Firewalls (continued) and Intrusion Detection

CS 161 Spring 2022 - Lecture 22

Subverting Packet Filters

- Consider a simple example: Deny all connections containing the string **`root`**
  - Deny packets that contain the sequence of bytes **`r`**, **`o`**, **`o`**, and **`t`**
  - Allow all other packets

From: A	To: B
Seq = 4	
`Hello world`	

✓

From: C	To: D
Seq = 2	
`Log in`	

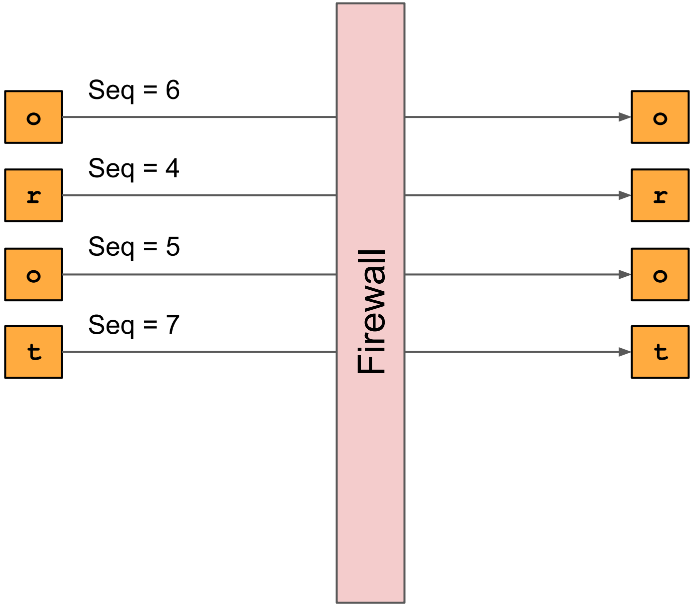
✓

From: C	To: D
Seq = 8	
`as root`	

✗

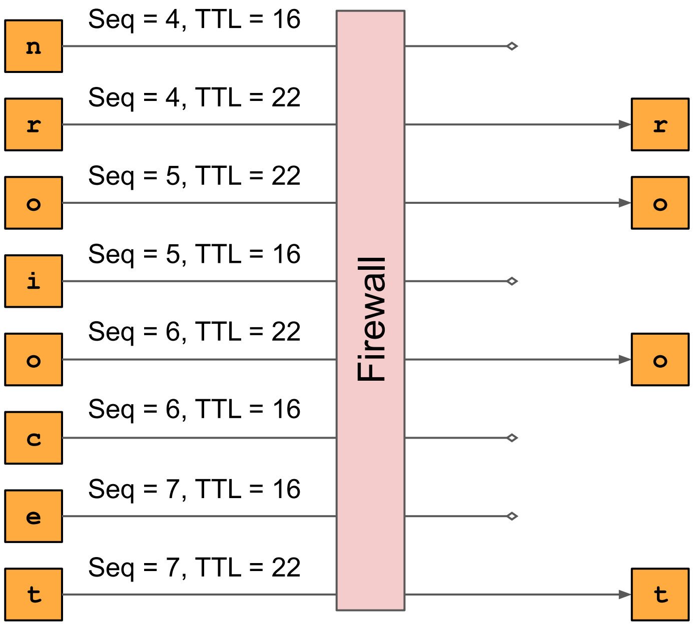
Subverting Packet Filters

- Recall TCP
  - Messages are split into packets before being sent
  - Packets can arrive out of order: The application will use sequence numbers to reorder packets
- Attack: Split the word **`root`** across packets
  - No single packet contains **`root`**, so the firewall won't stop any of these packets
- Attack: Send the split packets out of order
  - Now the firewall has to reconstruct TCP connections to detect the **`root`** message



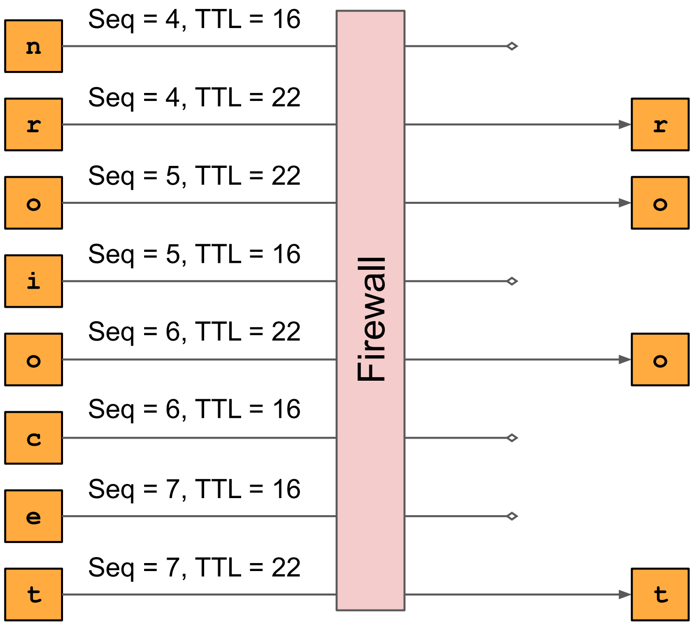
Subverting Packet Filters

- IP packets have a time-to-live (TTL)
  - The number of hops a packet may take before the packet is dropped
- The attacker can easily find how many hops away a given server is
  - Technique: Send ping packets with increasing TTLs until the server responds
- If the destination takes more hops than the firewall, the attacker can exploit this
  - Send multiple packets with the same sequence number, setting the TTLs on the dummy packets so that they are dropped before the reach the destination



Subverting Packet Filters

- Difficult for the stateful packet filter to defend against
  - TTLs for different packets naturally vary, since packets may take different routes
  - Storing all possible combinations takes exponential space
  - Hard to predict which packets will reach the destination and which won't
- Example of **evasion**, constructing network traffic that is parsed by the monitor in one fashion but the end host by another



Other Types of Firewalls

- **Proxy firewall**: Instead of forwarding packets, form two TCP connections: One with the source, and one with the destination
  - The firewall is really just a MITM, so it can easily spoof the addresses of the end hosts
  - Avoids problems with packets, since the firewall has direct access to the TCP byte streams
  - May require substantially more resources
- **Application proxy firewall**: Certain protocols allow for proxying at the application layer
  - Example: HTTP proxies will make an HTTP request on behalf of the user then return the HTTP response to the client

Alternatives to Allowing Firewall Traffic

- **Virtual private network** (**VPN**): A set of protocols that allows direct access to an internal network via an external connection
  - Creates an encrypted tunnel to allow internal network traffic to be sent securely over the Internet
  - Intuition: The encrypted tunnel is an emulated Ethernet cable that allows you to connect "inside" the network
    - Making "outside" computers treated as "inside"
  - The firewall allows VPN traffic, which allows arbitrary traffic to be tunneled inside
  - Relies on the VPN server to authenticate users

