

Process Query Systems for Network Security Monitoring *

Vincent Berk and Naomi Fox

Thayer School of Engineering, Dartmouth College, Hanover, NH 03755 USA

FirstName.LastName@Dartmouth.EDU

ABSTRACT

In this paper we present the architecture of our network security monitoring infrastructure based on a Process Query System (PQS). PQS offers a new and powerful way of efficiently processing data streams, based on process descriptions that are submitted as queries. In this case the data streams are familiar network sensors, such as Snort, Netfilter, and Tripwire. The process queries describe the dynamics of network attacks and failures, such as worms, multistage attacks, and router failures. Using PQS the task of monitoring enterprise class networks is simplified, offering a priority-based GUI to the security administrator that clearly outlines events that require immediate attention. The PQS-Net system is deployed on an unsecured production network; the system has successfully detected many diverse attacks and failures.

1. INTRODUCTION

The PQS-Net system is a valuable proof-of-concept that the PQS framework is ideally suitable to do high-level network security monitoring on large, enterprise-class networks. Enterprise-class networks are split up into many different subnets, often geographically diverse in nature. These subnets are then connected through WAN connections, allowing the network to spread worldwide. Enterprise networks may contain as many as two million connected hosts, many of which will be mission critical servers. Any form of central security monitoring will be very difficult. Standard intrusion detection sensors such as *Snort*¹ have a tendency to generate large quantities of alerts, most of which are false positives, therefore often obscuring the important information. Most notably, they only collect data for the subnet that they are connected to. Needless to say, security administrators need ways to collect this, and other data from the many networks at different sites, and be able to sift through the data in a meaningful manner.

In addition to IDS logs (such as *Snort*¹ or *Dragon*²), security administrators will often also consult services such as *DShield*³ to get an idea of which attacks are most popular at the current time. Combined with *server logs*, *system logs*, and *Tripwire logs*⁴ collected from the many sites, security administrators will have a flood of data to review. It is not difficult to see how this data can grow up to several gigabytes per day that need to be analyzed, most of which will be false positives.

The PQS-Net system collects data from many different sensors and implements PQS models based on methodical procedures commonly followed by security administrators. These models evaluate the data coming from the many sensors and the conclusions are published through a web interface. The system eliminates most of the false positives, focusing on detecting the processes that are indicative of malicious hacker behavior, insider threat behavior, and autonomic attacks, such as worms and viruses. To improve the power of the PQS-Net system, models were also written that specifically search for network failures, such as faulty routers, or misconfigured servers. This way the system is able to disambiguate between malicious hacker attacks, network failures, or benign behavior.

To test the PQS models and verify the results a test network was constructed that is complete with several subnets, network servers, and actual user hosts. Since the setup of this network is important to understanding the PQS-Net system, the next section will focus on this basic infrastructure. The sections afterward will focus on the sensors that were deployed, and the models that were written. Finally, this paper will close with a discussion of our experiences.

*Supported under ARDA P2INGS Award No. F30602-03-C-0248. Points of view in this document are those of the author(s) and do not necessarily represent the official position of ARDA.

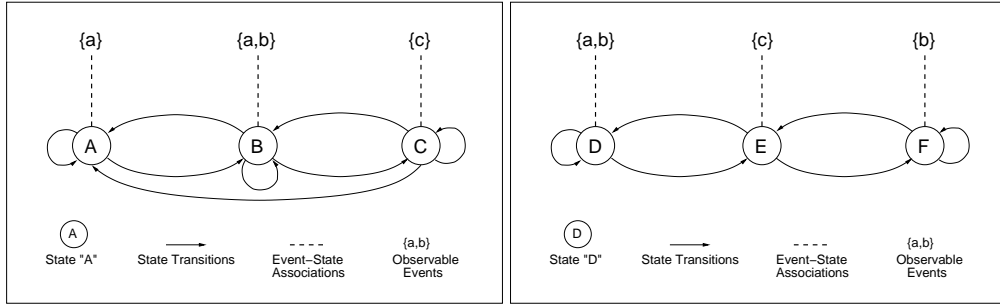


Figure 1. a: A Simple Process Model, \mathcal{M}_1 **b:** Another Process Model, \mathcal{M}_2

