Anomaly Detection of Web-based Attacks

Paper by Christopher Kruegel, Giovanni Vigna

Presented by Farhan Jiva

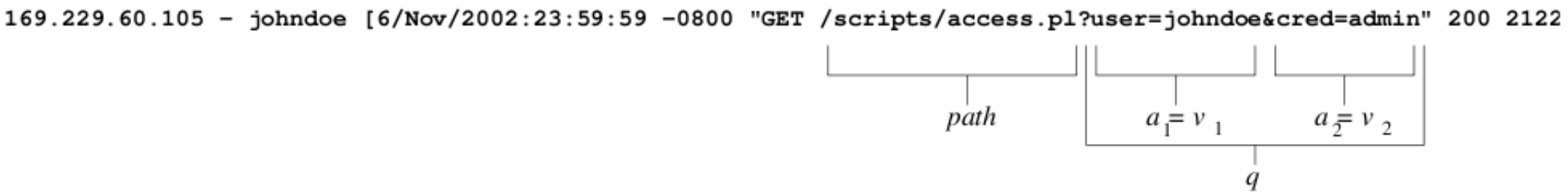
Overview

- Motivation
- Description of the data model
- Description of the detection models/techniques used
- Evaluation/results

Motivation

- Many IDSs rely on a finite set of signatures of known attacks
- Many vulnerabilities are discovered on a daily basis
  - It is difficult to keep the signatures up-to-date
- Solution: IDSs should be accompanied by anomaly detectors to accommodate for this lag

Data Model



- Approach based on analyzing HTTP GET requests via logs
- Only looking at successful requests
  - Responses with return code 2xx
- POST/HEAD requests left for future work

Detection Model

- Uses a number of different techniques to identify anomalous GET request entries
- Each detection model outputs a probability value (p_m) which is used in the Anomaly Score equation
- An anomalous event should yield a high anomaly score
- A model has two modes, training and detection
  - Training is used to establish the normality of a model, using the first X number of requests seen for a particular program
  - Detection is used to finding deviants

$$\text{Anomaly Score} = \sum_{m \in \text{Models}} w_m * (1 - p_m)$$

Model 1: Attribute Length

- The length of an attribute can be used to detect anomalous requests
- Values are either fixed-sized tokens or short strings from human inputs
  - Lengths should not vary much between different requests
- Goal is to approximate the actual but unknown distribution of the lengths of these values

Attribute Length: Learning

- Using the value-lengths for the attributes seen during the learning phase, calculate the mean μ and variance σ^2 for these samples

Attribute Length: Detection

- Using the sample mean μ and sample variance σ^2 of the length distribution from the learning phase, determine the regularity of a newly encountered value with length L.
- The probability of L is found using the Chebyshev inequality
  - The computed bound is generally very weak, has a high degree of tolerance to deviations
  - Only obvious outliers will be flagged as suspicious

Model 2: Attribute Character Distribution

- The characters used in any particular value for a certain attribute is drawn from small subset of the ASCII characters.
- The intuition is that these characters occur with different frequencies
- For this model, use what's called an Idealized Character Distribution
  - The character distribution of an attribute which is perfectly normal

Attribute Character Distribution: Learning

- For each observed query attribute, store its character distribution
- The Idealized Character Distribution is then calculated by averaging all stored character distributions

Attribute Character Distribution: Detection

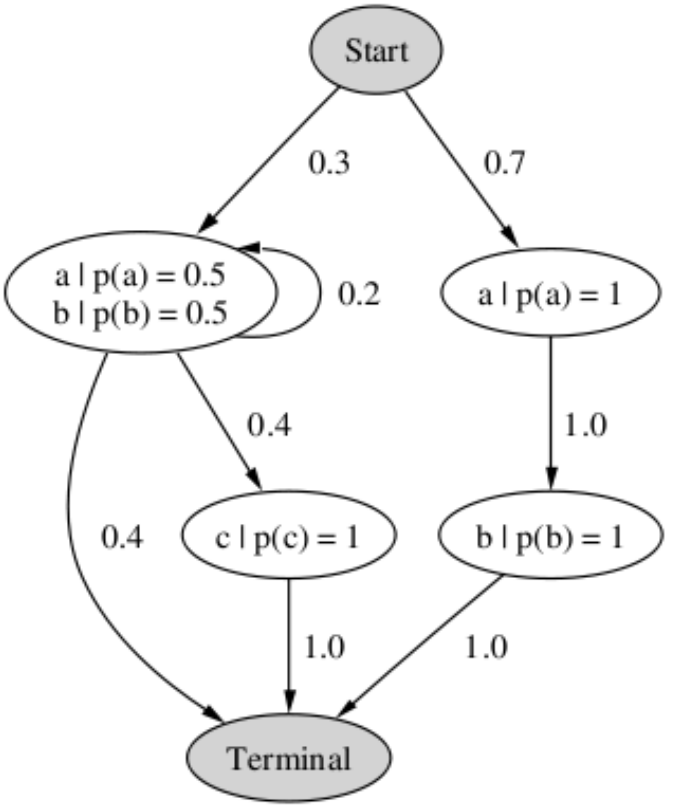
- When a new query attribute is encountered, determine the number of occurrences of each character in the string
- Use a Chi-square test to calculate and return the probability that the given sample has been drawn from the Idealized Character Distribution

