UNIVERSITÀ DEGLI STUDI
DI TRENTO

Network Security

AA 2015/2016

System hardening

(IDS, Vuln Management)

Dr. Luca Allodi

Some slides from M. Cremonini

Intrusion Detection Systems

Function of an IDS

- Firewalls prevent unwanted access to network resources that should be isolated w.r.t. another network
- IDS monitors incoming connections
  - Depending on its position in the network may provide different functionalities
    - More on this later
- Intrusion Prevention Systems (IPS) can act over "malicious" behaviour
- IDS → passive monitoring
- IPS → active monitoring
- In reality functionalities are not entirely distinct
  - Commercial lingo rather than actually different technology

IDS – 3 phases

1. Data collection
   - Host-based IDS → Sit on an host (client, server)
   - Network-based IDS → Collects network data
2. Data analysis
   - Two distinct approaches
   - Misuse detection → list unwanted behaviour, report if detected
   - Anomaly detection → build average profile, report if current activity significantly different from average
3. Action
   - IDS → report, log entry
   - IPS → report, log entry, block/alert

Misuse detection

- IDS equivalent of “default allow” policies
- ”blacklist” patterns that are believed to be related to malicious activities
  - System calls
  - Payloads in network protocols
- Signature-based
  - Very diffused detection technique
  - Easy to deploy
  - Typical implementation for network-based IDSs
- As all blacklisting approaches (signature-based) it can only detect patterns that are *already known*

Anomaly detection

- Assumes intruder behaviour differs from legitimate profile
- Building legitimate profile may be an issue
  - Depends on data used for profiling (e.g. sampled vs whole dataset)
  - Profile can evolve → new "legitimate activity" looks suspicious
- Can be used both for HIDS and NIDS
  - HIDS → syscall, system file hashing, system states, ..
  - NIDS → protocol analysis, similar to application proxy
    - Monitoring as opposed to filtering

Network IDS

- Baseline implementation is of type *misuse detection*
  - Easier to implement
  - Network traffic is hard to predict even on well-controlled environments
- Signature example:

```
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
```

The base-rate fallacy – or, can we have actually good detection rates?

- Both anomaly and misuses detection necessarily lead to false positives and false negatives
- A NIDS with 99% true positive rate and 99% true negative rate seems to have high-reliability alarms
  - → an alarm fires up → you should worry
  - → no alarm fires up → all is good
  - But is it?
- Base-rate fallacy
  - Simple derivation from Bayes theorem
  - Very well known by medics and doctors
  - Still making its way through in InfoSec

The base-rate fallacy [Axelsson 2000]

- Tests with high true positives and negatives rates yield much "worse" results than expected by the average user
- Remember Bayes theorem

$$P(A|B) = \frac{P(A) \cdot P(B|A)}{\sum_{i=1}^{n} P(A_i) \cdot P(B|A_i)}$$

This is P(B) expanded to all "n" cases for A that B comprises

- Let's make the classic medical example
  - Attack = illness
  - IDS Alarm = medical test

Base-rate fallacy example

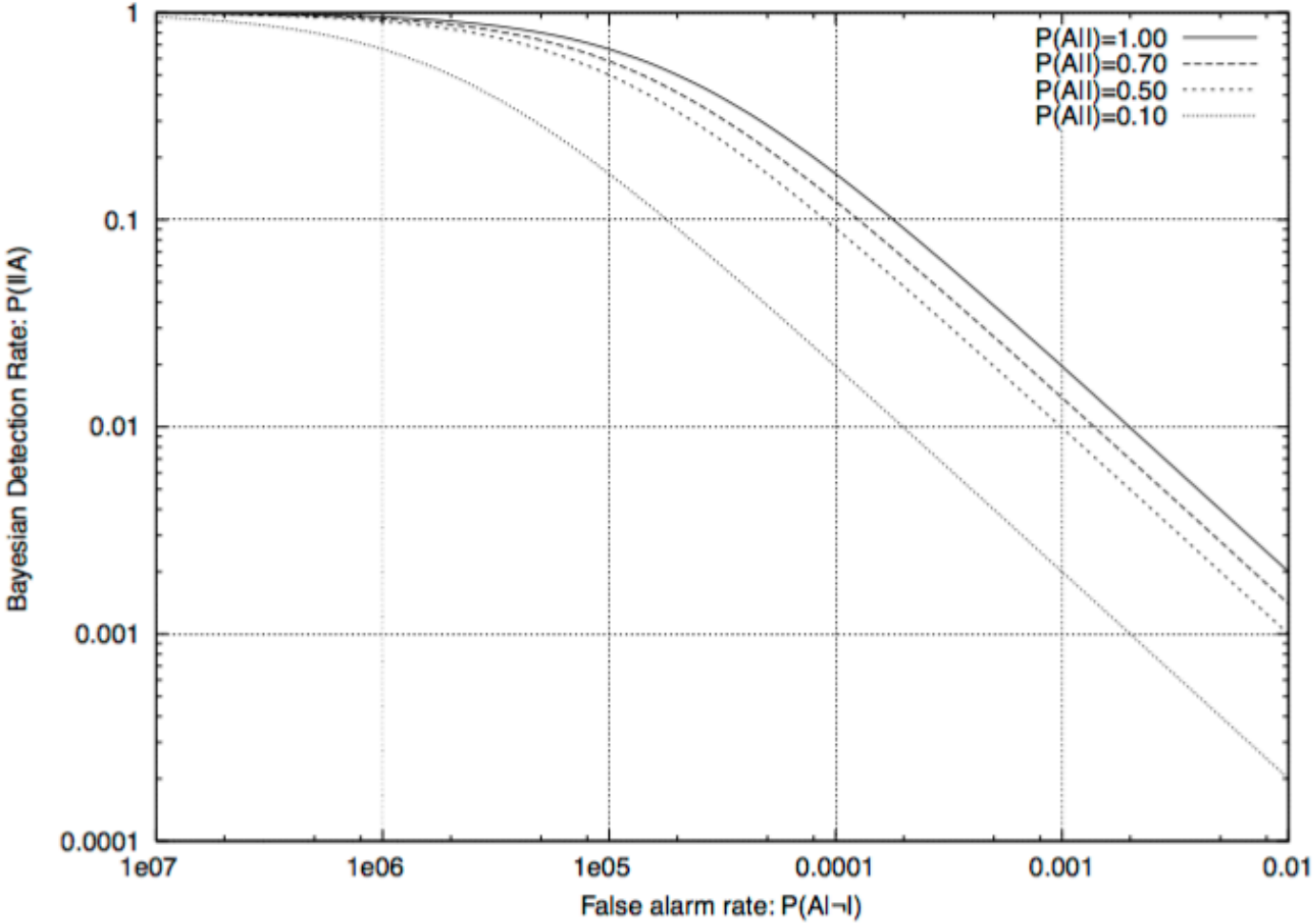
$$P(A|B) = \frac{P(A) \cdot P(B|A)}{\sum_{i=1}^{n} P(A_i) \cdot P(B|A_i)}$$