2. PROCESS QUERY SYSTEMS

Process Query Systems are a new paradigm in which user queries are expressed as process descriptions. This allows a PQS to solve large and complex information retrieval problems in dynamic, continually changing environments where sensor input is often unreliable. The system can take input from arbitrary sensors and then forms hypotheses regarding the observed environment, based on the process queries given by the user. Figure 1-a shows a simple example of such a model. Model \mathcal{M}_1 represents a state machine $S_1 = (Q_1, \Sigma_1, \delta_1)$, where the set of states $Q_1 = \{A, B, C\}$, the set of observable events $\Sigma_1 = \{a, b, c\}$, and the set of possible associations $\delta_1 : Q_1 \times \Sigma_1$ consists of $\delta_1 = \{\{A, a\}, \{B, a\}, \{B, b\}, \{C, c\}\}$. Notice how this process is able to produce observed event a in both state A and state B . A possible event sequence recognized by this model would be:

$$e_1 = a, e_2 = a, e_3 = b, e_4 = c, e_5 = b$$

which we will write as $e_{1:5} = aabcb$ for convenience. Possible state sequences that match this sequence of observed events could be $AABCB$, or $ABBCB$, both of which are equally likely given \mathcal{M}_1 . A rule-based model would need a lot of rules to identify this process, based on all the possible event sequences. Below is a set of all the rules necessary for detecting single transitions:

$$\begin{aligned} AA &\rightarrow \{aa\} \\ AB &\rightarrow \{aa\}, \{ab\} \\ BB &\rightarrow \{aa\}, \{ab\}, \{ba\}, \{bb\} \\ BA &\rightarrow \{aa\}, \{ba\} \\ BC &\rightarrow \{ac\}, \{bc\} \\ CC &\rightarrow \{cc\} \\ CB &\rightarrow \{ca\}, \{cb\} \\ CA &\rightarrow \{ca\} \end{aligned}$$

Needless to say the list of possible event sequences for double transitions is massive (eg. transitions AAA , AAB , ABB , ABC , ... etc.) and only gets bigger when we consider dealing with the possibility of missed observations. Rules would then have to include sequences that have two transitions for a single observed event, albeit with a lower priority accounting for the fact that we expect only few observations to go missing.

Now lets introduce a second model \mathcal{M}_2 in Figure 1-b defined by state machine $S_2 = (Q_2, \Sigma_2, \delta_2)$. Consider that both processes are regularly and concurrently occurring in the observed environment. Note that the process states are labelled differently, i.e. $Q_1 \cap Q_2 = \emptyset$, although both processes produce the same set of observable events, i.e. $\Sigma_1 \cap \Sigma_2 \equiv \Sigma_1 \equiv \Sigma_2$. Now consider the following sequence of events:

$$e_{1:24} = abaacabbacabaccbacabbc$$

where each observation may have been produced by instances of model \mathcal{M}_1 , model \mathcal{M}_2 , or be totally unrelated. It must be noted that any modelled process may very well be occurring multiple times concurrently. A Process Query System

uses multiple hypotheses, multiple model techniques to disambiguate observed events and associate them with a “best fit” description of which processes are occurring and in what state they are. In comparison, a rule-based system would get impossibly complex for the above situation. Additional problems with rule-based approaches arise when probabilities are assigned to the transitions, and/or the event productions.

Consider the following example. Assume the first model describes the dynamics of a propeller plane, and the second model describes the dynamics of a fighter jet, both observed by radar. It may very well be possible that there are several propeller planes and a group of jet fighters in the same airspace, all within radar range. The PQS will use the radar data as input observations together with the two models to disambiguate which radar observations were triggered by which aircraft by associating radar observations using the models. Subsequently, the hypothesis will be that there are several instances of the model \mathcal{M}_1 (the propeller plane) and a group of instances of model \mathcal{M}_2 in the observed environment. Since the environment is dynamic the *top hypothesis* will be changing continuously as planes move in and out of radar range.

A PQS is a very general and flexible core that can be applied to many different fields. The only things that change between different applications of a PQS are the format of the incoming observation stream(s) and the submitted model(s). Compare this with a traditional Database Management System (DBMS); inventory tracking systems, accounting, customer databases, etc. are all different applications that, at the core, are all based on the same DBMS. Likewise we have implemented vehicle tracking systems, server farm monitoring applications, enterprise network security trackers, and covert timing channel detectors using the same PQS software core by simply supplying a different observation stream and a different set of models. Internally a PQS has four major components that are linked in the following order:

1. Incoming observation handling and sensor subscription.
2. Multiple hypothesis generation.
3. Hypothesis evaluation by the models.
4. Selection, pruning, and publication.

