Selecting and Improving System Call Models for Anomaly Detection

(or, 30 minutes before CIPHER 5's CTF results)

Alessandro Frossi Federico Maggi Gian Luigi Rizzo
Stefano Zanero

July 10, 2009

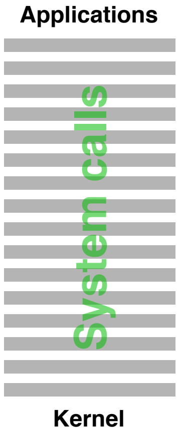
Topic of this talk

System Call Based Anomaly Detection

Detecting intrusions using system call flows w/ data models

System Call Based Anomaly Detection

Detecting intrusions using system call flows w/ data models

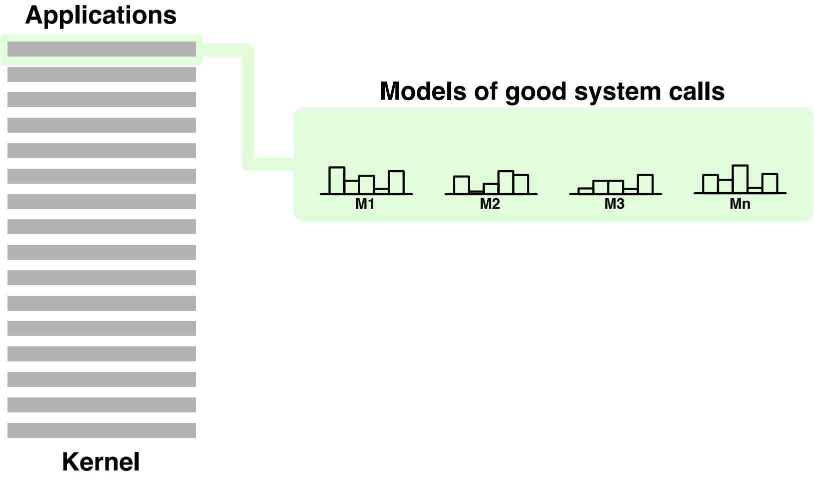


1. run applications into processes
2. intercept system calls
3. create models of good system calls
4. flag deviations to detect anomalies

Let's take a look at a simple, generic example.

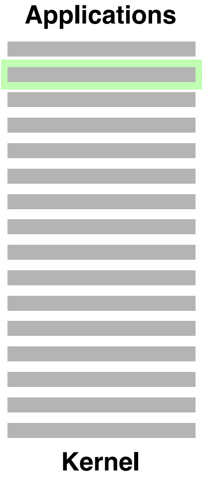
System Call Based Anomaly Detection

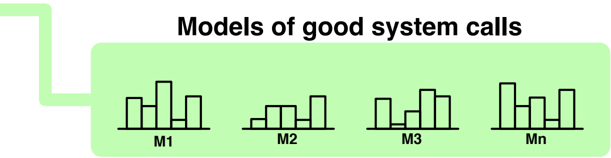
A set of models is created based on certain features of the system calls



System Call Based Anomaly Detection

Models estimate feature values observed in “good” system calls



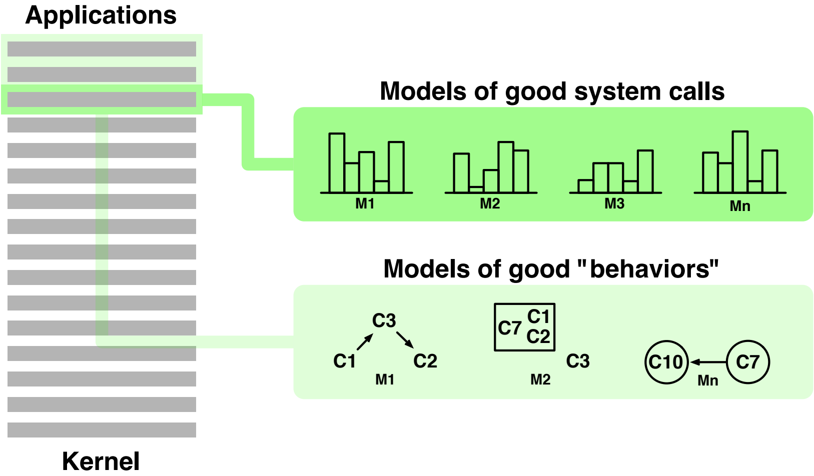


Example of models

— string length
— number of arguments
— function name
— string character distribution

