Intrusion Detection Systems

Intrusion Detection Systems (IDS)

An IDS is any combination of hardware & software that monitors a system or network for malicious activity.

An IPS (Intrusion Prevention System) is a network IDS that can cap network connections.

Examples of IDSs in real life

- Car alarms
- Fire detectors
- House alarms
- Surveillance systems

What should be detected?

- Attempted and successful break-ins
- Attacks by legitimate users
  - For example, illegitimate use of root privileges
  - Unauthorized access to resources and data
- Trojan horses
- Viruses and worms
- Denial of service attacks

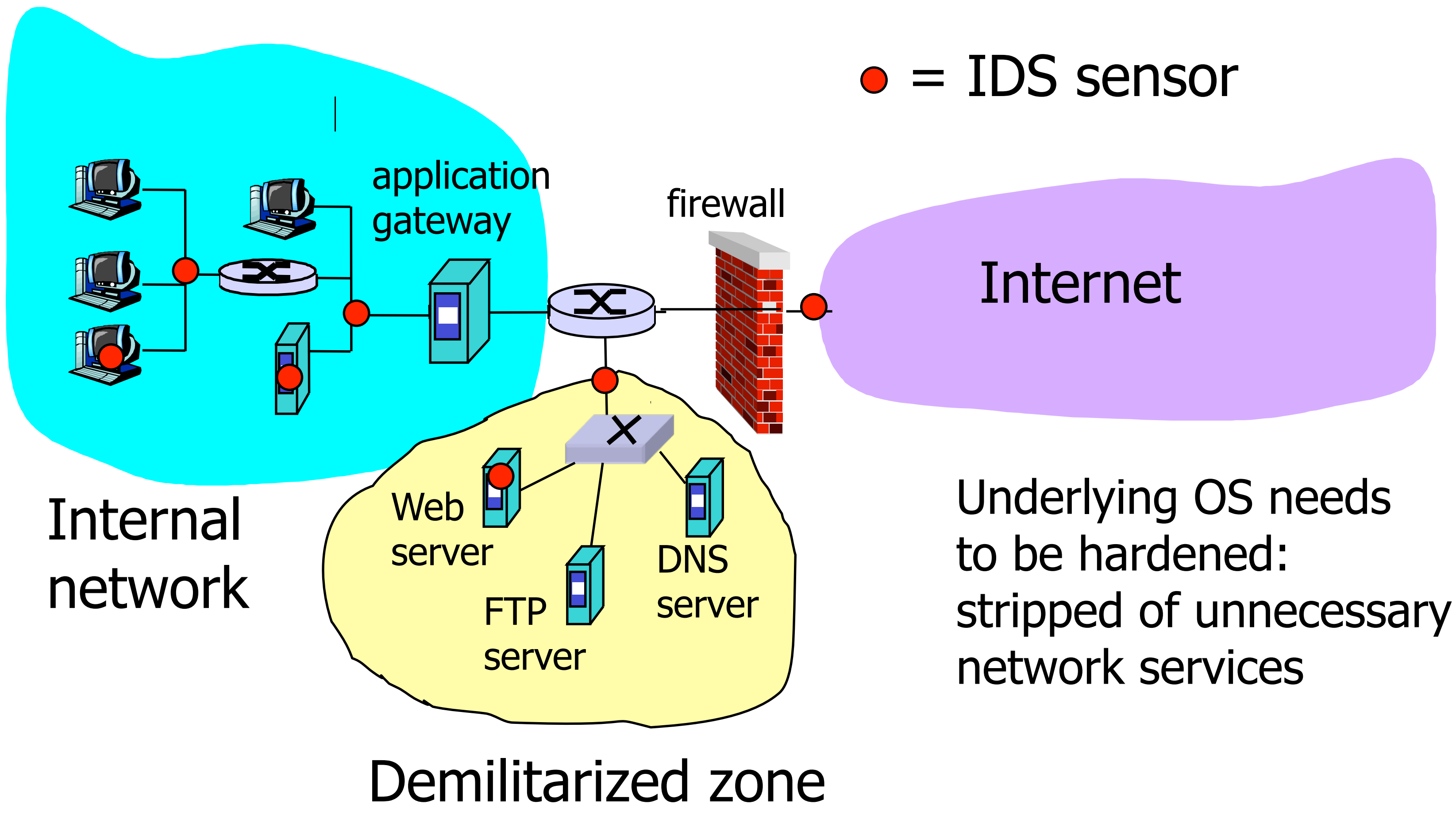
Where are IDS deployed?