To conclude, the big benefits of a PQS are that its superior scalability and applicability. The application programmer simply connects the input event streams and then focuses on writing process models. Models can be constructed as state machines (above), formal language descriptions, Hidden Markov Models, kinematic descriptions, or a set of rules. All these different model types are first compiled down to the fundamental PQML (*Process Query Modeling Language*) representation and then submitted to the PQS. The PQS is now ready to track processes occurring in a dynamic environment and continuously present *the best possible explanation of the observed events* to the user.

3. ARCHITECTURE

Figure 2 shows a toplevel schematic view of the network that was constructed for the PQS-Net project. Although the network is small compared to an actual enterprise class network, it features a wide range of systems and servers that are typical of larger organizations. Looking at the models it will become evident that the size of the test network is irrelevant because of the scalability of the PQS framework. Behind the firewall there are 64 addresses available (.192/26), which are split up between a *Workstations* network and a *Demilitarized Zone* server network, both (/27s). The uplink connects directly to the Internet, and all addresses are globally routable[†]. Both subnets contain more hosts than are displayed in the image, however, they are immaterial to the discussion below and have therefore been omitted. The workstations network features several Windows XP clients, Linux 2.4 and 2.6 systems, several Solaris 9 workstations, and a Solaris 9 server to which multiple thin-clients are connected. All these systems are used daily by students, and so the traffic on this network is typical for a normally operating organization where users browse the web, print documents, and download files during business hours.

The DMZ features several servers implemented on several different operating systems. The *DNS* server resolves names for all the hosts connected to the network, *Mail* handles inbound and outbound email traffic for the students, and *WWW*

[†]The first three octets of these addresses have been obscured by replacing them with nonroutable addresses.

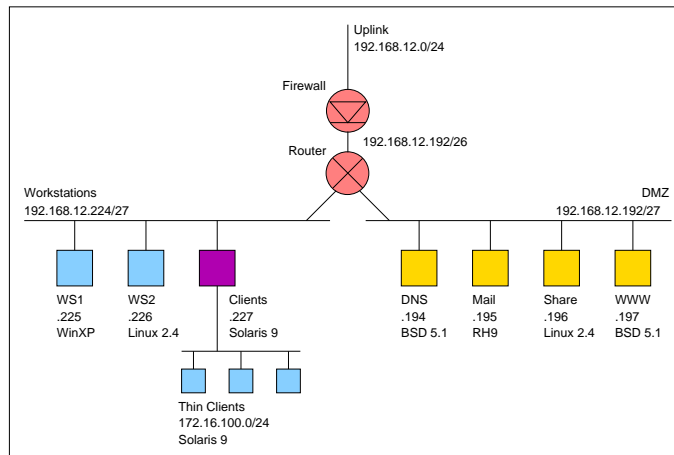


Figure 2. Schematic overview of the most important systems in the PQS-Net network.

serves up the results produced by PQS-Net system, as well as personal webpages. Finally, the *Share* server holds home-directories, as well as general fileshare areas that can be used to store and back-up data. The firewall is configured to allow nearly all inbound traffic in, and only protects the core PQS infrastructure, which is not considered part of the test network, and therefore not shown in the image. Although this would be uncommon for a normal organization, it does allow a lot of scans and attacks, that naturally come from the Internet, through. These scans form a nice baseline to test the PQS-Net system on actual malicious traffic. In addition to the expected malicious traffic coming from the Internet, the PQS-Net network is also attacked from dedicated *Attack Systems* that are located both inside the network, as well as outside on the Internet. These attacks include network scans, vulnerability scans, and directed attacks, all intended on testing the PQS-Net detection capabilities.

The graphical user interface for viewing PQS results is featured in 3. This snapshot was taken after a particularly aggressive noisy worm simulation was run.

3.1. Sensors

In order to apply a *Process Query System* to any area the programmer must do two things: connect the sensors and implement the process models. This section describes the first step: installing a variety of sensors in the test network and connecting them to the PQS system. Generally, the sensors can be split up into three categories: *Global* (Internet wide data), *Network* (network specific data), and *Host-based* (data pertaining to one specific host). In the subsections below all connected sensors are outlined, together with the data that they provide to the PQS system. Figure 4 shows where the sensors are located in the test network. The placement of the sensors is roughly similar to the common data collection points that security administrators use.

