Detecting Attacks, Part 1

CS 161: Computer Security

Prof. Vern Paxson

TAs: Devdatta Akhawe, Mobin Javed
& Matthias Vallentin

http://inst.eecs.berkeley.edu/~cs161/

April 12, 2011

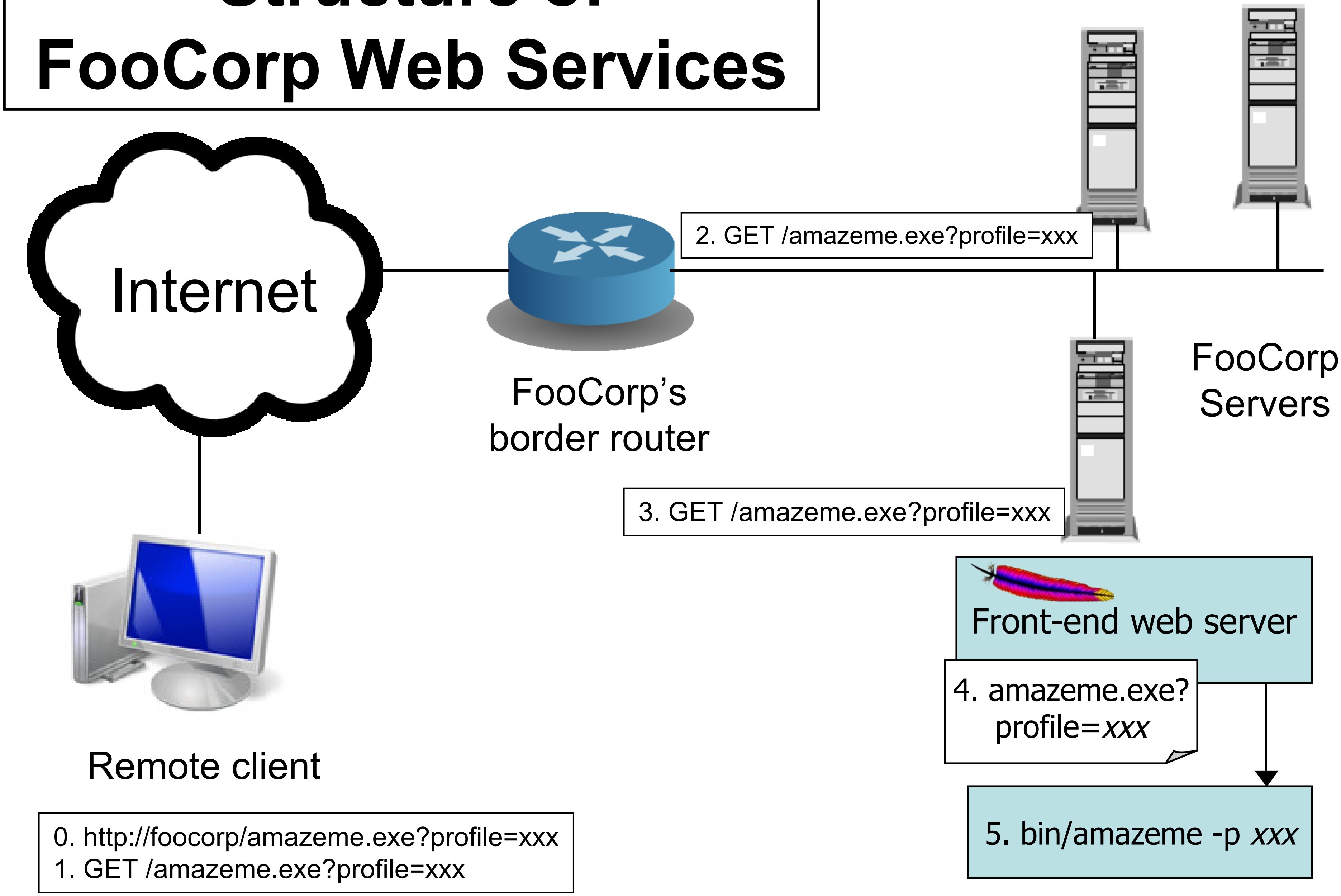
The Problem of Detecting Attacks

- 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: can be prudent for us to do this even if we think system is airtight - 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