- Host-based
  - Monitors host activity
  - Advantage: visibility of individual applications on host
  - Disadvantage: attackable from host
- Network-based (NIDS)
  - Often placed on a router or firewall
  - Monitor traffic == examine pkt headers/payloads
  - Advantages:
    - Single NIDS for many hosts
    - Can look for global patterns
  - Disadvantage: Has to reverse engineer app. behavior

Intrusion detection techniques

- Misuse detection
  - Use attack "signatures" (need a model of attack)
    - Sequences of system calls, patterns of network traffic, etc.
  - Must know what attacker will do (how?)
  - Can only detect known attacks
- Anomaly detection
  - Tries to detect deviations and abnormalities based on a model of normal system behavior
  - Can detect unknown attacks
  - Abnormal behavior not necessarily attack
- Most IDS use a mix of both, although misuse detection dominates

Possible IDS deployments



Underlying OS needs to be hardened: stripped of unnecessary network services

Misuse vs. anomaly

❒ Password file modified	Misuse
❒ Four failed login attempts	Anomaly/Misuse
❒ Failed connection attempts on 50 sequential ports	Anomaly/Misuse
❒ User who usually logs in around 10am from Berlin dorm logs in at 4:30am from a Russian IP address	Anomaly
❒ UDP packet to port 1434	Misuse
❒ "DEBUG" in body of a SMTP message	Most likely: not an attack!

Misuse detection (signature-based)

- Rules that define a behavioral signature associated with certain attacks
  - Example: buffer overflow
    - Setuid program spawns shell with certain arguments
    - Packet with lots of NOPs
    - Very long argument to string function
  - Example: SYN flooding (Denial of Service)
    - Large number of SYN packets without ACKs coming back
- Attack signatures disadvantage:
  - Very specific
  - May miss variants of known attacks
  - Hard for unknown attacks

Extracting misuse signatures

- Use invariant characteristics of known attacks
  - Bodies of known viruses and worms
  - Port numbers of apps with known buffer overflows
  - Return addresses of overflow exploits
  - Hard to handle mutations
    - Polymorphic viruses: each copy has different body
- Disadvantages (research challenges):
  - No knowledge of intention of activity
  - Large signature sets (=> performance issues)
  - Fast, automatic extraction of new attack signatures
  - Honeypots: Easy targets to attract malicious activity
    - Useful for signature extraction

Anomaly detection

- Based on deviation from normal behavior
- Define profile of "normal" behavior
  - Good for "small", well-defined systems (single program vs. multi-user OS)
- IDS flags deviations from the "normal" profile
  ⇨ Abnormal behavior might or might not be attack
- Profile can be statistical
  - Build manually (hard)
  - Use machine learning/data mining techniques
    - Log activities for some time
    - "train" IDS to differentiate normal and abnormal patterns
    - Risk: attacker trains IDS to accept his activity as normal
      e.g., low-volume port scan may train IDS to accept port scans

What is a "profile?"

- Login/session activity
  - Frequency; last login; password failures; elapsed time; session output, CPU, I/O
- Command/program execution
  - Frequency; program CPU, I/O, other resources (watch for exhaustion); denied executions
- File access activity
  - Read/write/create/delete frequency; failed reads, writes, creates, deletes; resource exhaustion
- How can that be done in a scalable manner?