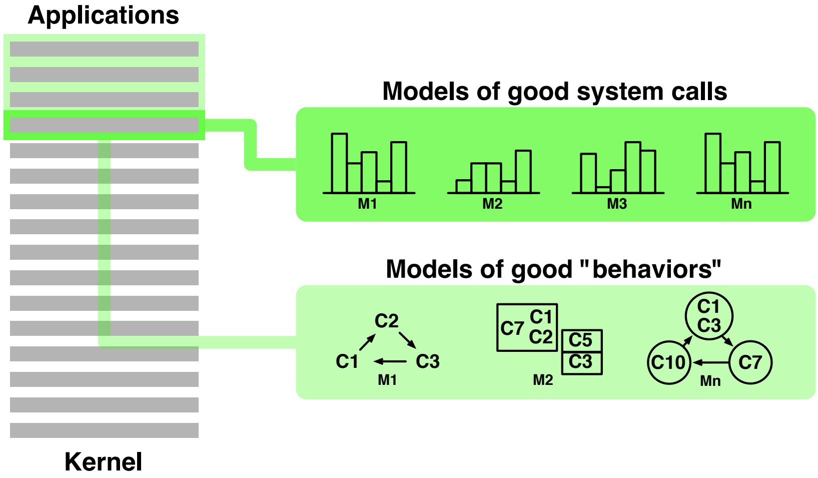
System Call Based Anomaly Detection

Estimations become more accurate as more system calls are analyzed



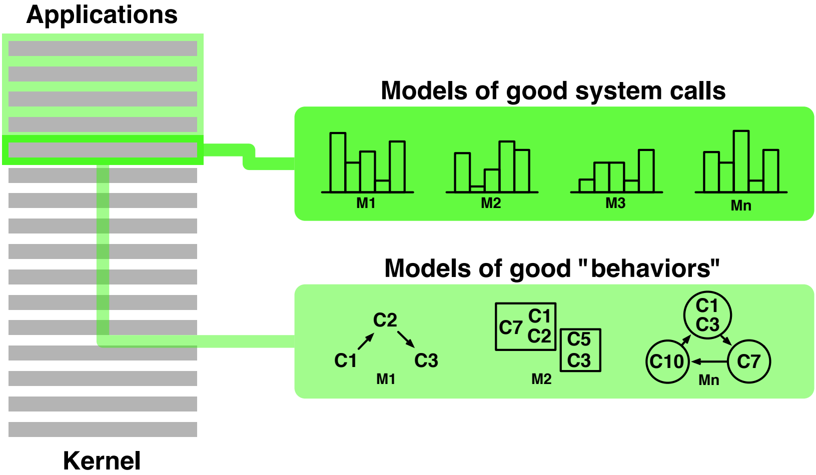
System Call Based Anomaly Detection

Also, models based on sets of system calls can be constructed



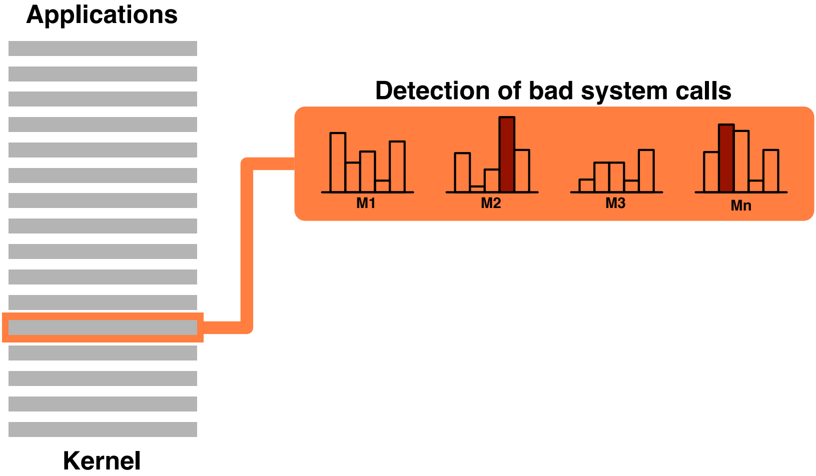
System Call Based Anomaly Detection

Knowledge about system calls' context is learned



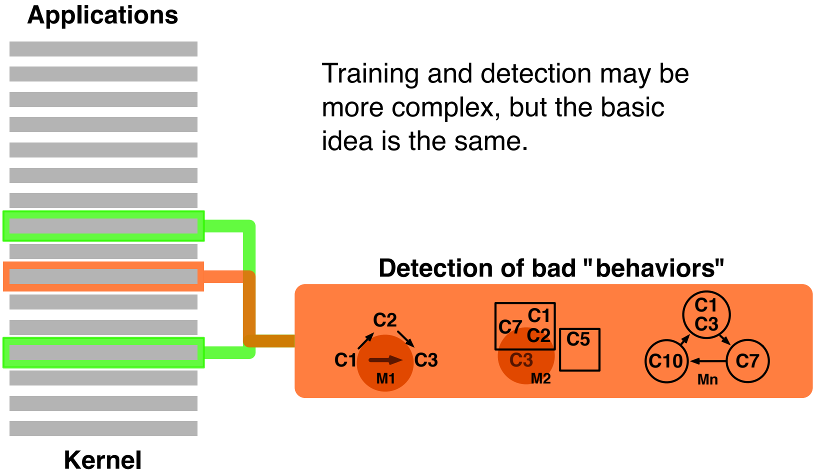
System Call Based Anomaly Detection

In detection mode, the same models can be used to spot malicious system calls...



System Call Based Anomaly Detection

...or malicious execution contexts



Training and detection may be more complex, but the basic idea is the same.

The systems we analyzed

Different Approaches: Deterministic vs. Stochastic

We analyzed two anomaly detectors based on different approaches

Different Approaches: Deterministic vs. Stochastic

We analyzed two anomaly detectors based on different approaches

FSA-DF [IEEE S&P 2006]

- Deterministic
- Control-flow: FSA
- Data-flow: unary/binary relations

Different Approaches: Deterministic vs. Stochastic

We analyzed two anomaly detectors based on different approaches

FSA-DF [IEEE S&P 2006]

- Deterministic
- Control-flow: FSA
- Data-flow: unary/binary relations

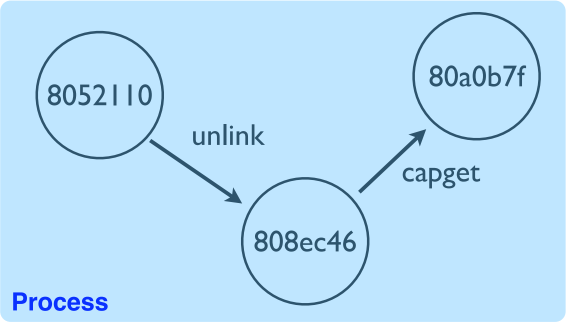
S^{2}A^{2}DE [IEEE TODS 2009]

