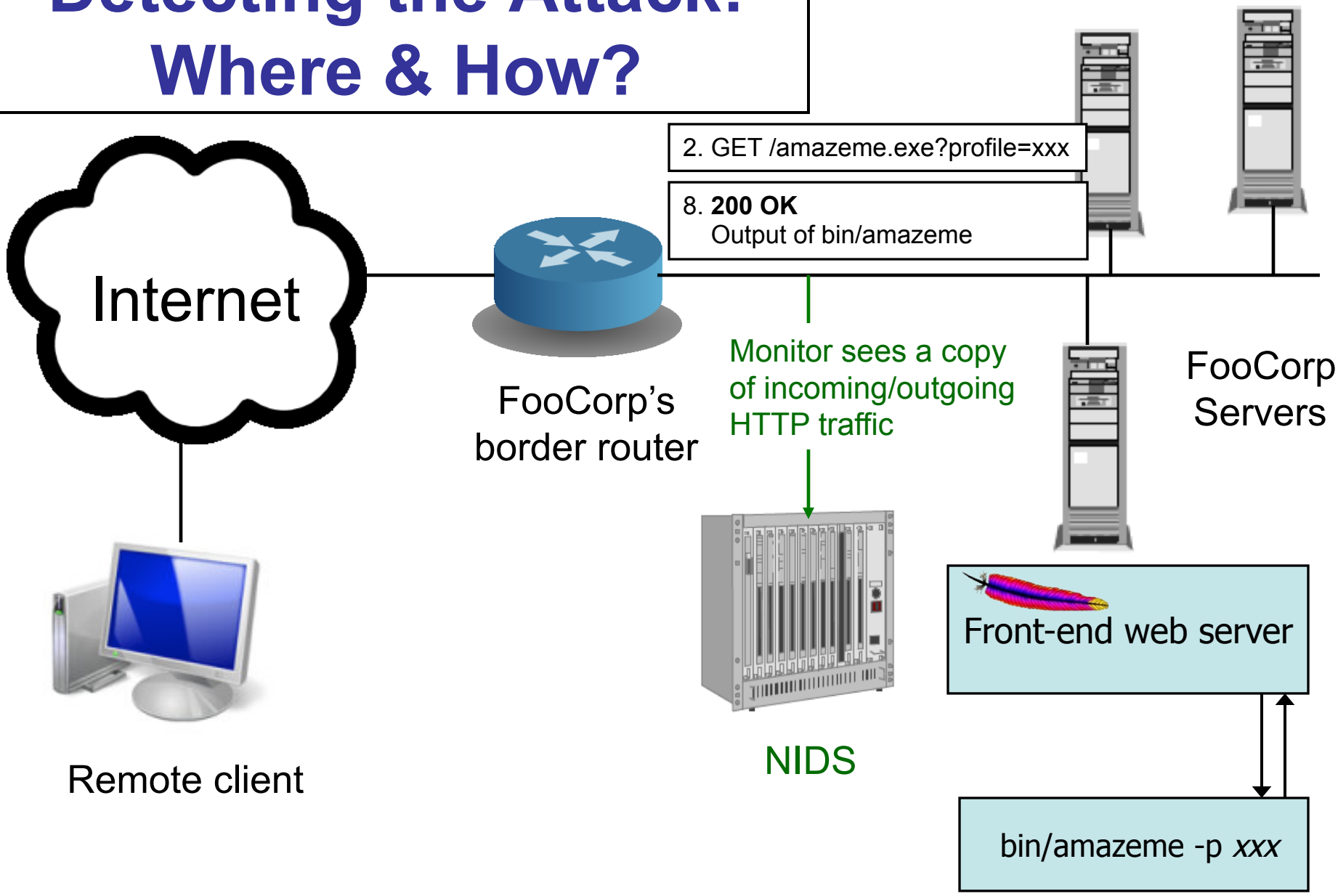
# Detecting the Attack: Where & How?