The Berkeley Campus VPN

- Berkeley has a VPN service for this purpose
  - Configure some or all of the traffic to route through the VPN
- Usage
  - Protect against a local network adversary?
    - Though it's pretty easy for on-path and MITM attackers to block the VPN connection
  - Your computer becomes "inside Berkeley" for authentication purposes
    - Internal services (e.g. purchasing) pages are only available to computers "inside Berkeley"
    - External services (e.g. library) pages only allow computers "inside Berkeley" to bypass the paywall

Firewall Pros and Cons

- Pros
  - Centralized management of security policies (single point of control)
  - Transparent operation to end users
  - Mitigates security vulnerabilities on end hosts (e.g. block anything that looks like shellcode)
- Cons
  - Reduced network connectivity
    - Some applications don’t work well inside a firewall
  - Vulnerability to “insiders”
    - Employees could be bribed or threatened
    - Devices are often brought from into the network outside (e.g. cell phones, laptops)
    - Once one device is compromised, attackers can quickly spread through the network
    - Could be mitigated by layering firewalls for more sensitive devices

Summary: Denial of Service

- **Availability**: Making sure users are able to use a service
  - DoS attacks availability of services
- **Application-level DoS**: Attacks the high-level applications
  - Algorithmic complexity attacks: Attack using inputs that cause the worst-case runtime of an algorithm
  - Defense: Identification, isolation, and quotas
  - Defense: Proof of work
- **Network-level DoS**: Attacks the network of a service
  - Typically floods either the network bandwidth or the packet processing capacity
  - Distributed DoS: Use multiple computers to flood a network at the same time
  - Amplified DoS: Use an amplifier to turn a small input into a large output, spoofing packets so the reply goes to the victim
  - Defense: Packet filtering
- All DoS attacks can be defended against by overprovisioning

Summary: SYN Cookies

- **SYN flooding**: A type of DoS that causes a server to allocate state for unfinished TCP connections, upon receiving a SYN packet
  - **SYN cookies**: Instead of allocating state when receiving a SYN, send the state back to the client in the sequence number
  - The client returns the state back to the server, which it only then allocates state for

Summary: Firewalls

- **Firewalls**: Defend many devices by defending the network
  - **Security policies** dictate how traffic on the network is handled
- **Packet filters**: Choose to either forward or drop packets
  - **Stateless packet filters**: Choose depending on the packet only
  - **Stateful packet filters**: Choose depending on the packet and the history of the connection
  - Attackers can subvert packet filters by splitting key payloads or exploiting the TTL
- **Proxy firewalls**: Create a connection with both sides instead of forwarding packets

Next: Intrusion Detection

- Path traversal attacks: An example exploit we want to detect
- Types of detectors
  - Network intrusion detection system (NIDS)
  - Host-based intrusion detection system (HIDS)
- Detection accuracy
- Styles of detection
  - Signature-based detection
  - Specification-based detection
  - Anomaly-based detection
  - Behavioral detection
- Other intrusion detection strategies

Intrusion Detection

- We’ve talked about many ways to prevent attacks
- However, some not all methods are perfect: attacks will slip through our defenses
- Recall: “Detect if you can’t prevent”
- How can we detect network attacks when they happen?

Path Traversal Attacks

Top 25 Most Dangerous Software Weaknesses (2020)

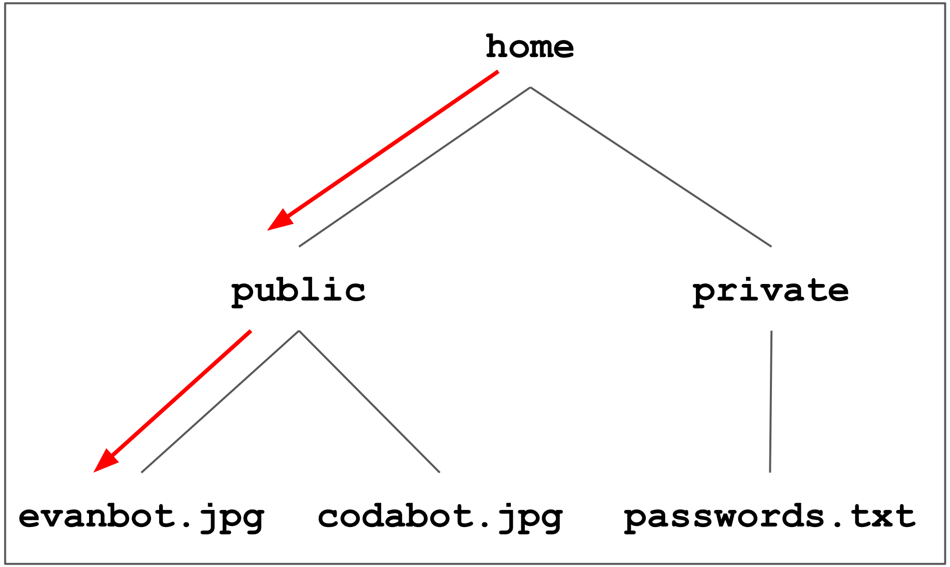
Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47
[4]	CWE-125	Out-of-bounds Read	26.50
[5]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16
[8]	CWE-416	Use After Free	18.87
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
[10]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44
[11]	CWE-190	Integer Overflow or Wraparound	15.81
[12]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	13.67
[13]	CWE-476	NULL Pointer Dereference	8.35
[14]	CWE-287	Improper Authentication	8.17
[15]	CWE-434	Unrestricted Upload of File with Dangerous Type	7.38
[16]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.95
[17]	CWE-94	Improper Control of Generation of Code ('Code Injection')	6.53

Unix File Paths

- A file path points to a file or a directory (folder) on a Unix system
- File paths have special characters
  - `/` (slash): Separates directories
  - `.` (one period): Shorthand for the current directory
  - `..` (two periods): Shorthand for the parent directory