- A=event is *patient is sick*
- B=medical test says patient is sick
- P(A|B) = patient is actually sick given that test said so
  - Equivalent to "there is an actual attack given that NIDS fired alarm"
- Set TP=99%; TN=99% → P(B|A) = 0.99
- Diseases are rare. Say 1/10.000 people have the illness → P(A)=1/10.000
  - Most network traffic is legitimate

$$P(A|B) = \frac{1/10000 \cdot 0.99}{1/10000 \cdot 0.99 + (1 - 1/10000) \cdot 0.01} = 0.00980... \approx 1\%$$

- There is only 1% chance that patient is sick when test says so
  - An alarm is not very meaningful → IDS alarms are hard to manage → log analysis

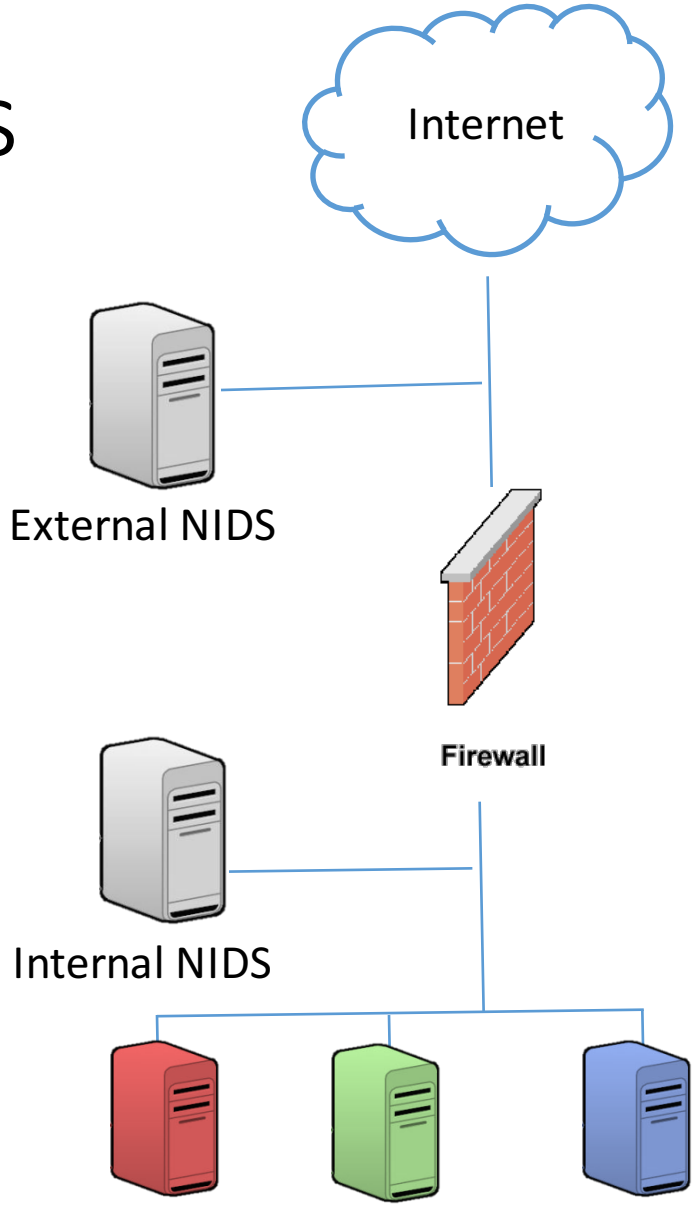
Base-rate fallacy and IDSs



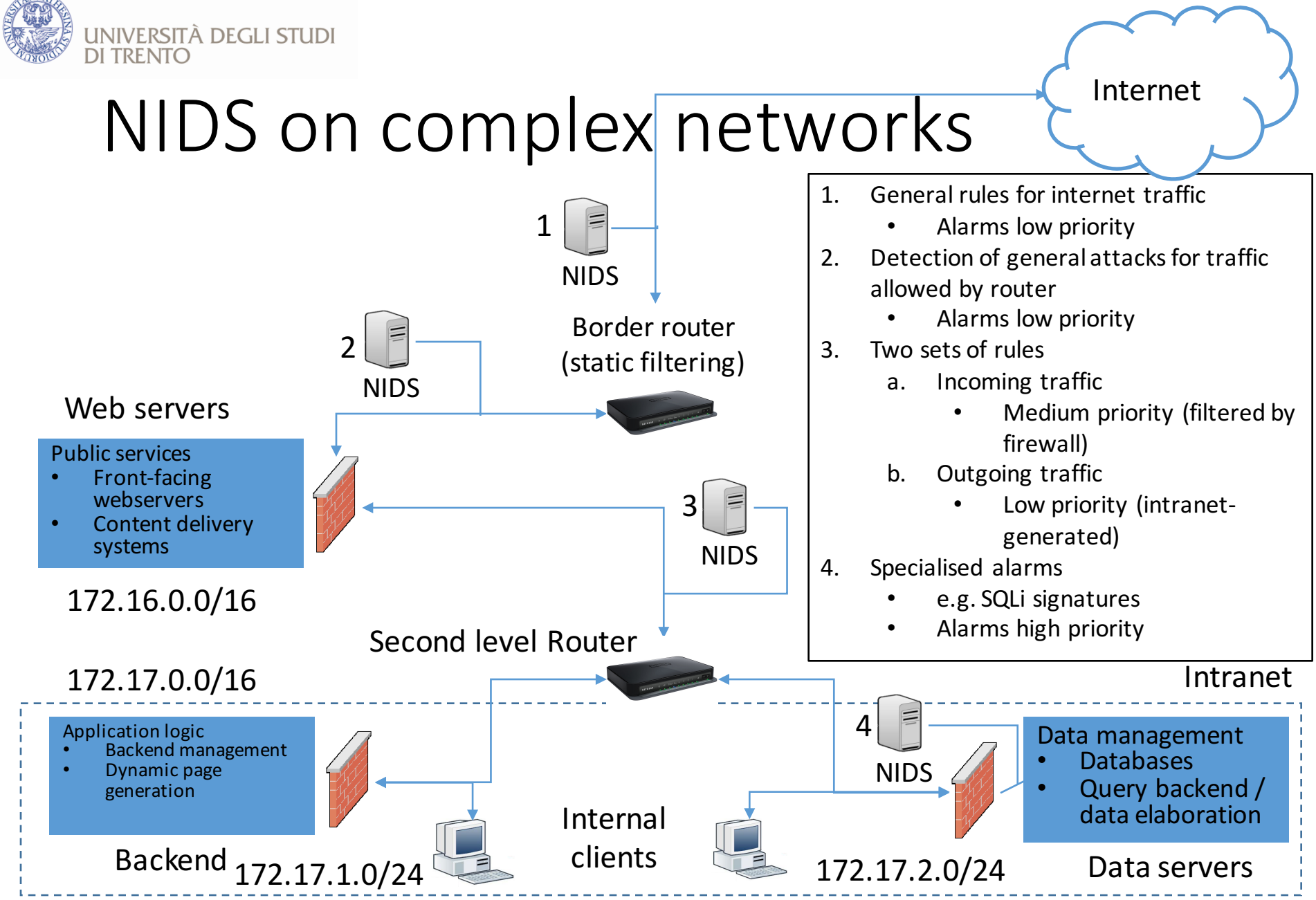
Notice that the false positives rate is the one that dominates the curve