# Detecting the Attack: Where & How?

- Devise an *intrusion detection system*
  - An IDS: "eye-dee-ess"
- Approach #1: look at the network traffic
  - (a "NIDS": rhymes with "kids")
  - Scan HTTP requests
  - Look for "/etc/passwd" and/or "../../"

# Detecting the Attack: Where & How?



# Detecting the Attack: Where & How?

- Devise an *intrusion detection system*
  - An IDS: "eye-dee-ess"
- Approach #1: look at the network traffic
  - (a "NIDS": rhymes with "kids")
  - Scan HTTP requests
  - Look for "/etc/passwd" and/or "../../"
- Pros:
  - No need to touch or *trust* end systems
    - Can "bolt on" security
  - Cheap: cover many systems w/ single monitor
  - Cheap: centralized management

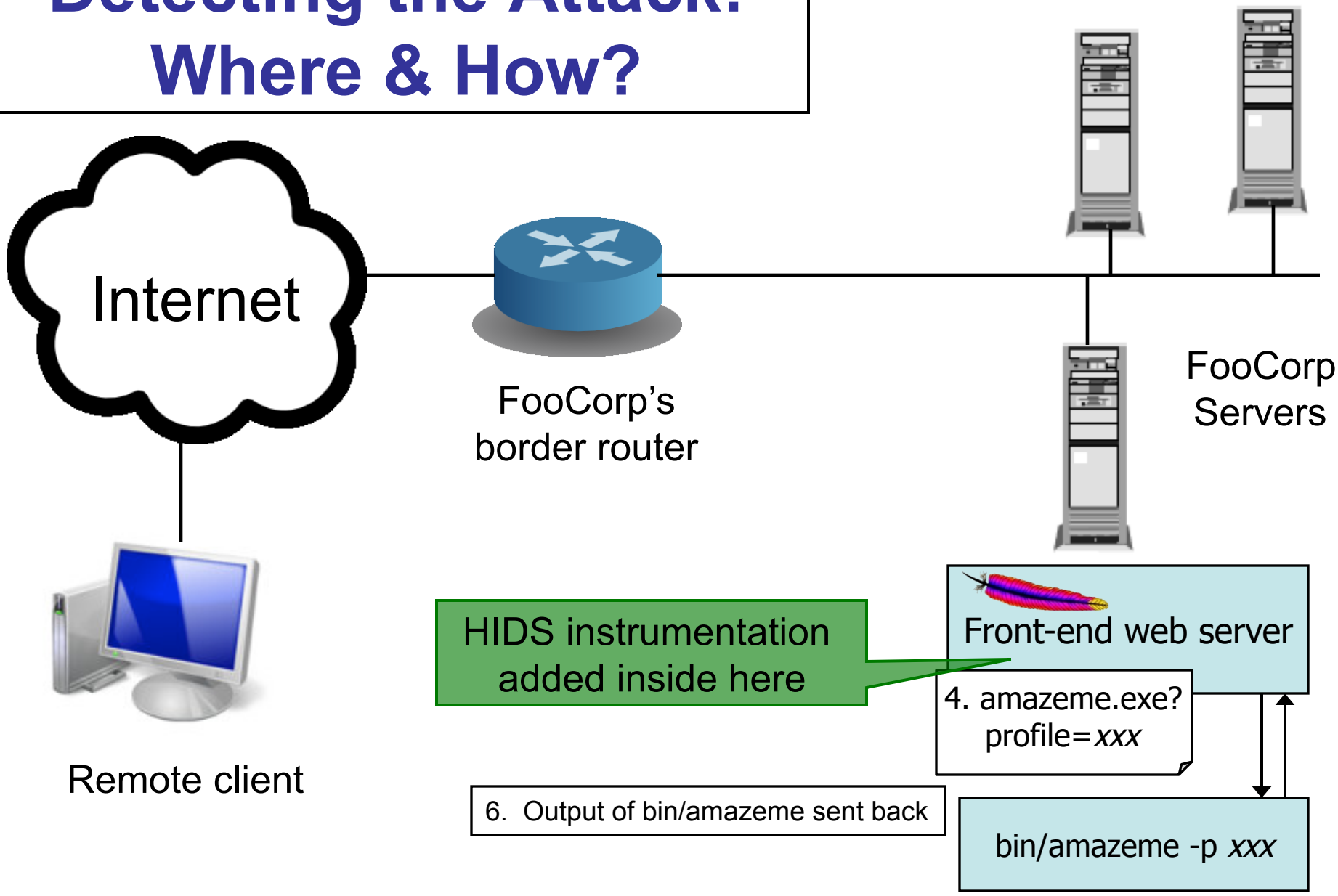
# Network-Based Detection

- Issues?
  - Scan for "/etc/passwd"?
    - What about *other* sensitive files?
  - Scan for "../../"?
    - Sometimes seen in legit. requests (= *false positive*)
    - What about "%2e%2e%2f%2e%2e%2f"? (= *evasion*)
      - Okay, need to do full HTTP parsing
    - What about "..///.///..////"?
      - Okay, need to understand Unix filename semantics too!
  - What if it's HTTPS and not HTTP?
    - Need access to decrypted text / session key - **yuck!**

