Detecting Attacks

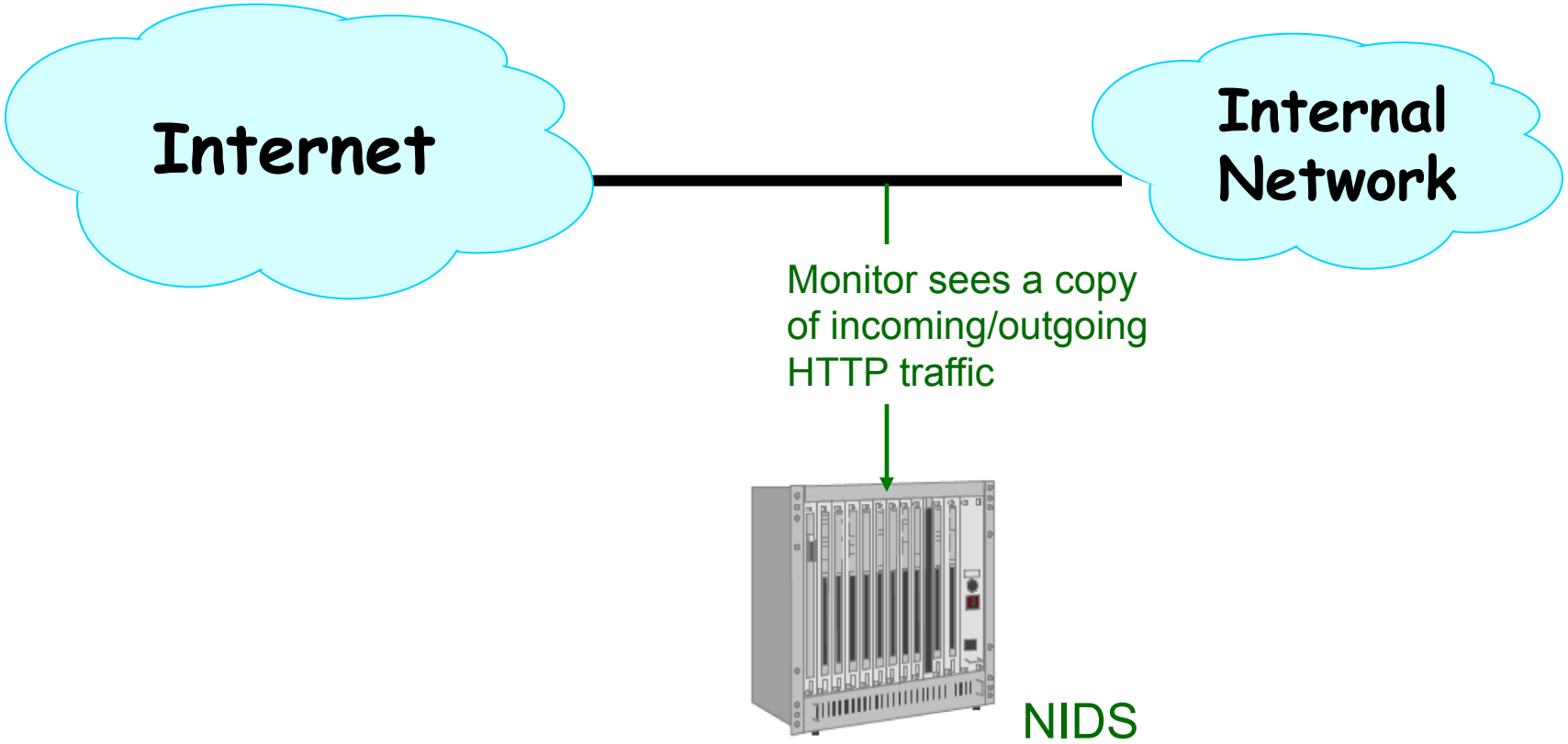
CS 161: Computer Security

Prof. David Wagner

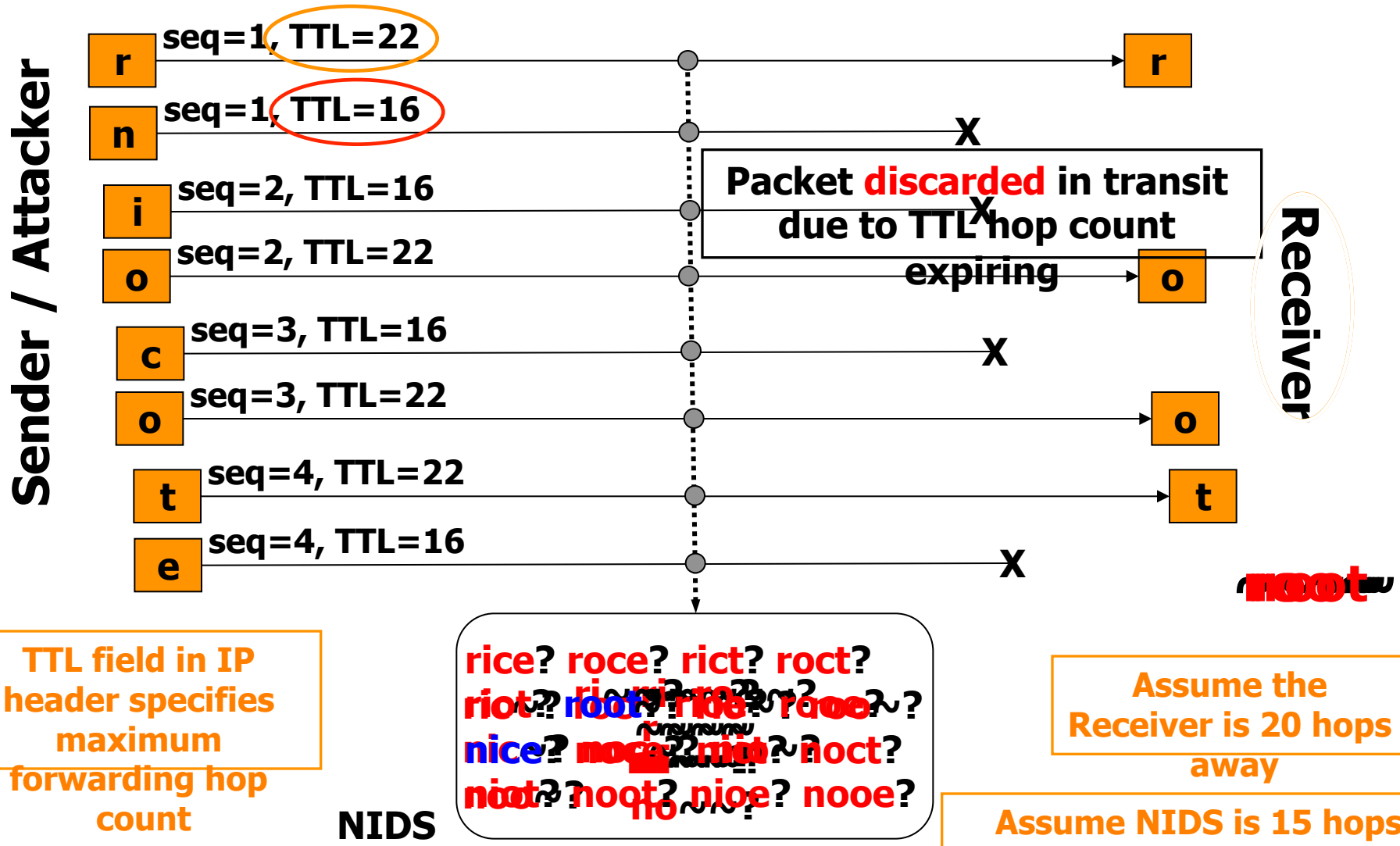
April 6, 2016

Network Intrusion Detection (NIDS)

- Passively monitor network traffic for signs of attack
  - Look for /etc/passwd



Evasion attacks



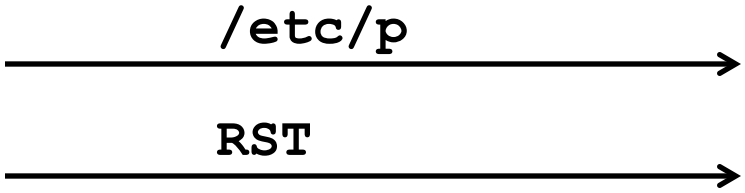
Network Intrusion Detection (NIDS)

- NIDS has a table of all active connections, and maintains state for each
  - e.g., has it seen a partial match of /etc/passwd?

