Anomaly detection through system call argument analysis

Stefano Zanero
Ph.D. Student, Politecnico di Milano
CTO & Founder, Secure Network S.r.l.

Black Hat Briefings – Amsterdam, 03/04/06
Presentation Outline

- Building a case for Anomaly Detection Systems
  - Bear with me if you already heard this rant :)  
  - Intrusion Detection Systems, not Software!
  - Why do we need Anomaly Detection?

- State of the art in host-based anomaly detection
  - System call sequence analysis (a lot of)
  - System call argument analysis (a few of)

- Combining both, along with other ingredients

- Detecting 0-day attacks: hope or hype?

- Conclusions
The defender's problem
- The defender needs to plan for everything... the attacker needs just to hit one weak point
- Being overconfident is fatal: King Darius vs. Alexander Magnus, at Gaugamela (331 b.C.)

Acting *sensibly* is the key ("Beyond fear", by Bruce Schneier: a must read!)

"The only difference between systems that can fail and systems that cannot possibly fail is that, when the latter actually fail, they fail in a totally devastating and unforeseen manner that is usually also impossible to repair" (Murphy's law on complex systems)
Murphy says: plan for the worst

- The mantra is: **plan for the worst** (and pray it will not get even worse than that) and act accordingly

- At the end of the day, we must keep in mind that every defensive system will, at some time, fail, so we must plan for failure
  - We must design systems to *withstand* attacks, and fail gracefully (failure-tolerance)
  - **We must design systems to be tamper evident** (detection)
  - We must design systems to be capable of recovery (reaction)
An information system must be designed for \emph{tamper evidence} (because it \emph{will} be broken into, sooner or later)

An IDS is a \emph{system} which is capable of detecting intrusion attempts on an \emph{information system}

- An IDS is a system, not a software!
- An IDS works on an information system, not on a network!

The so-called IDS software packages are a \emph{component} of an intrusion detection system

An IDS system usually closes its loop on a human being (who is an essential part of the system)
Breaking some hard-to-kill myths

- An IDS is a system, not a software
  - A skilled human looking at logs is an IDS
  - A skilled network admin looking at TCPdump is an IDS
  - A company maintaining and monitoring your firewall is an IDS
  - A box bought by a vendor and plugged into the network is not an IDS by itself

- An IDS is not a panacea, it’s a component
  - Does not substitute a firewall, nor it was designed to (despite what Gartner thinks)
  - It’s the last component to add to a security architecture, not the first

- Detection without reaction is a no-no
  - Like burglar alarms with no guards!

- Reaction without human supervision is a dream
  - “Network, defend thyself!”
<table>
<thead>
<tr>
<th>Anomaly Detection Model</th>
<th>Misuse Detection Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describes normal behaviour, and flags deviations</td>
<td>Uses a knowledge base to recognize the attacks</td>
</tr>
<tr>
<td>Uses statistical or machine learning models of behaviour</td>
<td>Can recognize only attacks for which a “signature” exists in</td>
</tr>
<tr>
<td>Theoretically able to recognize any attack, also 0-days</td>
<td>the KB</td>
</tr>
<tr>
<td>Strongly dependent on the model, the metrics and the thresholds</td>
<td>When new types of attacks are created, the language used to</td>
</tr>
<tr>
<td></td>
<td>express the rules may not be expressive enough</td>
</tr>
<tr>
<td>Generates statistical alerts: “Something’s wrong”</td>
<td>Problems for polymorphism</td>
</tr>
<tr>
<td></td>
<td>The alerts are precise: they recognize a specific attack,</td>
</tr>
<tr>
<td></td>
<td>giving out many useful informations</td>
</tr>
</tbody>
</table>
Misuse detection alone is an awful idea

- Misuse detection systems rely on a knowledge base (think of the anti-virus example, if it’s easier to grasp)
- Updates continuously needed, and not all the attacks become known (as opposed to viruses)
  - A misuse based IDS will not, in general, recognize a zero-day attack
- Attacks are polymorphs, more than computer viruses (human ingenuity vs computer program)
  - Think of ADMutate, UTF encoding...
  - A misuse based IDS will not, in general, recognize a new way to exploit an old attack, unless there is an unescapably necessary characteristic in the attack
- If we need intrusion detection as a complementary mean to patching and secure design, detecting known attacks is clearly not the solution
- Traditionally, network based IDS are mostly misuse based
Anomaly Detection, perhaps not better