# Detecting the Attack, con't

- Approach #2: instrument the web server
  - Host-based IDS (sometimes called "HIDS")
  - Scan ?arguments sent to back-end programs
    - Look for "/etc/passwd" and/or "../../"

# Detecting the Attack: Where & How?



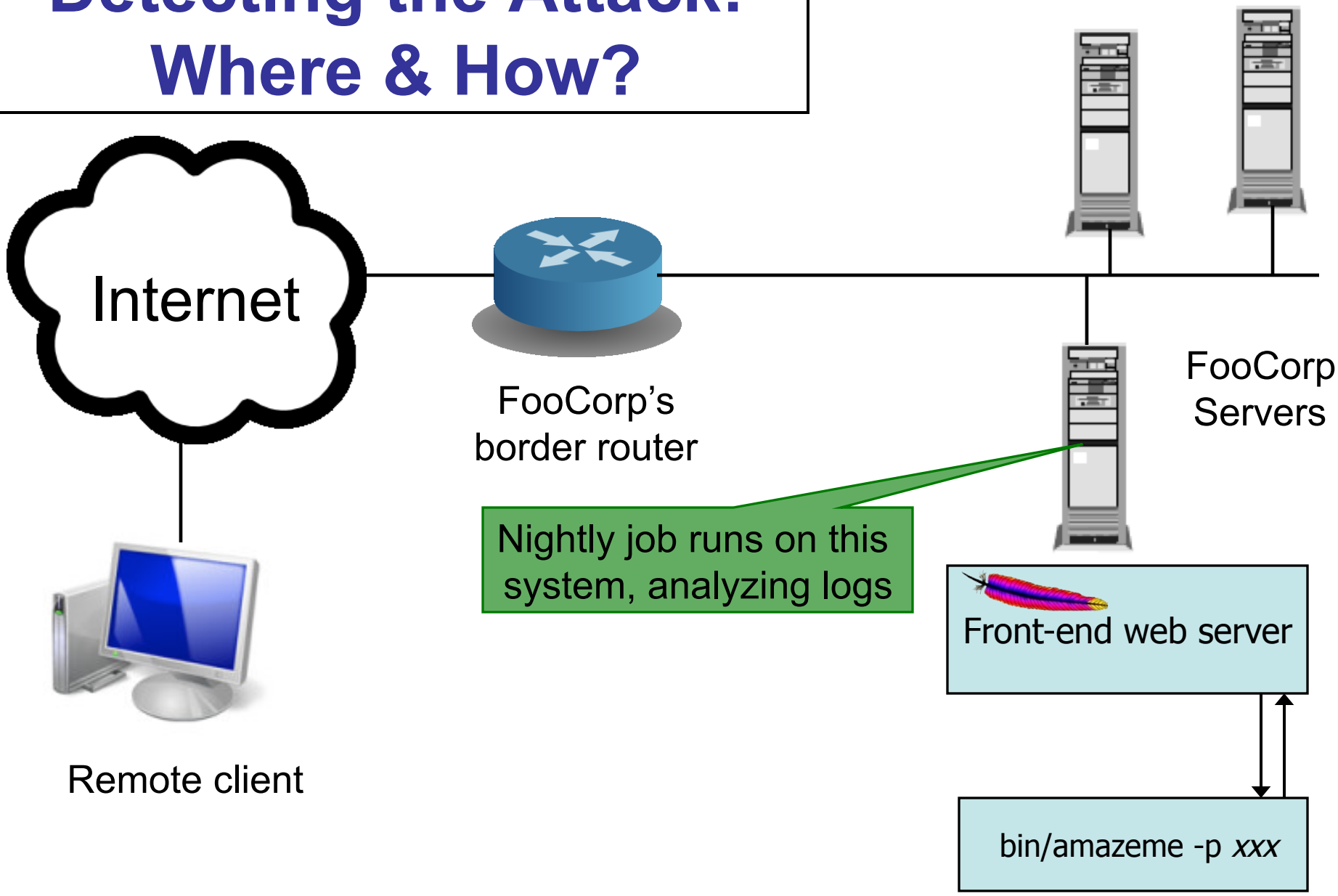
# Detecting the Attack, con't

- Approach #2: instrument the web server
  - Host-based IDS (sometimes called "HIDS")
  - Scan ?arguments sent to back-end programs
    - Look for "/etc/passwd" and/or "../../"
- Pros:
  - No problems with HTTP complexities like %-escapes
  - Works for encrypted HTTPS!
- Issues?
  - Have to add code to each (possibly different) web server
    - And that effort only helps with detecting web server attacks
  - Still have to consider Unix filename semantics ("..////.//")
  - Still have to consider other sensitive files