Efficiency of IDS systems

- Accuracy:
  - Proper detection of attacks
  - Absence of false alarms
  - Trade-off between those two goals
- Performance: Processing traffic and audit events
  - Not all IDS are able to handle traffic at Gigabit rates
  - Solution: Use multiple NIDSs; use clusters of NIDSs
- Fault tolerance: Resistance to attacks
  - Should run on dedicated hardened hosts
- Timeliness: Time elapsed between intrusion and detection

Accuracy: Intrusion detection errors

- False negatives:
  Attack is not detected
  - E.g., signature-based misuse detection
- False positives: Harmless behavior classified as attack
  - E.g., statistical anomaly detection
- Both types of IDS suffer from both error types

- Which is the bigger problem?
  - Attacks are fairly rare events
  - IDS often suffer from base-rate fallacy

Base-rate fallacy

- 1% of traffic is SYN floods; IDS accuracy is 90%
  - SYN flood classified as attack: prob. 90%
  - Benign connection classified as attack: prob. 10%
- Probability conn. flagged as SYN flood is benign?

$$\Pr(\text{benign} \mid \text{alarm}) = \ ?$$

Conditional probability

- Suppose events A and B occur with probability Pr(A) and Pr(B)
- Let Pr(AB) be probability that both A and B occur
- Conditional probability that A occurs assuming B has occurred?

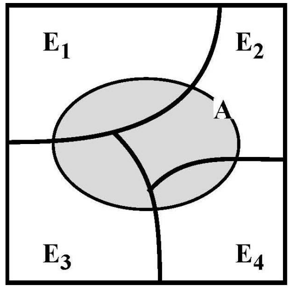
$$\Pr(A \mid B) = \frac{\Pr(AB)}{\Pr(B)}$$

Bayes' theorem

- Mutually exclusive events $E_1, \dots, E_n$
- Probability of <u>any</u> event A is

$$Pr(A) = \Sigma_{1 \le i \le n} Pr(A \mid E_i) \bullet Pr(E_i)$$

  - Intuition: whenever A occurs, some event E_i must have occurred



=>

$$Pr(E_i \mid A) = \frac{Pr(A \mid E_i) \bullet Pr(E_i)}{Pr(A)}$$

Base-rate fallacy

- 1% (=Pr(SYN flood) = 1-Pr(benign)) of traffic is SYN floods; IDS accuracy is 90%
  - SYN flood classified as attack: prob. 90% = Pr(SYN flood)
  - Benign connection classified as attack: prob. 10%
- Probability conn. flagged as SYN flood is benign?

$$\Pr(\text{benign} \mid \text{alarm}) = \frac{\Pr(\text{alarm} \mid \text{benign}) \cdot \Pr(\text{benign})}{\Pr(\text{alarm})}$$

$$= \frac{\Pr(\text{alarm} \mid \text{benign}) \cdot \Pr(\text{benign})}{\Pr(\text{alarm} \mid \text{benign}) \cdot \Pr(\text{benign}) + \Pr(\text{alarm} \mid \text{SYN flood}) \cdot \Pr(\text{SYN flood})}$$

$$= \frac{0.10 \cdot 0.99}{0.10 \cdot 0.99 + 0.90 \cdot 0.01}$$

=> 92% chance of false alarm!!!

Host-based IDS

- Monitor attacks on OSs, applications.
- Has access to audit logs, error messages, any resources that can be monitored on host
  - OS system calls
  - Command lines
  - Network data
  - Processes
  - Keystrokes
  - File and device accesses
  - Registry in Windows

Advantages

- Tuned for system/OS/apps
- High detection accuracy

Disadvantages

- Only covers one host
- IDS on every critical host
- Need versions for each OS
- Can be disabled by viruses, worms, …