- **Task:** describe the normal behaviour of a system
  - Which features/variables/metrics would you use?
  - Infinite models to fit them
- **Thresholds** must be chosen to minimize false positive vs. detection rate: a difficult process
- **The base model** is fundamental
  - If the attack shows up only in variables we discarded, or only in variations we do not check, we cannot detect it
  - Think of detecting oscillations when you just check the average of a variable on a window of time
- In any case, what we get as an alert is “hey, something’s wrong here”. What? Your guess!
- **Difficult** to be relied upon for automatic defense (i.e. IPS)
Our approach: unsupervised learning

- At the Politecnico di Milano Performance Evaluation lab we are working on anomaly-based intrusion detection systems capable of unsupervised learning

- What is a learning algorithm?
  - It is an algorithm whose performances grow over time
  - It can extract information from training data

- Supervised algorithms learn on labeled training data
  - “This is a good packet, this is not good”
  - Think of your favorite bayesian anti-spam filter
  - It is a form of generalized misuse detection

- Unsupervised algorithms learn on unlabeled data
  - They can “learn” the normal behavior of a system and detect variations (remembers something ... ?)

- We have already presented in past our network based IDS, we are presenting today our host based IDS
State of the art

- Host-based, anomaly based IDS have a long academic tradition, and there's a gazillion papers on them
- Let us focus on one observed feature: the sequence of system calls executed by a process during its life
- Assumption: this sequence can be characterized, and abnormal deviations of the process execution can be detected
- Earlier studied focused on the sequence of calls
  - Used markovian algorithms, wavelets, neural networks, finite state automata, N-grams, whatever, but just on the sequence of calls
  - Markov models comprise other models
- An interesting and different approach was introduced by Vigna et al. with “SyscallAnomaly/LibAnomaly”, but we'll see that in due time
Time series learning

- A time series is a sequence of observations on a variable made over some time
- If a syscall is an observation, then a program is a time series of syscalls
- If our observations are descriptive of the behavior of systems... attacks probably are outliers
  - An outlier is an observation that deviates so much from other observations as to arouse suspicions that it was generated from a different mechanism
- What is an outlier in a time series?
  - Traditional definitions are based on wavelet transforms but are not adequate for categorical values such as ours
- Markov chains give us an approach
What is a Markov chain?

- A stochastic process is a finite-state, k-th order Markov chain if it has:
  - A finite number of states
  - The Markovian property (probability of next state depends only on \( k \) most recent states)
  - Stationary transition probabilities (i.e. they do not change with time)

- Probabilities, in a first-order chain with \( s \) states can be expressed as a matrix with \( s \) rows and cols
  - In \( n \)-th order, with a matrix with \( s^n \) rows and cols

- Chain is irreducible if all states are reachable
  - Transient, recurrent and absorbing states

- They comprise other models
  - N-grams are simplified \( n \)-th order markov chains
An example of Markov chain

Markov Chain Models

Transition probabilities
\[
\begin{align*}
\Pr(x_i = a \mid x_{i-1} = g) &= 0.16 \\
\Pr(x_i = c \mid x_{i-1} = g) &= 0.34 \\
\Pr(x_i = g \mid x_{i-1} = g) &= 0.38 \\
\Pr(x_i = t \mid x_{i-1} = g) &= 0.12
\end{align*}
\]
Training a Markov chain

- We can compute the likelihood of a sequence in a model with a simple conditional probability.
- We can build the model which fits a given sequence or set of sequences by calculating the maximum likelihood model, the one which gives the various observations the maximum probability.
- Can be done through simple calculations (problem of null probabilities), or through Bayesian ones.
- Comparison of probability of sequences of different length is difficult (can use the logarithm or other tricks to smooth).
Which Markov chain does this fit?

- **Simple answer:** you compute the likelihood
- **If you need to compare multiple models, this is more complex**
  - You need to take into account the prior probability, or probability of the model, since:
    \[ P(M|O) = P(O|M) \frac{P(M)}{P(O)} \]
  - \( P(O) \) is fixed and cancels out, but you usually don't know \( P(M) \): depending on the choice, you can have varying results
- **S. Zanero, “Behavioral Intrusion Detection” explains the trick**
A Hidden Markov Model is one where we do not really see the state, but a set of symbols which can be generated with some probability from each state.