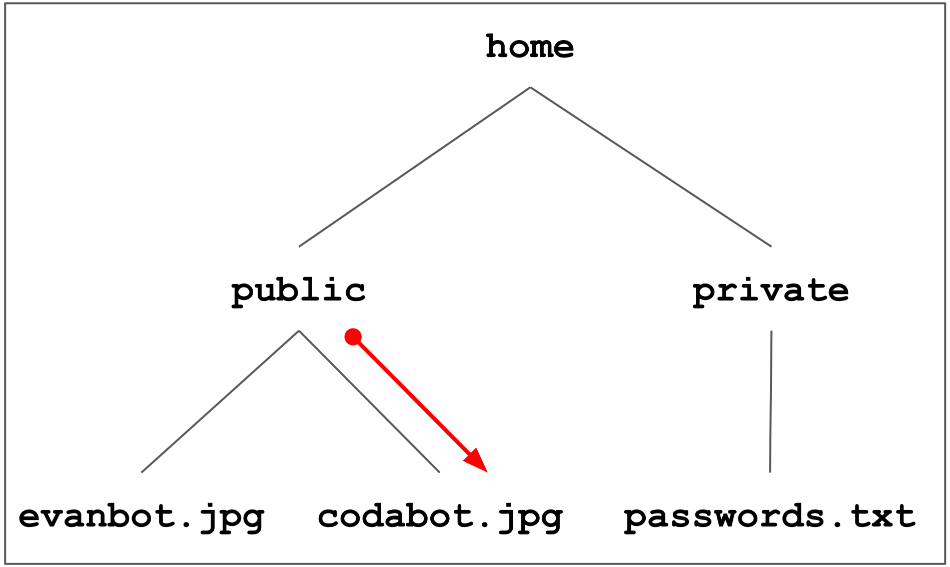
Unix File Paths

`/home/public/evanbot.jpg`



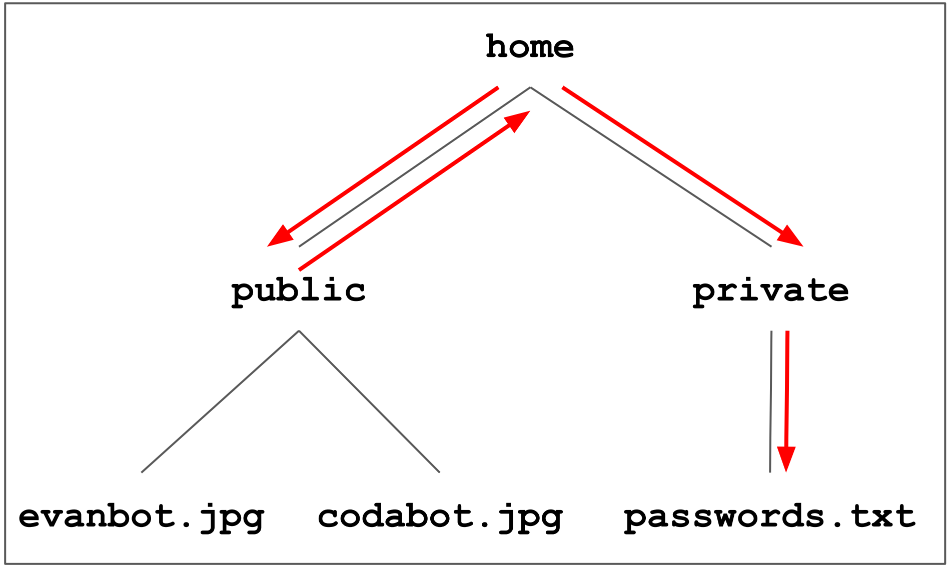
Unix File Paths

`./codabot.jpg` (Assume we're currently in **`public`**)

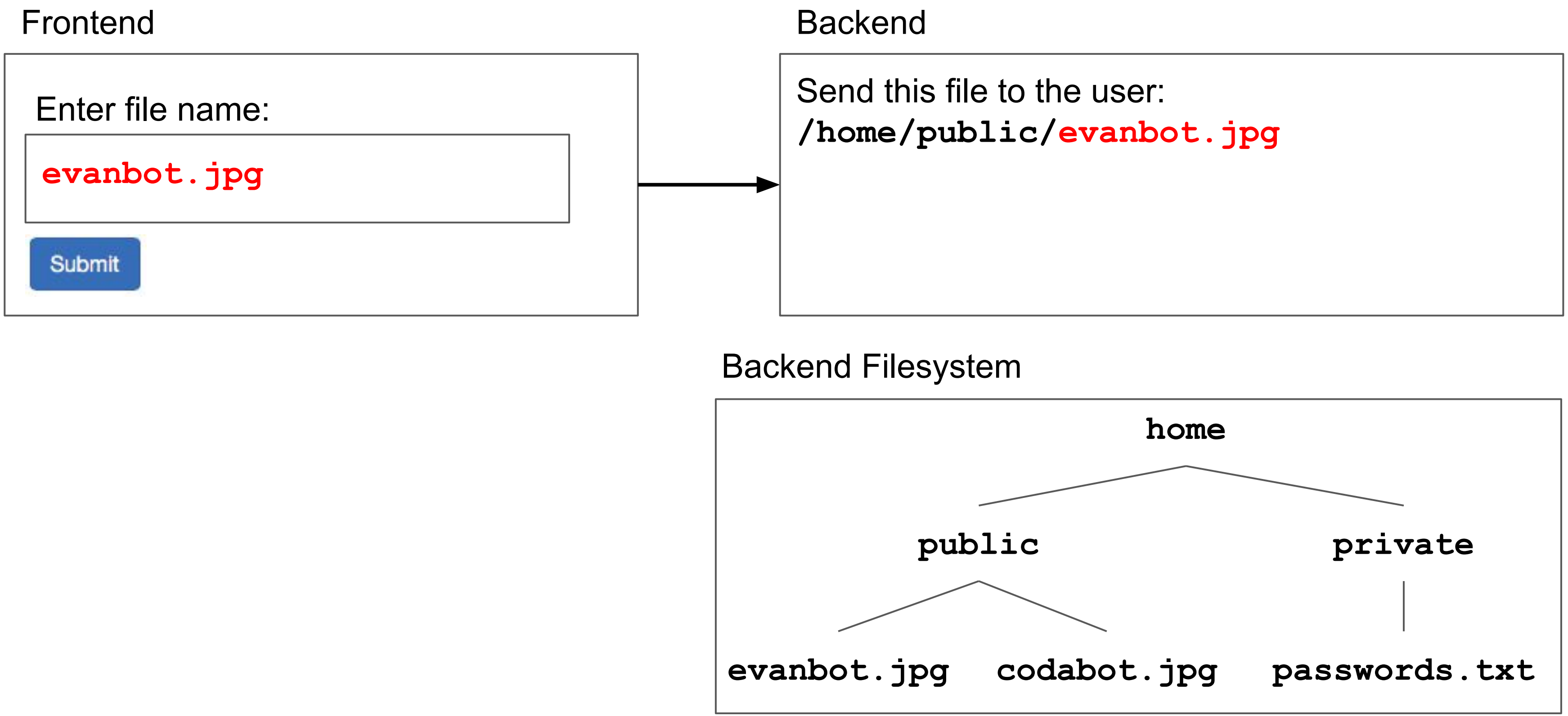


Unix File Paths

`/home/public/../private/passwords.txt`



Path Traversal Intuition



Path Traversal Intuition

Frontend

Enter file name:

`../private/passwords.txt`

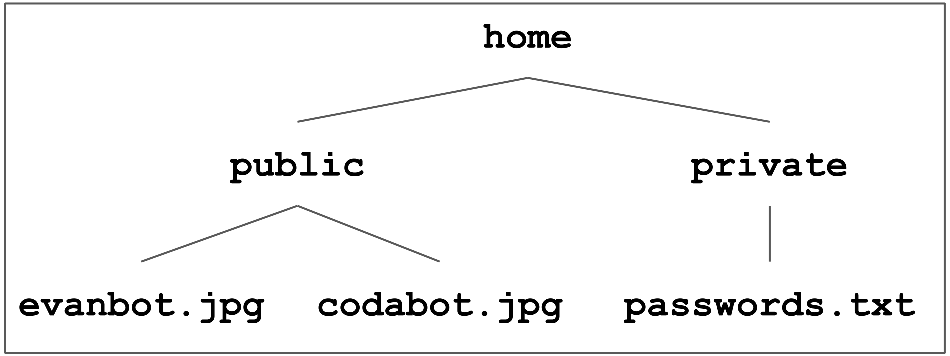
Submit

Backend

Send this file to the user:

`/home/public/../private/passwords.txt`

Backend Filesystem



Path Traversal Attacks

- **Path traversal attack**: Accessing unauthorized files on a remote server by exploiting Unix file path semantics
  - Often makes use of `../` to enter other directories
  - Vulnerability: User input is interpreted as a file path by the Unix file system
- Defense: Check that user input is not interpreted as a file path
  - How can we detect this kind of attack?

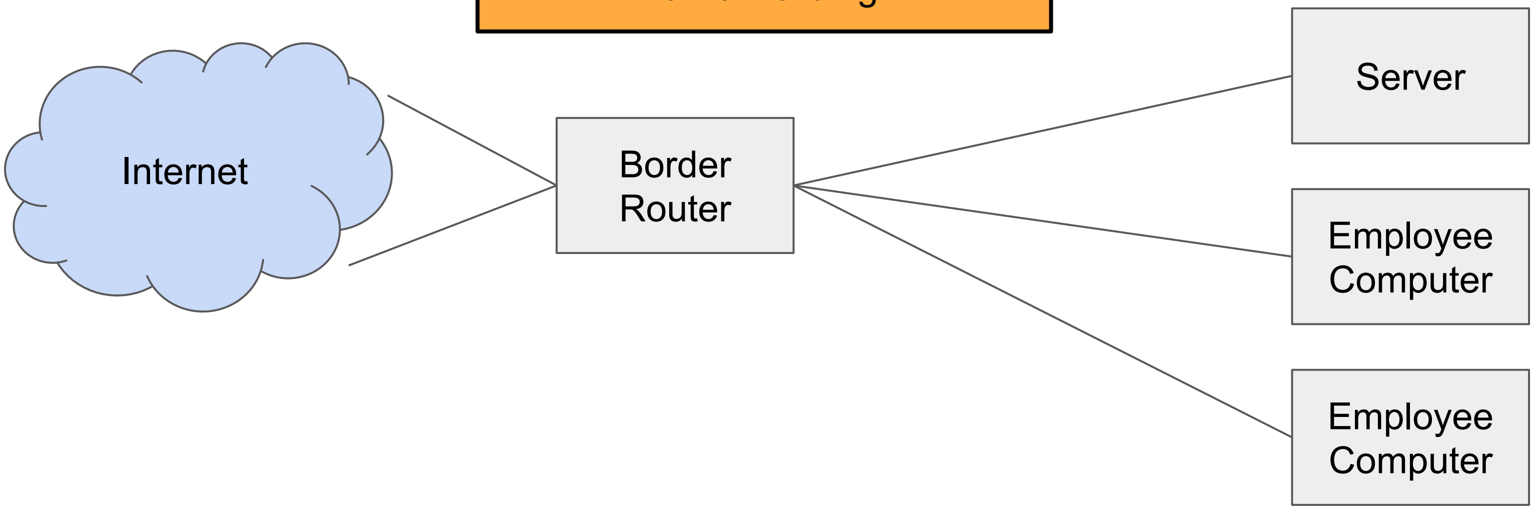
Types of Detectors

Types of Detectors

- Three types of detectors
  - Network Intrusion Detection System (NIDS)
  - Host-based Instruction Detection System (HIDS)
  - Logging
- The main difference is where the detector is deployed

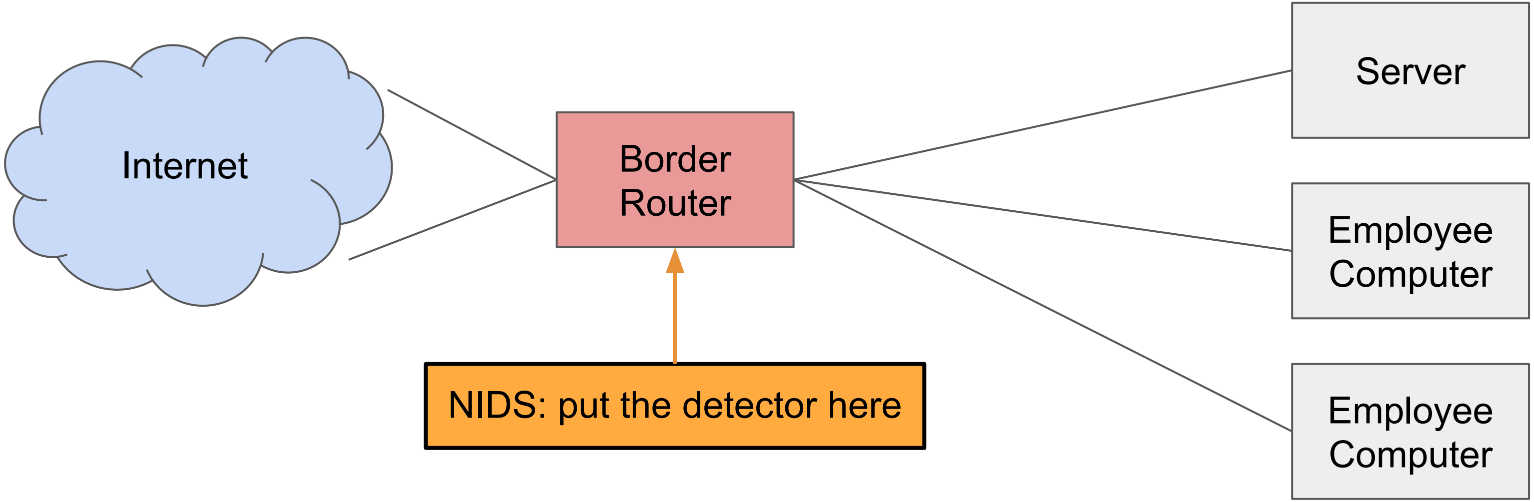
Structure of a Network

End hosts in the local network send packets to the Internet by sending it to the border router for forwarding