Network-Based IDS

- Passively inspect network traffic
  - Watches for protocol violations
  - Unusual connection patterns
  - Attack strings in packet payloads
  - Etc.
- If we actively change traffic ⇨ Intrusion Prevention System
- Disadvantage:
  - Limited possibilities for encrypted traffic (IPSec, VPNs)
  - Not all attacks via the network
  - Large amount of traffic

Example: Port scan detection

- Many vulnerabilities are OS specific
  - Bugs in specific implementations
  - Oversights in default configuration
- Port scan often prelude to attack
  - Attacker tries many ports and/or many IP addresses
    - Looking for old versions of daemons with unpatched buffer overflows
  - Then mount attack
    - Example: SGI IRIX responds on TCPMUX port (TCP port 1)
    - If response detected use IRIX vulnerabilities to break in

Example: Port scan detection (2.)

- Scan suppression: Block traffic from addresses that have too many failed connection attempts
  - Requires network filtering, state maintenance
  - Susceptible to slow scans
- False positives possible, e.g.:
  - Web proxies
  - P2P hosts
  - Other innocent hosts due to stale IP caches, i.e., got an IP address that was previously used by P2P host

Popular open-source NIDS

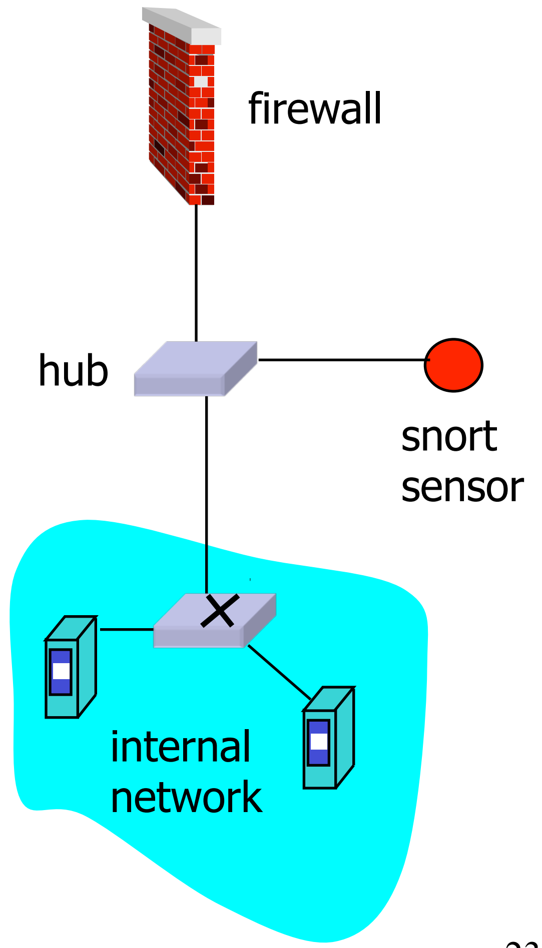
- Snort (widely deployed (unfortunately))
  - Large rule sets for known vulnerabilities, e.g.:
    - 2007-03-22: Microsoft Windows Server Service Controller is prone to a buffer overflow vulnerability that may allow an attacker to take complete control of the target host.
    - 2007-03-08: The HP Mercury LoadRunner agent suffers from a programming error that may allow a remote attacker to cause a stack-based buffer overflow condition to occur.
- Bro (from Vern Paxson at ICSI)
  - Separates data collection and security decisions
    - Event Engine distills packet stream into higher-level events
    - Policy Script Interpreter uses a script defining network's security policy to decide response

Snort

Good book: Intrusion Detection with Snort, by Jack Koziol

- Popular open source IDS
  - 200,000 installations
- Enhanced sniffer
  - Runs on Linux, Unix, Windows
  - Generic sniffing interface libpcap
- Signatures
  - Largest collection of signatures for NIDS
  - Written and released by Snort community within hours
  - Anyone can create one
  - Signature often undocumented and/or poor quality