Architectural aspects

- External NIDS
  - Analysis of all set of incoming traffic
  - Only general signatures are possible
    - high incidence of FP
  - All detected "attempted attacks" are logged
  - "normal" Internet traffic may generate many alarms
- Internal NIDS
  - Analysis of traffic allowed by the firewall
  - More specific signatures are possible
    - e.g. based on services behind firewall, subnet characteristics, ..
  - Says nothing about attacks attempted but blocked by firewall



NIDS on complex networks



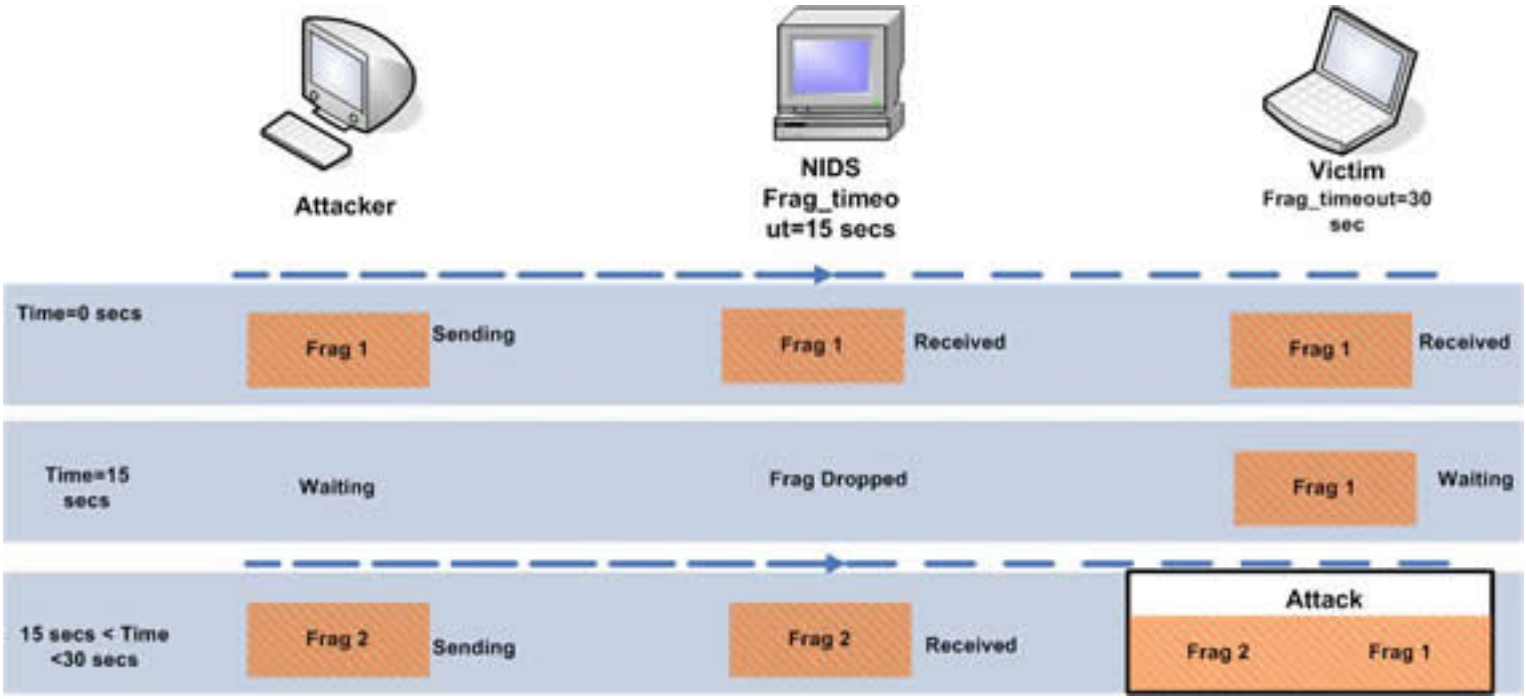
1. General rules for internet traffic
   - Alarms low priority
2. Detection of general attacks for traffic allowed by router
   - Alarms low priority
3. Two sets of rules
   a. Incoming traffic
      - Medium priority (filtered by firewall)
   b. Outgoing traffic
      - Low priority (intranet-generated)
4. Specialised alarms
   - e.g. SQLi signatures
   - Alarms high priority

NIDS evasion [Siddharth 2005]

- Signature-based evasion can be fairly trivial
- Depends on implementation of actual signature
  `content:"/bin/bash"`
  - → detects remote calls to bash
  - Does not detect string "/etc/../bin/bash", etc.
- More advanced techniques are typically based on IP fragmentation
  - All techniques have common goal: NIDS sees different packet than client
  - Look at these keeping in mind you may want to prevent the attacker from performing
    - Network mapping
    - OS fingerprinting