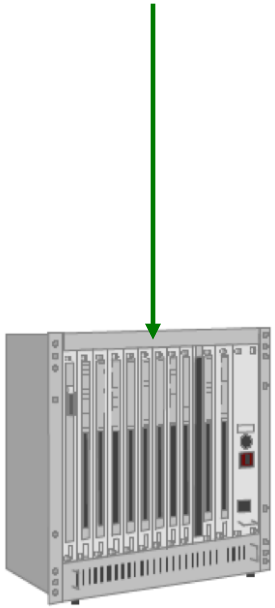
- What do you do when you see a new packet not associated with any known connection?
  - Create a new connection: when NIDS starts it doesn't know what connections might be existing

Evasion

- What should NIDS do if it sees a RST packet?



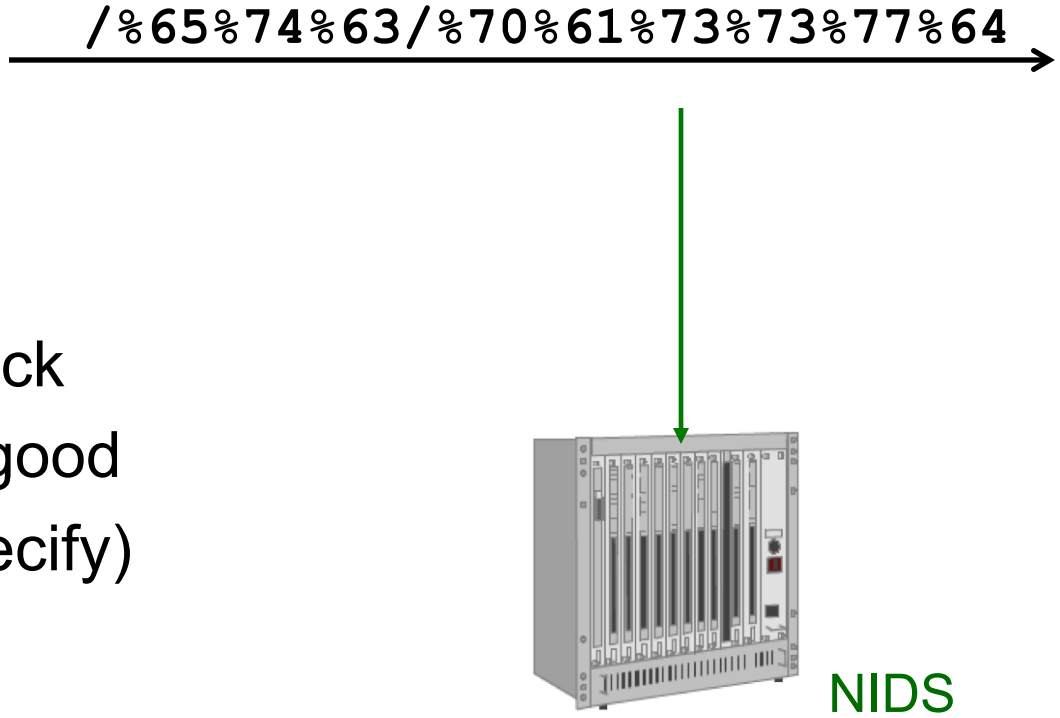
(a) Assume RST will be received
(b) Assume RST won't be received
(c) Other (please specify)



NIDS

Evasion

- What should NIDS do if it sees this?