- Stochastic
- Control-flow: Markov-chain
- Data models: clusters

Deterministic Data-flow Anomaly Detection

The system calls generated by each process are examined

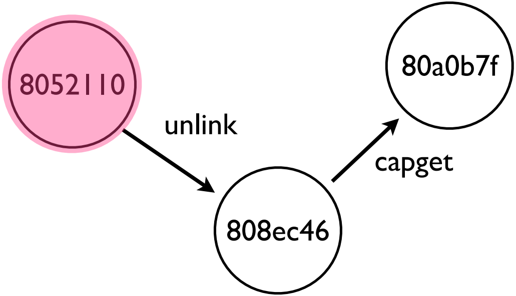
5866	8052110	unlink("/usr/local/var/proftpd/test.sock") = 0
5866	808ec46	capget(DONT_CARE, DONT_CARE) = 0
5866	80a0b7f	timer_gettime (DONT_CARE, DONT_CARE) = 3



Deterministic Data-flow Anomaly Detection

Different PCs means different process states

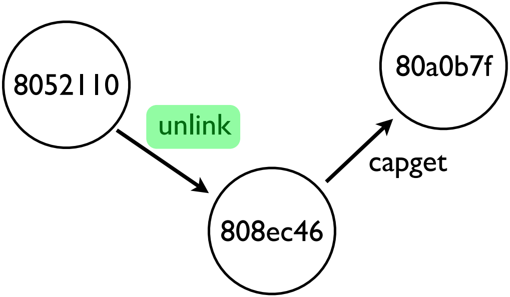
5866 8052110 unlink("/usr/local/var/proftpd/test.sock") = 0
5866 808ec46 capget(DONT_CARE, DONT_CARE) = 0
5866 80a0b7f timer_gettime (DONT_CARE, DONT_CARE) = 3



Deterministic Data-flow Anomaly Detection

A system call changes the process' state...

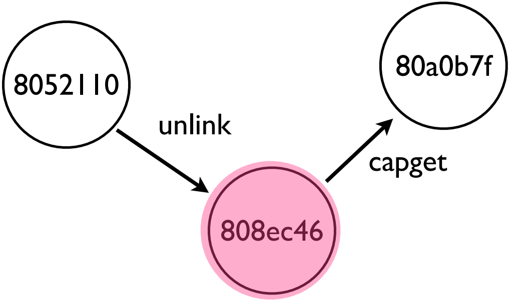
5866	8052110	unlink("/usr/local/var/proftpd/test.sock") = 0
5866	808ec46	capget(DONT_CARE, DONT_CARE) = 0
5866	80a0b7f	timer_gettime (DONT_CARE, DONT_CARE) = 3



Deterministic Data-flow Anomaly Detection

...and so forth

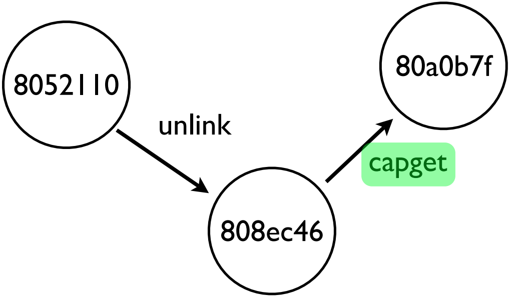
5866	8052110	unlink("/usr/local/var/proftpd/test.sock") = 0
5866	808ec46	capget(DONT_CARE, DONT_CARE) = 0
5866	80a0b7f	timer_gettime (DONT_CARE, DONT_CARE) = 3



Deterministic Data-flow Anomaly Detection

This analysis is repeated until termination

5866	8052110	unlink("/usr/local/var/proftpd/test.sock") = 0
5866	808ec46	capget(DONT_CARE, DONT_CARE) = 0
5866	80a0b7f	timer_gettime (DONT_CARE, DONT_CARE) = 3

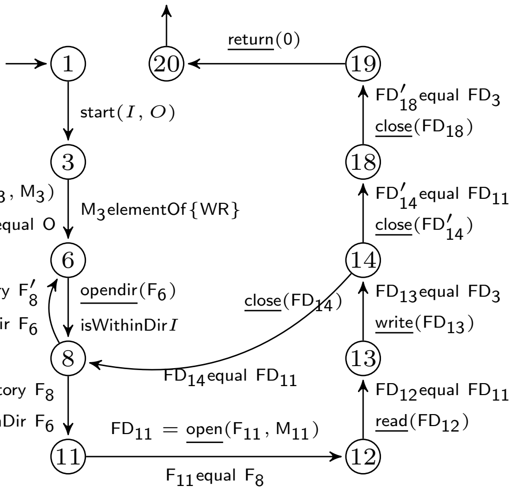


Deterministic Data-flow Anomaly Detection

A network of unary/binary data-flow relations on top of the process' FSA

Deterministic Data-flow Anomaly Detection

A network of unary/binary data-flow relations on top of the process' FSA



Deterministic Data-flow Anomaly Detection

Other types of relations

- **Unary**: capture properties of a single argument.
  - **equal**
  - **elementOf**
  - **subsetOf**
  - **range**
  - **isWithinDir**
  - **hasExtension**
- **Binary**: capture relations between two arguments.
  - **equal**
  - **isWithinDir**
  - **contains**
  - **hasSameDirAs**
  - **hasSameBaseAs**
  - **hasSameExtensionAs**

Major Drawback: False Positives

Mostly due to the deterministic relations

Major Drawback: False Positives

Mostly due to the deterministic relations

$$\text{open}(\underline{\textbf{“/tmp/php1553”}},\ 0,\ 0\text{x1b6}) = 5$$

$$\text{unary } \textbf{elementOf}(\{/tmp/php1553, /tmp/php9022\})$$

$$\text{open}(\underline{\textbf{“/tmp/php9022”}},\ 0,\ 0\text{x1b6}) = 5$$

Major Drawback: False Positives

Mostly due to the deterministic relations

open(<u>**"/tmp/php1553"**</u>, 0, 0x1b6) $=$ 5

unary **elementOf**$(\{/tmp/php1553, /tmp/php9022\})$

open(<u>**"/tmp/php9022"**</u>, 0, 0x1b6) $=$ 5

What if "/tmp/php1990" is found?

