CS259D: Data Mining for Cybersecurity

Polymorphic Blending Attacks

Slides by Jelena Mirkovic

Motivation

- Polymorphism is used by malicious code to evade signature-based IDSs
  - Anomaly-based IDSs detect polymorphic attacks because their byte frequency differs from the one seen in the legitimate traffic
- Polymorphic blending attacks adjust their byte frequency to match that of legitimate traffic
  - Evade anomaly-based IDSs
- This paper investigates how polymorphic blending attacks can be created and proposes countermeasures

Outline

- Polymorphic attacks
- Defenses against polymorphism
- Polymorphic blending attacks
- Case study with PAYL
- Conclusions

Polymorphic Attacks

- Freely available tools for making code polymorphic
  - ADMutate, PHATBOT and CLET
- Transform assembly commands into other commands with the same semantics
- Use encryption to hide malicious code
- Insert garbage, shuffle registers
- Each instance of malicious code looks different but does not resemble normal code (CLET may come near)

Polymorphic Attacks

- Three components:
  1. Attack vector used for exploiting vulnerability of the target
     - Some parts can be modified but there is always a set of invariant parts
     - If invariant parts are small and exist in legitimate traffic detection can be evaded
  2. Attack body - malicious code for the attacker's purpose; shell code
     - Can be transformed or encrypted
  3. Polymorphic decryptor
     - Decrypts the shell code, can be transformed
- Byte frequency of malicious code should be anomalous

Related Work

- Polymorphic attacks:
  - IP/TCP transformations
  - Mutation exploits (Vigna et al.)
  - Fragroute, Whisker, AGENT, Mistfall, tPE, EXPO, DINA, ADMutate, PHATBOT, JempiScodes
  - CLET
- Defenses against polymorphism
  - Looking for executable code (Toth et al.)
  - Looking for similar structure in multiple code instances (Kruegel et al.)
  - Looking for common substrings present in multiple code instances (Polygraph) - defeated by noise
  - Looking for any exploit of a known vulnerability (Shield)

Related Work

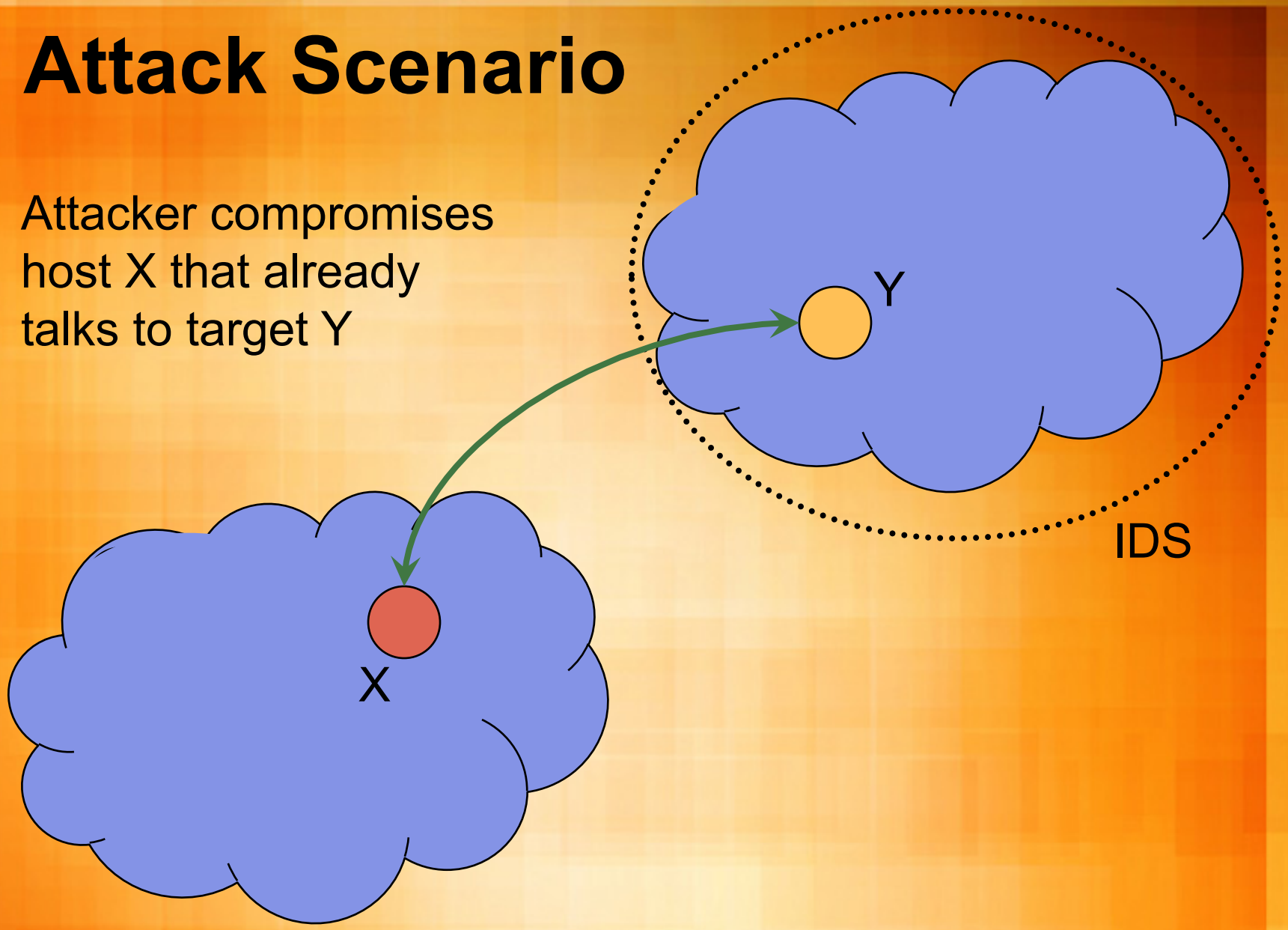
- Defenses against polymorphism
  - Looking for instruction semantics, detect known code transformations (Cristodorescu et al.)
  - Detect sequence of anomalous system calls (Forest et al.) - can be defeated through mimicry attacks.
    - New approaches use stack information but they can also be defeated
  - Payload-based anomaly detection: use length, character distribution, probabilistic grammar and tokens to model HTTP traffic (Kruegel et al.); record byte frequency for each port's traffic (PAYL)

Polymorphic Blending Attacks

- Attack can blend in if it can mimic simple statistics observed in legitimate traffic
  - Average size and rate of packets, byte frequency distribution, range of tokens at different offsets
  - To avoid PAYL the attack must carefully choose encryption key and pad its payload to replicate desired byte frequency
- There are more sophisticated approaches to attack detection that cannot be evaded so simply
  - Simple approaches are used because they are affordable at high packet speeds