Internet

FooCorp's border router

2. GET /amazeme.exe?profile=xxx

FooCorp Servers

3. GET /amazeme.exe?profile=xxx

Front-end web server

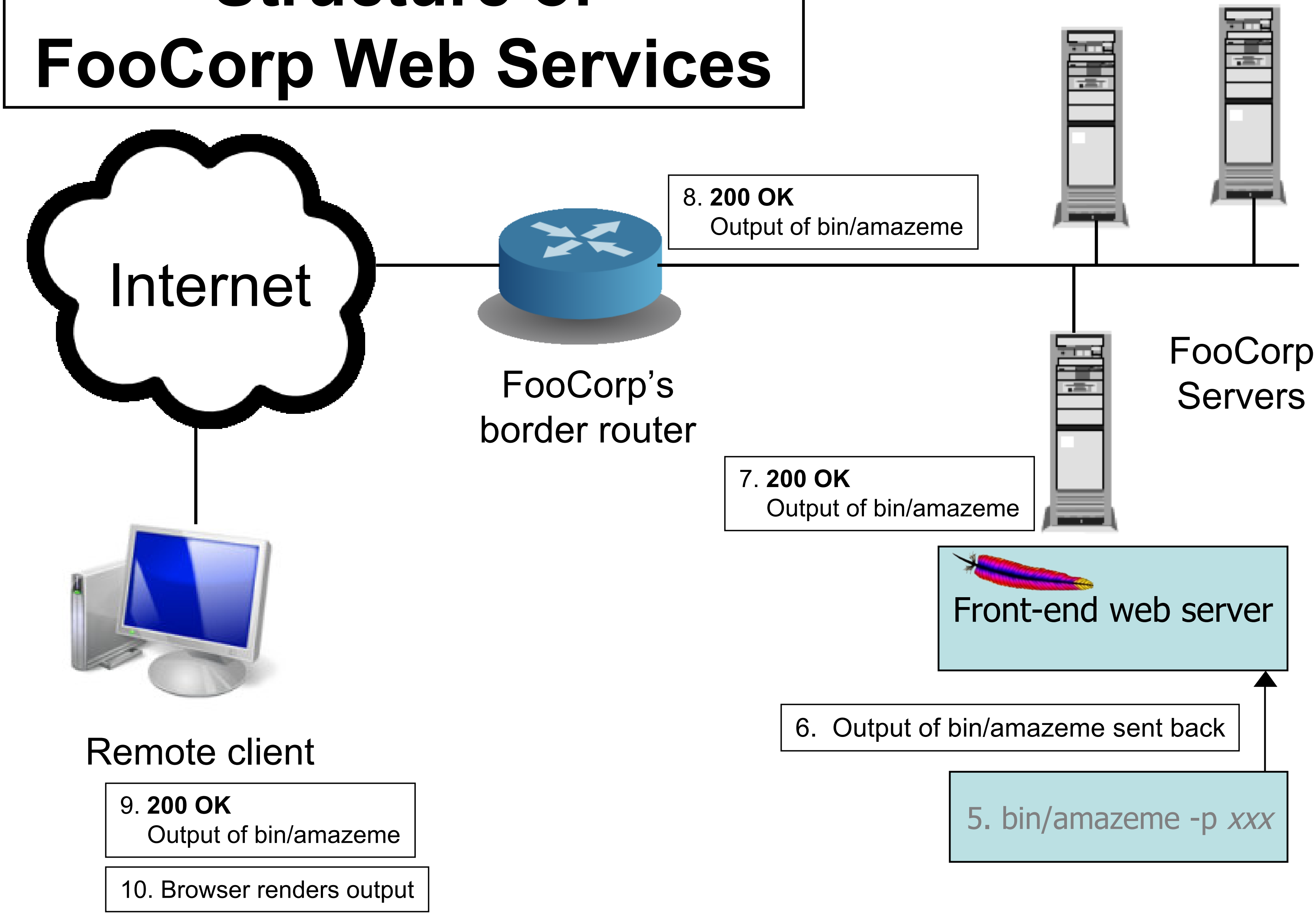
4. amazeme.exe?
profile=*xxx*

5. bin/amazeme -p *xxx*

Remote client