Network Intrusion Detection System (NIDS)

- **Network intrusion detection system** (**NIDS**): A detector installed on the network, between the local network and the rest of the Internet
  - Monitors network traffic to detect attacks



Network Intrusion Detection System (NIDS)

- Operation:
  - NIDS has a table of all active connections and maintains state for each connection
  - If the NIDS sees a packet not associated with any known connection, create a new entry in the table
    - Example: A connection that started before the NIDS started running
  - NIDS can be used for more sophisticated network monitoring: not only detect attacks, but analyze and understand all the network traffic

NIDS: Benefits

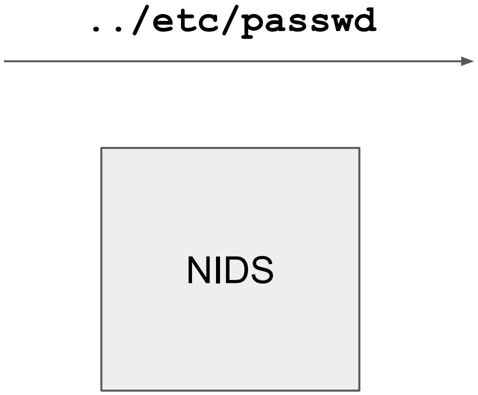
- Cheap: A single detector can cover a lot of systems
- Easy to scale: As the network gets larger, add computing power to the NIDS
  - Linear scaling: Investing twice as much money gives twice as much bandwidth
- Simple management: Easy to install and manage a single detector
- End systems are unaffected
  - Doesn’t consume any resources on end systems
  - Useful for adding security on an existing system
- Smaller trusted computing base (TCB)
  - Only the detector needs to be trusted

NIDS: Drawbacks

- Inconsistent or ambiguous interpretation between the detector and the end host
- How does the NIDS monitor encrypted traffic?

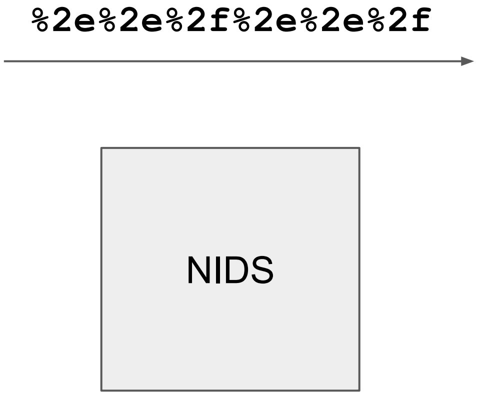
Drawback: Inconsistent Interpretation

- What should the NIDS do if it sees this packet?
  - This looks like a path traversal attack… Maybe it should alert
- What if the packet's TTL expires before it reaches any end host?
- Problem: What the NIDS sees doesn't exactly match what arrives at the end system



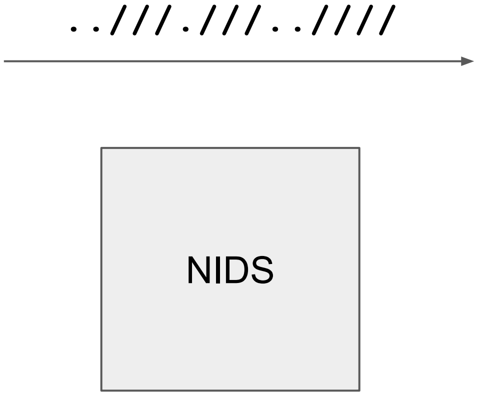
Drawback: Inconsistent Interpretation

- What should the NIDS do if it sees this packet?
  - This doesn't look like a path traversal attack… maybe it shouldn't alert
- This input is using URL percent encoding. If you decode it, you get **`../etc/passwd`**!
- Problem: Inputs are interpreted differently between the NIDS and the end system



Drawback: Inconsistent Interpretation

- What should the NIDS do if it sees this packet?
- What file on the file system does this file path refer to? It's hard for the NIDS to know
- Problem: Information needed to interpret correctly is missing



Evasion Attacks

- Problem: Imperfect observability
  - What the NIDS sees doesn't match what the end system sees
  - Example: The packet's time-to-live (TTL) might expire before reaching the end host
- Problem: Incomplete analysis (double parsing)
  - Inconsistency: Inputs are interpreted and parsed differently between the NIDS and the end system
  - Ambiguity: Information needed to interpret correctly is missing
- **Evasion attack**: Exploit inconsistency and ambiguity to provide malicious inputs that are not detected by the NIDS

Evasion Attacks: Defenses

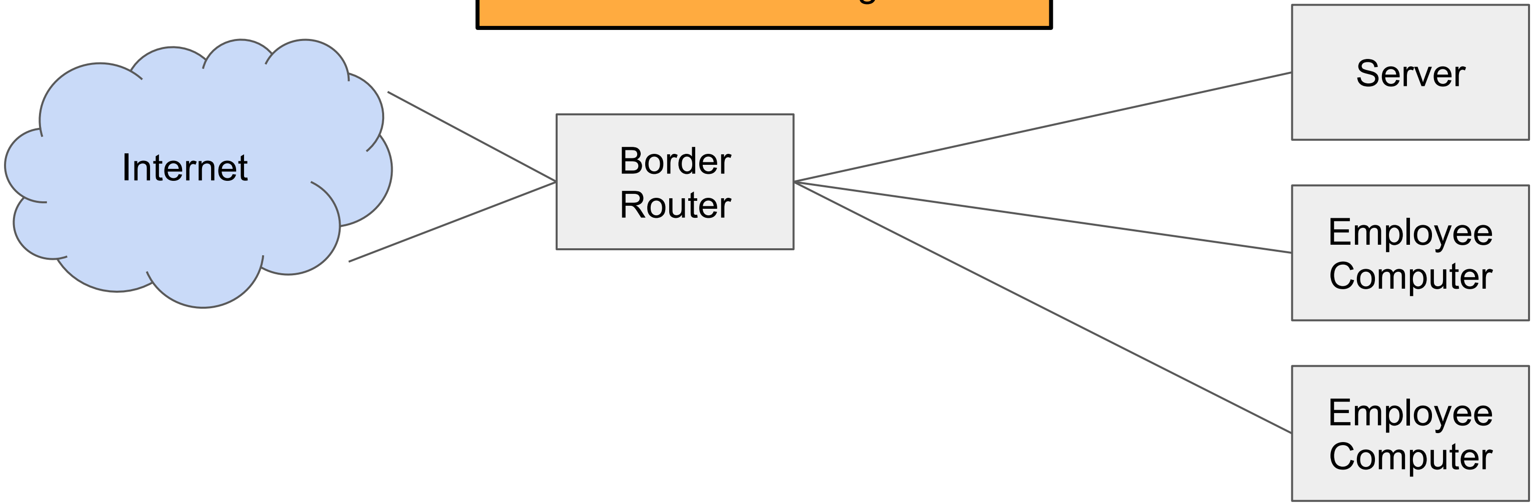
- Make sure that the NIDS and the end host are using the same interpretations
  - This can be very challenging
  - How do we detect the URL-encoded attack `%2e%2e%2f%2e%2e%2f`?
    Now the NIDS has to parse URL encodings!
  - How do we detect a more complicated path traversal attack `..///.///..////`?
    Now the NIDS has to parse Unix file paths!
- Impose a canonical (“normalized”) form for all inputs
  - Example: Force all URLs to expand all URL encodings or not expand all URL encodings
- Analyze all possible interpretations instead of assuming one
- Flag potential evasions so they can be investigated further

