Overview
System Architecture
Detection State Machine
Redirection
Anomaly Detection
Embedding and Similarity Measures
Anomaly Score
Implementation
Experiments
Runtime
Accuracy
Conclusions

An Architecture for Inline Anomaly Detection

Tammo Krueger, Christian Gehl,
Konrad Rieck and Pavel Laskov
Fraunhofer Institute FIRST
Intelligent Data Analysis, Berlin, Germany

12.12.2008 @ EC2ND 2008

Outline

1 System Architecture

2 Detection State Machine

3 Redirection

4 Anomaly Detection
- Embedding and Similarity Measures
- Anomaly Score

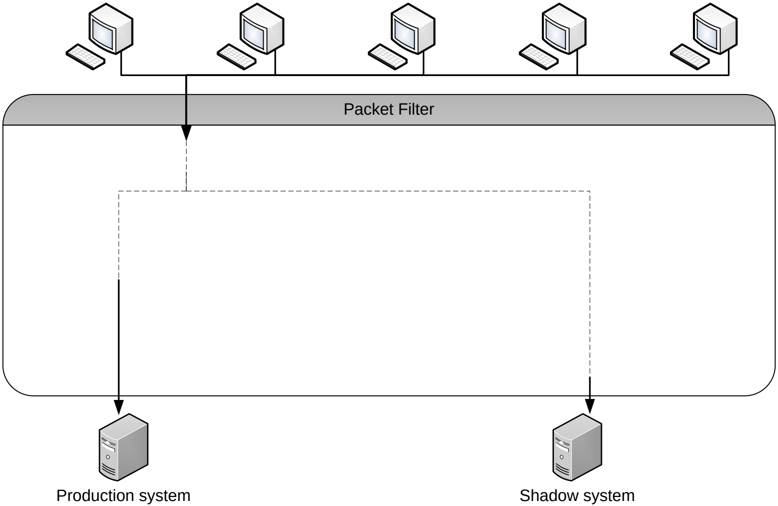
5 Implementation

6 Experiments
- Runtime
- Accuracy

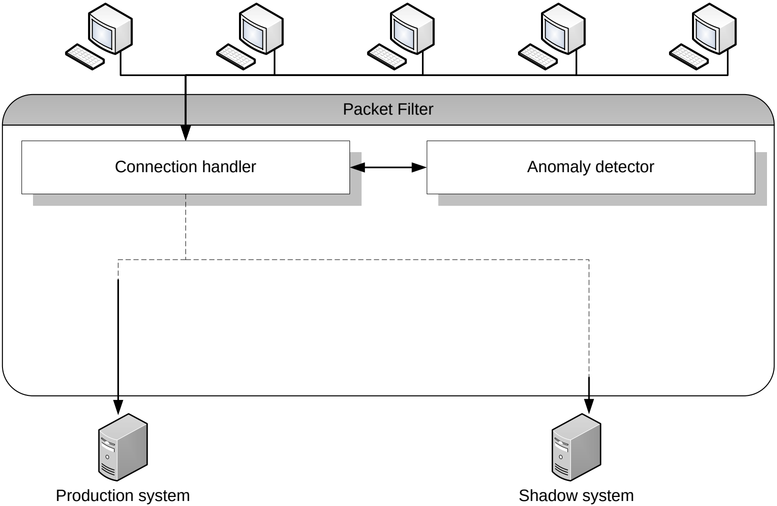
Overview

- **Goal**: exploit anomaly detection in an *inline* intrusion prevention system:
  - ... with an *application-independent* architecture
  - ... where decision-making is performed at the *network layer*
  - ... where anomaly detection runs at the *application layer*
- Inline defense policies
  1. forwarding to a production system
  2. redirection to a hardened system (*shadow system*)
  3. redirection to a monitored network sink (*forensic sink*)