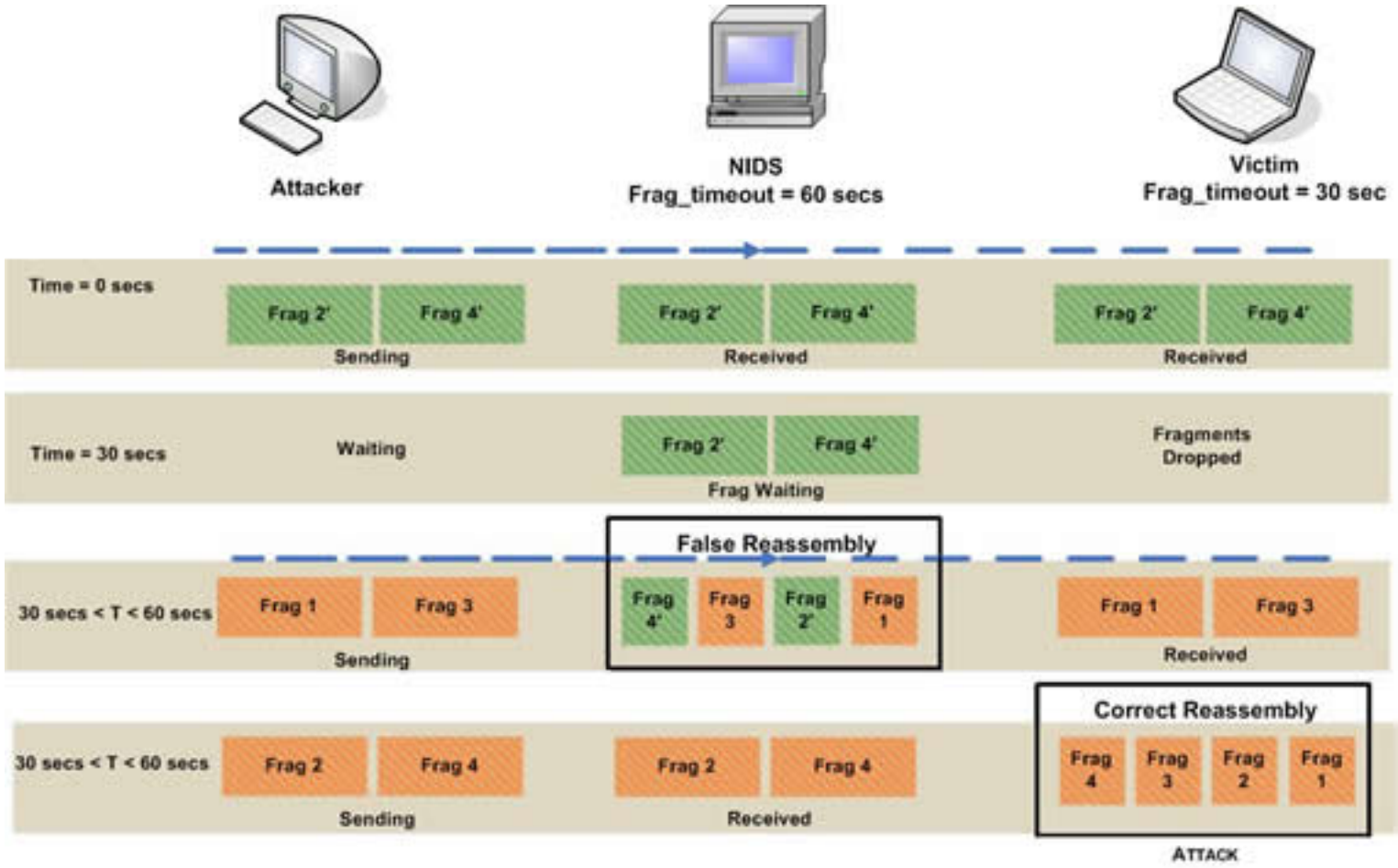
Evasion technique – Reassembly time-out

- NIDS has lower reassembly timeout than receiving client



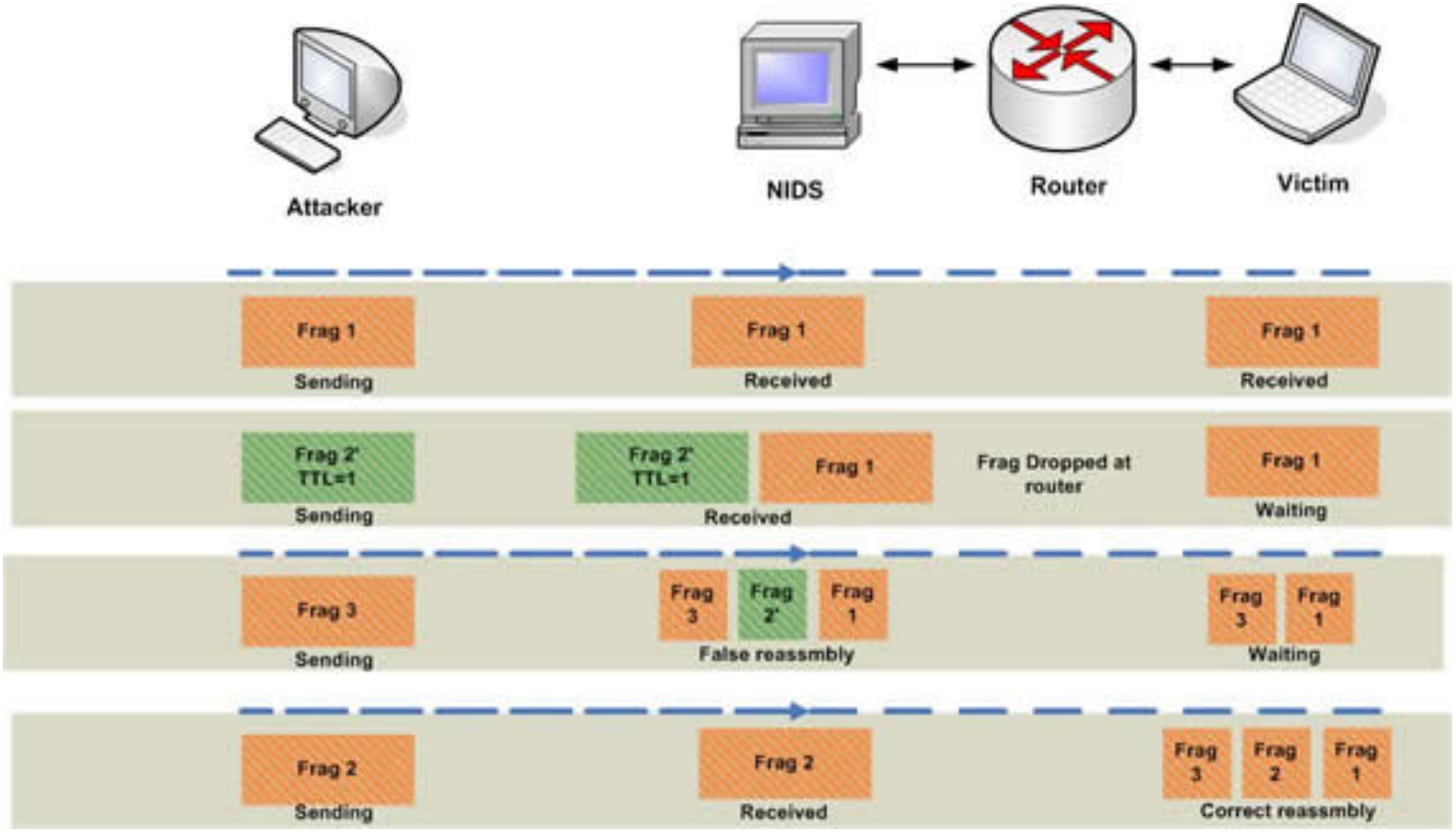
Evasion technique – Reassembly time-out (2)