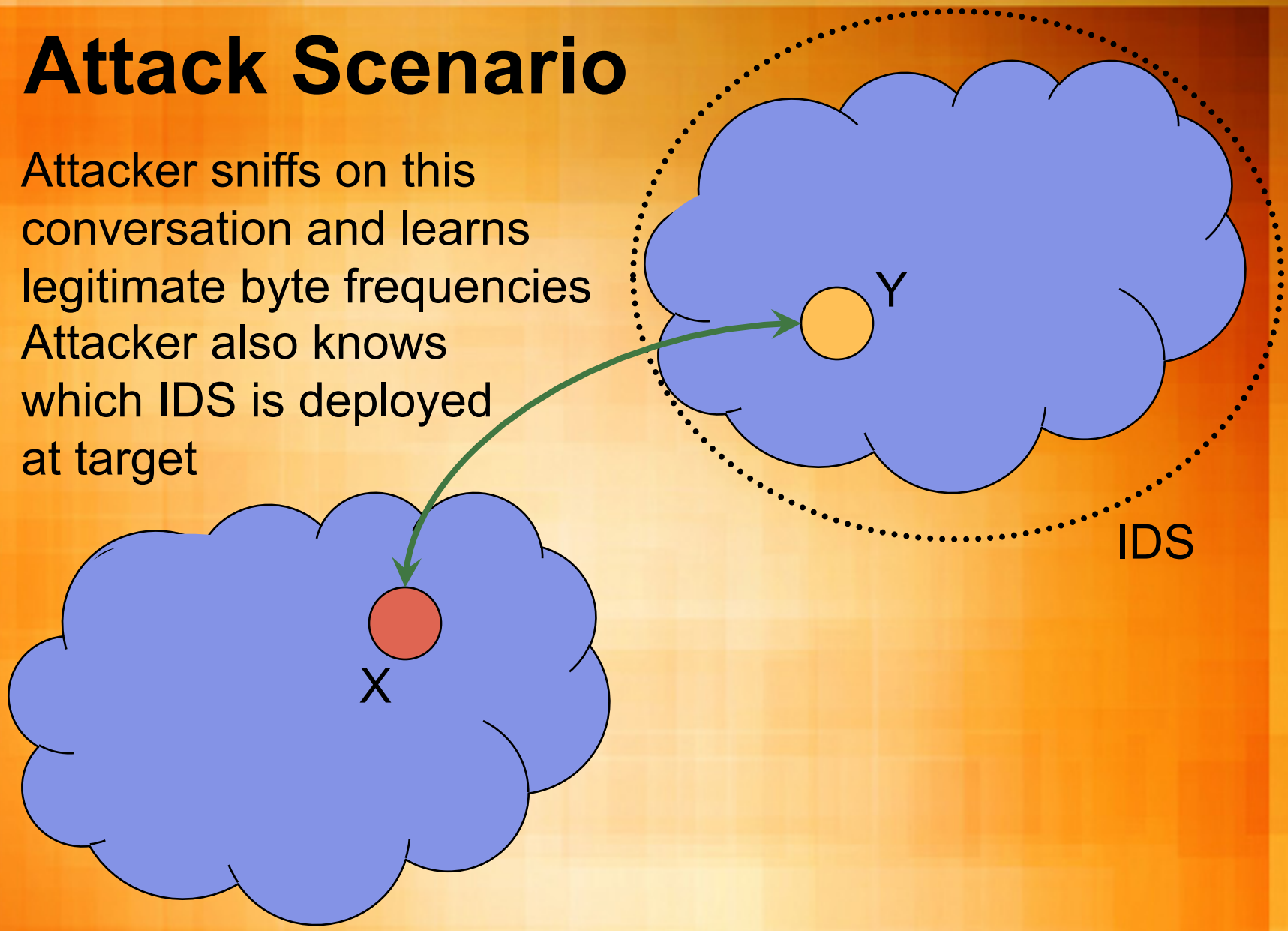
Attack Scenario

Attacker compromises host X that already talks to target Y



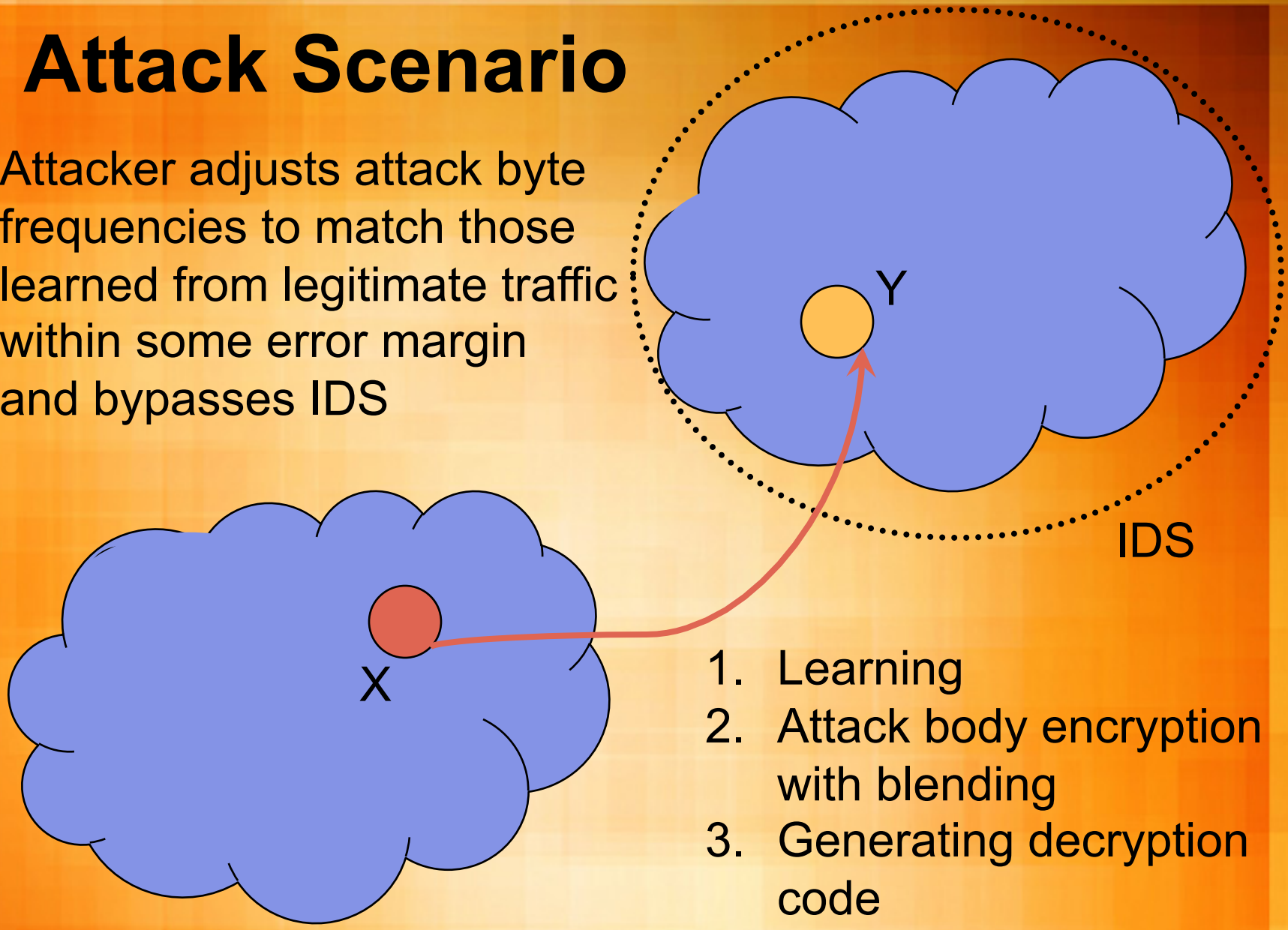
Attack Scenario

Attacker sniffs on this conversation and learns legitimate byte frequencies
Attacker also knows which IDS is deployed at target



Attack Scenario

Attacker adjusts attack byte frequencies to match those learned from legitimate traffic within some error margin and bypasses IDS



1. Learning
2. Attack body encryption with blending
3. Generating decryption code

Polymorphic Blending Attacks

- Desirable properties of an attack:
  - Match legitimate traffic's byte frequencies
  - Do not result in large attack size
  - Economical (time and memory) blending process
  - Short learning time → small traffic sample
    - Even a single packet may suffice to learn good traffic pattern for traffic of this size
    - If an attacker can sniff he can collect any amount of traffic
- For encryption use a substitution cipher
  - Each byte is transformed into a byte from a legitimate traffic sample to match desired byte frequency
  - Possibly some padding is added
  - Use greedy algorithm to create mapping