Typical setup



Bro: a flexible NIDS

- Facts
  - Open source
  - Developed since 1995 by Vern Paxson
  - Used in many research environments, e.g., UCB, LBL, TUM, The Grid, NERSC, ESnet, NCSA
  - Supports anomaly as well as misuse detection
- Design goals
  - Reliable detection of attacks
  - High-performance
  - Separation of base functionality from site specific security policy
  - Robust against attacks on itself

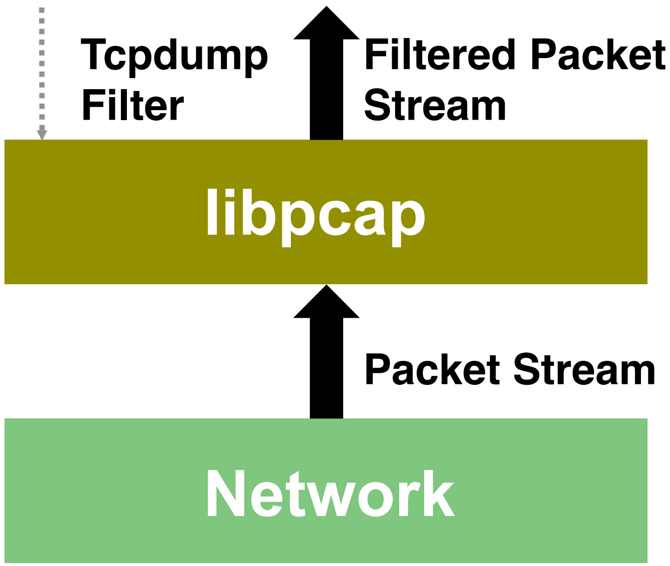
Bro features

- Full TCP stream reassembly
- Stateful protocol analysis
- Can import (some) SNORT signature rulesets
- Dynamic Protocol Detection
- BinPAC – a network protocol description language
- Very flexible policy scripting language (called Bro as well)
  - Specialized for traffic analysis
  - Strongly typed for robustness (conn_id, addr, port, time, …)
  - Can trigger alarms and/or program execution
  - Supports dynamic timeouts
- Clustering support for analysis of multi Gbps links
- Cooperates with Network Time Machine