0. http://foocorp/amazeme.exe?profile=xxx
1. GET /amazeme.exe?profile=xxx

Structure of FooCorp Web Services



5. bin/amazeme -p *xxx*

6. Output of bin/amazeme sent back

7. **200 OK**
Output of bin/amazeme

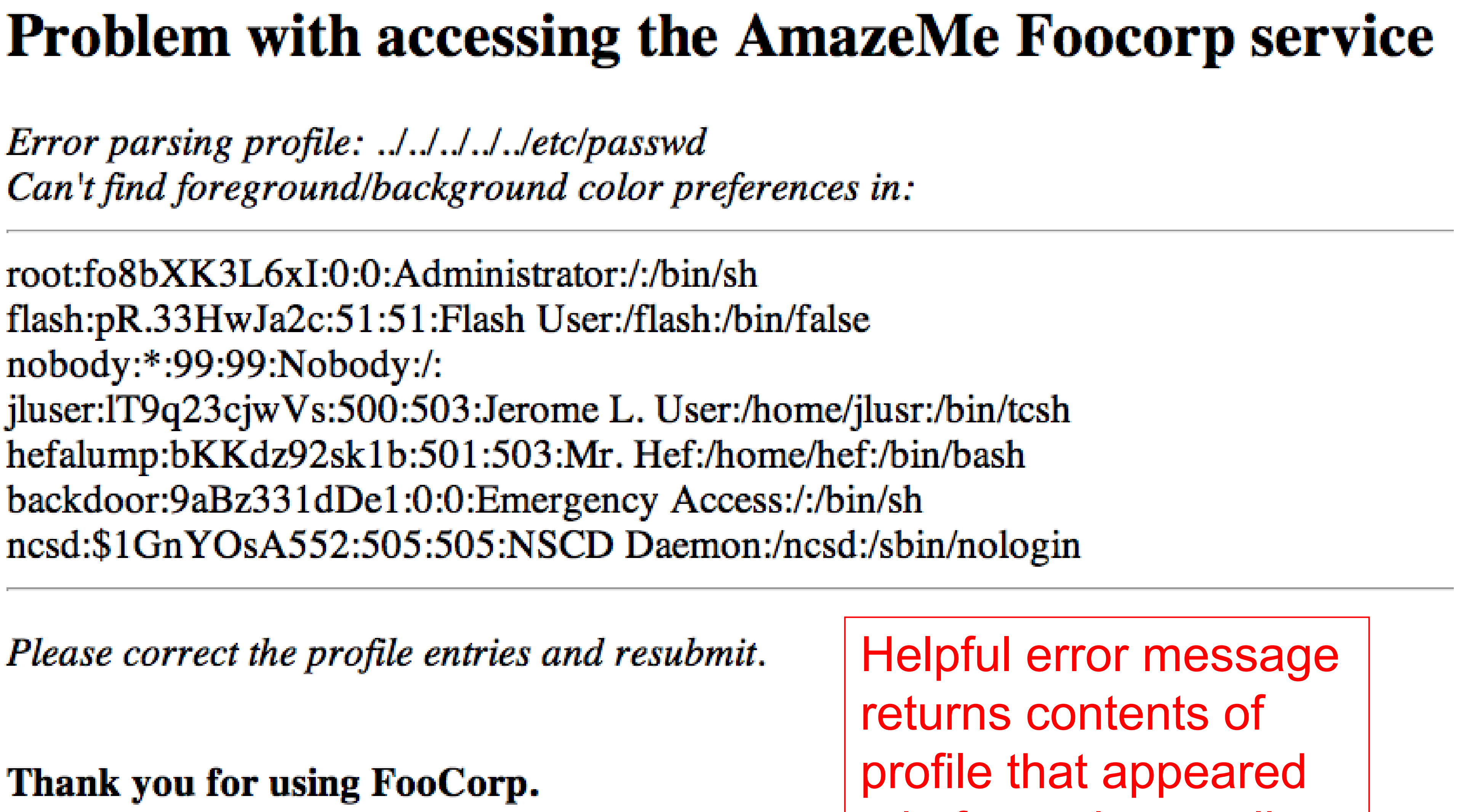
8. **200 OK**
