

# Detecting Attacks, Part 2

***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 18, 2017**

# Goals For Today

- General approaches (“styles”) to **detecting attacks**
- The fundamental problem of **evasion**
- Analyzing successful attacks: **forensics**
- (Operation of a modern HIDS/NIDS)

# 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

# Sample Higher-Layer *Snort* Signature

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS
                                $HTTP_PORTS
(msg:"ET Piranha default passwd attempt";
 flow:to_server,established;
 uricontent:"/piranha/secure/control.php3";
 content:"Authorization\ : Basic
         cGlyYW5oYTp";
reference:bugtraq,1148;
reference:cve,2000-0248;
reference:nessus,10381;
classtype:attempted-recon;
sid:2002331; rev:5;)
```

# Signature-Based Detection, con't

- 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
  msg: "Web-IIS ISAPI .ida attempt"
  reference: bugtraq,1816
  reference: cve,CAN-2000-0071
  classtype: attempted-admin
```
- That is, match URIs that invoke **\*.ida?\*** (in any combination of lower/uppercase) with more than **239 bytes** of payload
- 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, con't

- What's nice about this approach?
  - Conceptually fairly simple *Benefits of attack signatures*
  - 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

# 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, con't

- 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*

# Anomaly Detection in ML Terms

- In machine-learning terms, traditional anomaly detection corresponds to **unsupervised** *one-class classification*
  - Known to be very challenging; only works if data has a well-defined natural cluster that algorithms can discover
- More powerful **supervised** techniques can work much better
  - However #1: requires **labels**, which can be difficult to obtain
  - However #2: **Base Rate Fallacy** can still be a big problem
    - But for domains with plenty of “attacks”, such as detecting spam, can work well

*Somewhat in use 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!
  - **Note: no false positives!**

# Behavioral-Based Detection, con't

- 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

# 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/track** intruders
  - **Study** what they're up to
  - **Divert** them from legitimate targets

# Honeypots, con't

- 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**

# Honeypots, con't

- 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



**Joe Stewart**

@joestewart71

 Follow

Future of host IDS: Just a Bitcoin wallet with small amount of BTC. When emptied it means time to wipe/reinstall + change all your PWs.



RETWEETS

**31**

FAVORITES

**9**



12:10 PM - 21 Nov 2013

**5 Minute Break**

**Questions Before We Proceed?**

# The Problem of Evasion

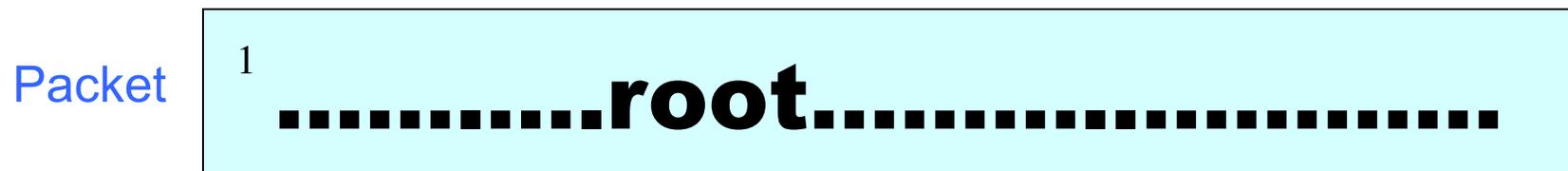
- For any detection approach, we need to consider how an adversary might (try to) **elude** it
  - *Note: even if the approach is evadable, it can still be useful to operate in practice*
  - **But:** if it's very easy to evade, that's especially worrisome (security by obscurity)
- Some evasions reflect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

# The Problem of Evasion, con't

- Some evasions exploit *deviation from the spec*
  - E.g., *double-escapes* for SQL injection:  
`%25%32%37`  $\Rightarrow$  `%27`  $\Rightarrow$  `'`
- Some can exploit more *fundamental* ambiguities:
  - Problem grows as monitoring viewpoint increasingly removed from ultimate endpoints
    - Lack of *end-to-end* visibility
- 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?*

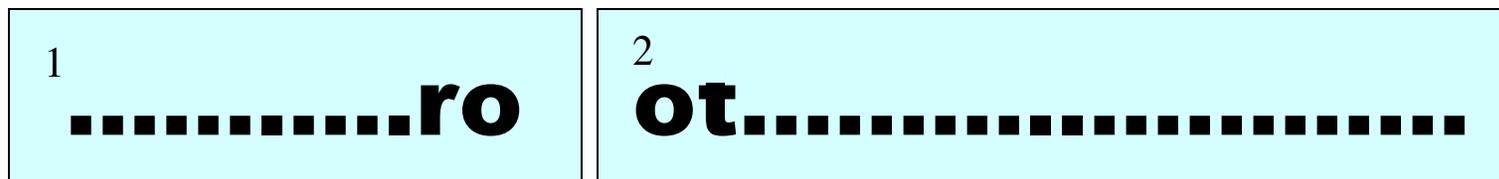
# Detecting “root”: Attempt #1

- 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*



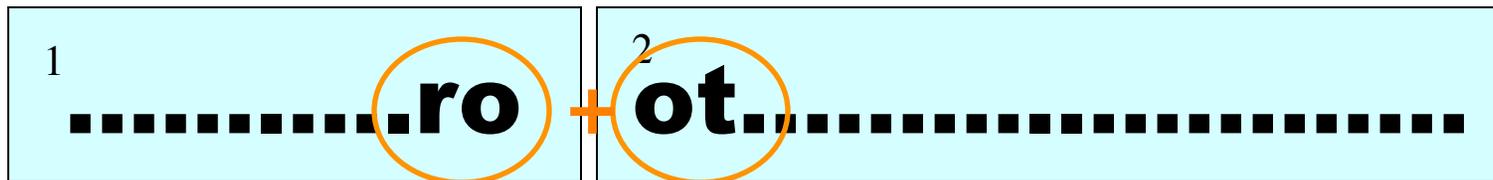
Packet #1

Packet #2

Fix?