Inside Bro

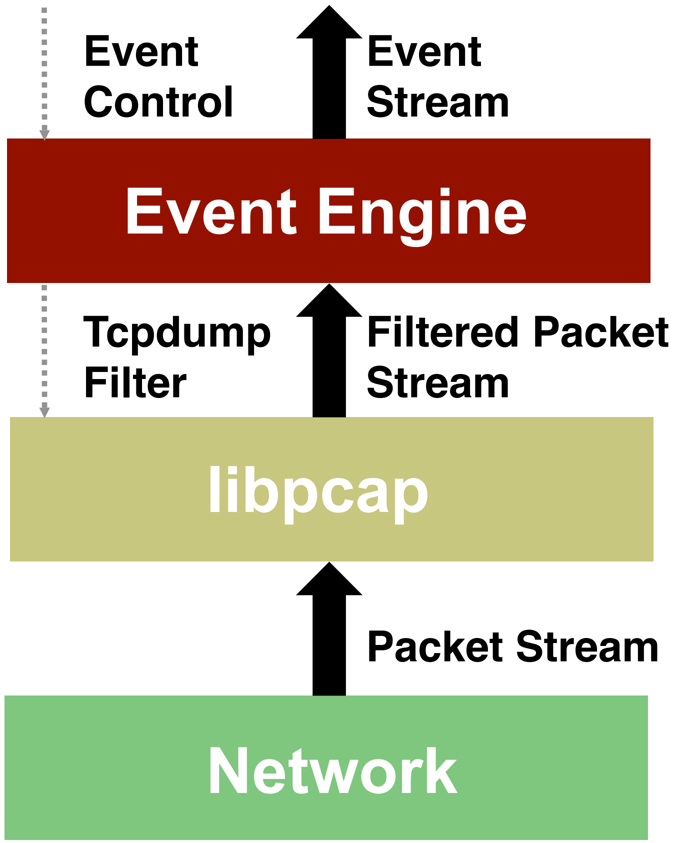
- Passive link tap copies all traffic

Inside Bro



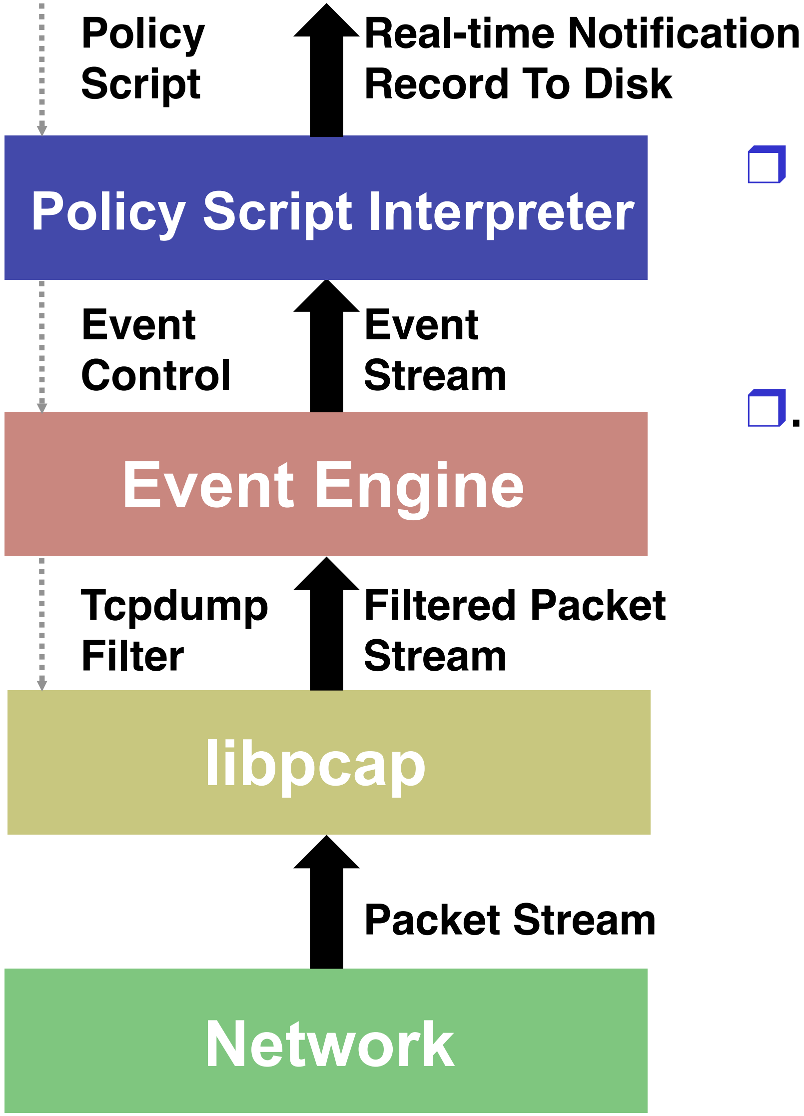
- Kernel filters high-volume stream

Inside Bro



- "Event engine" produces *policy-neutral* events, e.g.:
  - Connection-level:
    - connection attempt
    - connection finished
  - Application-level:
    - ftp request
    - http_reply
  - Activity-level:
    - login success

Inside Bro



- "Policy script" incorporates:
  - Context from past events
  - Site's particular policies
- … and *takes action*:
  - Records to disk
  - Generates alerts
  - Executes programs as <u>response</u>