(a) Alert – it's an attack
(b) No alert – it's all good
(c) Other (please specify)

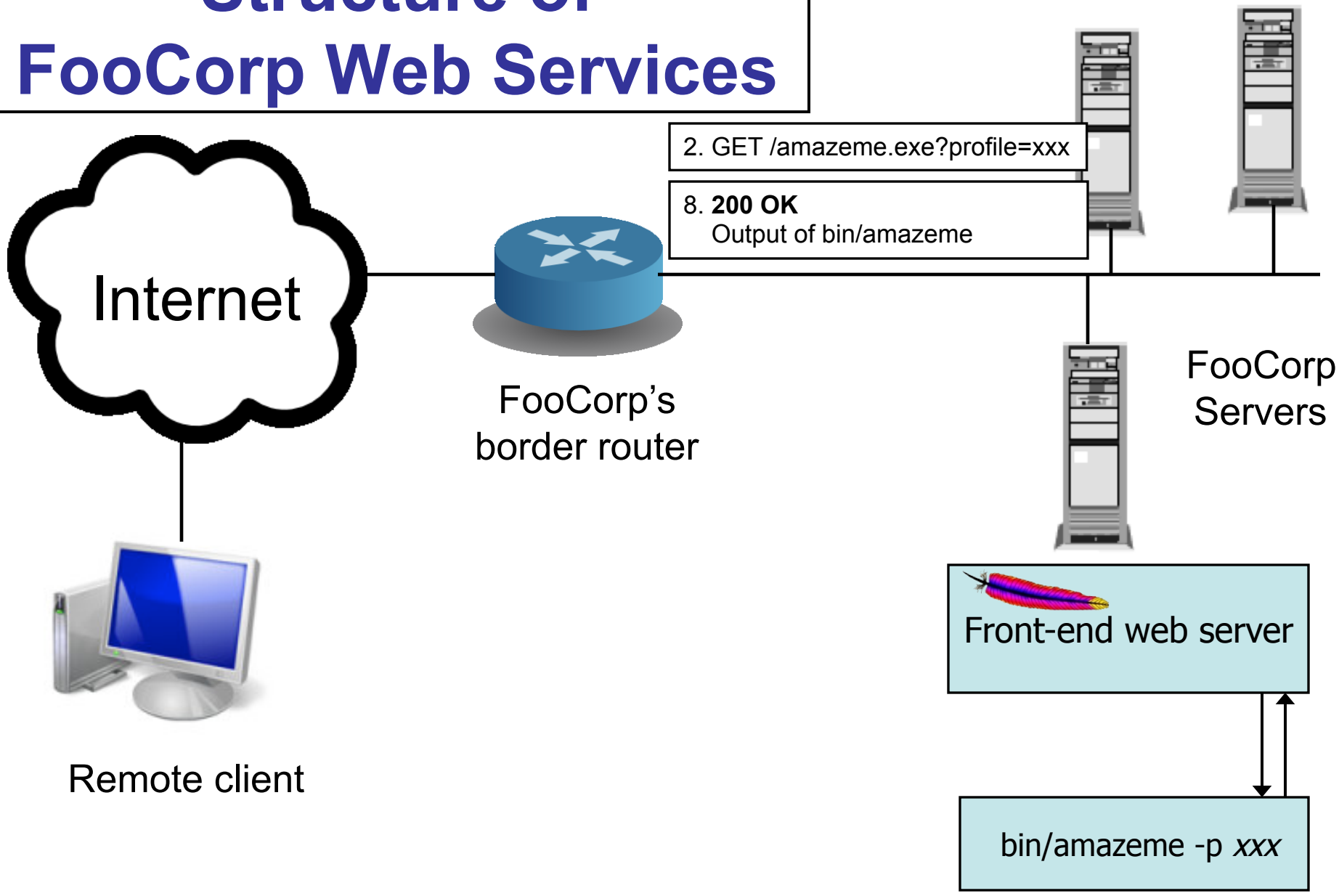
Evasion

- Evasion attacks arise when you have "double parsing"
- *Inconsistency* – interpreted differently
- *Ambiguity* – information needed to interpret is missing

Evasion Attacks (High-Level View)

- Some evasions reflect incomplete analysis
  - In our FooCorp example, hex escapes or "..////.//../" alias
  - In principle, can deal with these with implementation care (make sure we fully understand the spec)
- Some are due to imperfect observability
  - For instance, if what NIDS sees doesn't exactly match what arrives at the destination