Drawback: Encrypted Traffic

- Recall: TLS is end-to-end secure, so a NIDS can’t read any encrypted traffic
- One possible solution: Give the NIDS access to all the network’s private keys
    - Now the NIDS can decrypt messages to inspect them for attacks
    - Problem: Users have to share their private key with someone else

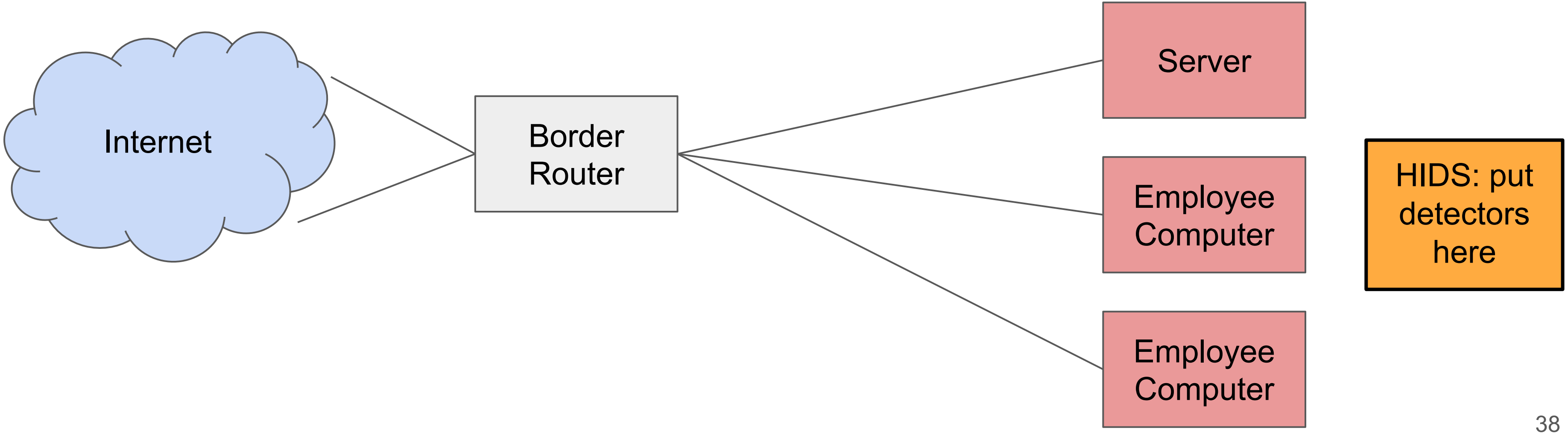
Recall: Structure of a Network

End hosts in the local network send packets to the Internet by sending it to the border router for forwarding



Host-Based Intrusion Detection System (HIDS)

- **Host-based intrusion detection system (HIDS)**: A detector installed on each end system



Host-Based Intrusion Detection System (HIDS)

- Benefits
  - Fewer problems with inconsistencies or ambiguities: The HIDS is on the end host, so it will interpret packets exactly the same as the end host!
  - Works for encrypted messages
  - Can protect against non-network threats too (e.g. malicious user inside the network)
  - Performance scales better than NIDS: one NIDS is more vulnerable to being overwhelmed than many HIDS
- Drawbacks
  - Expensive: Need to install one detector for every end host
  - Evasion attacks are still possible (consider Unix file name parsing)

Logging

- **Logging**: Analyze log files generated by end systems
  - Example: Each night, run a script on the log files to analyze them for attacks
- Benefits
  - Cheap: Modern web servers often already have built-in logging systems
  - Fewer problems with inconsistencies or ambiguities: The logging system works on the end host, so it will interpret packets exactly the same as the end host!
- Drawbacks
  - Unlike NIDS and HIDS, there is no real-time detection: attacks are only detected **after the attack has happened**
  - Some evasion attacks are still possible (again, consider Unix file name parsing)
  - The attacker could change the logs to erase evidence of the attack

Detection Accuracy

Detection Errors

- Two main types of detector errors
    - **False positive**: Detector alerts when there is no attack
    - **False negative**: Detector fails to alert when there is an attack
- Detector accuracy is often assessed in terms of the rates at which these errors occur
    - **False positive rate** (**FPR**): The probability the detector alerts, given there is no attack
    - **False negative rate** (**FNR**): The probability the detector does not alert, given there is an attack

Perfect Detectors

- Can we build a detector with a false positive rate of 0%? How about a detector with a false negative rate of 0%?
  - Recall false positive rate: The probability the detector alerts, given there is no attack
  - Recall false negative rate: The probability the detector does not alert, given there is an attack

```
void detector_with_no_false_positives(char *input) {
    printf("Nope, not an attack!");
}
```

```
void detector_with_no_false_negatives(char *input) {
    printf("Yep, it's an attack!");
}
```