Major Drawback: False Positives

Mostly due to the deterministic relations

$$\text{open}(\underline{\textbf{“/tmp/php1553”}},\ 0,\ 0\text{x}1\text{b}6) = 5$$

$$\text{unary } \textbf{elementOf}(\{/tmp/php1553, /tmp/php9022\})$$

$$\text{open}(\underline{\textbf{“/tmp/php9022”}},\ 0,\ 0\text{x}1\text{b}6) = 5$$

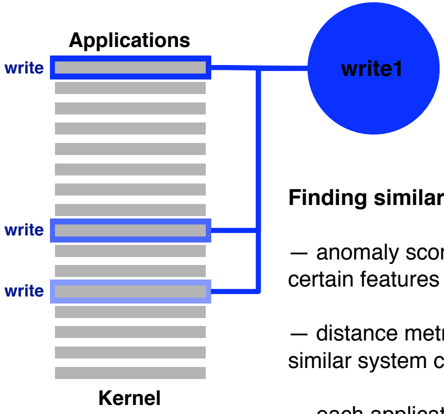
What if "/tmp/php1990" is found?
These false positives occur pretty often.

Stochastic Behavior Profiling of Processes

Clusters of similar system calls interconnected by Markov-chains

Stochastic Behavior Profiling of Processes

Clusters of similar system calls interconnected by Markov-chains



Finding similar system calls

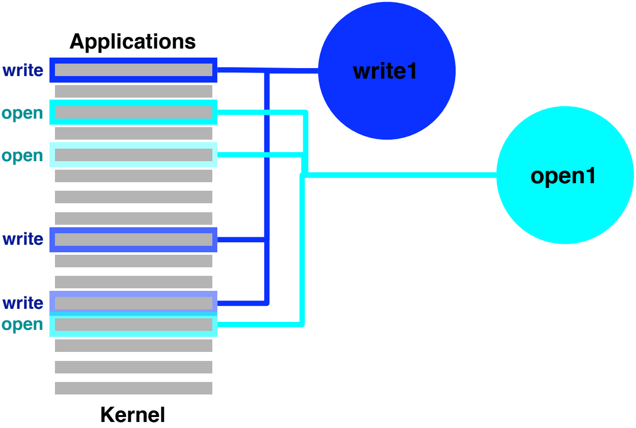
— anomaly scores [ACM TISSEC 2006] based on certain features of the arguments

— distance metrics [ACM TODS 2009] used to cluster similar system calls

— each application's process creates different clusters

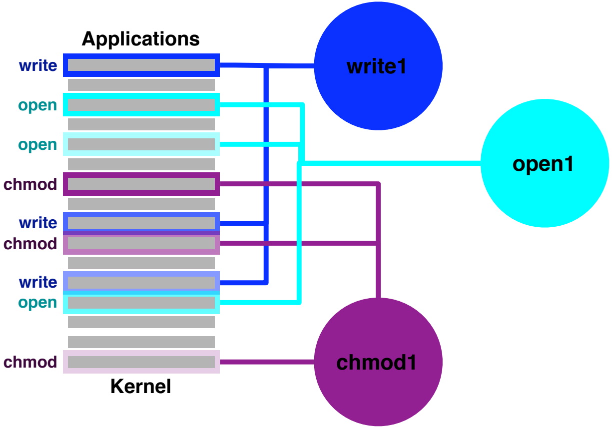
Stochastic Behavior Profiling of Processes

Clusters of similar system calls interconnected by Markov-chains



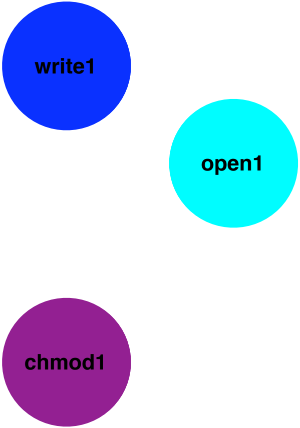
Stochastic Behavior Profiling of Processes

Clusters of similar system calls interconnected by Markov-chains



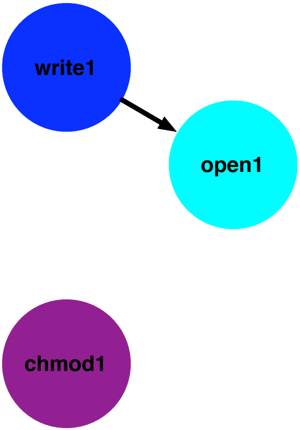
Stochastic Behavior Profiling of Processes

Clusters of similar system calls interconnected by Markov-chains



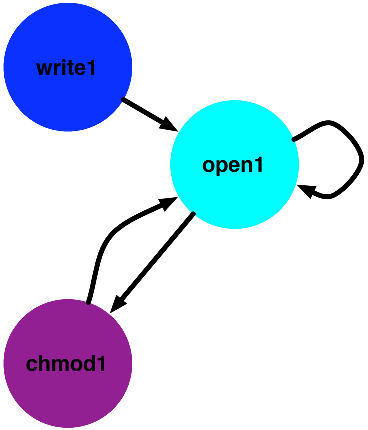
Stochastic Behavior Profiling of Processes