- There are two *Global* sensors deployed in PQS-Net: the Dartmouth ICMP Bcc: system (DIB:s)⁵ and the Border Gateway Protocol (BGP)⁶ sensor.⁷ The DIB:s sensor receives ICMP type-3 destination unreachable messages from widely-dispersed routers on the Internet, and aggregates these messages to create observations on scanning activity. It was originally designed to detect the spread of new worms on the Internet. The BGP sensor reports on the stability of the Internet, with the Global Reachability Index (GRI) and the Global Instability Index (GII). The GRI is the percentage of the assigned network address ranges that are actually reachable, normally around 98%. The GII is a number which reflects how much the reachability tables on each of the border gateways changes, and how frequently. In a more stable Internet, no border gateways would ever become overloaded or breakdown, but in the real-world Internet, this is expected to happen and can be an important clue to new malware on the loose.
- The *Network* sensors deployed include Snort, Spade, and Netfilter/IPTables sensors. Snort¹ is a traditional signature-matching IDS. Its sensor reports on such attributes as source and destination IPs and ports of packets that have triggered certain rules. Spade⁸ is an anomaly detector that generates alerts on statistical outliers based on specific

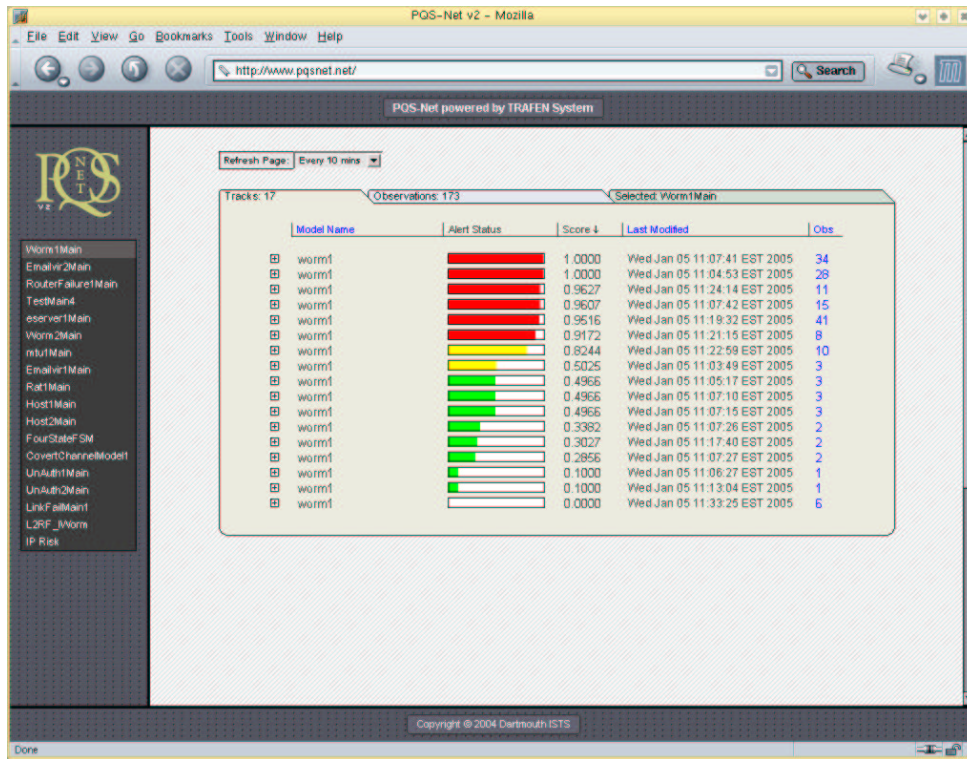


Figure 3. Web-based graphical user interface for viewing hypotheses.

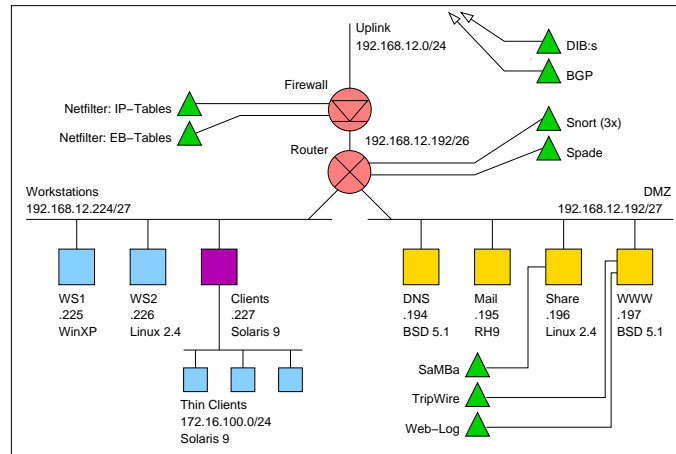


Figure 4. Overview of the location of sensors in the test network.

packet attributes such as source and destination IPs and ports, and packet length. The Spade sensor generates observations that contain pertinent information (IPs, ports, timestamp) and an anomaly score. The Netfilter⁹ sensor creates observation similar to Snort, but on triggered firewall rules.