- NIDS has higher reassembly timeout than receiving client



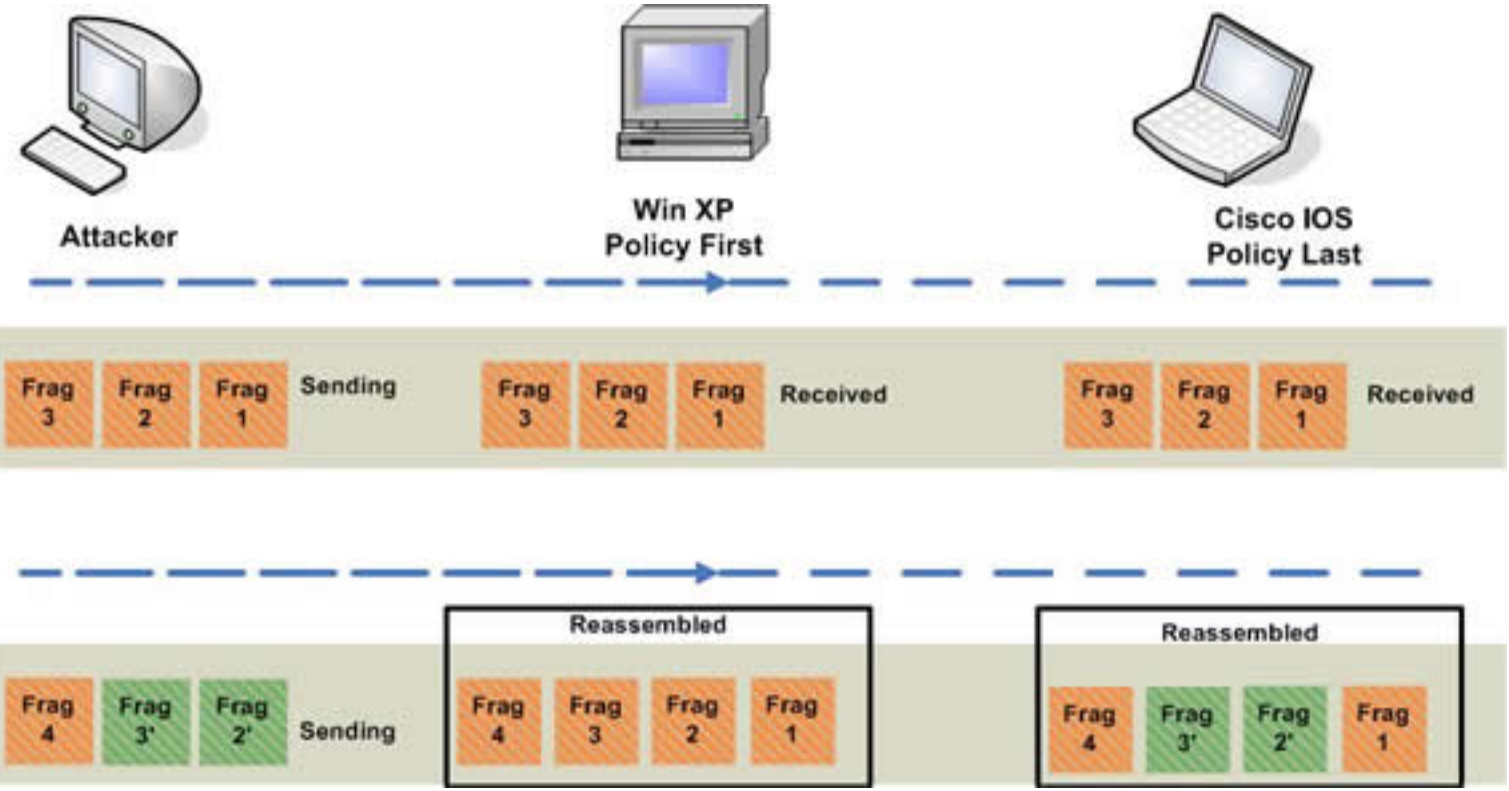
Evasion technique – Time-to-live

- Router drops packet analysed by NIDS that will not be delivered to victim



Evasion technique – Fragment replacement

- Some operating systems replace fragments with newer ones, others keep old fragments