Model 3: Structural Inference

- It could be the case that an attacker can make the attack string of an attribute more regular in terms of character distribution
- In these situations, it is useful to know the structure of all the normal, legitimate values
  - Use a regular grammar to analyze this

Structural Inference: Learning

- The items in the training set make up the output of a probabilistic grammar
  - Probabilities are assigned for each of the grammar's productions
  - Considered a Markov Model
- Use the Bayesian theorem to derive a Markov model from empirical data



$$p(\text{ab}) = (1.0 * 0.3 * 0.5 * 0.2 * 0.5 * 0.4) + (1.0 * 0.7 * 1.0 * 1.0 * 1.0 * 1.0)$$

$$= 0.706$$

Structural Inference: Detection

- Using the Markov model which was built in the learning phase, determine the probability for a a newly encountered attribute
- Sometimes, legitimate input might return a low probability score
  - Easy fix, return 1 if the word is a valid output from the Markov model and 0 if it cannot be derived from the grammar

Model 4: Token Finder

- Sometimes, the value for a particular attribute is drawn from a limited set of enumerated elements
- This model attempts to detect if any attributes are using enumeration

Token Finder: Learning

- An argument is classified as an enumeration if the number of different occurrences of values is bound by some unknown threshold
  - There is no bound restriction for random values
- If the number of different arguments grows proportional to the total number of arguments, assume it is using random values.
- Otherwise, assume enumeration and store all the values witnessed for the particular attribute

Token Finder: Detection

- If it has been determined that the attribute draws its tokens from an enumeration, any new values encountered should be in the list stored previously
  - If new value encountered is in the list, return 1
  - Else return 0
- If no enumeration was detected in the learning phase, this model always returns 1

Model 5: Attribute Presence or Absence

- This model detects the presence or absence of an attribute
  - Some hand-crafted attacks will leave out some attributes
- Learning:
  - During training, keep a list of acceptable subsets of attributes that appear together
- Detection:
  - In a newly seen request, if the set of attributes has been seen during the training phase, return 1. Else return 0.

Model 6: Attribute Order

- This model analyzes the relative order of attributes seen in a request
- Learning:
  - Create a directed graph such that an edge (ai,aj) : attribute ai precedes attribute aj
  - Use Tarjan's algorithm to find all strongly connected components, remove edges and vertices of the same SCC
- Detection:
  - Traverse this graph for each request
  - If there any any violations, return 0. Else return 1.

Evaluation: Datasets

Data Set	Time Interval	Size (MByte)	HTTP Queries	Program Requests	Attributes	Programs
Google	1 hour	236	640,506	490,704	1,611,254	206
UCSB	297 days	1,001	9,951,174	7,993	4,617	395
TU Vienna	80 days	251	2,061,396	713,500	765,399	84

- Evaluation was performed on 3 datasets
  - Google, UCSB, TU Vienna
- Datasets were Apache log files