# Detecting the Attack, con't

- Approach #3: each night, script runs to analyze log files generated by web servers
  - Again scan ?arguments sent to back-end programs

# Detecting the Attack: Where & How?



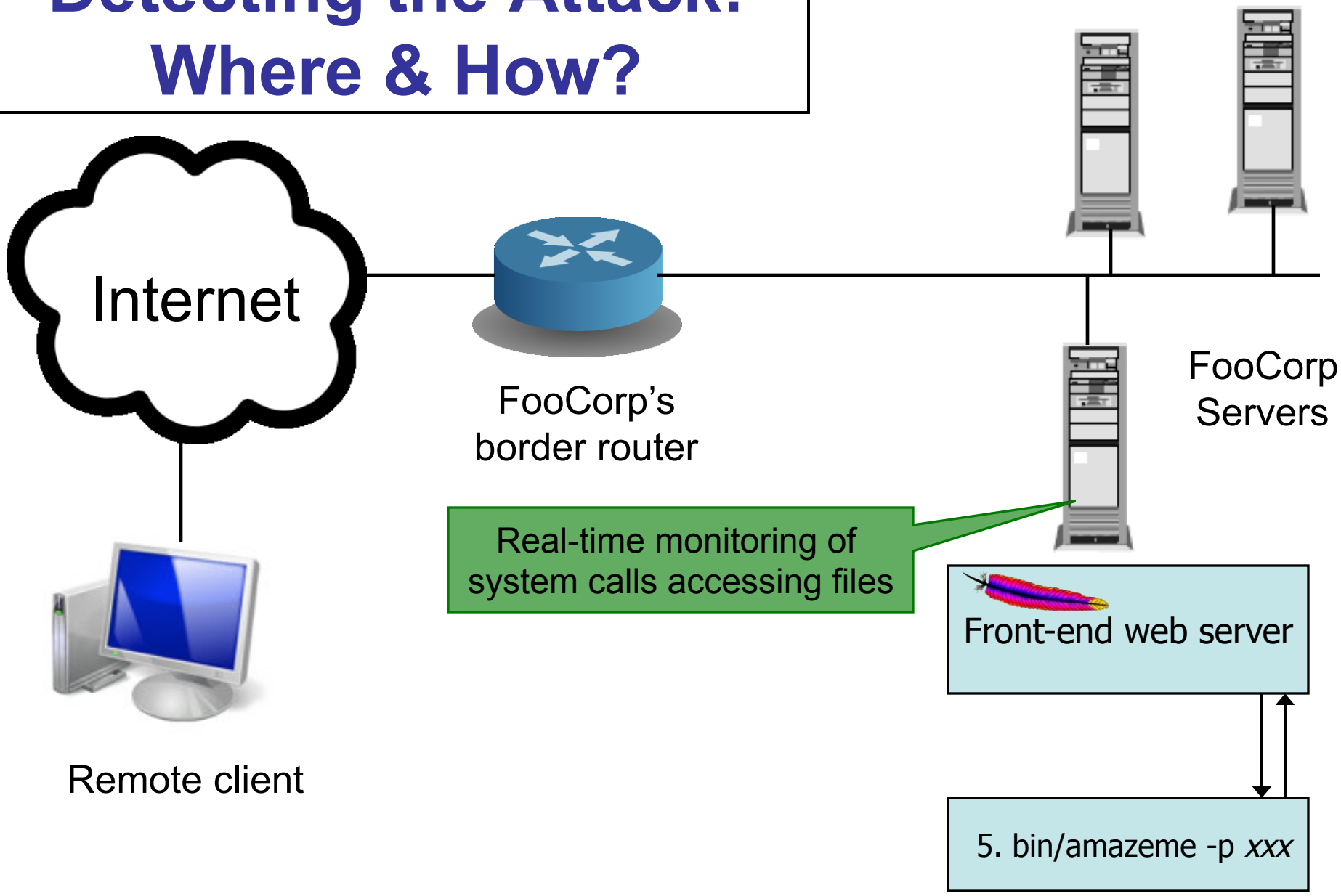
# Detecting the Attack, con't

- Approach #3: each night, script runs to analyze log files generated by web servers
  - Again scan ?arguments sent to back-end programs
- Pros:
  - Cheap: web servers generally already have such logging facilities built into them
    - Can "bolt on" security
  - No problems like %-escapes, encrypted HTTPS
- Issues?
  - Again must consider filename tricks, other sensitive files
  - Can't block attacks & prevent from happening
  - Detection delayed, so attack damage may **compound**
  - If the attack is a compromise, then malware might be able to alter the logs before they're analyzed
    - (Not a problem for directory traversal information leak example)