Detection Tradeoffs

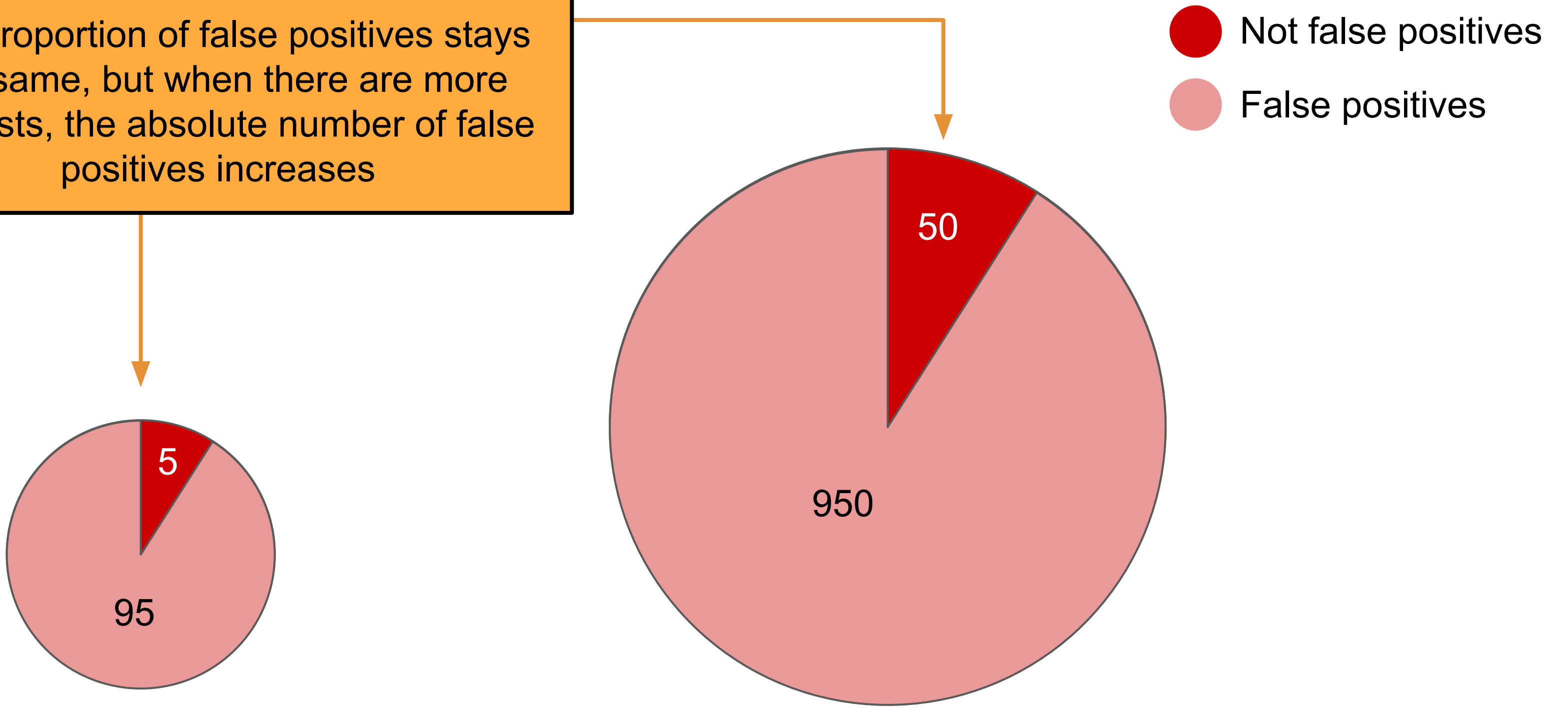
- The art of a good detector is achieving an effective balance between false positives and false negatives
- The quality of the detector depends on the system you're using it on
  - What is the rate of attacks on your system?
  - How much does a false positive cost in your system?
  - How much does a false negative cost in your system?
- Example of cost analysis: Fire alarms
  - Which is better: a very low false positive rate or a very low false negative rate?
  - Cost of a false positive: The fire department needs to inspect the building
  - Cost of a false negative: The building burns down
  - In this situation, false negatives are much more expensive!
  - We want a detector with a low false negative rate

Detection Tradeoffs

- Example of changing the base rate of attacks
  - Consider a detector with a 0.1% false positive rate (for every 1,000 non-attacks, there is one mistaken alert)
  - Scenario #1: Our server receives 1,000 non-attacks and 5 attacks per day
    - Expected number of false positives per day: 1,000 × 0.1% = 1
  - Scenario #2: Our server receives 10,000,000 non-attacks and 5 attacks per day
    - Expected number of false positives per day: 10,000,000 × 0.1% = 10,000
    - Possibly expensive if the false positives cost money to investigate
    - Example: Maybe a human has to manually examine 10,000 requests per day
  - Nothing changed about the detector: Only our environment changed
- **Takeaway**: Accurate detection is very challenging if the base rate of attacks is low!

Detection Tradeoffs

The proportion of false positives stays the same, but when there are more requests, the absolute number of false positives increases



Base Rate Fallacy

- Consider the detector from before: 0.1% false positive rate
  - Assume a 0% false negative rate: Every attack is detected
  - Scenario from before: Our server receives 10,000,000 non-attacks and 5 attacks per day
  - Expected number of false positives per day: 10,000,000 × 0.1% = 10,000
- You see the detector alert. What is the probability this is really an attack?
  - Of the 10,005 detections, 5 are real attacks, and 10,000 are false positives
  - There is an approximately 0.05% probability that the detector found a real attack
- **Base rate fallacy**: Even though the detector alerted, it's still highly unlikely that you found an attack, because of the high false positive rate

Combining Detectors

- Can you combine two independent detectors to create a better detector?
- Parallel composition
  - Alert if either detector alerts
  - Intuition: The combination generates more alerts
  - Reduces false negative rate
  - Increases false positive rate
- Series composition
  - Alert only if both detectors alert
  - Intuition: The combination generates fewer alerts
  - Reduces false positive rate
  - Increases false negative rate
- There is no free lunch: Reducing one rate usually increases the other

Styles of Detection

Styles of Detection

- So far we've talked about types of detectors: *what* the detector is scanning
- Now we'll talk about styles of detection: *how* the detector scans data to find attacks
- Four main styles of detection
  - Signature-based detection
  - Specification-based detection
  - Anomaly-based detection
  - Behavioral detection

Signature-based Detection

- **Signature-based detection**: Flag any activity that matches the structure of a known attack
- Signature-based detection is **blacklisting**: Keep a list of patterns that are not allowed, and alert if we see something on the list
- Signatures can be at different network layers
    - Example: TCP/IP header fields
    - Example: URLs
    - Example: Payload of the HTTP request

Signature-based Detection: Examples

- Example: Path traversal attacks
  - We know that `../` is often part of a path traversal attack
  - Strategy: Alert if any request contains `../`
- Example: Buffer overflows
  - We know that buffer overflows usually contain shellcode
  - Strategy: Keep a list of common shellcodes and alert if any request contains shellcode
- **Vulnerability signatures**: Instead of keeping an entire exploit as a signature, keep only the parts necessary to exploit a vulnerability
  - Limits evasion opportunities since every exploit needs to have parts of a vulnerability signature

Signature-based Detection: Tradeoffs

- Benefits
  - Conceptually simple
  - Very good at detecting known attacks
  - Easy to share signatures and build up shared libraries of attacks
- Drawbacks
  - Won’t catch new attacks without a known signature
  - Might not catch variants of known attacks if the variant doesn’t match the signature
  - If an attacker knows what the signature is, they can easily modify their attack to avoid it
  - Simpler versions only look at raw bytes, without parsing them in context
    - May miss variants
    - May generate lots of false positives

Specification-based Detection

- **Specification-based detection**: Specify allowed behavior and flag any behavior that isn't allowed behavior
- Specification-based detection is **whitelisting**: Keep a list of allowed patterns, and alert if we see something that is not on the list