How likely is a given sequence in a HMM?
- the Forward algorithm

What is the most probable “path” for generating a given sequence?
- the Viterbi algorithm

How can we learn the HMM parameters given a set of sequences?
- the Forward-Backward (Baum-Welch) algorithm
SyscallAnomaly: analyzing the variables

- SysCall Anomaly, proposed by Vigna et al.
  - Each syscall separately evaluated on 4 separated models
    - (maximum) string length
    - Character distribution
    - Structural inference
    - Token search

- Each model is theoretically interesting, but exhibits flaws in real-world situations
  - Structural inference
    - Realized as a markov model with no probabilities...
    - Too sensitive!
  - Token search
    - No “search”, really: you must predefine what is a token
    - Again, no probabilities
Our proposal

- We evolved the models
  - Structural inference: abolished (halving false positives...)
  - Implemented a model for filesystem paths (depth – structural similarities)
  - Token Search: probabilistic model
    - UID/GID specialization, considering three categories: superuser, system id, regular id

- Now, we wanted to add
  - Correlation among the arguments of a single syscall
    - Hierarchical clustering algorithm to create classes of use
  - Correlation among system calls over time
    - First order Markov model (a Markov chain)
What is clustering?

- *Clustering* is the grouping of pattern vectors into sets that maximize the intra-cluster similarity, while minimizing the inter-cluster similarity.

- Here “pattern vectors” are the values of various models.

- We used a hierarchical agglomerative algorithm.
  - Pick up the two most similar items.
  - Group them.
  - Compute distance from the new group to other groups.
  - Repeat.

- What is similarity?
  - Two patterns are similar if they are “close.”
  - We had to define similarity for each model type.
    - e.g. is `/usr/local/lib` similar to `/usr/lib`? And to `/usr/local/doc`?
Results of clustering

- The clustering process aggregates similar uses of a same system call

  - E.g.: let us take the open syscalls in fdformat:
    
    ```
    /usr/lib/libvolmgt.so.1, -rwxr-xr-x
    /usr/lib/libintl.so.1, -rwxr-xr-x
    /usr/lib/libc.so.1, -rwxr-xr-x
    /usr/lib/libadm.so.1, -rwxr-xr-x
    /usr/lib/libw.so.1, -rwxr-xr-x
    /usr/lib/libdll.so.1, -rwxr-xr-x
    /usr/lib/libelf.so.1, -rwxr-xr-x
    /usr/platform/sun4u/lib/libc_psr.so.1, -rwxr-xr-x
    /devices/pseudo/mm@0:zero, crw-rw-rw-
    /devices/pseudo/vol@0:volctl, crw-rw-rw-
    /usr/lib/locale/iso_8859_1/LC_CTYPE/ctype,-r-xr-xr-x
    ```

- Each of the clusters is a separate type of syscall (e.g. “open_1”, “open_2”, “open_3”)
A matter of sequence

- We can now build a Markov chain which uses as states the clusters of syscalls, as opposed to the syscalls per se.
- *We can train the model easily on normal program executions.*
- At runtime we will have three “outlier indicators”:
  - The likelihood of the sequence so far
  - The likelihood of this syscall in this position
  - The “similarity” of this syscall arguments to the best-matching cluster
- The first is an indicator of likely deviation of program course, the others are punctual indicators of an anomaly.
Conclusions & Future Work

- **Conclusions:**
  - IDS are going to be needed as a complementary defense paradigm (detection & reaction vs. prevention)
  - In order to detect unknown attacks, we need better anomaly detection systems
  - We can successfully use unsupervised learning for anomaly detection in a host-based environment using
    - System call sequence
    - System call arguments

- **Future developments:**
  - Integrating this to become an Intrusion Prevention system, maybe using CORE FORCE?
  - More extensive real-world evaluation on the go
  - Integration with our network-based system
Thank you!

Any question?

I would greatly appreciate your feedback!

Stefano Zanero
zanero@elet.polimi.it
www.elet.polimi.it/upload/zanero/eng