# Detecting the Attack, con't

- Approach #4: monitor system call activity of backend processes
  - Look for access to /etc/passwd

# Detecting the Attack: Where & How?



# Detecting the Attack, con't

- Approach #4: monitor system call activity of backend processes
  - Look for access to /etc/passwd
- Pros:
  - No issues with any HTTP complexities
  - Can avoid issues with filename tricks
  - Attack only leads to an "alert" if attack succeeded
    - Sensitive file was indeed accessed
- Issues?
  - Might have to analyze a huge amount of data
  - Maybe other processes make legit accesses to the sensitive files (*false positives*)
  - Maybe we'd like to detect attempts even if they fail?
    - "situational awareness"

# Detecting the Attack, con't

- *Only generates an "alert" if the attack succeeded*
  - How does this work for other approaches?
- Instrumenting web server:
  - Need to inspect `bin/amazeme`'s output
  - What do we look for?
    - Can't just assume failure = empty output from `bin/amazeme` …

# Problem with accessing the AmazeMe Foocorp service

*Error parsing profile: ../../../../../etc/passwd*
*Can't find foreground/background color preferences.*

*Please correct the profile entries and resubmit.*

***Thank you for using FooCorp.***

With this version of the *Not Found* page, the attack fails, but there's still a full-fledged web page. All that indicates failure is the lack of the contents of the password file

# Detecting the Attack, con't

- *Only generates an "alert" if the attack succeeded*
  - How does this work for other approaches?
- Instrumenting web server:
  - Need to inspect `bin/amazeme`'s output
  - What do we look for?
    - Can't just assume failure = empty output from `bin/amazeme` …
- Monitoring log files
  - Same, but only works if servers log details about output they generate
- Network-based
  - Same, but have to worry about encoding issues
    - E.g., what if server reply is gzip-compressed?

# NIDS vs. HIDS

- NIDS benefits:
  - Can cover a lot of systems with single deployment
    - Much simpler management
  - Easy to "bolt on" / no need to touch end systems
  - Doesn't consume production resources on end systems
  - Harder for an attacker to subvert / less to trust
- HIDS benefits:
  - Can have direct access to semantics of activity
    - Better positioned to block (prevent) attacks
    - Harder to evade
  - Can protect against non-network threats
  - Visibility into encrypted activity
  - Performance scales much more readily (no chokepoint)
    - No issues with "dropped" packets

# 5 Minute Break

Questions Before We Proceed?

# An Alternative Paradigm

- Idea: rather than detect attacks, launch them yourself!
- Vulnerability scanning: use a tool to probe your own systems with a wide range of attacks, fix any that succeed
- Pros?
  - Accurate: if your scanning tool is good, it finds real problems
  - Proactive: can prevent future misuse
  - Intelligence: can ignore later IDS alarms that you know can't succeed
- Issues?
  - Can take a lot of work
  - Not so helpful for systems you can't modify
  - Dangerous for disruptive attacks
    - And you might not know which these are …
- In practice, this approach is prudent and widely used today
  - Good complement to also running an IDS

# Detection Accuracy

- Two types of detector errors:
  - False positive (FP): alerting about a problem when in fact there was no problem
  - False negative (FN): failing to alert about a problem when in fact there was a problem

- Detector accuracy is often assessed in terms of rates at which these occur

# Detection Accuracy, con't

- Define:
  - I to be the event of an instance of intrusive behavior occurring (something we want to detect)
  - A to be the event of detector generating an alert
  - *False positive rate* = P[A | ¬ I]
    - "How often do we misclassify benign activity?"
  - *False negative rate* = P[¬A | I]
    - "How often do we misclassify malicious activity?"
- Another common framework (ML-based classifiers):
  - *Precision* = P[I | A]
    - "If we get an alert, how often is it relevant?"
    - *Varies with proportion of attacks-vs-non-attacks*
  - *Recall* = P[A | I]
    - "How often do we get alerts when we would expect to?
      = 1 - *False negative rate* *(= True positive rate)*

# Perfect Detection

- Is it possible to build a detector for our example with a false negative rate of 0%?
- Algorithm to detect bad URLs with 0% FN rate:

```
void my_detector_that_never_misses(char *URL)
  {
      printf("yep, it's an attack!\n");
  }
```

  - In fact, it works for detecting **any** bad activity with no false negatives! Woo-hoo!