Polymorphic Blending Attacks

- Decryption removes the padding and reverses the substitution steps
  - This code cannot be blended but can be transformed into equivalent instructions
  - Decoding table can be stored in a positional array, thus code contains only legitimate characters
- Attack vector and decoding information influence the byte frequency distribution so we may need several iterations to achieve desired match
- It may happen that the IDS has different profiles for different packet lengths
  - In this case we must match the byte frequency for a given length

Evaluation

- Create polymorphic blending attacks to evade PAYL
- First create polymorphic attacks using CLET and verify that they are all detected by PAYL
- Next create polymorphic blending attacks and demonstrate that they can evade detection
  - Evaluate easy of attack construction and cost
- 1-gram and 2-gram PAYL is used to evaluate performance when IDS has more complex models

PAYL

- Measure frequency distribution of *n*-grams in traffic payloads
- Use sliding window of size *n*
- Generate a separate model for each packet length
  - Cluster models at the end to reduce memory cost
- Packets with unusual length are also flagged as anomalous
- Model consists of frequency $f(x_i)$ and stdev $\sigma(x_i)$
- Anomaly score is calculated as:

$$score(P) = \sum_i \frac{\overset{\circ}{f}(x_i) - f(x_i)}{\sigma(x_i) + \alpha}$$

1-gram PAYL Evasion: Padding

- Let $\hat{\omega}$ and ω' be the attack body before and after the padding, *n* is the number of distinct chars in normal traffic and λ_i denotes the number of occurrences of character x_i in padding
- It holds $\|\omega'\| = \|\hat{\omega}\| + \sum_i \lambda_i$
- Let $f(x_i)$ and $\hat{f}(x_i)$ be the frequencies of char x_i in legitimate and in blended attack traffic, it holds:

$$\lambda_i = \|\omega'\| f(x_i) - \|\hat{\omega}\| \hat{f}(x_i)$$

- There may be some characters for which $f(x_i) < \hat{f}(x_i)$ and the most frequent such character need not be padded
- Let $\delta = \max \frac{\hat{f}(x_i)}{f(x_i)}$ be the maximum overuse, then

$$\lambda_i = \|\omega'\| (\delta f(x_i) - \hat{f}(x_i))$$

1-gram PAYL Evasion: Substitution

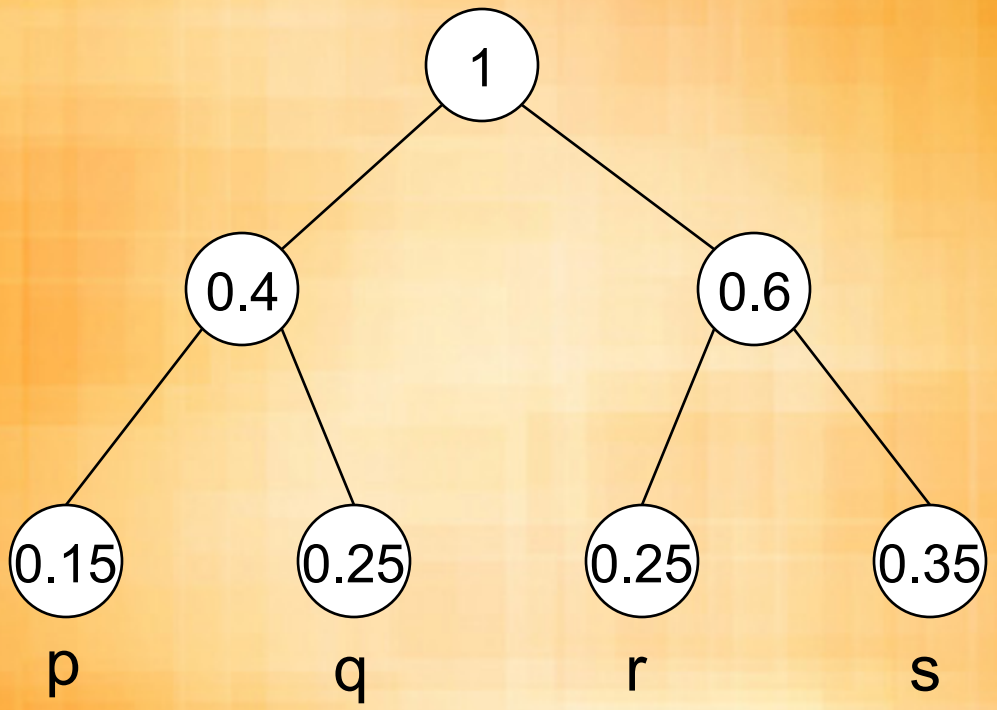
- To minimize padding we need to minimize δ
- Case 1: attack chars are less numerous than legitimate chars
  - A greedy algorithm that generates one-to-many mapping
  - Sort characters by frequency in attack and leg. Traffic
  - Match frequencies in decreasing order
  - Remaining legitimate characters are assigned to attack characters that have highest δ to bring it down
  - For example, we want to map attack string *qpqppqpq* into chars *a, b, c* with frequences 0.3, 0.4 and 0.3
    - Choose b to replace p, a to replace q and because 0.5/0.3 = 1.66 then c will also replace q

1-gram PAYL Evasion: Substitution

- Case 2: attack chars are more numerous than legitimate chars
  - A greedy algorithm that generates n-gram-to-one mapping
  - Construct a Huffman tree where leaves are characters in the attack traffic, and smallest two nodes are iteratively connected (thus most frequent characters have shortest n-gram length)
  - We must choose the labels for the links so to preserve the original legitimate character frequency
    - Sort vertices in the tree by weight
    - Sort legitimate characters by their frequency
    - Choose the highest frequency character for the highest weight vertex
    - Remove the vertex from the list and remove the given portion of the character's frequency from further consideration; then resort the characters

Example For Case 2

- Legitimate characters a and b have frequency 0.5, and attack characters p, q, r, s have frequency 0.15, 0.25, 0.25 and 0.35

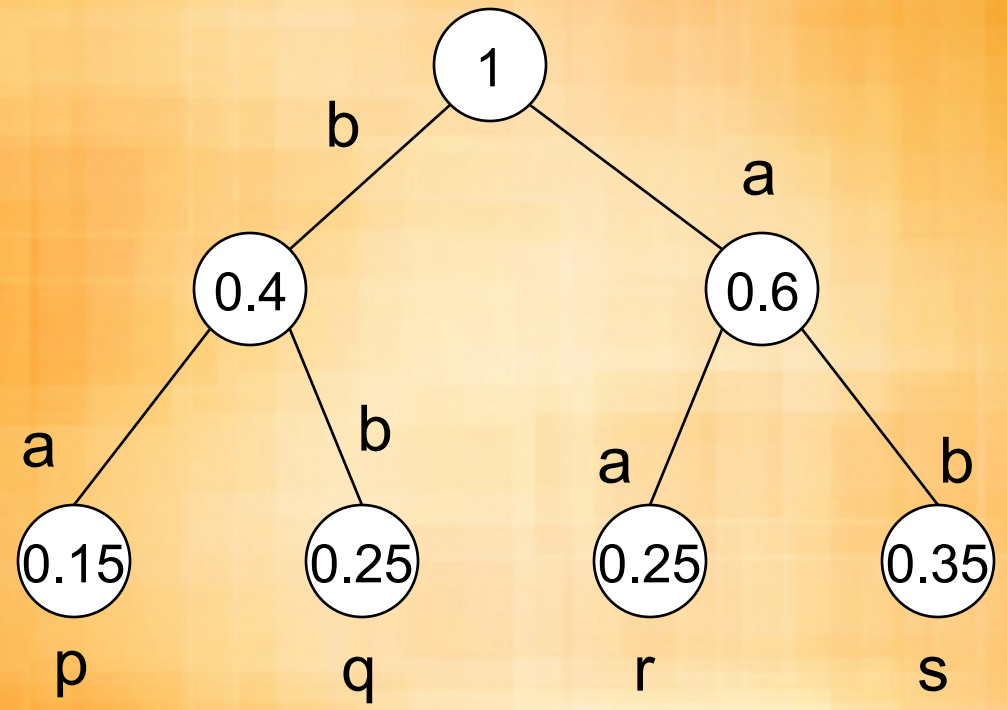


0.5, 0.5
a b

0.6, 0.4, 0.35, 0.25, 0.25, 0.15

Example For Case 2

- Legitimate characters a and b have frequency 0.5, and attack characters p, q, r, s have frequency 0.15, 0.25, 0.25 and 0.35