Clusters of similar system calls interconnected by Markov-chains



Stochastic Behavior Profiling of Processes

Clusters of similar system calls interconnected by Markov-chains



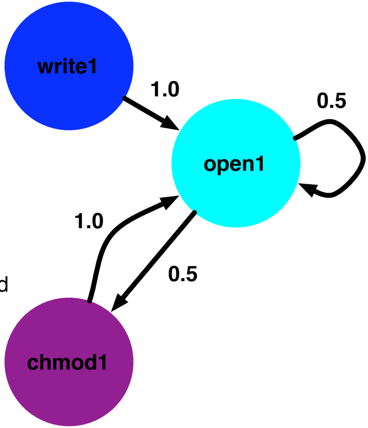
Stochastic Behavior Profiling of Processes

Clusters of similar system calls interconnected by Markov-chains

Markov-chains encode process behavior

— transitions between system calls can occur with different probabilities [ACM TODS 2009]

— a call is anomalous if either there is no matching state (i.e., cluster) or transition probability is violated



Major Drawbacks: False Negatives

Mostly due to the stochastic nature of Markov-chains

Major Drawbacks: False Negatives

Mostly due to the stochastic nature of Markov-chains

Clustering

- clustering depends on configuration parameters
- different paramenters $\longrightarrow$ different results

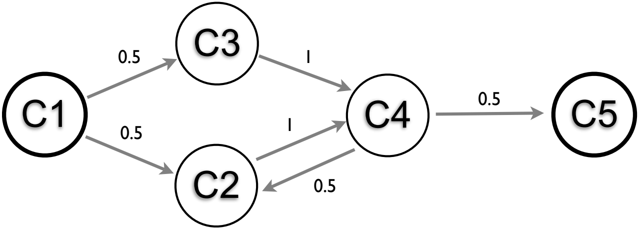
Major Drawbacks: False Negatives

Mostly due to the stochastic nature of Markov-chains

Clustering

- clustering depends on configuration parameters
- different paramenters $\longrightarrow$ different results

Markov-chains (example)



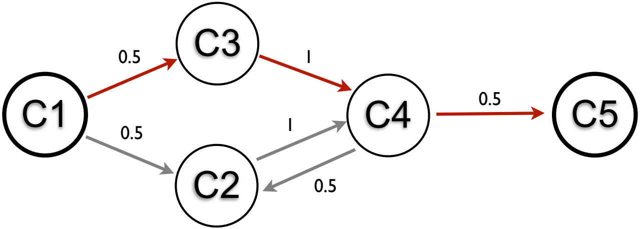
Major Drawbacks: False Negatives

Mostly due to the stochastic nature of Markov-chains

Clustering

- clustering depends on configuration parameters
- different paramenters $\longrightarrow$ different results

Markov-chains (example)



$$\text{Threshold} = 0.5 * 1 * 0.5 = 0.25$$

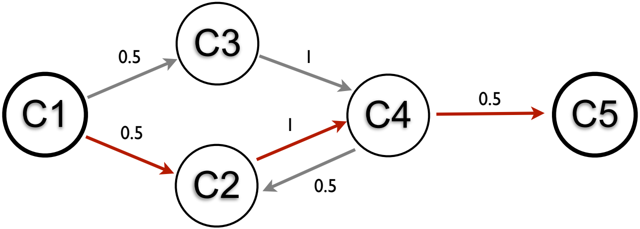
Major Drawbacks: False Negatives

Mostly due to the stochastic nature of Markov-chains

Clustering

- clustering depends on configuration parameters
- different paramenters $\longrightarrow$ different results

Markov-chains (example)



$$\text{Threshold} = 0.5 * 1 * 0.5 = 0.25$$

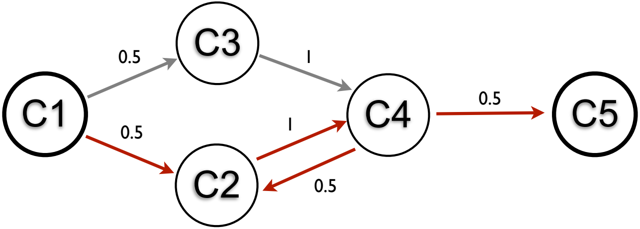
Major Drawbacks: False Negatives

Mostly due to the stochastic nature of Markov-chains

Clustering

- clustering depends on configuration parameters
- different paramenters $\rightarrow$ different results

Markov-chains (example)



$$\text{Threshold} = 0.5 * (1 * 0.5)^n * 0.5 \rightarrow 0$$

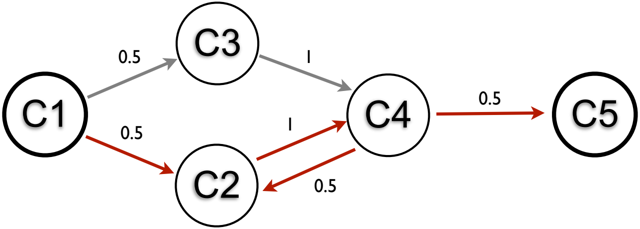
Major Drawbacks: False Negatives

Mostly due to the stochastic nature of Markov-chains

Clustering

- clustering depends on configuration parameters
- different paramenters $\rightarrow$ different results

Markov-chains (example)



$$\text{Threshold} = 0.5 * (1 * 0.5)^n * 0.5 \rightarrow 0$$

No valid threshold can be found if cycles are not of fixed length.

For instance, DoS attacks may not be detected.