Evaluation: Model Validation

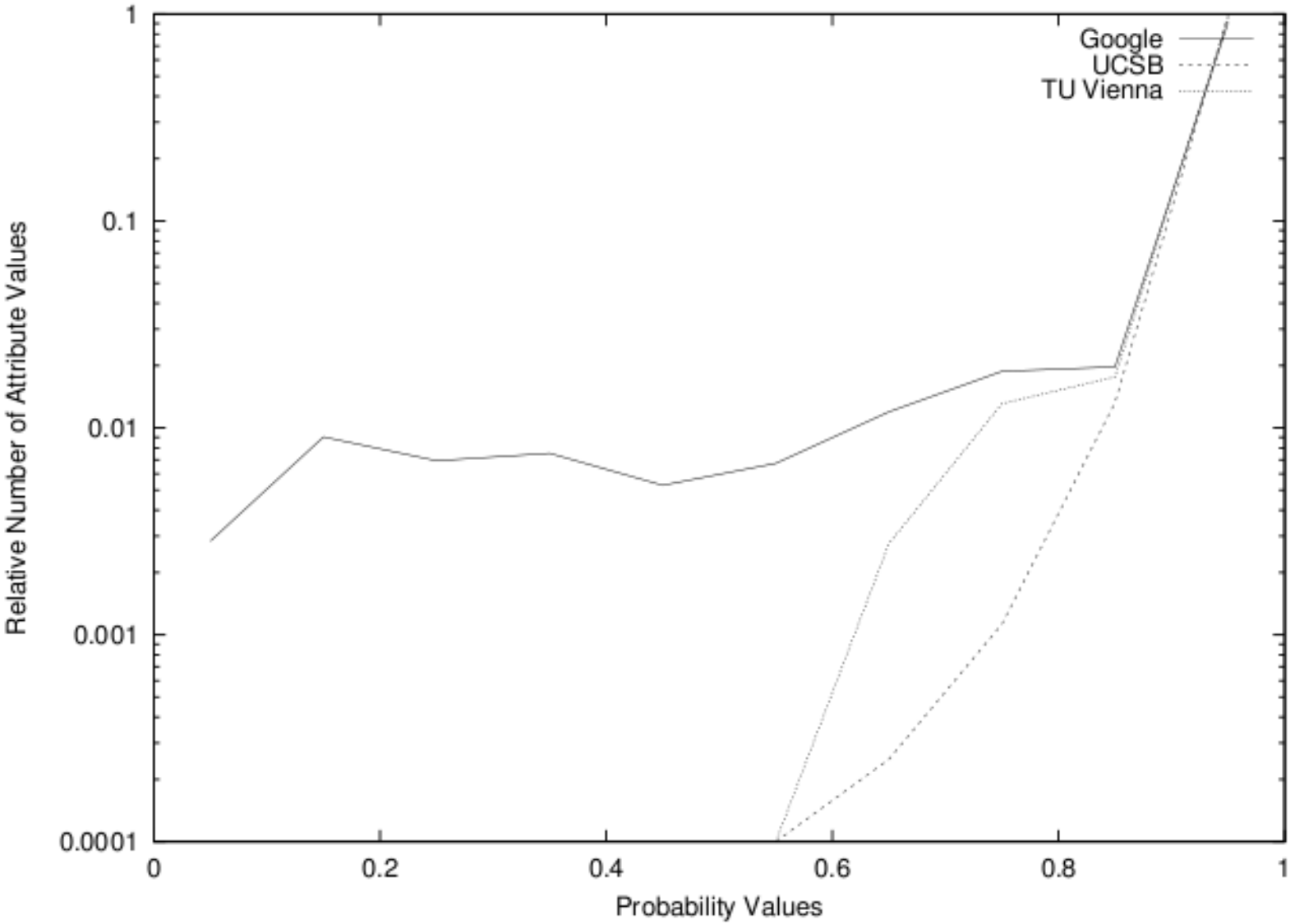


Figure 3: Attribute Length

- Google has the highest variability
- Google includes the search string in the GET request