b a

0.6, 0.4, 0.35, 0.25, 0.25, 0.15

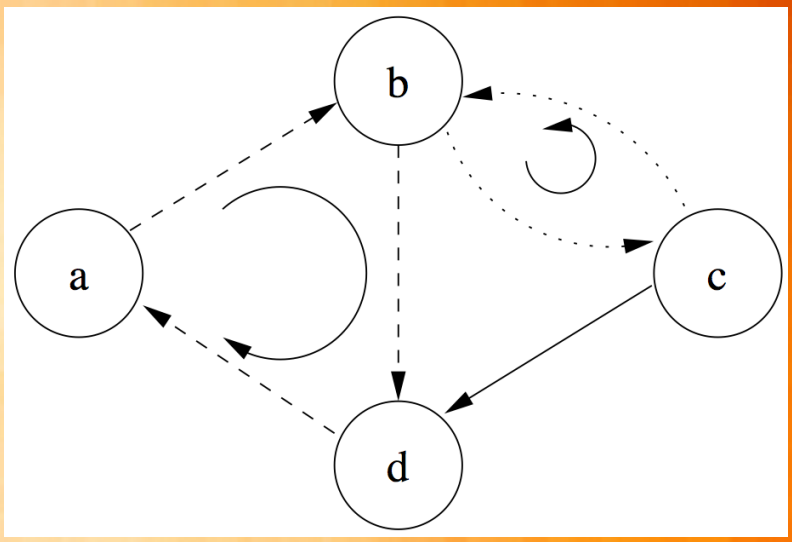
a b b b a a

2-gram PAYL Evasion

- Must match all 2-byte pairs
- Represent valid 2-grams as states in FSM
- A simple approach will enumerate valid paths in FSM and map attack characters to paths randomly but this generates large code size
  - Better mapping can be obtained by using entropy information, i.e., mapping frequent characters to short paths
- Another approach will attempt to find single byte mappings so that 2-grams are also matched
  - Greedy algorithm sorts 2-grams by frequencies in legitimate and attack traffic and matches them greedily taking care not to violate any existing mappings
- Generate padding so to match the target distribution greedily

2-gram PAYL Evasion

- e_0 = da
- e_1 = bc
- Input: 01101010
- Output: bdabcbcbdabcbdabcbda



Attack Complexity

- For 1-gram blending greedy algorithms are proposed that generate small padding and can closely match the target byte frequency
- For 2-gram blending it is difficult to meet both the goal of accurate frequency match and of small code size
  - In general finding a good substitution is NP-hard
  - Proposed heuristics can achieve good frequency match but at the expense of code size

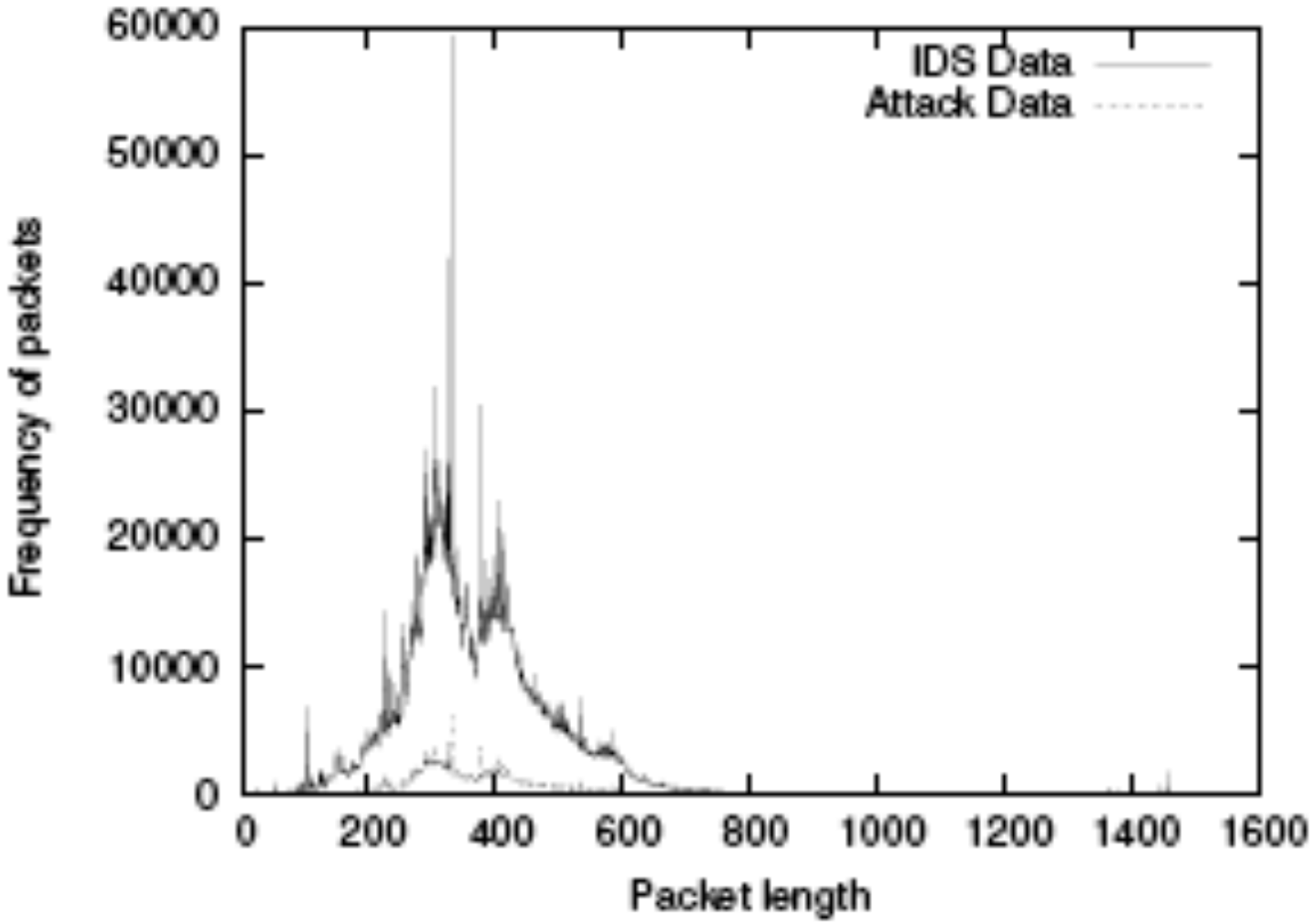
Evaluation

- Attack on Windows Media Services
  - Exploits a vulnerability with logging of user requests
- Attack vector is 99 bytes long and must be present at the start of the HTTP request
- For buffer overflow attack must send 10KB of data
- Attack body opens a TCP connection and sends registry files
- Size of attack body is 558B and contains 109 unique characters
- Attack was divided into multiple packets and, after blending, padded with legitimate traffic to achieve required 10KB size