Bro's protocol analyzers

- Full analysis
  - HTTP, FTP, telnet, rlogin, rsh, RPC, DCE/RPC, DNS, Windows Domain Service, SMTP, IRC, POP3, NTP, ARP, ICMP, Finger, Ident, Gnutella, BitTorrent, NNTP
- Partial analysis
  - NFS, SMB, NCP, SSH, SSL, IPv6, TFTP, AIM, Skype
- In progress
  - BGP, DHCP, Windows RPC, SMB, NetBIOS, NCP, …
- Data sources
  - DAG, libpcap, NetFlow

Protect your NIDS

Sourcefire Snort Remote Buffer Overflow

- Notification Type: IBM Internet Security Systems Protection Advisory
- Notification Date: Feb 19, 2007
- Description: Snort IDS and Sourcefire Intrusion Sensor IDS/IPS are vulnerable to stack-based buffer overflow, which can result in remote code execution.

… patched since then

Attacking and evading NIDS

- Looking for patterns / signatures seems pretty easy and straightforward
- But .....

Attacking and evading NIDS

- Attackers do not want to be detected by IDS
  - Often attackers are intimately familiar with popular IDS products, including their weaknesses
- Ideas:
  - Overload NIDS then attempt the intrusion
    - E.g., huge workload, packets requiring detailed analysis, massive SYN floods
  - Manipulate attack data
    - Use encryption to hide packet contents
    - Use data fragmentation (either physical or logical)

NIDS evasion: Fragmentation

- Send flood of fragments
  - May saturate NIDS
  - Once saturated, NIDS unable to detect new attacks
- Fragment packets in unexpected ways (possibly violating RFCs)
  - NIDS may not understand how to properly reassemble attack packets
  - Network stacks are resilient => will try and often succeed
  - Network stack may reassemble fragments differently (OS dependent) => state explosion