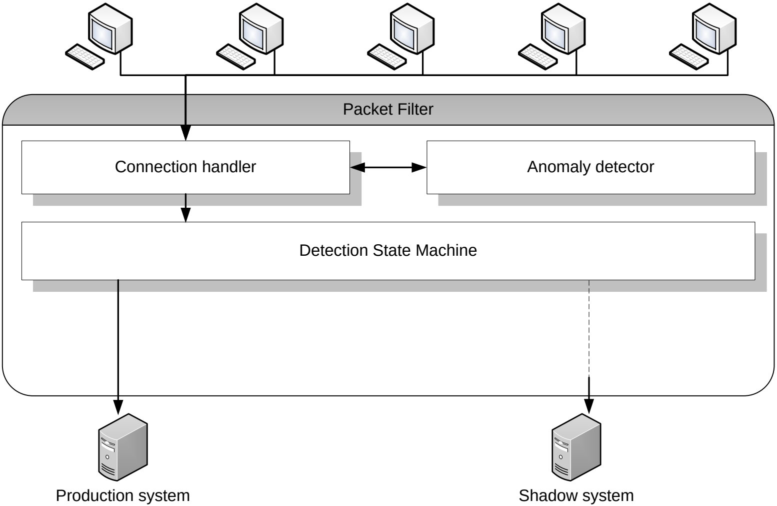
System Architecture



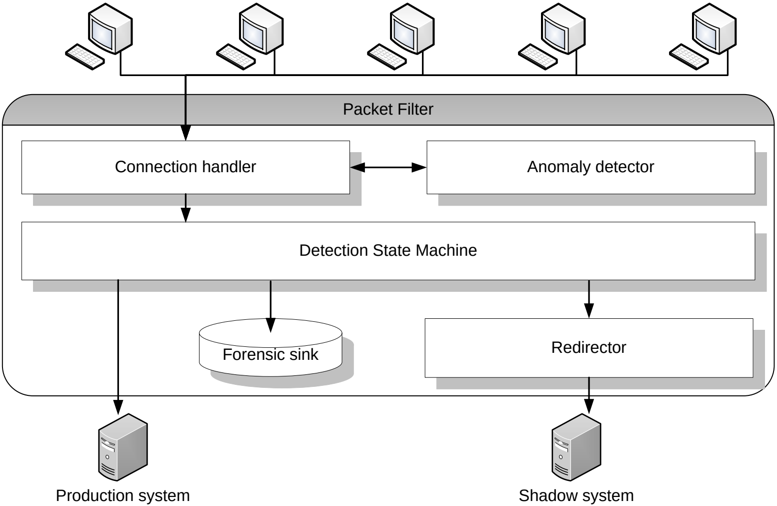
System Architecture



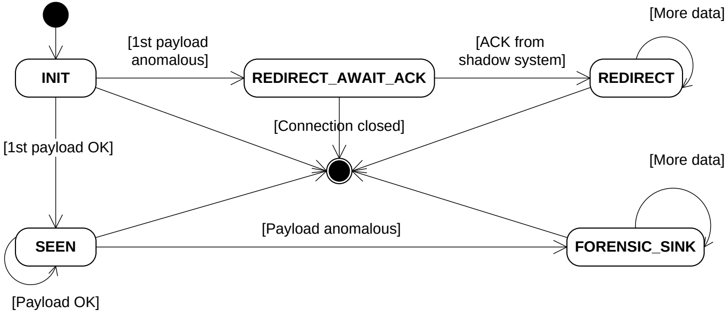
System Architecture



System Architecture

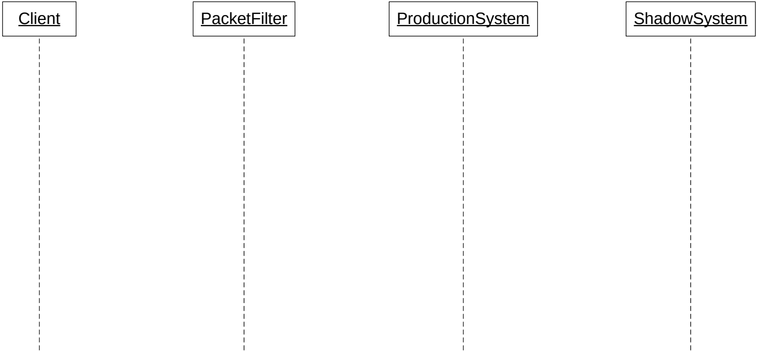


Detection State Machine



- Each connection has a detection state
- Each detection state triggers specific action for each packet of the connection

Redirection



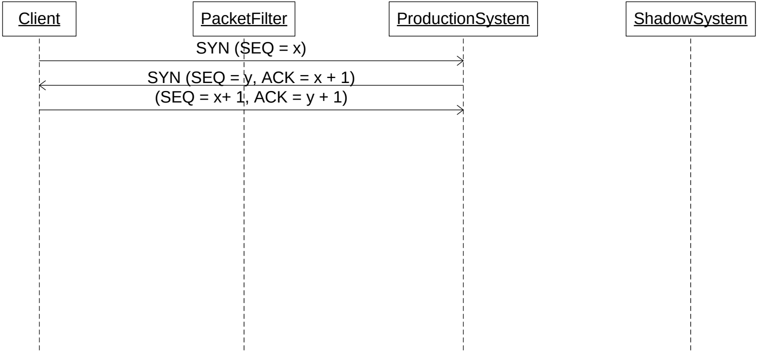
- Memorize difference in the sequence numbers (here $d = z - y$)
- Adjust corresponding sequence / ACK numbers of packets