# Detecting “root”: Attempt #2

- Okay: remember match from end of previous packet



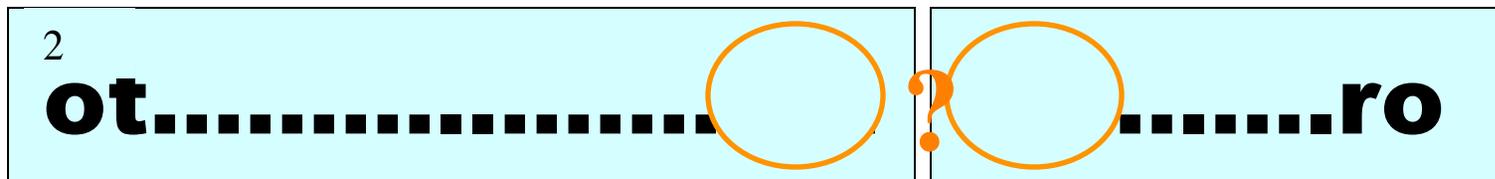
Packet #1

Packet #2

When 2nd packet arrives, continue working on the match

- Now we're managing **state** :-(  
Are we done?

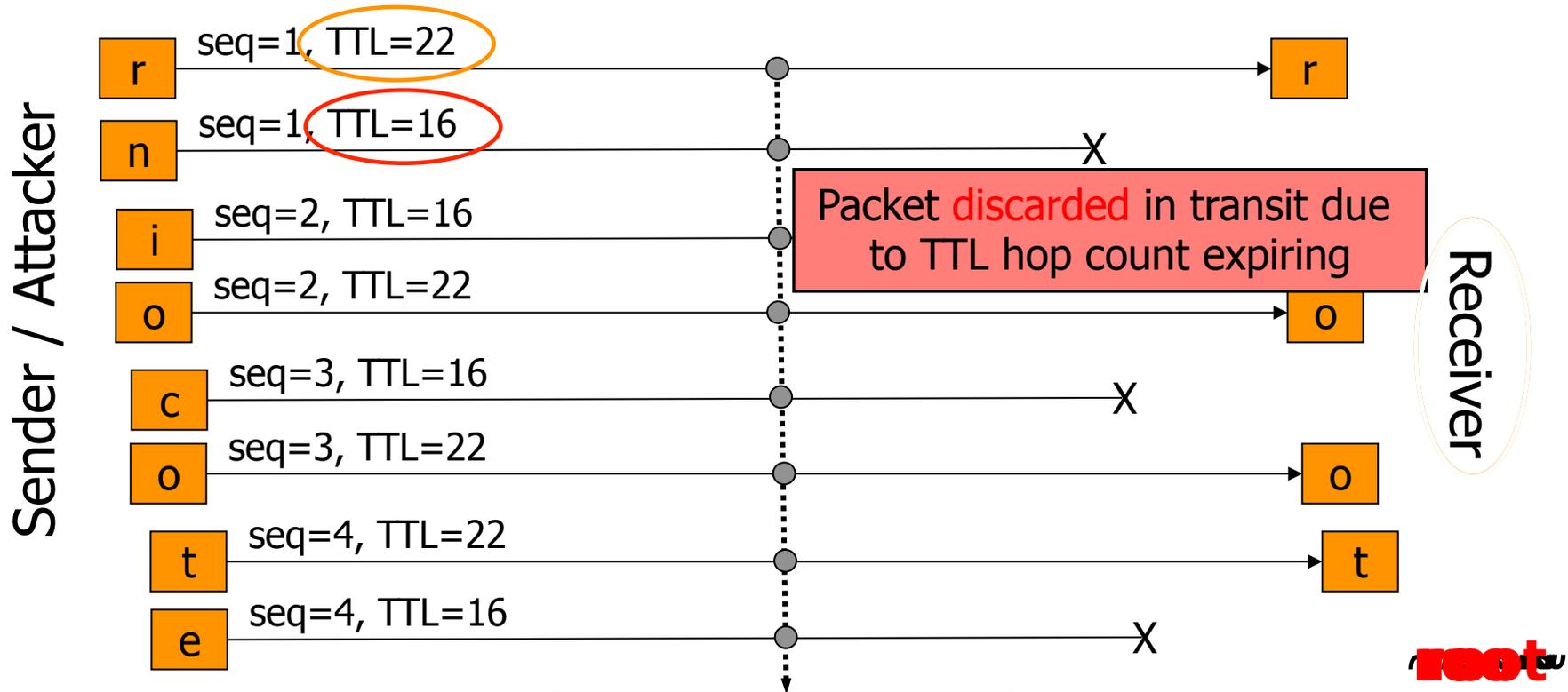
Oops: IP doesn't guarantee in-order arrival



# 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



TTL field in IP header specifies maximum forwarding hop count

NIDS

rice? roce? rict? roct?  
 riot? riot? riot? riot?  
 nice? nice? nice? nice?  
 riot? riot? nice? noce?

Assume the Receiver is 20 hops away

Assume NIDS is 15 hops away

# 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 benignly in live traffic
  - Fairly rare ( $< 1$  conn. in  $10^5$  at LBL; but 350M conn./day!)
  - 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: fix the basic observation problem
  - E.g., monitor **directly** at end systems