Suggested reading

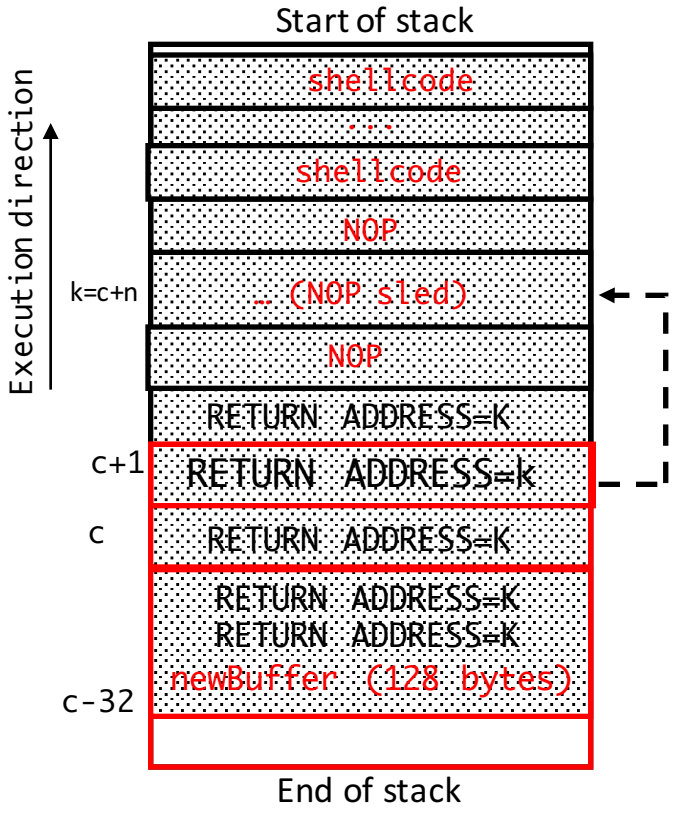
- Wool, Avishai. "A quantitative study of firewall configuration errors." *Computer* 37.6 (2004): 62-67.
- Axelsson, Stefan. "The base-rate fallacy and the difficulty of intrusion detection." *ACM Transactions on Information and System Security (TISSEC)* 3.3 (2000): 186-205.
- [Siddharth 2005] http://www.symantec.com/connect/articles/evading-nids-revisited

Vulnerability mitigation

Attack surface minimisation in practice - recap

- Network hardening
  - Firewalls → block unwanted traffic
    - Default allow → easier configuration, less secure in general
    - Default deny → can cause disservices for the users, high security
  - IDS → analyse traffic payload to check for malicious packets
    - Misuses detection → signatures that match known payloads
    - Anomaly detection → signals behaviour (host, network) significantly different from expected
- System hardening
  - "can't break what's not there" → trim system configuration to only allow actions that are needed for system functionality
  - Authentication → minimise set of user actions to minimal
  - **Open vulnerabilities represent a risk of incoming attacks**
    - Vulnerabilities patches not always (immediately) possible
    - Mitigation techniques

OS vulnerability mitigation – BoF vs DEP protection



- Buffer overflow
  - attacker can overwrite data in stack with executable shellcode
  - Redirect execution to shellcode
- But in stack there should never be code, only data
- **Data Execution Protection (DEP)**
  - Data areas in memory are marked as **non-executable**
  - Hw support → AMD NX bit, Intel XD bit
  - Defeats code execution via stack corruption
  - Does not prevent corruption of Heap or redirection to other functions in memory

OS vulnerability mitigation- BoF vs ASLR

- With DEP attacker can still redirect execution to code areas in memory
  - e.g. write a stack frame in memory and point to lib-c or other known functions (that are of course executable)
- Most memory corruption attacks rely on the attacker being able to guess start address of stack frame/heap/other areas in memory
  - e.g. write n bytes with n=offset between buffer and RET
- **Address Space Layout Randomization → ASLR**
  - Randomise location in memory of stack, heap, libraries
  - Randomisation happens in a n-bits space
    - Windows Vista → 8 bit → 1/256 guesses work
    - Linux → ExecShiel/PaX → 16 bits

DEP + ASLR

- DEP → prevents execution of data in memory
  - Can still jump to existing libraries
- ASLR → makes it more difficult for the attacker to correctly guess memory address of libraries
  - In some cases (e.g. low memory, older implementations) still possible to make a guess
- Advanced exploitation techniques redirect execution to existing code in memory
  - Return Oriented Programming → Turing-complete
  - Bypass DEP
  - ASLR can be bypassed too (most applications run sw modules in non-randomised memory areas)
- DEP+ASLR should be used together
  - Not perfect protection
- → Vulnerability patching

Vulnerability patching

- Software patch fixes vulnerability in code
- ”Just install the patch” approach does not always work well
  - OS patches often require system reboot
  - A patch modifies software code
    - Software functionalities may change
    - Deprecated third-party libraries
  - Production systems need to be up and running
    - Can’t always install patch
    - Test patch before install
- Vulnerability patching is costly process
  - “get rid of all vulnerabilities” is not always viable

Counting vulnerabilities != security assessment

- More vulnerabilities do not translate directly into "risk of attack"
- We already know that vulnerabilities enable threat scenarios with a certain impact and a certain probability
  - Risk != sum_v(severity_v)
    - **CVSS measures severity**
  - Risk = f(impact x likelihood)
    - **CVSS does not measure risk**
- Yet, security status is often measured by how many vulnerabilities we have
  - Symantec Threat report 2015
  - Secunia Vuln report 2011-2015
  - *"The grayed out section represents the vendor with the **worst security of the month**."*