- Finally, representing *Host-based* sensors on PQS-Net are the Samba and Tripwire sensors. The Samba¹⁰ sensor reports on fileshare accesses. The same approach can be applied to other fileshare servers, such as NFS. The power of the Samba sensor can be increased by monitoring certain "honeytoken"¹¹ files placed on the fileshare server that should not be accessed by anyone. If a honeytoken is accessed, it indicates someone performing reconnaissance on

the fileshare server. The Tripwire⁴ sensor generates observations when important system files have been modified.

Currently, the PQS subscribes to sensor observations by connecting to a raw socket on the sensor, which then transmits tightly packed packets. Security could be greatly improved by encrypting the channels. Portability of sensors can be improved by using the Intrusion Detection Message Exchange Format (IDMEF),¹² but this would increase the amount of traffic required to transmit each observation. Also, IDMEF in particular is good as a guideline, but there is still no standard way to implement IDMEF. Because there is no real convention for transmitting sensor data, the PQS core is designed to be very flexible to easily handle new sensor data formats.

3.2. Models

The second part of applying a Process Query System to a specific task is the models for evaluating the fidelity that certain scenarios are occurring in the network. In the construction of the PQS-Net system many models were written, this section outlines a few of these models in detail, separated into attacks, network failures, and higher-level models. Recall that models are asked by the PQS to “score” sequences of observed events. If the sequence of events is evidence of the process that the model is describing, then the score should be high, otherwise it should be low. In the discussion of models in the following sections, the scores are normalized between zero and one, unless otherwise specified. Some of the earlier models were extremely complex and needed a lot of tuning. We quickly learned, however, that the power of PQS encourages simpler, more general models: either two or three process states, or just several decision rules.

Most of the sensors discussed above (Snort, Spade, Samba, ...) have a tendency to produce a large flood of observations. It is the task of a good model to sift through this and find relevant combinations of events that is evidence of the modeled process. Additionally, some observations have more “weight” than others. For instance, consider a stream of Snort scan alerts towards the webserver versus a single firewall observation of unexpected outbound traffic from the webserver. The stream of Snort scan alerts is not necessarily alarming, considering that any Internet-connected DMZ will experience regular scanning, however, unexpected outbound traffic from the webserver is almost always related to a compromise. Therefore, based on the firewall observation, the Snort scan alerts become more important and can be used as evidence to support that the webserver may have been compromised.

Terminology:

$S_{\mathcal{M}}(T_n)$ New score of track T at time n given by model \mathcal{M} .
 $S_{\mathcal{M}}(T_{n-1})$ Old score previously assigned to this track by model \mathcal{M} .

3.2.1. Attacks

Often the observed events can be explained in various different ways. For instance, DIB:s observations may indicate that the network is being scanned, or that a server has failed. Similarly, Tripwire observations may indicate that a system was compromised and important files were changed, however, it can also indicate that a system software update was performed recently. Because of the multiple hypothesis capability of the PQS it is possible to keep both hypotheses around in the above cases. As more observations arrive, the models may favor one hypothesis over the other. In this way we load multiple models into the PQS and allow them to compete for observations, while forming a conclusion. Models were built for such scenarios as worm propagation, email virus propagation, multi-step intrusion, and unauthorized insider document access. This section describes two successful attack detection models. This is a small subset of the models there are currently successfully operational.

Noisy Worm Propagation

sensors: DIB:s
 Snort
 BGP

This model looks for worms that show aggressive scanning patterns. The most typical and early evidence for worms comes from the DIB:s sensor. During the initial stages of worm propagation an exponential growth in infected systems is expected. Since DIB:s reports on hosts that display massive parallel scan behavior it is expected that the number of DIB:s

alerts for the destination port of the worm will also increase exponentially. A track consisting entirely of same destination port DIB:s alerts gets scored as follows:

$$S_{\mathcal{M}}(T_n) = \begin{cases} \frac{1}{2} \times (1 + S_{\mathcal{M}}(T_{n-1})) & : \Delta t < 10 \\ \frac{1}{2} \times \left(\frac{\Delta t - 10}{890} + S_{\mathcal{M}}(T_{n-1}) \right) & : 10 \leq \Delta t \leq 900 \\ \frac{1}{2} \times S_{\mathcal{M}}(T_{n-1}) & : \Delta t > 900 \\ 0 & : \text{when destination ports mismatch} \end{cases}$$