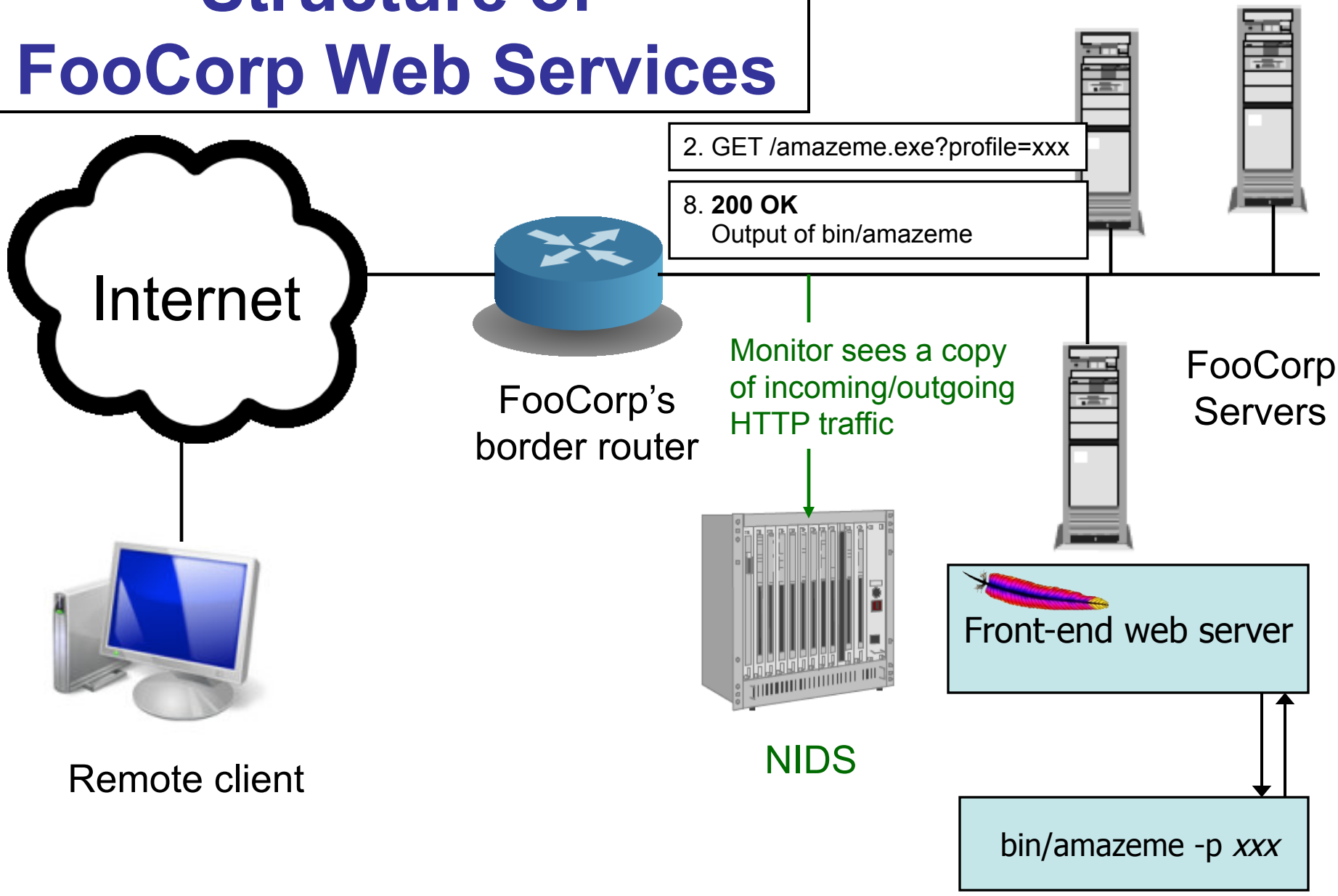
Structure of FooCorp Web Services



Network Intrusion Detection

- 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



Network Intrusion Detection

- 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

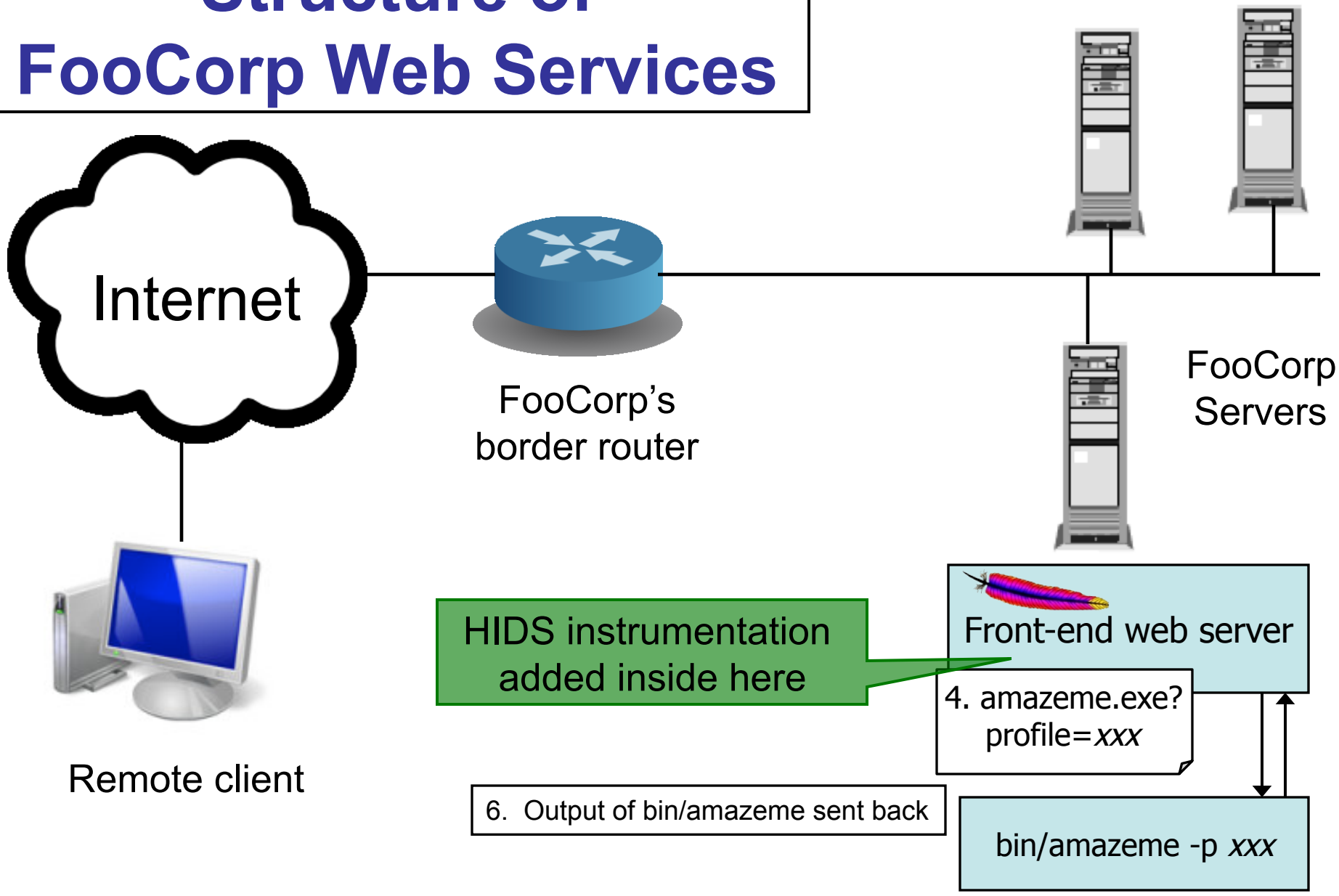
Network-Based Detection

- 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!