Output of bin/amazeme

9. **200 OK**
Output of bin/amazeme

10. Browser renders output

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

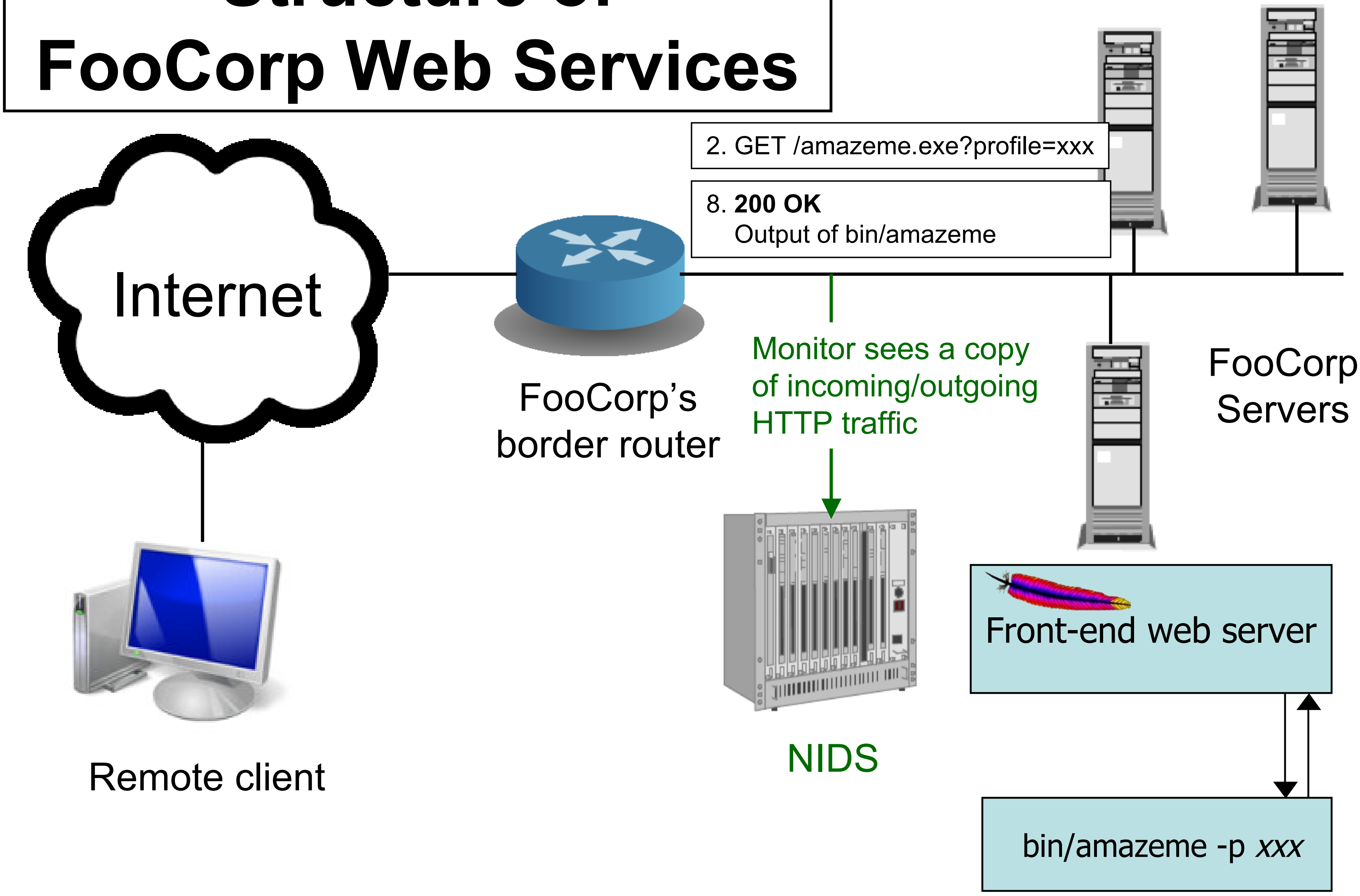
Example Scenario

- 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?