- Wow, so what about a detector for bad URLs that has NO FALSE POSITIVES?!
  - `printf("nope, not an attack\n");`

# Detection Tradeoffs

- The art of a good detector is achieving an effective balance between FPs and FNs
- Suppose our detector has an FP rate of 0.1% and an FN rate of 2%. Is it good enough? Which is better, a very low FP rate or a very low FN rate?
  - Depends on the **cost** of each type of error …
    - E.g., FP might lead to paging a duty officer and consuming hour of their time; FN might lead to $10K cleaning up compromised system that was missed
  - … but also **critically** depends on the **rate** at which actual attacks occur in your environment

# Base Rate Fallacy

- Suppose our detector has a FP rate of 0.1% (!) and a FN rate of 2% (not bad!)
- Scenario #1: our server receives 1,000 URLs/day, and 5 of them are attacks
  - Expected # FPs each day = 0.1% * 995 ≈ 1
  - Expected # FNs each day = 2% * 5 = 0.1 (< 1/week)
  - Pretty good!
- Scenario #2: our server receives 10,000,000 URLs/day, and 5 of them are attacks
  - Expected # FPs each day ≈ 10,000 :-(
- *Nothing changed about the detector*; only our environment changed
  - Accurate detection very challenging when base rate of activity we want to detect is quite low

# Same Scenarios, Precision/Recall

- Detector: FP rate = 0.1% (!), FN rate = 2% (not bad!)
- Scenario #1: 1,000 URLs/day, 5 are attacks
  - Expected # FPs each day = 0.1% * 995 ≈ 1
  - Expected # FNs each day = 2% * 5 = 0.1  (< 1/week)
  - Pretty good!
  - Precision = P[I | A] = (0.98 * 5) / (0.98 * 5 + 0.1% * 995) ≈ 83%
    - About 5 out of every 6 alerts are relevant.  Quite good.
  - Recall = P[A | I] = (0.98 * 5) / (0.98 * 5 + 0.02 * 5) = 98%
    - (Equals 1 – FN rate.  We detect nearly all the attacks, cool.)
- Scenario #2: $10^7$ URLs/day, 5 are attacks
  - Expected # FPs each day ≈ 10,000 :-(
  - Precision = P[I | A] = (0.98 * 5) / (0.98 * 5 + 0.1% * ($10^7$-5))
    ≈ 0.05% (only about one alert in 2,000 is relevant – terrible!)
  - Recall = P[A | I] = (0.98 * 5) / (0.98 * 5 + 0.02 * 5) = 98%
    - (doesn't change, since only concerns false-vs-true negatives)

# Detection vs. Blocking

- If we can detect attacks, how about blocking them?
- Issues:
  - Not a possibility for retrospective analysis (e.g., nightly job that looks at logs)
  - Quite hard for detector that's not in the data path
    - E.g. How can NIDS that passively monitors traffic block attacks?
      - Change firewall rules dynamically; forge RST packets
      - There's a race though regarding what attacker does before blocked

# Detection vs. Blocking

- If we can detect attacks, how about blocking them?
- Issues:
  - Not a possibility for retrospective analysis (e.g., nightly job that looks at logs)
  - Quite hard for detector that's not in the data path
    - E.g. How can NIDS that passively monitors traffic block attacks?
      - Change firewall rules dynamically; forge RST packets
      - There's a race though regarding what attacker does before blocked
  - False positives get more expensive
    - You don't just bug an operator, you damage production activity
- Today's technology/products pretty much all offer blocking
  - *Intrusion prevention systems* (IPS - "eye-pee-ess")

# Can We Build An IPS That Blocks *All* Attacks?


**The Ultimately Secure DEEP PACKET INSPECTION AND APPLICATION SECURITY SYSTEM**
**Featuring signature-less anomaly detection and blocking technology with application awareness and layer-7 state tracking!!!**

**Now available in Petabyte-capable appliance form factor!***

**(Formerly: The Ultimately Secure INTRUSION PREVENTION SYSTEM Featuring signature-less anomaly detection and blocking technology!!)**