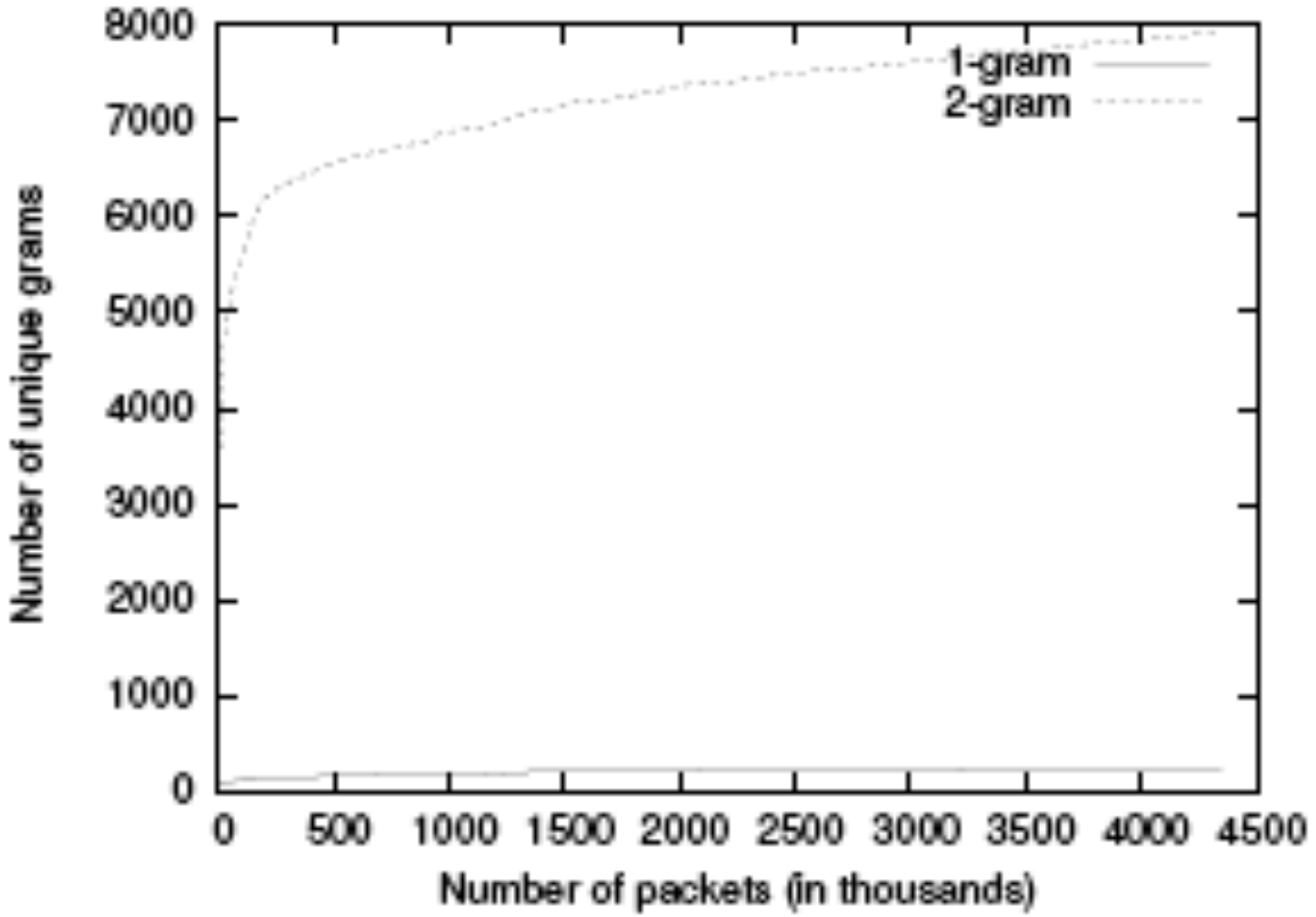
IDS Training

- Captured 15 days of HTTP traffic and used 14 days' traffic to train the IDS
  - Only TCP data packets are used that do not contain known attacks
- IDS builds profiles per packet length
- Last day's traffic is used by the attacker to learn character distributions
- Selected three frequent packet sizes for the attack
  - Used packets of these lengths observed in the 15th day to extract byte frequencies for blending

Packet Length Distributions



Unique 1-grams and 2-grams



Evaluation Results

- PAYL training time increases with the size of the training data because new packets carry more unique n-grams
- Tested CLET-generated polymorphic attacks against PAYL
  - CLET only adds padding to match byte frequency
  - Other polymorphic engines perform worse than CLET against PAYL
  - CLET attack sequence will avoid PAYL detection only if all packets have an anomaly score above the threshold
  - Both 1-gram and 2-gram PAYL detected all attacks with chosen threshold setting

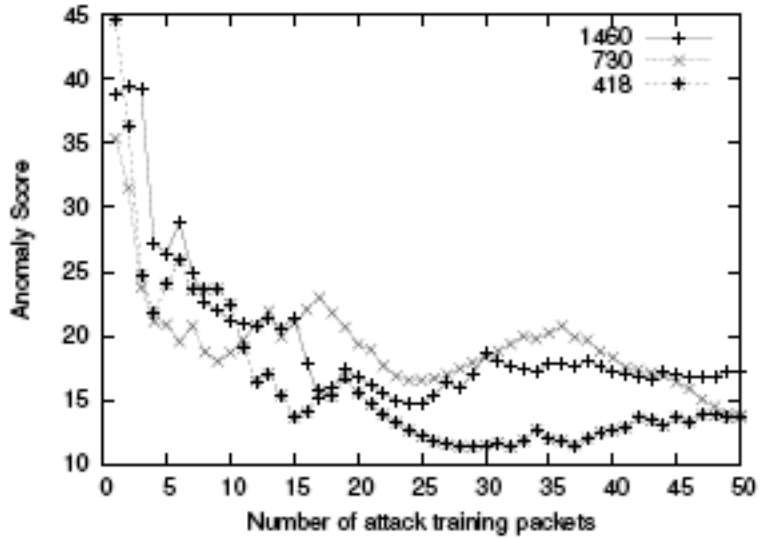
Packet Length	1-gram	2-gram
418	872	1,399
730	652	1,313
1460	355	977

Evaluation Results

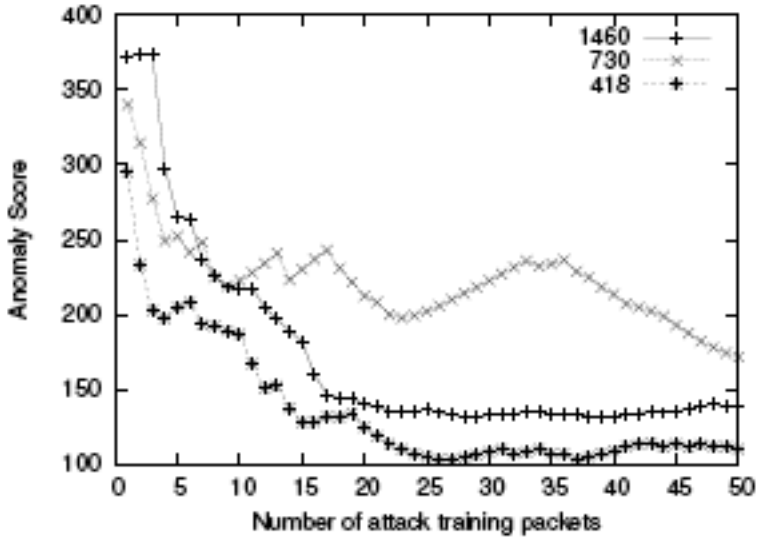
- Training of the artificial profile is stopped when there is no significant improvement over existing profile (measured using Manhattan distance) within two packets
- Number of packets required for convergence