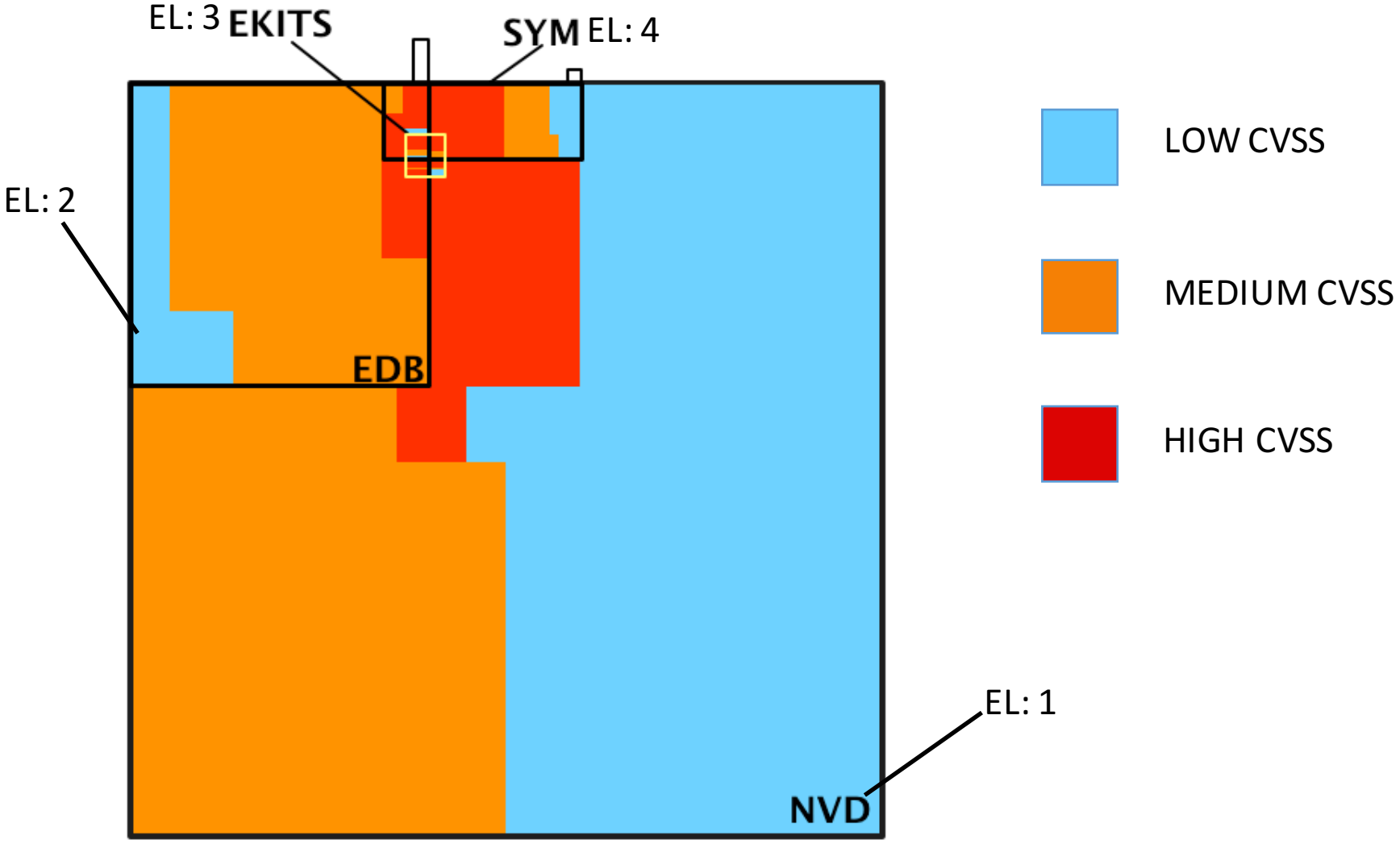
→

	Mac OS X						Windows XP					
Month	Extreme	Unpatched	High	Unpatched	Moderate	Unpatched	Extreme	Unpatched	High	Unpatched	Moderate	Unpatched
Feb-06	1	1							1		1	
Jan-06									1			
Dec-05			1				1				1	1
Nov-05			13						2			
Oct-05			10						2		10	
Sep-05					5							
Aug-05			35						1		3	
Jul-05					2		1				1	
Jun-05			15						4		3	
May-05			19		5							
Apr-05					8				1		2	
Mar-05			14								5	
Feb-05			1						5		2	
Jan-05			12						1			
Dec-04			30						4	1	3	
Nov-04												
Oct-04			8						7		2	
Sep-04					16				1			
Aug-04			7									
Jul-04									2		2	
Jun-04												
May-04					8				1			
Apr-04	4		8		5				15		5	
Mar-04					1							
Feb-04					9				1		1	1
Total	5	1	173	0	59	0	2	0	49	1	41	2

Do we need to patch all vulns?

- Let's look at the numbers
- Exploitation Level = EL
- EL1 → NVD: vulnerability is disclosed
- EL2 → EDB: Exploit-DB, PoC exists and is public
- EL3 → EKITS: dataset collected @ UniTn, infiltration in underground markets
  → exploit is traded in the Russian Cybercrime Markets
- EL4 → SYM: vulnerability is reported as exploited in Symantec's Threat Explorer dataset (at least one exploit has been detected)
- EL5 → WINE: Symantec dataset of detected attacks in the wild over more than 1M sensors