Host-based Intrusion Detection

- 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



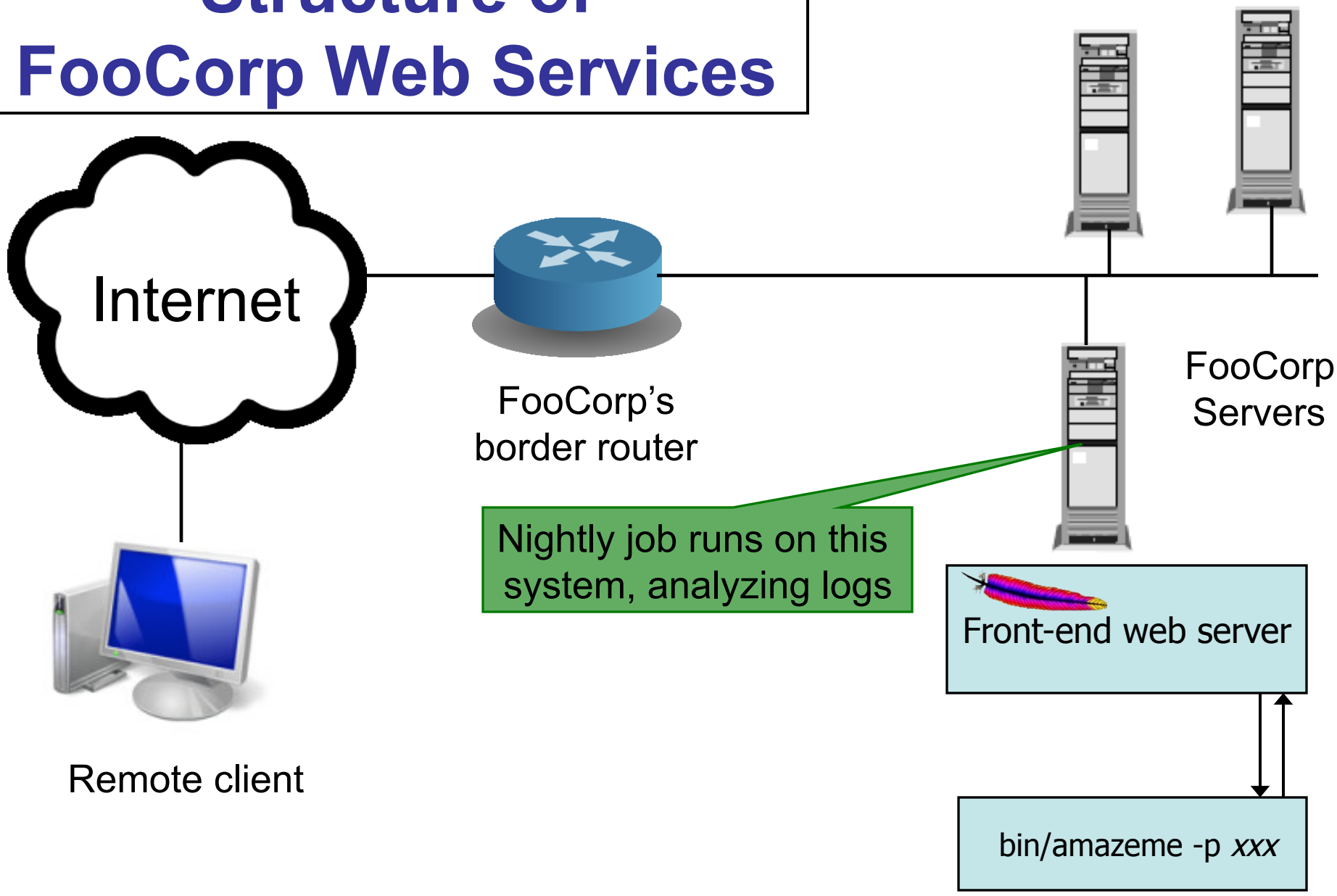
Host-based Intrusion Detection

- 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

Log Analysis

- 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



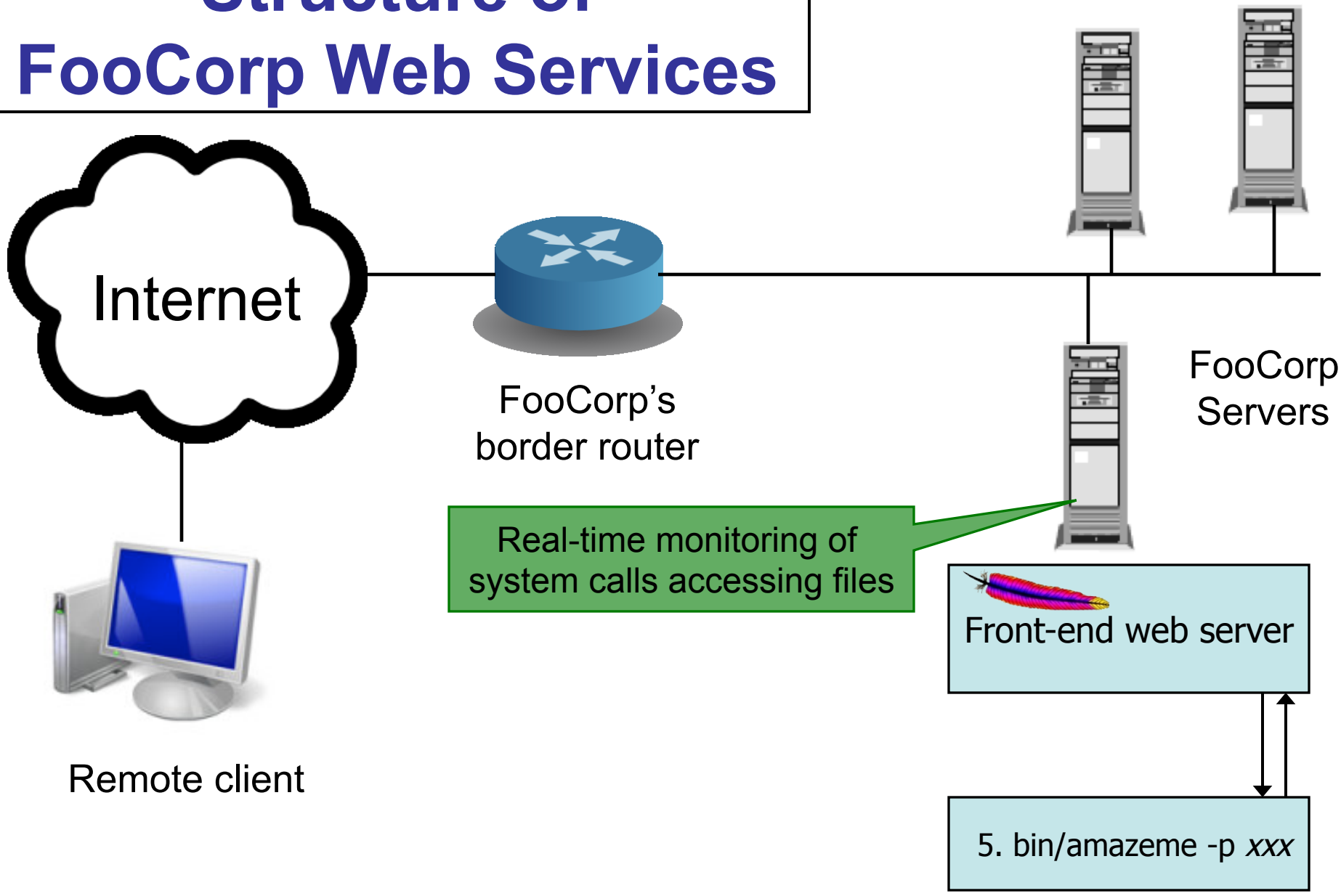
Log Analysis

- 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)