Pros and cons of the two approaches

Pros and cons of the two approaches

	FSA-DF		S^2A^2DE	
FSA	• Perfectly models a software behavior • No False Negatives • Doesn't allow deviations	Control Flow	• Introduces a statistical approach • False Negatives • Few False Positives	MCM
Relations	• Deterministic approach • No new input adaptation • Prone to False Positives • No False Negatives	Data Flow	• Stochastic approach • Can adapt to new inputs • Few False Positives • False Negatives	Clusters

First contribution:
combination of the two approaches

Combining Complementary Approaches

Deterministic control-flow + stochastic data models

	Hybrid IDS			
	FSA-DF		S^2A^2DE	
FSA	• Perfectly models a software behavior • No False Negatives • Doesn't allow deviations	Control Flow		
		Data Flow	• Stochastic approach • Can adapt to new inputs • Few False Positives • False Negatives	Models

Combining Complementary Approaches

The learning algorithm is similar to that used in FSA-DF

- $\forall couple\langle syscall_{i-1}, syscall_i\rangle \in \{TrainingSet\}$
  - make state
  - learn relations

Combining Complementary Approaches

The learning algorithm is similar to that used in FSA-DF

- $\forall couple\langle syscall_{i-1}, syscall_i\rangle \in \{TrainingSet\}$
  - make state
  - learn relations
    - equal
    - elementOf
    - subsetOf
    - range
    - isWithinDir
    - hasExtension
    - isWithinDir
    - contains
    - hasSameDirAs
    - hasSameBaseAs
    - hasSameExtensionAs

Combining Complementary Approaches

The learning algorithm is similar to that used in FSA-DF

- $\forall couple\langle syscall_{i-1}, syscall_i\rangle \in \{TrainingSet\}$
  - make state
  - learn relations
    - ~~equal~~ save model of similar strings
    - ~~elementOf~~ save model of similar strings
    - subsetOf
    - range
    - isWithinDir
    - hasExtension
    - isWithinDir
    - ~~contains~~ save model of similar strings
    - hasSameDirAs
    - hasSameBaseAs
    - hasSameExtensionAs

Combining Complementary Approaches

The learning algorithm is similar to that used in FSA-DF

- learn string domains
- $\forall couple\langle syscall_{i-1}, syscall_i\rangle \in \{TrainingSet\}$
  - make state
  - learn relations
    - ~~equal~~ save model of similar strings
    - ~~elementOf~~ save model of similar strings
    - subsetOf
    - range
    - isWithinDir
    - hasExtension
    - isWithinDir
    - ~~contains~~ save model of similar strings
    - hasSameDirAs
    - hasSameBaseAs
    - hasSameExtensionAs

Paths And Filenames

How to find groups of good strings into execve/open/read/... args?

/var/log/http.0 . . . /etc/ftp.conf . . . /tmp/php1231
. . . /var/run/nfsd.pid . . . /etc/smb/samba.conf
. . . /opt/local/lib/libncurses.a . . . /usr/lib/libkmod.a
. . . /tmp/uscreens/427.ttys000 . . . /var/db/ntp.drift . . .

Paths And Filenames

How to find groups of good strings into execve/open/read/... args?

/var/log/http.0 . . . /etc/ftp.conf . . . /tmp/php1231
. . . /var/run/nfsd.pid . . . /etc/smb/samba.conf
. . . /opt/local/lib/libncurses.a . . . /usr/lib/libkmod.a
. . . /tmp/uscreens/427.ttys000 . . . /var/db/ntp.drift . . .

Self-Organizing Map

Type of artificial neural network, trained using unsupervised learning to produce a multi dimensional discretized representation of the input space of the training samples, called map.

Idea

- SOM to capture classes of good strings.
- Model of good strings $\rightarrow$ nodes.
- Similar strings $\rightarrow$ neighbor nodes.

Paths And Filenames

How to find groups of good strings into execve/open/read/... args?

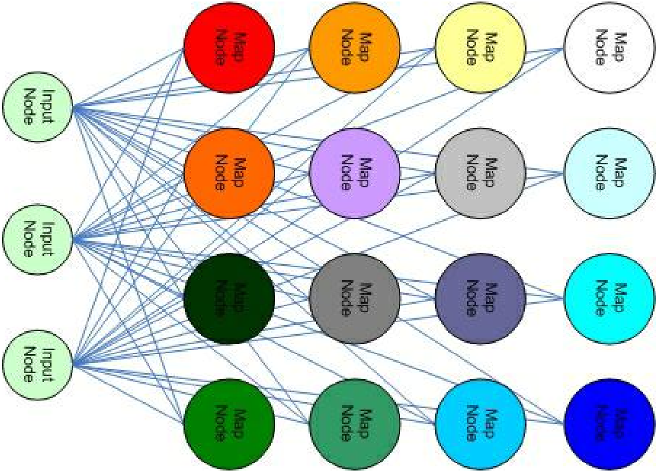
/var/log/http.0 . . . /etc/ftp.conf . . . /tmp/php1231 . . . /var/run/nfsd.pid . . . /etc/smb/samba.conf . . . /opt/local/lib/libncurses.a . . . /usr/lib/libkmod.a . . . /tmp/uscreens/427.ttys000 . . . /var/db/ntp.drift . . .

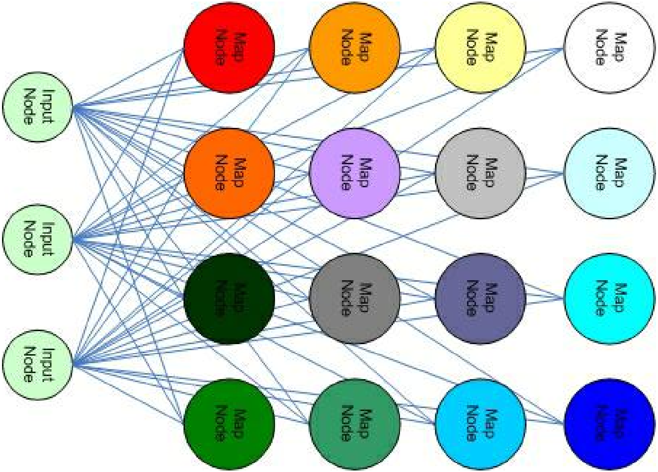
Self-Organizing Map

Type of artificial neural network, trained using unsupervised learning to produce a multi dimensional discretized representation of the input space of the training samples, called map.

Idea

- SOM to capture classes of good strings.
- Model of good strings → nodes.
- Similar strings → neighbor nodes.