Redirection

An Architecture for Inline Anomaly Detection

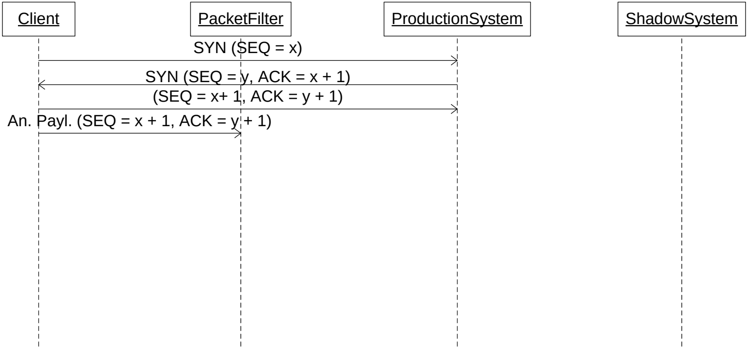
Tammo Krueger

Overview
System Architecture
Detection State Machine
Redirection
Anomaly Detection
Embedding and Similarity Measures
Anomaly Score
Implementation
Experiments
Runtime
Accuracy
Conclusions



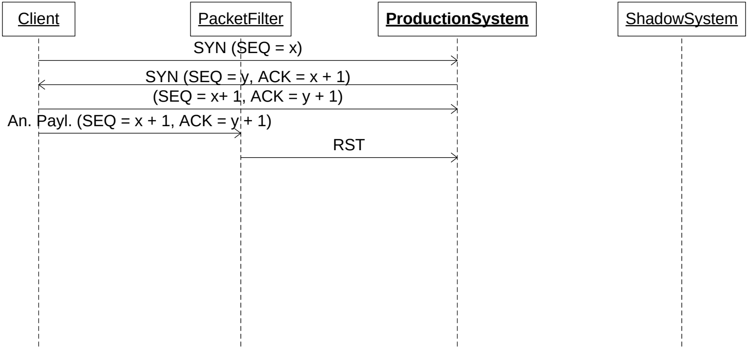
- Memorize difference in the sequence numbers (here $d = z - y$)
- Adjust corresponding sequence / ACK numbers of packets

Redirection



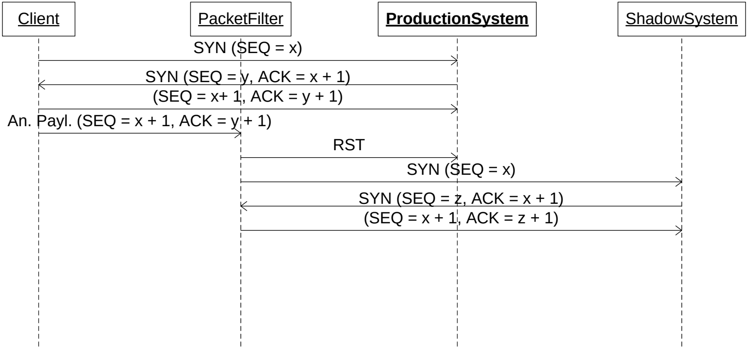
- Memorize difference in the sequence numbers (here $d = z - y$)
- Adjust corresponding sequence / ACK numbers of packets

Redirection



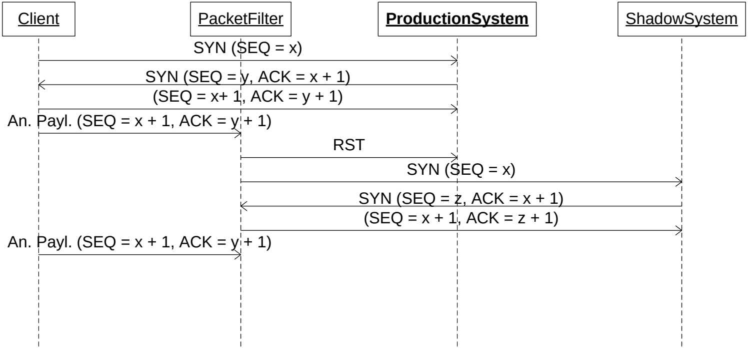
- Memorize difference in the sequence numbers (here $d = z - y$)
- Adjust corresponding sequence / ACK numbers of packets