System Call Monitoring (HIDS)

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

Structure of FooCorp Web Services



System Call Monitoring (HIDS)

- 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
  - Attack only leads to an "alert" if attack succeeded
    - Sensitive file was indeed accessed
- Issues:
  - 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"

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 the event of an instance of intrusive behavior occurring (something we want to detect)
  - Define A to be the event of detector generating 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

Styles of Detection: Signature-Based

- Idea: look for activity that matches the structure of a known attack
- Example (from the freeware *Snort* NIDS):

```
alert tcp $EXTERNAL_NET any -> $HOME_NET
  139 flow:to_server,established
content:"|eb2f 5feb 4a5e 89fb 893e 89f2|"
msg:"EXPLOIT x86 linux samba overflow"
reference:bugtraq,1816
reference:cve,CVE-1999-0811
classtype:attempted-admin
```

- Can be at different semantic layers
  e.g.: IP/TCP header fields; packet payload; URLs

Signature-Based Detection

- E.g. for FooCorp, search for "../../" or "/etc/passwd"
- What's nice about this approach?
  - Conceptually simple
  - Takes care of known attacks (of which there are zillions)
  - Easy to share signatures, build up libraries
- What's problematic about this approach?
  - Blind to novel attacks
  - Might even miss *variants* of known attacks ("..///.//../")
    - Of which there are zillions
  - Simpler versions look at low-level syntax, not semantics
    - Can lead to weak power (either misses variants, or generates lots of false positives)

Vulnerability Signatures

- Idea: don't match on known attacks, match on known problems
- Example (also from *Snort*):

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80
uricontent: ".ida?"; nocase; dsize: > 239; flags:A+
msg:"Web-IIS ISAPI .ida attempt"
reference:bugtraq,1816
reference:cve,CAN-2000-0071
classtype:attempted-admin
```

- That is, match URIs that invoke *.ida?*, have more than 239 bytes of payload, and have ACK set (maybe others too)
- This example detects any* attempt to exploit a particular buffer overflow in IIS web servers
  - Used by the "Code Red" worm
  - * (Note, signature is not quite complete)

Vulnerability Signatures

- What's nice about this approach?
  - Conceptually fairly simple
  - Takes care of known attacks
  - Easy to share signatures, build up libraries
  - Can detect variants of known attacks
  - Much more concise than per-attack signatures
- What's problematic?
  - Can't detect novel attacks (new vulnerabilities)
  - Signatures can be hard to write / express
    - Can't just observe an attack that works …
    - … need to delve into how it works

Benefits of attack signatures

Styles of Detection: Anomaly-Based

- Idea: attacks look peculiar.
- High-level approach: develop a model of normal behavior (say based on analyzing historical logs). Flag activity that deviates from it.
- FooCorp example: maybe look at distribution of characters in URL parameters, learn that some are rare and/or don't occur repeatedly
  - If we happen to learn that '.'s have this property, then could detect the attack *even without knowing it exists*
- Big benefit: potential detection of a wide range of attacks, including novel ones

Anomaly Detection

- What's problematic about this approach?
  - Can fail to detect known attacks
  - Can fail to detect novel attacks, if don't happen to look peculiar along measured dimension
  - What happens if the historical data you train on includes attacks?
  - Base Rate Fallacy particularly acute: *if prevalence of attacks is low,* then you're more often going to see benign outliers
    - High FP rate
    - OR: require such a stringent deviation from "normal" that most attacks are missed (high FN rate)

Hard to make work well - not widely used today

Specification-Based Detection

- Idea: don't learn what's normal; specify what's allowed
- FooCorp example: decide that all URL parameters sent to foocorp.com servers **must** have at most one '/' in them
  - Flag any arriving param with > 1 slash as an attack
- What's nice about this approach?
  - Can detect novel attacks
  - Can have low false positives
    - If FooCorp audits its web pages to make sure they comply
- What's problematic about this approach?
  - Expensive: lots of labor to derive specifications
    - And keep them up to date as things change ("churn")

Styles of Detection: Behavioral

- Idea: don't look for attacks, look for evidence of compromise

- FooCorp example: inspect all output web traffic for any lines that match a passwd file
- Example for monitoring user shell keystrokes:

```
unset HISTFILE
```

- Example for catching code injection: look at sequences of system calls, flag any that prior analysis of a given program shows it can't generate
  - E.g., observe process executing **`read`**(), **`open`**(), **`write`**(), **`fork`**(), **`exec`**() …
  - … but there's *no code path* in the (original) program that calls those in exactly that order!

Behavioral-Based Detection

- What's nice about this approach?
  - Can detect a wide range of novel attacks
  - Can have low false positives
    - Depending on degree to which behavior is distinctive
    - E.g., for system call profiling: **no false positives!**
  - Can be cheap to implement
    - E.g., system call profiling can be mechanized
- What's problematic about this approach?
  - Post facto detection: discovers that you definitely have a problem, w/ no opportunity to prevent it
  - Brittle: for some behaviors, attacker can maybe avoid it
    - Easy enough to not type "`unset HISTFILE`"
    - How could they evade system call profiling?
      - **Mimicry**: adapt injected code to comply w/ allowed call sequences

The Problem of Evasion

- For any detection approach, we need to consider how an adversary might (try to) elude it
  - *Note: even if the approach is evadable, it can still be useful to operate in practice*
  - **But**: if it's very easy to evade, that's especially worrisome (security by obscurity)

Evasion Attacks (High-Level View)

- Some evasions reflect incomplete analysis
  - In our FooCorp example, hex escapes or "..////.//../" alias
  - In principle, can deal with these with implementation care (make sure we fully understand the spec)
- Some are due to imperfect observability
  - For instance, if what NIDS sees doesn't exactly match what arrives at the destination

The Problem of Evasion

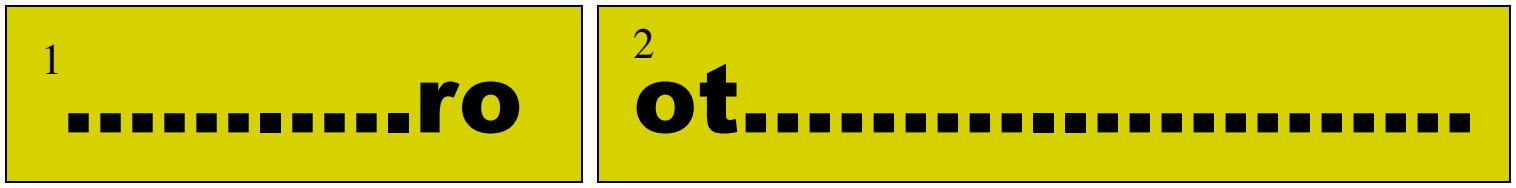
- Imperfect observability is particularly acute for network monitoring
- Consider detecting occurrences of the (arbitrary) string "`root`" inside a network connection …
  - We get a copy of each packet, how hard can it be?

Detecting "root": Attempt #1

- Method: scan each packet for 'r', 'o', 'o', 't'
  - Perhaps using Boyer-Moore, Aho-Corasick, Bloom filters …

Are we done?

Oops: TCP *doesn't preserve text boundaries*



Fix?

Detecting "root": Attempt #2

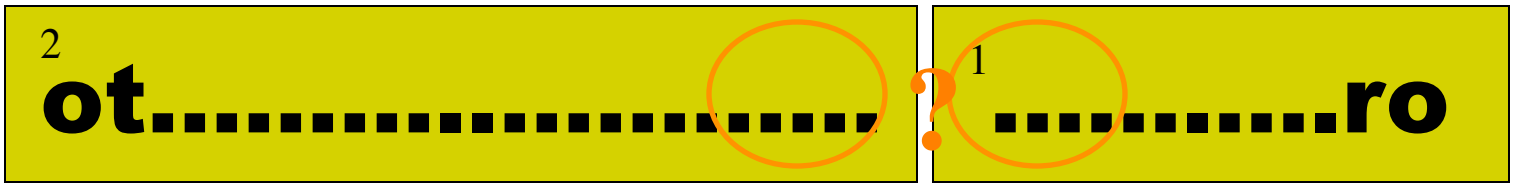
- Okay: remember match from end of previous packet

When 2nd packet arrives, continue working on the match

- Now we're managing state :-(

Are we done?

Oops: IP doesn't guarantee in-order arrival



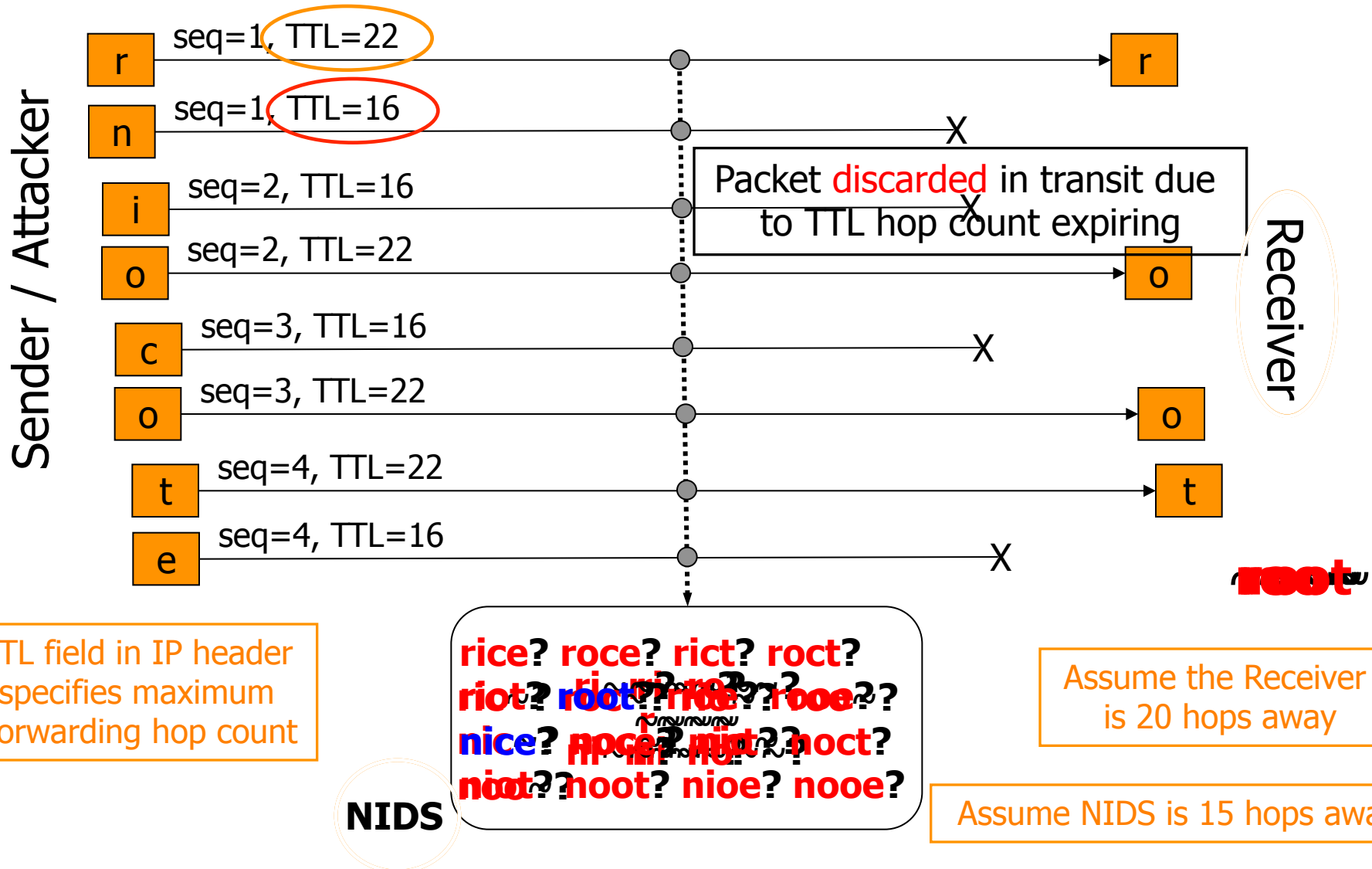
Detecting "`root`": Attempt #3

- Fix?

- We need to reassemble the entire TCP bytestream
  - Match sequence numbers
  - Buffer packets with later data (above a sequence "hole")
- Issues?
  - Potentially requires a lot of state
  - Plus: attacker can cause us to exhaust state by sending lots of data above a sequence hole

- But at least we're done, right?

Full TCP Reassembly is Not Enough



Inconsistent TCP Retransmissions

- Fix?
- Idea: NIDS can alert upon seeing a retransmission inconsistency, as surely it reflects someone up to no good
- This doesn't work well in practice: TCP retransmissions broken in this fashion occur in live traffic
  - Fairly rare (23 times in a day of ICSI traffic)
  - But real evasions much rarer still (Base Rate Fallacy)
  - ⇒ This is a *general problem* with alerting on such ambiguities
- Idea: if NIDS sees such a connection, kill it
  - Works for this case, since benign instance is already fatally broken
  - But for other evasions, such actions have collateral damage
- Idea: rewrite traffic to remove ambiguities
  - Works for network- & transport-layer ambiguities
  - But must operate in-line and at line speed

Summary of Evasion Issues

- Evasions arise from uncertainty (or incompleteness) because detector must infer behavior/processing it can't directly observe
  - A general problem any time detection separate from potential target
- One general strategy: impose canonical form ("*normalize*")
  - E.g., rewrite URLs to expand/remove hex escapes
  - E.g., enforce blog comments to only have certain HTML tags
- (Another strategy: analyze all possible interpretations rather than assuming one
  - E.g., analyze raw URL, hex-escaped URL, doubly-escaped URL …)
- Another strategy: fix the basic observation problem
  - E.g., monitor directly at end systems

Inside a Modern HIDS ("AV")

- URL/Web access blocking:
  - Prevent users from going to known bad locations
- Protocol scanning of network traffic (esp. HTTP)