- 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 “../../”

Structure of FooCorp Web Services



Internet

FooCorp's border router

2. GET /amazeme.exe?profile=xxx

8. **200 OK**
Output of bin/amazeme

Monitor sees a copy of incoming/outgoing HTTP traffic

NIDS

FooCorp Servers

Front-end web server

bin/amazeme -p *xxx*

Remote client

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

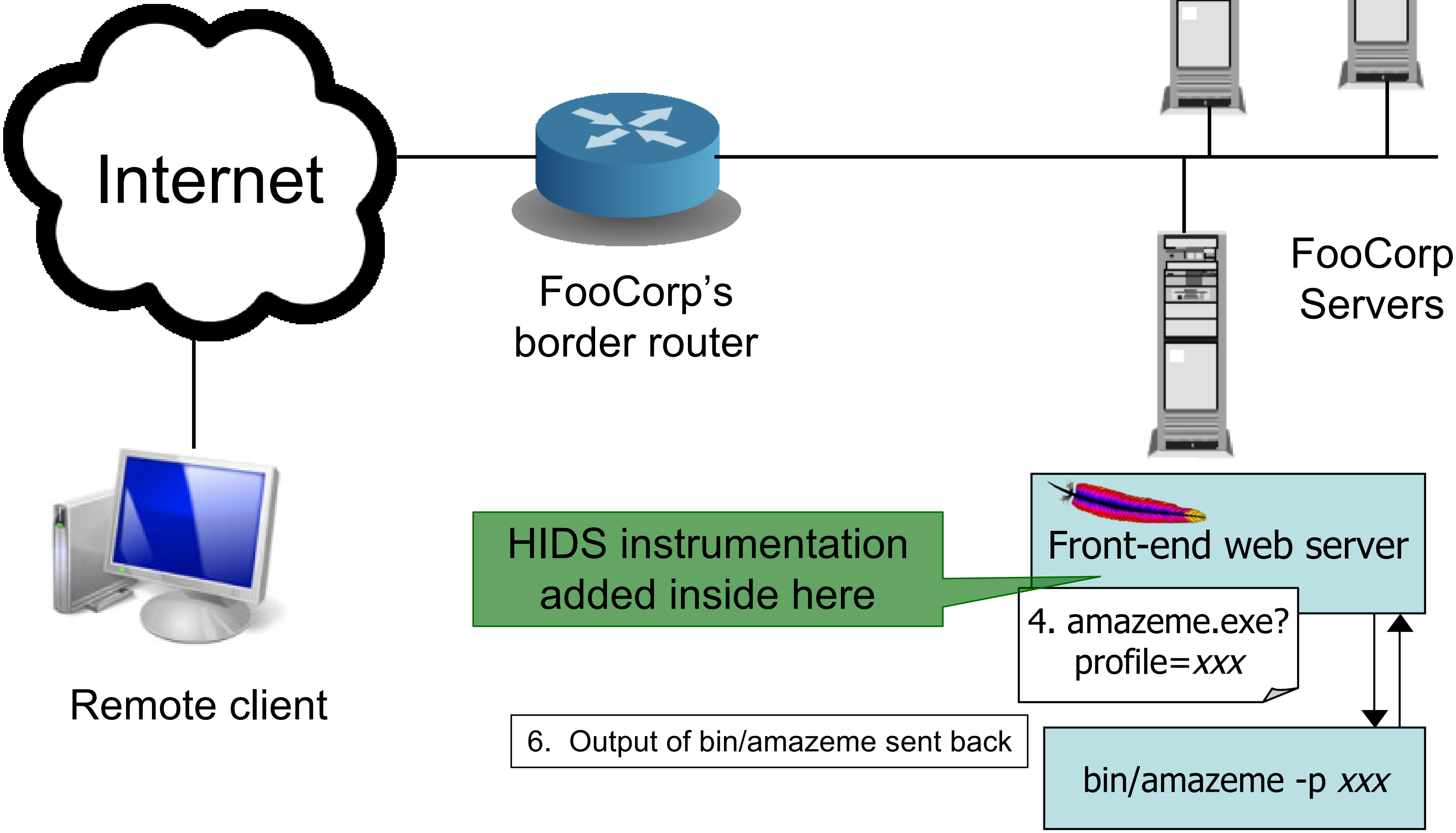
Network-Based Detection

- 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 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 “../../”