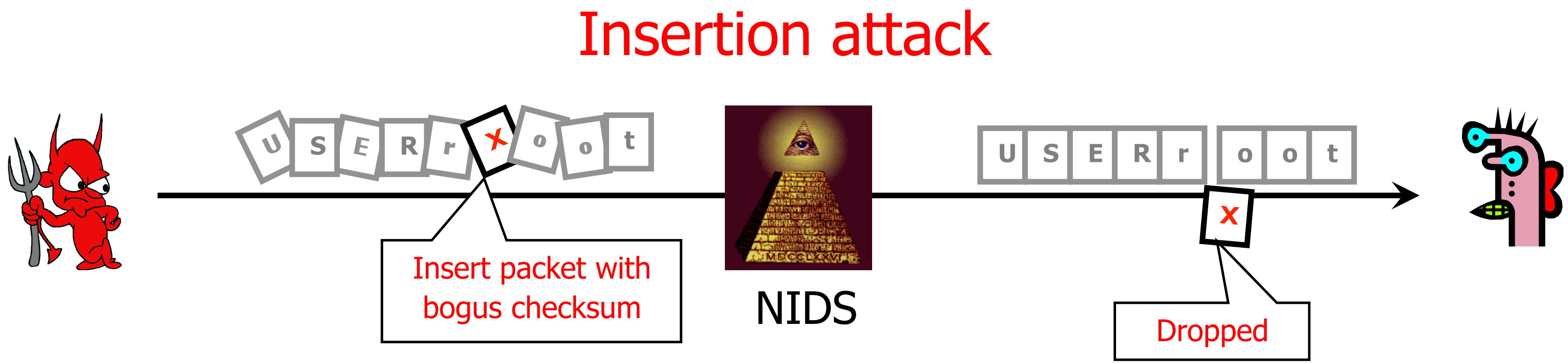
Example: Fragment overlap attack

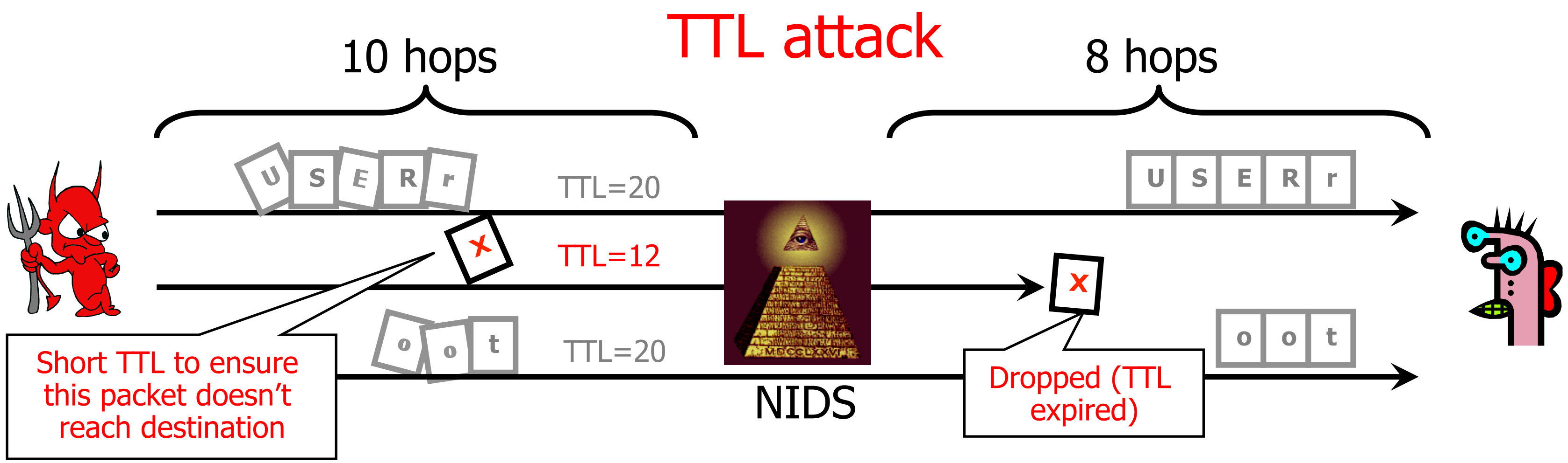
- Attacker uses two fragments for every attack datagram
  - First fragment: TCP header, incl. port number of innocuous service not monitored by NIDS
  - Second fragment: offset value overlaps with original and includes a different port number
- IDS might let both fragments pass:
  - First fragment to innocuous port
  - Second fragment part of same "good datagram"
- Once the two fragments arrive at target host:
  - IP reassembles datagram, possibly overwriting TCP header with port in fragment 2
  - Malicious segment delivered to monitored port!

Example: Payload ambiguity

- Want to detect "USER root" in packet stream
- Scanning every packet is not sufficient
  - Attacker can split attack string into several packets; defeats stateless NIDS
- Recording previous packet is not sufficient
  - Send packets out of order
- Full reassembly of TCP state is not sufficient
  - Attacker can use TCP tricks, e.g.:
    - Certain packets seen by NIDS but dropped at receiver
    - Manipulate checksums, TTL (time-to-live), fragmentation
    - Segment reassembly differs by OS
  - Use of application layer protocol polymorphism

NIDS evation:





Solving evasion: Easy?

- Just flag everything that's weird
  - E.g., Overlapping fragments
- Golden rule of protocol implementation: "be strict in what you send but liberal in what you accept"
  - Advantage: the Internet works
  - Impact: Lots of crud seen in every network:
    - Violation of RFCs but it still works
  - Problem for IDS, since it cannot flag weird stuff
- Different OSes, browsers, implementations handle crud differently
  - Impossible for the IDS to know how exactly a receiver is going to react

Developing an IDS: Intrusion detection problems

- Lack of training data with real attacks
  - But lots of “normal” network traffic, system call data
  - "Ground truth"
- Data drift
  - Statistical methods detect changes in behavior
  - Attacker can attack gradually and incrementally
- Main characteristics not well understood
  - By many measures, attack may be within bounds of “normal” range of activities
- False identifications are very costly
  - Sysadmin will spend many hours examining evidence