Redirection



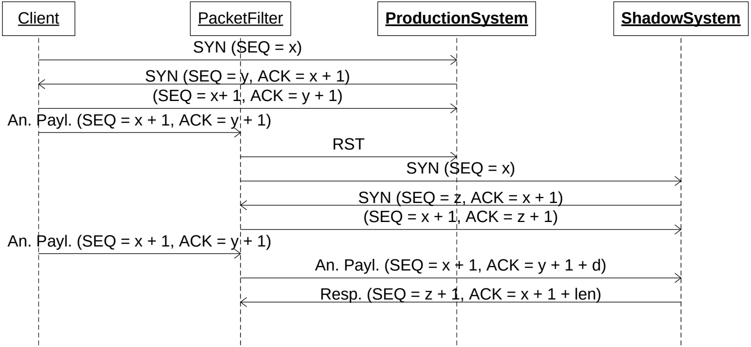
- Memorize difference in the sequence numbers (here $d = z - y$)
- Adjust corresponding sequence / ACK numbers of packets

Redirection



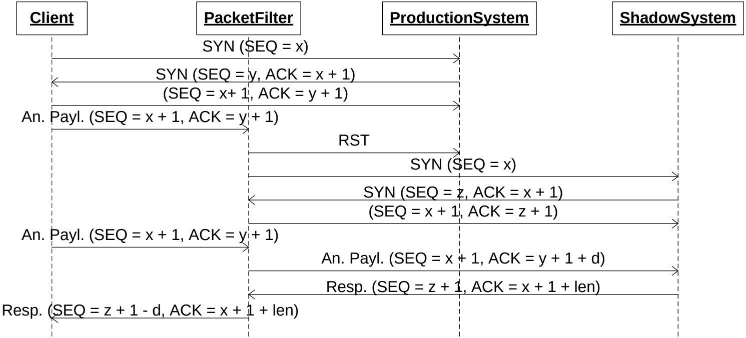
- Memorize difference in the sequence numbers (here $d = z - y$)
- Adjust corresponding sequence / ACK numbers of packets

Redirection



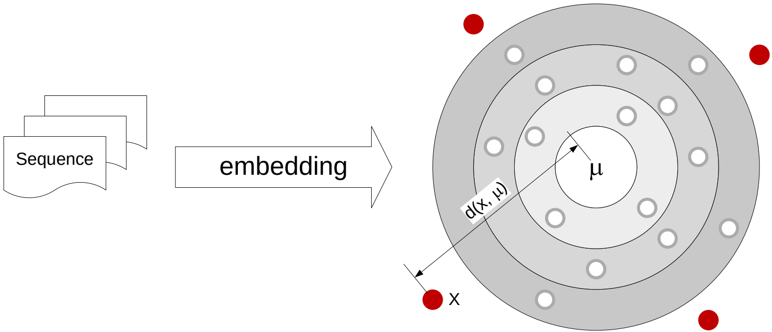
- Memorize difference in the sequence numbers (here $d = z - y$)
- Adjust corresponding sequence / ACK numbers of packets

Redirection



- Memorize difference in the sequence numbers (here $d = z - y$)
- Adjust corresponding sequence / ACK numbers of packets

Anomaly Detection – Overview



- Idea: An anomaly is a *deviation* from a model of *normality*
- Implementation:
  1. Embed data in *vector space* via embedding function
  2. Learn the center μ of the data as a model of normality
  3. Anomaly score for new data point is distance to μ

Embedding and Similarity Measures

- Given the set of all possible n-grams over byte sequences $S = \{0, \ldots, 255\}^n$, we define the embedding function ϕ as

$$\phi(x) = (\phi_s(x))_{s \in S} \in \mathbb{R}^{|S|} \quad \text{with} \quad \phi_s(x) = \#_s(x)$$

- Example ($n = 3$):

$$\phi('Hello') = (0, \ldots, \overset{Hel}{\frac{1}{3}}, \overset{ell}{\frac{1}{3}}, \overset{llo}{\frac{1}{3}}, \ldots, 0)^T \in \mathbb{R}^{16777216}$$

- With embedding function we can define distances between byte sequences, for instance Euclidean distance:

$$d(x, z) = \|\phi(x) - \phi(z)\|_2 = \sqrt{\sum_{s \in S} |\phi_s(x) - \phi_s(z)|^2}$$