Input Node
Input Node
Input Node
Map Node
Map Node
Map Node
Map Node
Map Node
Map Node
Map Node
Map Node
Map Node
Map Node
Map Node
Map Node
Map Node
Map Node
Map Node
Map Node



OK, they look pretty nice. But why SOMs?

Paths And Filenames

Integration in Hybrid IDS - algorithm

- create SOM of all paths
  - SOM initialization with linux directory structure.
  - Extract all the paths from the syscalls
  - SOM training [Kohonen 2004] with a randomized subset of the paths.
- $\forall couple\langle syscall_{i-1}, syscall_i\rangle \in \{TrainingSet\}$
  - make state
  - learn relations
    - if $syscall_{i-1}$ contains a path argument
find BMU from the SOM
add BMU to the edge
    - subsetOf
    - range
    - isWithinDir
    - hasExtension
    - isWithinDir
    - hasSameDirAs
    - hasSameBaseAs
    - hasSameExtensionAs

Second contribution:
improved system call models

Improved System Call Models

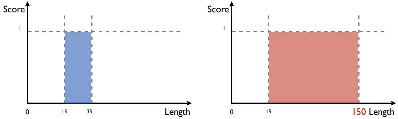
New models to reduce false detections

- **Goal 1: Resillience to spurious strings in the datasets.**

Improved System Call Models

New models to reduce false detections

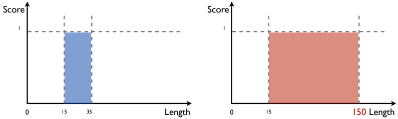
- **Goal 1: Resillience to spurious strings in the datasets.**
  - Long/short strings in the training data can bias interval based models.



Improved System Call Models

New models to reduce false detections

- **Goal 1: Resillience to spurious strings in the datasets.**
  - Long/short strings in the training data can bias interval based models.

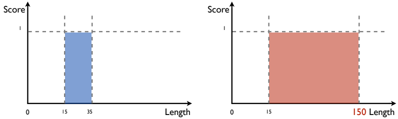


- **Goal 2: Detect simple DoS attacks.**

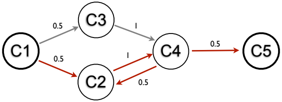
Improved System Call Models

New models to reduce false detections

- **Goal 1: Resillience to spurious strings in the datasets.**
  - Long/short strings in the training data can bias interval based models.



- **Goal 2: Detect simple DoS attacks.**
  - i.e., process forced to execute the same code region until crash.



Argument Length Using Gaussian Intervals

Yields to less false positives

Argument Length Using Gaussian Intervals

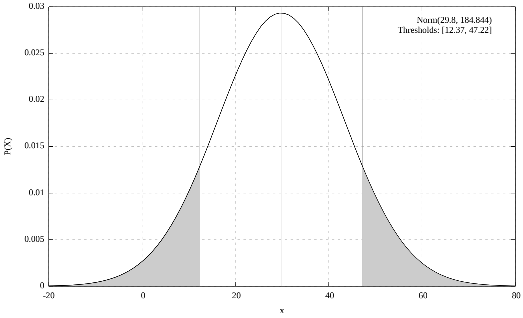
Yields to less false positives

Statistics: to estimate the distribution of args length

- $|args| = X_{args} \sim \mathcal{N}(\mu, \sigma^2)$
- Sample Mean, Sample Variance.

Model precision parameter:

- Kurtosis $\hat{\gamma}_X = \frac{\hat{\mu}_{X,4}}{\hat{\sigma}_X^4} - 3$
- If $\gamma_{X_{args}} < 0$ the sample is spread on a big interval



Anomaly threshold: percentile T_{args} centered on the mean.

Argument Length Using Gaussian Intervals

Integration in Hybrid IDS - algorithm

- create SOM of all paths
- $\forall couple \langle syscall_{i-1}, syscall_i \rangle \in \{TrainingSet\}$
  - make state
  - learn relations
    - save BMU
    - subsetOf
    - ~~range~~ save string length or num. value
    - isWithinDir
    - hasExtension
    - isWithinDir
    - hasSameDirAs
    - hasSameBaseAs
    - hasSameExtensionAs

Mitigating DoS Using Edge Frequency Models

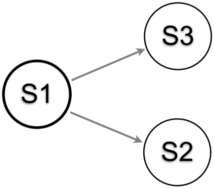
Yields to less false negatives

Mitigating DoS Using Edge Frequency Models

Yields to less false negatives

Given that:

- each FSA edge is traversed a variable number of times over multiple executions
- the traversal frequency has a range

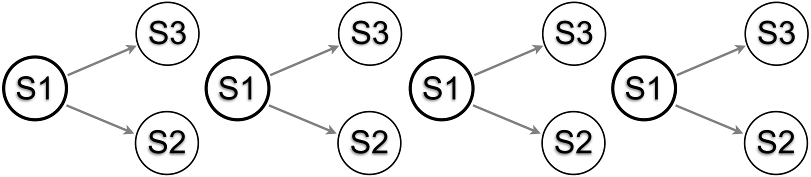


Idea: estimate a validity interval to detect DoS attacks.

Edge Traversal Frequency

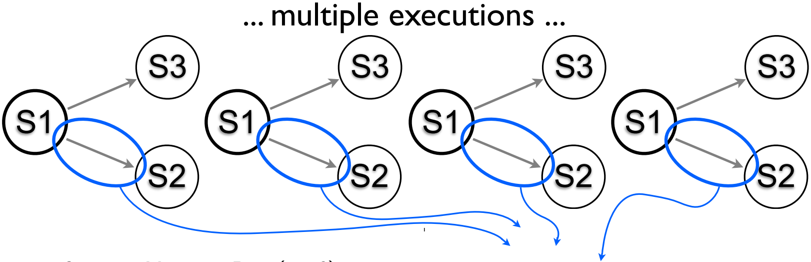
The Model

... multiple executions ...