Packet Length	1-gram	2-gram
418	8	20
730	8	18
1460	14	40

Anomaly Score of the Artificial Profile vs Training Length



(a) 1-gram



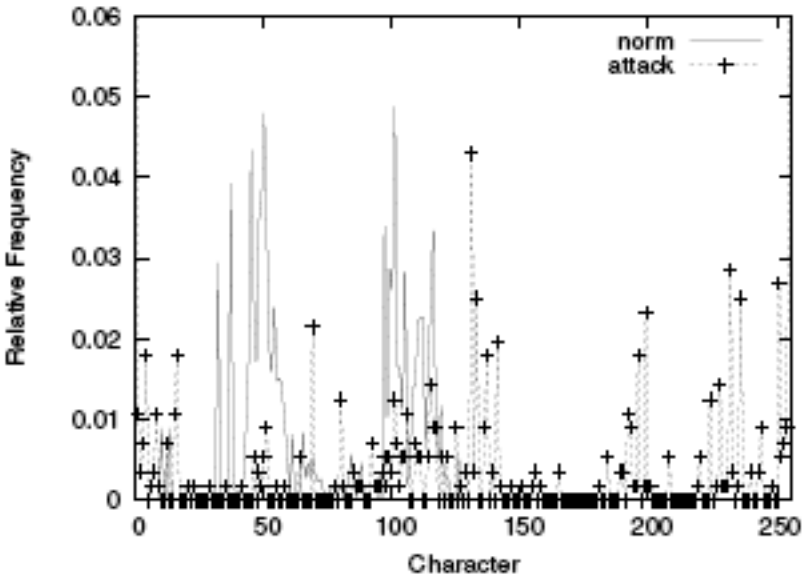
(b) 2-gram

Well under the PAYL threshold

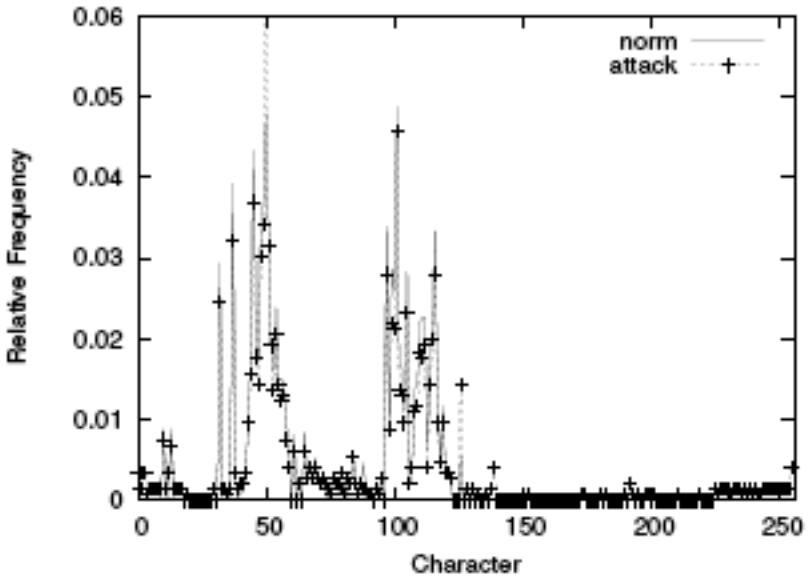
1-gram and 2-gram Attacks

- For 1-gram attacks used one-to-one substitution cipher
- For 2-gram attacks used single byte encoding scheme
- Two types of transformations were tested
  - Substitution table is constructed for entire attack body - global substitution
  - Substitution table is constructed for each packet separately - local substitution
  - If attack characters are more numerous than those in legitimate traffic, non-existing characters were used