Structure of FooCorp Web Services



Internet

FooCorp's border router

FooCorp Servers

Remote client

Front-end web server

HIDS instrumentation added inside here

4. amazeme.exe? profile=*xxx*

bin/amazeme -p *xxx*

6. Output of bin/amazeme sent back

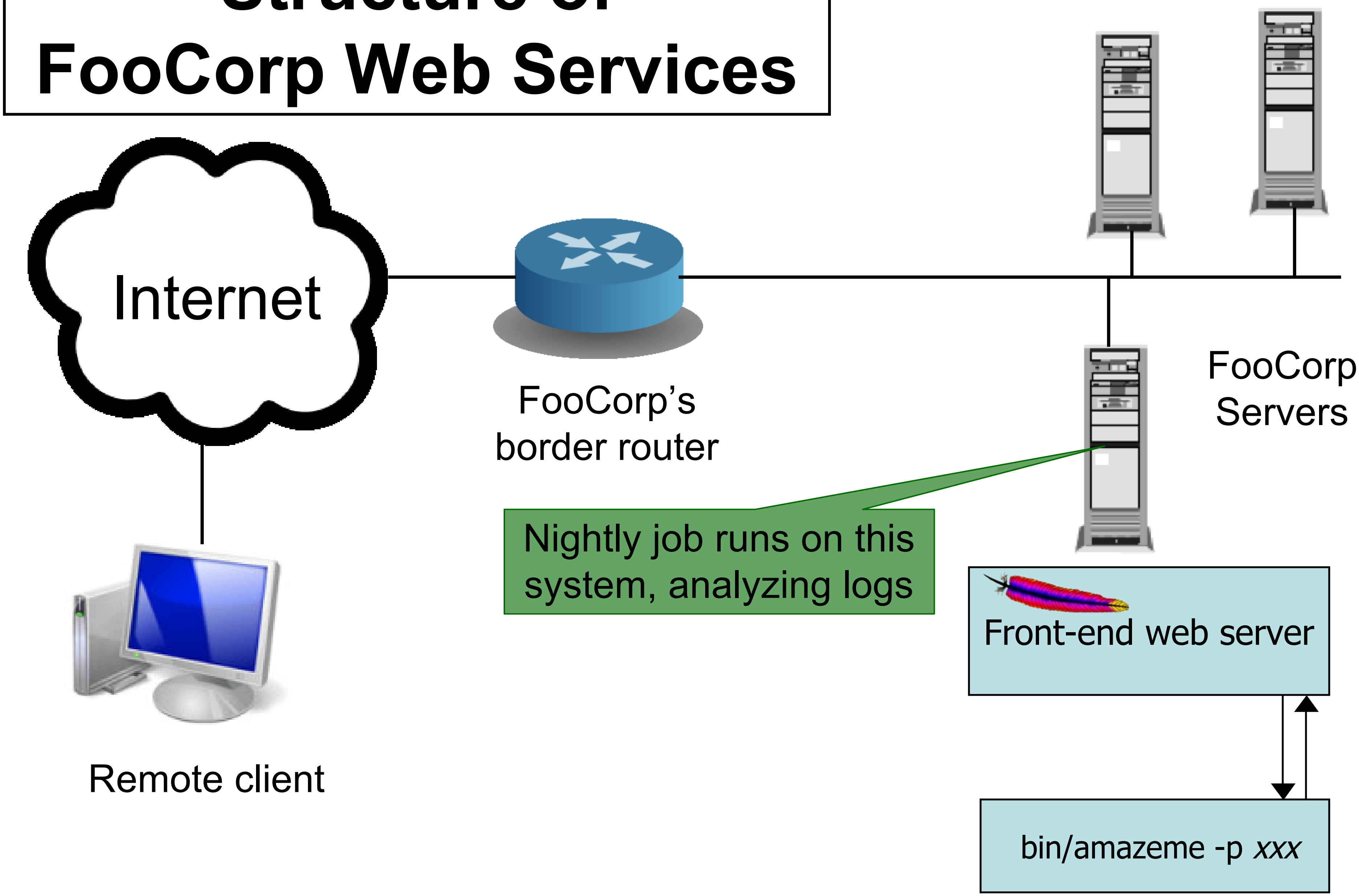
Detecting the Attack, con’t

- 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