Specification-based Detection: Examples

- Example: Path traversal attacks
  - We have a folder where all filenames are alphanumeric (a-z, A-Z, 0-9)
  - We specify that only alphanumeric characters are allowed as input
  - Strategy: Alert if any request contains something other than alphanumeric characters
  - If an attacker tries a path traversal attack (`../`), the detector will flag it
- Example: Buffer overflows
  - Consider a program that asks for the user's age as input
  - We know that ages are numerical, so we specify that only numbers are allowed
  - Strategy: Flag input that isn't numerical
  - If an attacker tries to input shellcode (not numbers), the detector will flag it

Specification-based Detection: Tradeoffs

- Benefits
  - Can detect new attacks we’ve never seen before
  - If we properly specify all allowed behavior, can have low false positive rate
- Drawbacks
  - Takes a lot of time and effort to manually specify all allowed behavior
  - May need to update specifications as things change

Anomaly-based Detection

- Idea: Attacks look unusual
- **Anomaly-based detection**: Develop a model of what normal activity looks like. Alert on any activity that deviates from normal activity.
  - Example: Analyze historical logs to develop the model
- Similar to specification-based detection, but learn a model of normal behavior instead of manually specifying normal behavior

Anomaly-based Detection: Examples

- Example: Path traversal attacks
  - Analyze characters in requests and learn that `..` only appears in attacks
  - Strategy: Alert if any request contains `..`
- Example: Buffer overflows
  - Study user inputs to a C program
  - Learn that user input usually contains characters that can be typed on a keyboard
  - Strategy: Alert if the input contains characters that can’t be typed on a keyboard
  - If an attacker inputs shellcode (can’t be typed on a keyboard), the detector will alert

Anomaly-based Detection: Tradeoffs

- Benefits
  - Can detect attacks we haven’t seen before
- Drawbacks
  - Can fail to detect known attacks
  - Can fail to detect new attacks if they don’t look unusual to our model
  - What if our model is trained on bad data (e.g. data with a lot of attacks)?
  - The false positive rate might be high (lots of non-attacks look unusual)
  - If we try to reduce false positives by only flagging the most unusual inputs, the false negative rate might be high (we miss slightly unusual attacks)
- Great subject for academic research papers, but not used in practice

Behavioral Detection

- **Behavioral detection**: Look for evidence of compromise
- Unlike the other three styles, we are not scanning the input: We’re looking at the actions triggered by the input
    - Instead of looking for the exploit, we’re looking for the result of the exploit
    - *Behaviors* can themselves be analyzed using blacklists (signature-based), whitelists (specification-based), or normal behavior (anomaly-based)

Behavioral Detection: Examples

- Example: Path traversal attacks
  - Strategy: See if any unexpected files are being accessed (e.g. the passwords file)
- Example: Buffer overflows
  - Strategy: See if the program calls unexpected functions
  - Consider a C program that never calls the `exec` function: if the program starts running `exec`, there is probably an attack in progress!

Behavioral Detection: Tradeoffs

- Benefits
  - Can detect attacks we haven't seen before
  - Can have low false positive rates if we're looking for behavior that rarely occurs in normal programs (e.g. in the `exec` example, there are probably no false positives!)
  - Can be cheap to implement (e.g. existing tools to monitor system calls for a program)
- Drawbacks
  - Legitimate processes could perform the behavior as well (e.g. accessing a password file)
  - Only detects attacks after they've already happened
  - Only detects successful attacks (maybe we want to detect failed attacks as well)
  - The attacker can modify their attack to avoid triggering some behavior

Other Intrusion Detection Strategies

Vulnerability Scanning

- Idea: Instead of detecting attacks, launch attacks on your own system first, and add defenses against any attacks that worked
- **Vulnerability scanning**: Use a tool that probes your own system with a wide range of attacks (and fix any successful attacks)
- Widely used in practice today
  - Often used to complement an intrusion detection system
  - One common form is **penetration testing** (**pentesting**) or **red teaming**: Hire a group of people to legally attempt to break into your system

Vulnerability Scanning: Tradeoffs

- Benefits
  - Accuracy: If your scanning tool is good, it will find real vulnerabilities
  - Proactive: Prevents attacks before they happen
  - Intelligence: If your intrusion detection system alerts on an attack you know you already fixed, you can safely ignore the alert
- Drawbacks
  - Can take a lot of work
  - Not helpful for systems you can't modify
  - Dangerous for disruptive attacks (you might not know which attacks are dangerous before you run the scanning tool)

Honeypots

- **Honeypot**: a sacrificial system with no real purpose
  - No legitimate systems ever access the honeypot
  - If anyone accesses the honeypot, they must be an intruder
  - False positives: Legitimate systems mistakenly accessing the honeypot
- Similar idea as stack canaries

Honeypots: Examples

- Example: Hospitals
  - Employees should not read patient records
  - The hospital enters a honeypot record with a celebrity name
  - Catch any staff member who reads the honeypot record
- Example: Unsecured Bitcoin wallet
  - Leave an unsecured Bitcoin wallet on your system with a small amount of money in it
  - If the money is stolen, you know that someone has attacked your system!
- Example: Spamtrap
  - Create a fake email address that is never used for legitimate emails
  - If email gets sent to the address, it's probably spam!

Honeypots: Tradeoffs

- Benefits
  - Can detect attacks we haven’t seen before
  - Can analyze the attacker's actions
    - Who is the attacker?
    - What are they doing to the system?
  - Can distract the attacker from legitimate targets
- Drawbacks
  - Can be difficult to trick the attacker into accessing the honeypot
  - Building a convincing honeypot might take a lot of work
  - These drawbacks matter less if the honeypot is aimed at automated attacks (e.g. the spam detection honeypot)

Forensics

- **Forensics**: Analyzing what happened after a successful attack
  - Important complement to detecting attacks
  - An entire security subfield: **digital forensics and incident response** (**DFIR**)
    - The network and systems are compromised. How? What was taken? What was encrypted? How can we recover?
- Tools needed
  - Detailed logs of system activity
  - Tools for analyzing and understanding logs

Blocking: Intrusion Prevention Systems

- Idea: If we can detect attacks, can we also block them?
- **Intrusion prevention system** (**IPS**): An intrusion detection system that also blocks attacks
  - Commonly used today
- Drawbacks
  - Not possible for retrospective analysis (e.g. logging)
  - Difficult for a detector that passively monitors traffic (e.g. an on-path NIDS)
    - Dynamically change firewall rules to block attacks?
    - Forge a RST packet to stop an attack?
    - Need to race against the attacker's malicious packets
  - False positives are expensive
    - Blocking a non-attack might affect legitimate users

Building the Perfect IPS?

0% false positive rate

0% false negative rate

Takeaway: You must always have tradeoffs between false positive and false negative rates