Byte Frequencies

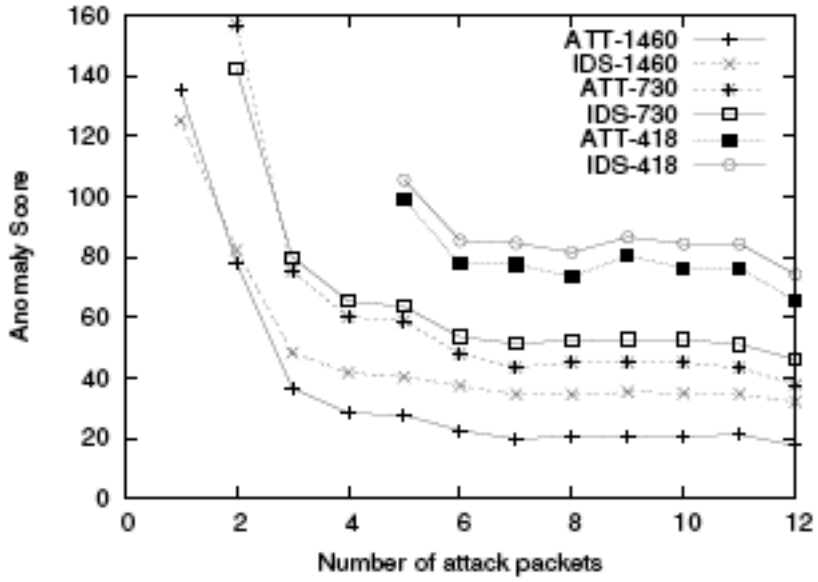


(a) Original attack packet

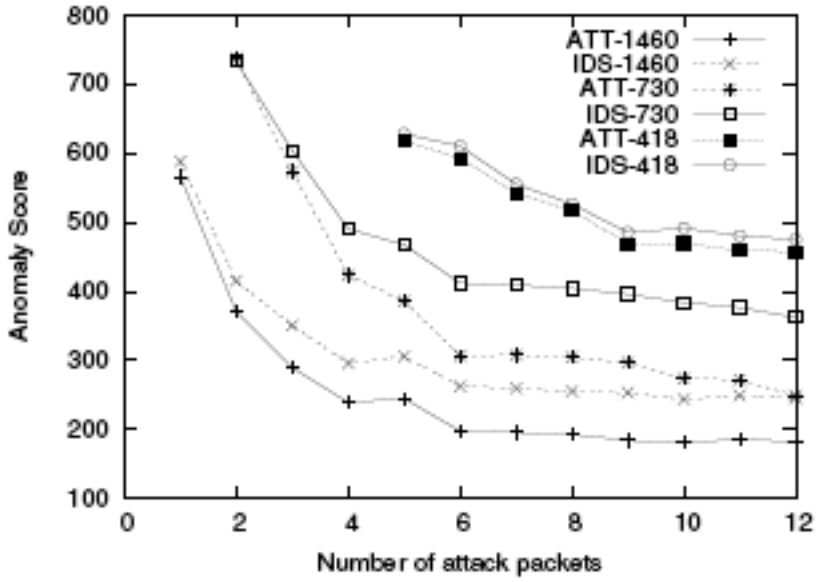


(b) 1-gram Blending Packet for packet length 1460

Local Substitution

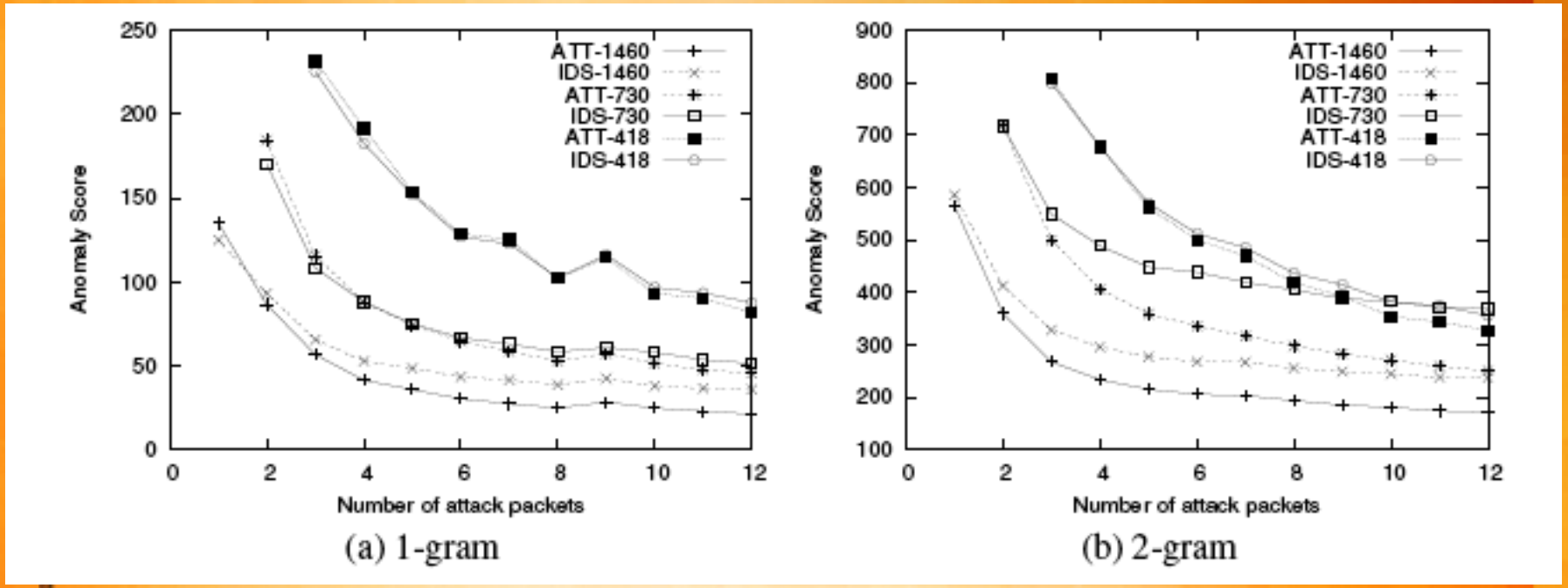


(a) 1-gram



(b) 2-gram

Global Substitution



Effect of False Positive Setting

- Higher false positives make IDS more sensitive so more packets are needed to successfully blend the attack

False Positive	418		730		1460	
	1-gram	2-gram	1-gram	2-gram	1-gram	2-gram
0.1	61.07 (17,-)	373.4 (-,12)	63.70 (5,7)	467.6 (5,5)	74.50 (3,3)	447.7 (2,2)
0.01	78.61 (12,15)	456.9 (22,8)	143.6 (2,3)	625.5 (3,3)	81.98 (3,3)	531.0 (2,2)
0.001	125.5 (5,7)	561.8 (7,6)	164.6 (2,3)	670.5 (3,3)	239.2 (1,1)	931.9 (1,1)
0.0001	166.8 (5,5)	582.6 (7,5)	244.5 (2,2)	805.0 (2,2)	243.4 (1,1)	935.0 (1,1)

Other Observations

- 2-gram IDS had consistently higher anomaly scores for attacks but it also had higher thresholds to avoid false positives
  - Overall similar performance as 1-gram IDS
  - More costly for IDS
- Local substitution always outperformed global substitution

Countermeasures

- More complex models are needed
  - Observe additional traffic features in addition to statistical ones, e.g., syntactic and semantic information
  - Key direction to explore is a more sophisticated (e.g., semantic IDS) that can perform at high speed
- Use multiple simple IDSs that model different features
- Introduce randomness into the IDS model
  - Model byte pairs that are *v* characters apart
  - Choose *v* at random and fix it for a given IDS
  - Combine several such systems

My Opinion

- The idea is neat and the proposed blending techniques are easy to understand
  - The attack is realistic
- Paper had a lot of repetitive text
  - Organization was poor too
- A lot of greedy heuristics without proof of their performance
  - Explanations of the proposed heuristics are also poor, but examples were helpful
- There is no firm evidence that statistical detection using legitimate traffic profile is much cheaper than detection using sintactic and semantic information

Conclusions

- There are many defenses against polymorphic attacks, but such defenses are simplistic
  - They can be tricked by a polymorphic blending attack, i.e. an attack that actively attempts to evade detection
- Polymorphic blending attacks are easily constructed and can evade PAYL in multiple scenarios
- A few countermeasures are proposed against polymorphic blending attacks
  - Left as future work

Infeasibility of Modeling Polymorphic Shellcode

Shell code background

- Format:

  [NOP][DECODER][ENCPAYLOAD][RETADDR]
- Polymorphism applied to decoding routine

Signature matching

- String-based signatures
  - Example: Snort
  - Identification of NOP sled
- Statistical measures of packet content

Problem definition

- Given n bytes, there exist 256^n possible strings
- x86 code of length n is a subspace
- How difficult is it to model this subspace?

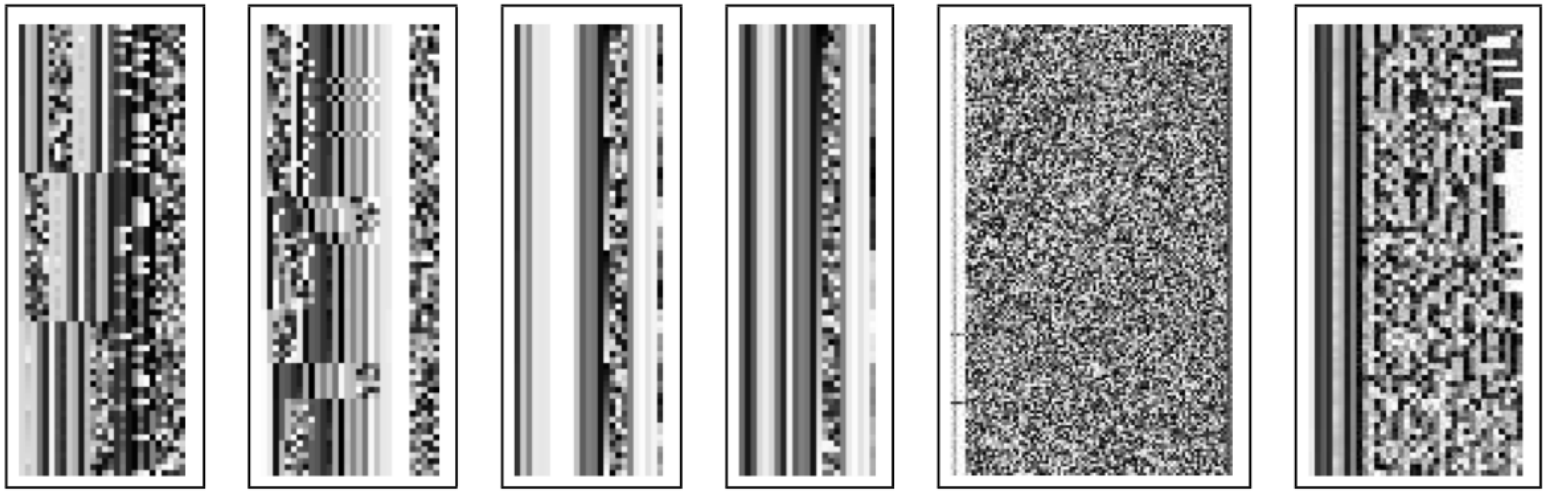
Measures

- Spectral image
- Minimum Euclidian distance
- Variation strength
- Propagation strength
- Overall strength