Structure of FooCorp Web Services



Internet

FooCorp's border router

FooCorp Servers

Nightly job runs on this system, analyzing logs

Front-end web server

bin/amazeme -p *xxx*

Remote client

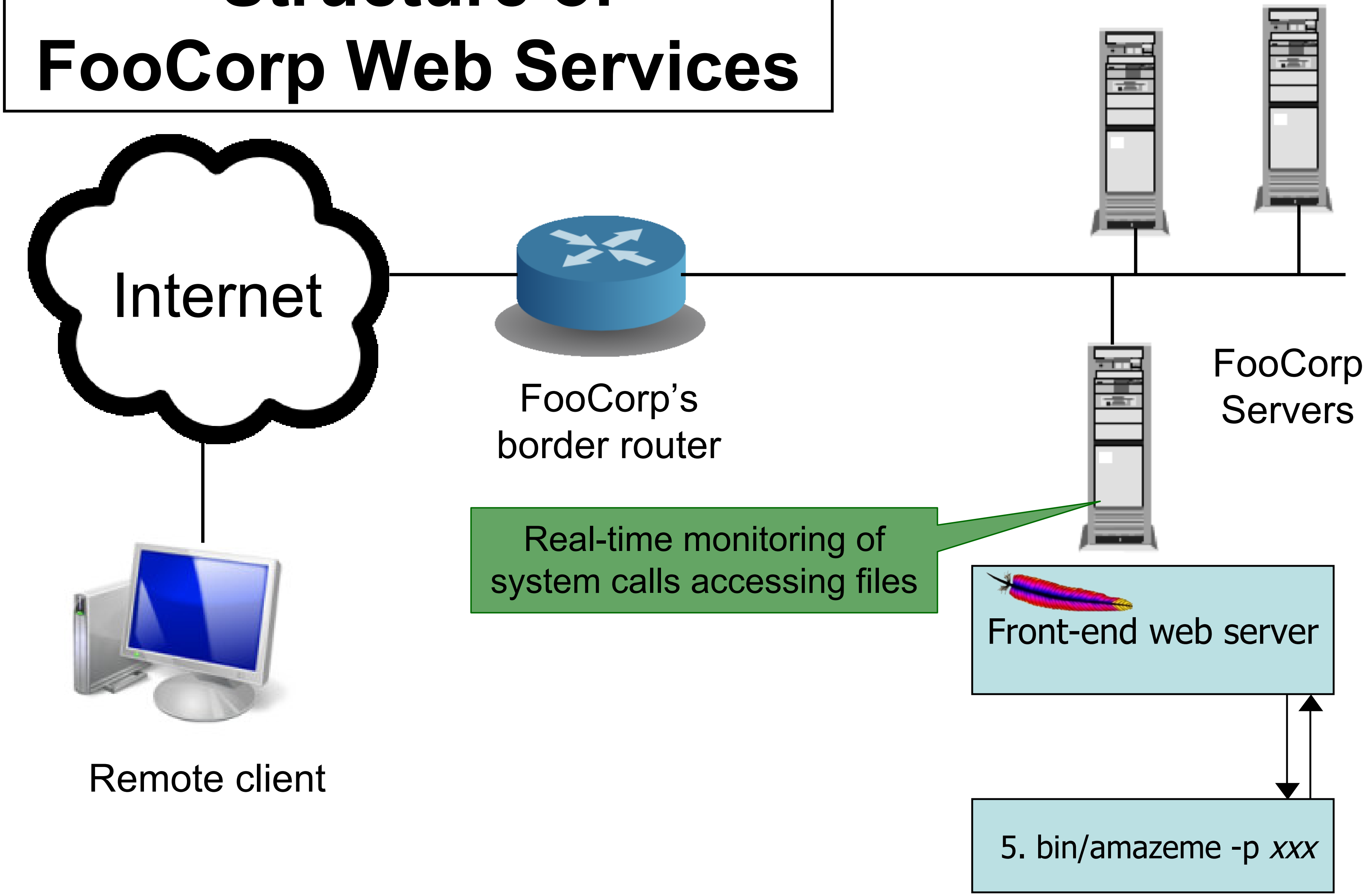
Detecting the Attack, con't

- 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
  - 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