Anomaly Score

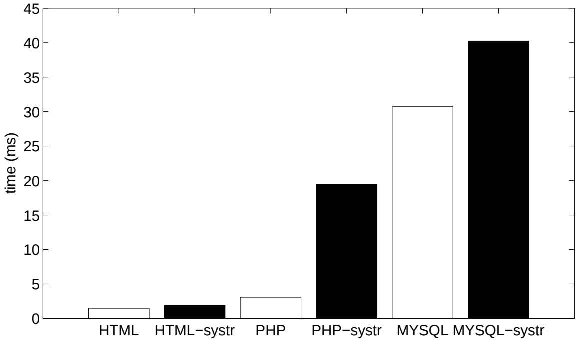
1. *Training:* collect normal data packets $X = \{x_1, \ldots, x_n\}$ and compute their mean $\mu = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i)$.
2. *Validation:*
   1. collect an independent set of normal packets $\tilde{X} = \{\tilde{x}_1, \ldots, \tilde{x}_m\}$
   2. pre-define a false-positive rate ν
   3. determine anomaly threshold t_a so that the ratio of packets $\tilde{x}_i$ for which $d(\mu, \tilde{x}_i) > t_a$ is smaller than ν
3. *Deployment*: for each incoming packet y, compute the anomaly score:

$$\text{score}(y) = \begin{cases} \text{normal}, & \text{if } d(\mu, y) \leq t_a \\ \text{anomaly}, & \text{otherwise} \end{cases}$$

Implementation Details

- Mechanism for performing *inline* anomaly detection:
  - `netfilter` linux firewall
  - `libnetfilter_queue` for queueing packets to user space
- `libnet` for packet creation and delivery in the redirection mechanism
- Prototype deployed on recent `Debian` system acting as a central router between client system and the production / shadow system
- Client system: `Apache Flood`
- Production system: `OpenBSD Apache` server
- Shadow system: `OpenBSD Apache` server with `Systrace`
- Everything hosted on `VMware ESX Server 3`

Experiments – Impact of Instrumentation



Different scenarios:

HTML returns just a static HTML page

PHP returns a dynamic, PHP generated page

MYSQL returns a dynamic, PHP generated page with values read from a MYSQL database.

Experiments – Packet Filter Actions

Type	normal	anomaly	sink	red-1st	red-next
HTML	1.47	2.05	1.64	235.63	1.62
PHP	3.08	3.59	3.36	238.25	3.13
MYSQL	30.71	31.09	30.72	242.32	30.75

Packet filter action scenarios:

anomaly the distance of each packet to a centroid is calculated and compared to t_a

sink each packet is logged to the forensic sink

red-1st each connection is redirected

red-next translation of sequence numbers, addresses and ports for redirection of subsequent packets

Experiments – Evaluation Dataset

- Normal data from incoming HTTP traffic of our institute:
  - 150k unsanitized connections (totaling to roughly 240k packets) of 10 consecutive days
  - Split into three equal parts of 50k connections each for training, validation and testing
- Attack Data:
  - 100 instances (470 connections totaling to 2960 packets) of recent exploits in the `Metasploit` framework
  - `Nessus` HTTP scans
- Evaluation criterion: $AUC_{0.01}$(area under ROC-curve with false positive rate ≤ 0.01)

Experiments – Accuracy I

- Results on test dataset:
  - 3102 ($\sim 0.05\%$) packets with payload are redirected
  - 111 ($\sim 0.001\%$) packets with payload are logged to the forensic sink
  - 58,369 packets with payload are processed as normal
- Ratios for the evaluation of the system:

$$\text{broken} = \frac{\#\ \text{normal conn. in SINK}}{\#\ \text{all normal conn.}} = 0.0008$$

$$\text{jailed} = \frac{\#\ \text{attack conn. in REDIRECT}}{\#\ \text{all attack conn.}} = 0.9760$$

Experiments – Accuracy II

Type	True positive rate	False positive rate
plain AD	0.9939 ± 0.0030	0.0092 ± 0.0105
AD with redirect	0.9952 ± 0.0022	0.0017 ± 0.0009

- Comparison against "plain anomaly detector", i.e. system without the REDIRECT/SINK extension
- Improves both true positive and false positive rate

Overview
System Architecture
Detection State Machine
Redirection
Anomaly Detection
Embedding and Similarity Measures
Anomaly Score
Implementation
Experiments
Runtime
Accuracy
Conclusions

Conclusions

- *Inline* intrusion prevention system which
  - ... is *application-independent*
  - ... decides at the *network layer*
  - ... performs anomaly detection at the *application layer*
- Minor performance impact ($\leq$ 0.5 ms per packet)
- System significantly improves both true positive and false positive rate
- Limitation: requires synchronization

Overview
System Architecture
Detection State Machine
Redirection
Anomaly Detection
Embedding and Similarity Measures
Anomaly Score
Implementation
Experiments
Runtime
Accuracy
Conclusions

Questions? Remarks?

Thanks for your attention!

Evaluation guideline

Target	Normal Traffic	Attack Traffic
REDIRECT	True neg.	True pos.
SINK	False pos.	True pos.