Spectral image

- D decoders of length N
- Compile into D×N matrix
- Display matrix as image

Spectral image



Minimum Euclidian distance

- String x as point in n-dim Euclidian space
  - Example: “ab” -> (97,98)
- Minimum Euclidian Distance: minimum normalized distance between two points under arbitrary byte-level rotations

$$\delta(x,y) = \min_{1 \le r \le n} \left\{ \frac{\| x - rot(y,r) \|}{\| x \| + \| y \|} \right\}$$

- Intuition: Decoders can shift order of operations

Variation strength

- Magnitude of the space covered by span of points in n-space corresponding to detectors
- Decoders $x_1, x_2, \ldots, x_N$ in n-space
- $\lambda_1, \lambda_2, \ldots, \lambda_n$ eigenvalues of covariance matrix
- Variation strength:

$$\psi = \frac{1}{n} \sum_{i=1}^{n} \sqrt{\lambda_i}$$

Propagation strength

- Efficacy in making sample pairs different
- Consider fully connected graph with decoders as nodes
- Edge weight = minimum Euclidian distance
- Propagation strength = average edge weight
- η = number of salient bytes in samples
- p(.) = prior (default: p(.) = 1)

$$\Phi(engine) = (1 - \frac{\eta}{n}) \int \int p(\delta(\mathbf{x}, \mathbf{y})))\delta(\mathbf{x}, \mathbf{y})\, d\mathbf{x}\, d\mathbf{y}$$

Overall strength

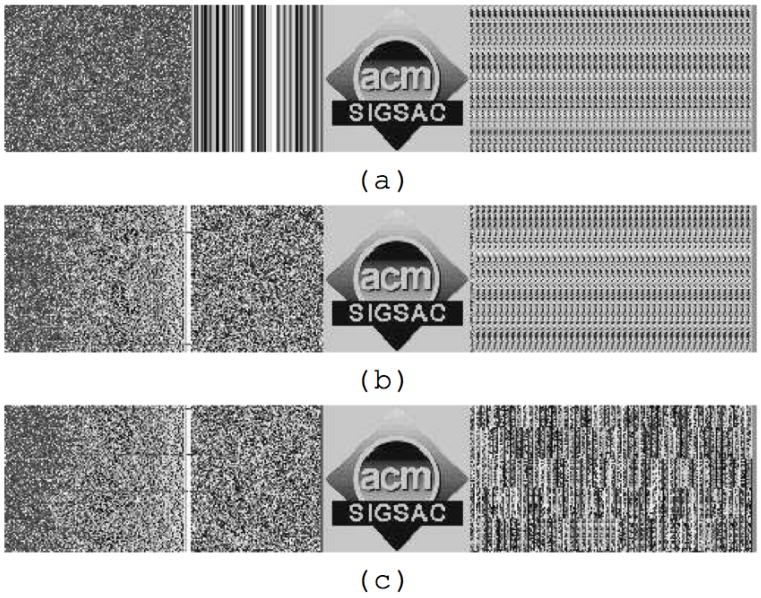
- For a polymorphic engine:

$$\Pi(engine) = \psi(engine) \times \varphi(engine)$$

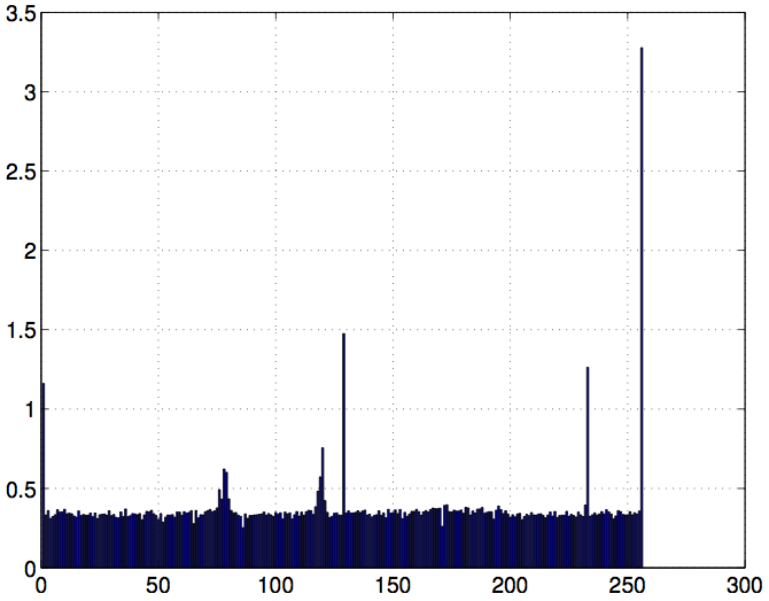
Engine	Prop. St.	Var. St.	Overall St.	p-score
Shikata	0.14	53.24	7.24	0.62
Jcadd	0.11	44.62	4.87	0.42
C4d	0.06	14.62	0.83	0.07
Fnstenv	0.07	15.70	1.05	0.09
Clet	0.14	53.00	7.37	0.63
Admmutate	0.15	68.76	10.59	0.91
$rand_{128}$	0.16	36.90	5.83	0.50
$rand_{256}$	0.16	73.74	11.61	1.00

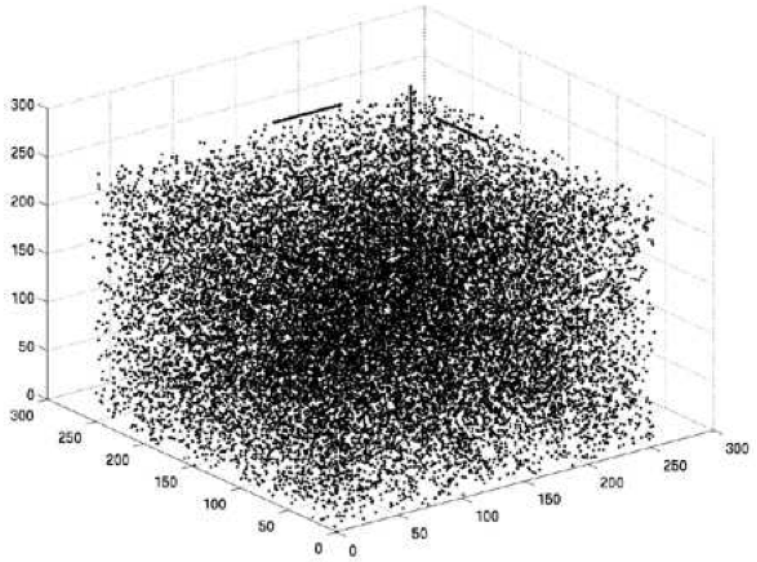
Hybrid engine: Full spectrum polymorphism and blending

- CLET: byte distribution blending
- ADMutate: Polymorphism
  - Random looking decoder, recursive NOP sled
- Combine CLET and ADMutate
  - Blend in with normal traffic
  - Blending bytes can be randomly permuted
  - RETADDR can be added with a random offset
  - 4-byte salient artifact too small to use as a signature
  - Essentially impossible to model



Hybrid engine: Full spectrum polymorphism and blending





References

- "Polymorphic Blending Attacks", Fogla et al, 2006
- "On the Infeasibility of Modeling Polymorphic Shellcode", Song et al, 2007