  - Detect & block known attacks
  - Detect & block known malware communication
- Payload scanning
  - Detect & block known malware
- (Auto-update of signatures for these)
- Cloud queries regarding reputation
  - Who else has run this executable and with what results?
  - What's known about the remote host / domain / URL?

Inside a Modern HIDS

- *Sandbox execution*
  - Run selected executables in constrained/monitored environment
  - Analyze:
    - System calls
    - Changes to files / registry
    - Self-modifying code (*polymorphism/metamorphism*)
- File scanning
  - Look for malware that installs itself on disk
- Memory scanning
  - Look for malware that never appears on disk
- Runtime analysis
  - Apply heuristics/signatures to execution behavior

Inside a Modern NIDS

- Deployment inside network as well as at border
  - Greater visibility, including tracking of user identity
- Full protocol analysis
  - Including extraction of complex embedded objects
  - In some systems, 100s of known protocols
- Signature analysis (also behavioral)
  - Known attacks, malware communication, blacklisted hosts/domains
  - Known malicious payloads
  - Sequences/patterns of activity
- *Shadow execution* (e.g., Flash, PDF programs)
- Extensive logging (in support of forensics)
- Auto-update of signatures, blacklists

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

Key Concepts for Detection

- Signature-based vs anomaly detection (blacklisting vs whitelisting)
- Evasion attacks
- Evaluation metrics: False positive rate, false negative rate
- Base rate problem

Extra Material

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")

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!!)

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 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

Styles of Detection: Honeypots

- Idea: deploy a sacrificial system that has no operational purpose
- Any access is by definition not authorized …
- … and thus an intruder
  - (or some sort of mistake)

- Provides opportunity to:
  - Identify intruders
  - Study what they're up to
  - Divert them from legitimate targets

Honeypots

- Real-world example: some hospitals enter fake records with celebrity names …
  - … to entrap staff who don't respect confidentiality
- What's nice about this approach?
  - Can detect all sorts of new threats
- What's problematic about this approach?
  - Can be difficult to lure the attacker
  - Can be a lot of work to build a convincing environment
  - Note: both of these issues matter less when deploying honeypots for automated attacks
    - Because these have more predictable targeting & env. needs
    - E.g. "spamtraps": fake email addresses to catching spambots

Forensics

- Vital complement to detecting attacks: figuring out what happened in wake of successful attack
- Doing so requires access to rich/extensive logs
  - Plus tools for analyzing/understanding them
- It also entails looking for patterns and understanding the implications of structure seen in activity
  - An iterative process ("peeling the onion")

Other Attacks on IDSs

- DoS: exhaust its memory
  - IDS has to track ongoing activity
  - Attacker generates lots of different forms of activity, consumes all of its memory
    - E.g., spoof zillions of distinct TCP SYNs …
    - … so IDS must hold zillions of connection records
- DoS: exhaust its processing
  - One sneaky form: *algorithmic complexity attacks*
    - E.g., if IDS uses a predictable hash function to manage connection records …
    - … then generate series of *hash collisions*
- Code injection (!)
  - After all, NIDS analyzers take as input network traffic under attacker's control …

Riverbed Technology | WinPcap

IPv4 ✔ IPv6 ✘

WIRESHARK

the world's foremost network protocol analyzer

Wireshark | Get Help | Develop

Google™ Custom Search | Search

Security Advisories

The following Wireshark releases fix serious security vulnerabilities. If you are running a vulnerable version of Wireshark you should consider upgrading.

wnpa-sec-2013-09: NTLMSSP dissector overflow, fixed in 1.8.5, 1.6.13
wnpa-sec-2013-08: Wireshark dissection engine crash, fixed in 1.8.5, 1.6.13
wnpa-sec-2013-07: DCP-ETSI dissector crash, fixed in 1.8.5, 1.6.13
wnpa-sec-2013-06: ROHC dissector crash, fixed in 1.8.5
wnpa-sec-2013-05: DTLS dissector crash, fixed in 1.8.5, 1.6.13
wnpa-sec-2013-04: MS-MMC dissector crash, fixed in 1.8.5, 1.6.13
wnpa-sec-2013-03: DTN dissector crash, fixed in 1.8.5, 1.6.13
wnpa-sec-2013-02: CLNP dissector crash, fixed in 1.8.5, 1.6.13