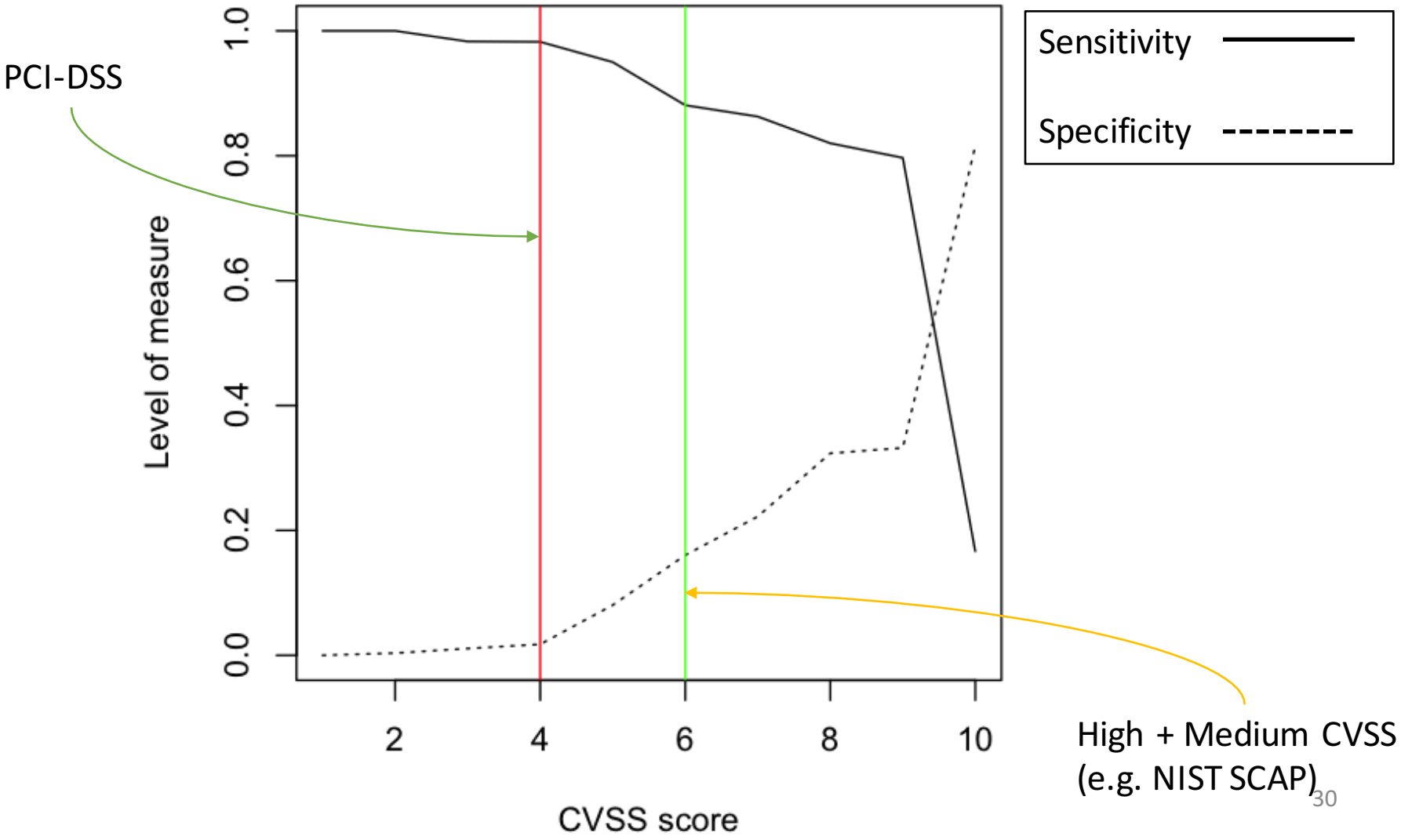
CVSS vs exploitation levels



How to evaluate a risk metric

- Much like we did before to evaluate effectiveness of IDS alarms
  - Evaluate true and false positives vs all alarms
- **Sensitivity** → true positives vs all "sick people"
  - HIGH→ the test correctly identifies exploited vulns
  - LOW → lots of “sick people” undetected
- **Specificity** → true negatives vs all healthy people
  - HIGH → the test correctly identifies non exploited vulns
  - LOW → lots of “healthy people” flagged

CVSS versus risk of exploitation



Numerical examples

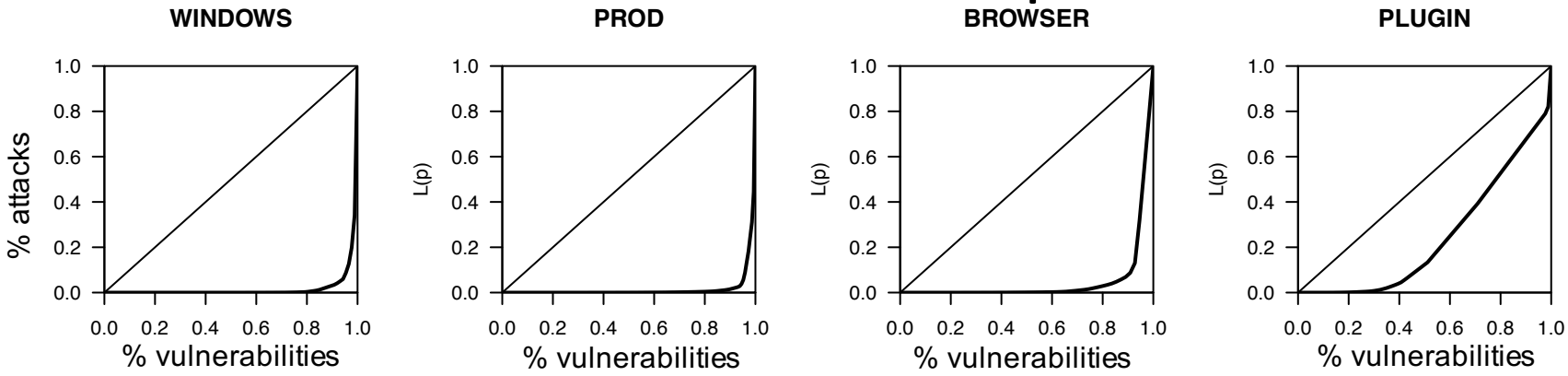
Test for Patching	Sensitivity	Specificity
Patch Everything	100%	0%
CVSS High+Med	91%	23%
CVSS + PoC in EDB	97%	22%
CVSS + EKITS	94%	50%
3BT: Down Syndrome	69%	95%
PSA: Prostate Cancer	81%	90%

CVSS does not correlate with risk, but how is risk distributed?

- Here we are at EL5
- Evaluate overall number of attacks in the wild
  - How many attacks does a vulnerability drive on average?
  - Answer is in next slide

Vuln. Category	Sample software names	No. of vulns	Attacks (Millions)
PLUGIN	Acrobat reader, Flash Player	86	24.75
PROD	Microsoft Office, Eudora	146	3.16
WINDOWS	Windows XP, Vista	87	47.3
BROWSER	Internet Explorer	55	0.55
Tot:		374	75.76

Distribution of attacks per vuln



- Lorentz curve of attacks per vulnerability
  - x-axis = percentage of vulnerabilities receiving an L(p) fraction of attacks
- All categories but **PLUGIN** see 10% of vulnerabilities responsible for 90%+ of attacks
- Example for **PROD:**
  - 7 vulnerabilities receive 3.000.000 attacks
  - 139 vulnerabilities receive 100.000 attacks

Category	Top p% vulns.	$L(p)$% of attacks
WINDOWS	20%	99.6%
	10%	96.5%
	5%	91.3%
PROD	20%	99.5%
	10%	98.3%
	5%	94.4%
BROWSER	20%	97.1%
	10%	91.3%
	5%	68.2%
PLUGIN	20%	46.9%
	10%	31%
	5%	24%

Risk of vulnerability exploitation - recap

- Some vulnerabilities are exploited several order of magnitude more than the ”average” vulnerability
  - Risk = likelihood of exploitation x impact of exploitation
- Risk is not uniformly distributed
  - CVSS measures vulnerability severity
    - **Does not make a claim to estimate exploit likelihood**
    - Currently, best available measure (worst case scenario is accounted for)
- How to calculate exploitation risk is still an open research problem
  - Technical evaluations
  - Attacker economics

Suggested reading

- Allodi, Luca, and Fabio Massacci. "Comparing vulnerability severity and exploits using case-control studies." *ACM Transactions on Information and System Security (TISSEC)* 17.1 (2014): 1.
- Nayak, Kartik, et al. "Some vulnerabilities are different than others." *Research in Attacks, Intrusions and Defenses*. Springer International Publishing, 2014. 426-446.