Evaluation: Model Validation

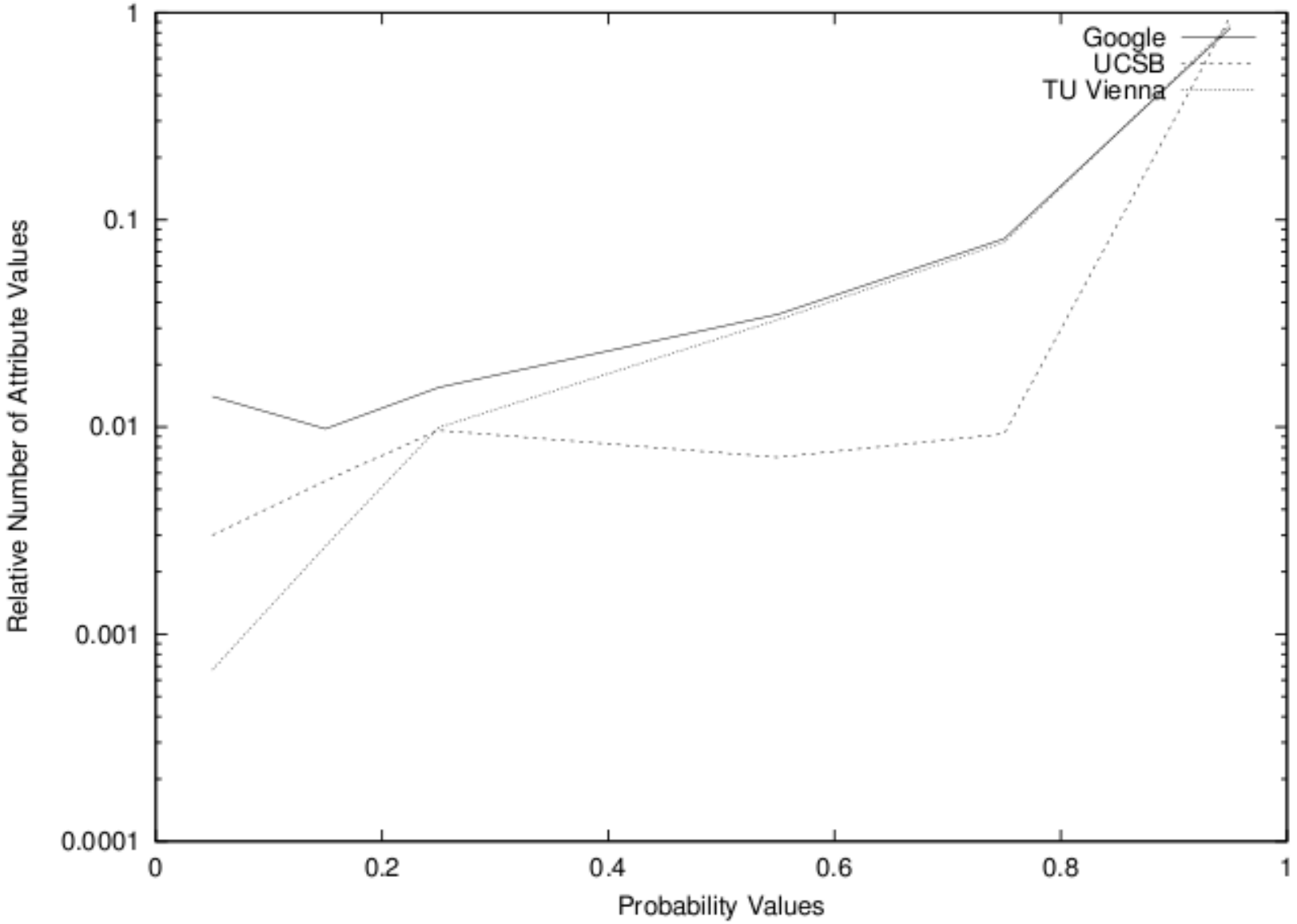


Figure 4: Attribute Character Distribution

Evaluation: Model Validation

	Structure (Attribute)		Token (Attribute)		Presence (Query)		Order (Query)	
Data Set	normal	anomalous	normal	anomalous	normal	anomalous	normal	anomalous
Google	1,595,516	15,738	1,603,989	7,265	490,704	0	490,704	0
UCSB	7,992	1	7,974	19	4,616	1	4,617	0
TU Vienna	765,311	98	765,039	370	713,425	75	713,500	0

Table 3: Probability Values

Detection Effectiveness

Data Set	Number of Alerts	Number of Queries	False Positive Rate	Alarms per Day
Google	206	490,704	0.000419	4,944
UCSB	3	4617	0.000650	0.01
TU Vienna	151	713,500	0.000212	1.89

Table 4: False Positive Rates

- The authors ran an IDS over the 3 datasets to assess the number of false positives that can be expected
- They assumed that the training set (first 1000) contained no real attacks
- Relative number of false positives were similar

Detection Capabilities

Attack Class	Length	Char. Distr.	Structure	Token	Presence	Order
Buffer Overflow	x	x	x		x	
Directory Traversal		x	x			
XSS (Cross-Site Scripting)	x	x	x		x	
Input Validation				x		x
Code Red	x	x	x			

Table 5: Detection Capabilities

- Used 11 real-world exploits and Code Red to test the capabilities of their system
  - Used on TU Vienna's web server
  - Mix of buffer overflow, XSS attacks
- All 11 attacks were detected

Questions?