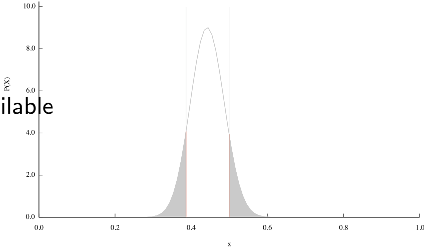


Edge Traversal Frequency

The Model



- *freqs* $= X_{freqs} \sim Beta(\alpha, \beta)$
- Estimated α and β
- Interval from x-th percentile
- **Not** estimated if few values available



Mitigating DoS Using Edge Frequency Models

Integration in Hybrid IDS - algorithm

- create SOM of all paths
- $\forall couple \langle syscall_{i-1}, syscall_i \rangle \in \{TrainingSet\}$
  - make state
  - learn relations
    - save BMU
    - subsetOf
    - save string length or num. value
    - isWithinDir
    - hasExtension
    - isWithinDir
    - hasSameDirAs
    - hasSameBaseAs
    - hasSameExtensionAs
  - save edge traverse count

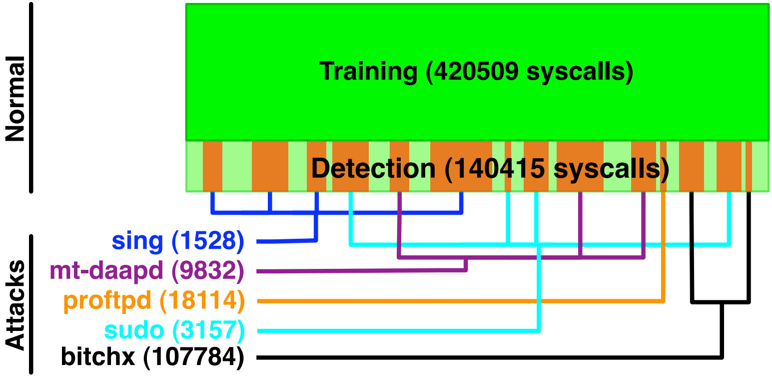
How We Built The Evaluation Dataset

How We Built The Evaluation Dataset

- “normal” execution of 5 tools (420509 syscalls)
- recent exploits from CVE (140415 syscalls)

How We Built The Evaluation Dataset

- "normal" execution of 5 tools (420509 syscalls)
- recent exploits from CVE (140415 syscalls)



Accuracy Evaluation

No false negatives (deterministic control-flow) + almost-zero false positives (stochastic data models)

	`sing`	`mt-daapd`	`profdtpd`	`sudo`	`BitchX`
Traces	22	18	21	22	15
Syscalls	1528	9832	18114	3157	107784
S^2A^2DE	10.0%	0%	0%	10.0%	0.0%
FSA-DS	5.0%	16.7%	28%	15.0%	0.0%
Hybrid IDS	0.0%	0%	0%	10.0%	0.0%

	`mcweject`	`bsdtar`
Traces	12	2
Syscalls	75	102
S^2A^2DE	0.0%	8.7%
SOM-S^2A^2DE	0.0%	0.0%

Table 2. Comparison of the FPR of S^2A^2DE vs. FSA-DF vs. Hybrid IDS and S^2A^2DE vs. SOM-S^2A^2DE. Values include the number of traces used. Accurate description of the impact of each *individual* model is in Section 4.2 (first five columns) and 4.3 (last two columns).

Accuracy Evaluation

No false negatives (deterministic control-flow) + almost-zero false positives (stochastic data models)

	sing	mt-daapd	profdtpd	sudo	BitchX	mcweject	bsdtar	
Traces	22	18	21	22	15	12	2	
Syscalls	1528	9832	18114	3157	107784	75	102	
S^2A^2DE	10.0%	0%	0%	10.0%	0.0%	0.0%	8.7%	**S^2A^2DE**
FSA-DS	5.0%	16.7%	28%	15.0%	0.0%	0.0%	0.0%	**SOM-S^2A^2DE**
Hybrid IDS	0.0%	0%	0%	10.0%	0.0%			

Table 2. Comparison of the FPR of S^2A^2DE vs. FSA-DF vs. Hybrid IDS and S^2A^2DE vs. SOM-S^2A^2DE. Values include the number of traces used. Accurate description of the impact of each *individual* model is in Section 4.2 (first five columns) and 4.3 (last two columns).

- sing - write on arbitrary file (data-flow).
- mt-daapd - arbitrary code execution (data-flow + DoS).
- proftpd - arbitrary command exeuction (data-/control-flow).
- sudo - arbitrary command execution (control-flow).
- bitchx - arbitrary code execution (control-flow + DoS).

Performance Evaluation

Not-so-negligible overhead, but mostly due to ptrace

	`sing`	`sudo`	`BitchX`	`mcweject`	`bsdtar`	**Avg. speed**
System calls	3470	15308	12319	97	705	
S^{2}A^{2}DE	0.4	0.8	1.9	0.1	0.1	8463
FSA-DF	1.3	1.5	1.2	-	-	7713
Hybrid IDS	29	5.8	27.7	-	-	1067
SOM-S^{2}A^{2}DE	-	-	-	8.8	19	25

Table 3. Detection performance measured in "seconds per system call". The average speed is measured in system calls per second (last column).

Conclusions and Future Works

Solve performance issues due to SOMs

- **determinisitc** models accurately capture the **control**-flow
- **stochastic** models accurately capture **data**-flow features
- a **hybrid** approach lowers false detections
- performance issues:
  - the optimization of BMUs lookup is the first item on our TODO list
  - the use of a faster system call interceptor the second one ;)