Structure of FooCorp Web Services



Internet

FooCorp's border router

FooCorp Servers

Real-time monitoring of system calls accessing files

Front-end web server

5. bin/amazeme -p *xxx*

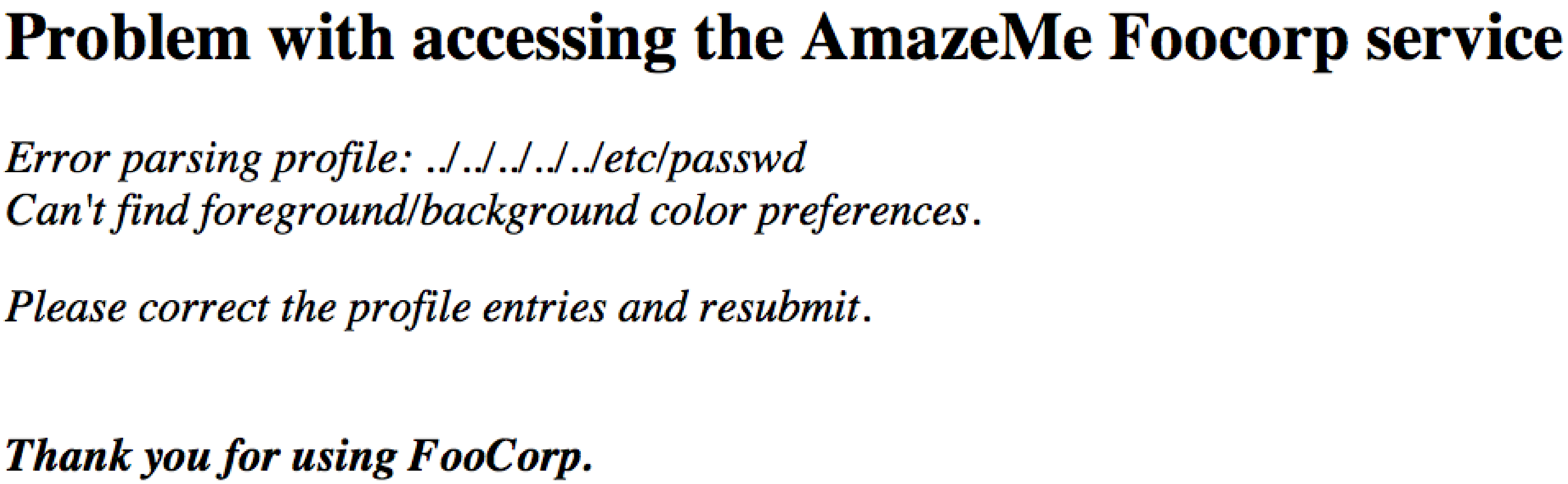
Remote client

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
  - May avoid issues with filename tricks
  - Only generates an "alert" if the 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?

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?
  - Proactive: can prevent future misuse
  - Intelligence: can ignore 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:
  - Define I to be an instance of intrusive behavior (something we want to detect)
  - Define A to be the presence of a detector alarm
- Define:
  - *False positive rate* = $P[A|\neg I]$
  - *False negative rate* = $P[\neg A|I]$

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

Detecting Successful Attacks

- Suppose we're worried about a version of the attack that modifies /etc/passwd rather than retrieves it
  - Say: GET /amazeme.exe?profile=/etc/passwd
    &newcolor=w00t:nlT9q23cjwVs:0:0:/:/bin/bash
- How can we detect if it succeeds?
- Maybe `amazeme.exe` generates specific output if file modified - if so, look for that
- But if not, then NIDS / web server instrumentation / log monitor all have difficulty in telling if succeeded
  - Note: similar problems arise with other successful attacks, such as "did attempted malware infection succeed"?
  - System call monitoring could identify change

Detecting Successful Attacks, con’t

- Alternative approach: periodic process that looks for changes to sensitive files, flags for operator
  - Not based on file modification time, as program can change that
- Instead: verify against a database of say SHA256 hashes
- Problem: what if malware compromised the kernel?
  - Could alter the hashes and/or the content returned when reading a given file
- Fix?
  - One approach:
    - Don’t store hashes on local system; send over net elsewhere
    - To verify, boot separate kernel from read-only media

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
      - And still there’s a race regarding what attacker does before block
  - 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”)