where Δt stands for the time between two consecutive DIB:s alerts in the track. The above function alone is a very powerful worm detection model when submitted to a PQS. To improve the model and increase the amount of information returned, the following functions may be added:

$$S_{\mathcal{M}}(T_n) = \begin{cases} \frac{1}{6} \times (1 + S_{\mathcal{M}}(T_{n-1})) & : \text{Snort port and protocol match} \\ \frac{1}{6} \times (1 + S_{\mathcal{M}}(T_{n-1})) & : \text{BGP GII} > 12.0 \\ 0 & : \text{otherwise} \end{cases}$$

however, all tracks must be started, and primarily be constructed from DIB:s observations. The rule for Snort observations encourages tracks when there are matching Snort rules for the vulnerability that the worm is exploiting. The BGP observation may be expected later into the propagation of the worm where the network load on the Internet is causing routing instabilities (a GRI > 12.0). Finally, the score of any new, single-observation DIB:s tracks is set to 0.1 for this model.

Unauthorized Insider Document Access

sensors: Samba
Spade

This model focuses primarily on users accessing honeytokens on the central fileshare. It monitors all accesses to the fileshare using the Samba sensor, and checks it against a list of known honeytokens. To improve performance this model also checks for fileshare packets that have a high anomaly score, for example, when the fileshare is accessed at an unusual time of day. Initial scores are 0.9 if a honeytoken was accessed, 0.6 for any packet with a very high anomaly score going to the fileserver, 0.4 for any fileshare access violation, and 0.0 otherwise. Tracks scores change according to this function:

$$S_{\mathcal{M}}(T_n) = \begin{cases} \frac{1}{2} \times (1.0 + S_{\mathcal{M}}(T_{n-1})) & : \text{honeytoken access} \\ \frac{1}{2} \times (0.5 + S_{\mathcal{M}}(T_{n-1})) & : \text{general access violation} \\ \frac{1}{2} \times (1.0 + S_{\mathcal{M}}(T_{n-1})) & : \text{anomalous traffic to fileshare} \\ S_{\mathcal{M}}(T_{n-1}) & : \text{otherwise} \end{cases}$$

where in all cases either the SMB username or the source IP address of the request must be the same for all observations in the track.

3.2.2. Failures

This category of models was specifically designed to catch cases where the observed events are more likely associated with a network device failure than an attack. However, it is not always clear which of the two may be going on, for instance, consider a DDOS attack on a large network. The border router of this large network is likely to fail under such a heavy load, and therefore it is more likely that the system reports router failure than DDOS attack. (Either way there is only one thing the administrator can do: restart the router.)

Router or Link Failure

sensors: DIB:s
Snort

Technically the models for router failure and link failure are separated, however, it seems very difficult to separate the two conditions based on the observations we have available. Router failure happens when a router, somewhere upstream, fails to route packets. In some cases traffic will be dynamically re-routed, although usually network connectivity simply stops. Link failures are all conditions where the physical link fails, this can be a cut in the wire, or a switch or hub failure. Both conditions are very similar and often require a lot more than network based sensors to diagnose the condition accurately.

The router failure model depends on DIB:s observations with a local source address, giving a score of 0.6 to all new tracks that fit this condition. More DIB:s observations with a local source address quickly let the total score grow to 1 according to $S_{\mathcal{M}}(T_n) = \frac{1}{2} \times (1.0 + S_{\mathcal{M}}(T_{n-1}))$. The drawback of this model is that it requires a local network installation of the DIB:s system, and at least one functioning router. If the local network router fails, then this model will be unable to detect the condition for any host on that local network. It will, however, work well for a larger enterprise class network that has multiple smaller (say class-C) networks, all monitored by DIB:s enabled routers. If any of these routers for a class-C subnet fails, other hosts in the larger network will be unable to reach this small subnet. The ICMP-T3 traffic coming from the other routers going to the DIB:s system will then quickly generate alerts that will trigger this model effectively.

The link failure model relies on Snort observations and starts out with the premise that only the local subnet is still reachable. Therefore all other network based and global sensors are unreachable. The Snort observations are assumed to be originating from a local sensor, and are only considered for this model when their source address is within the local network. Only two rules are important to this model: Snort rule 472 (ICMP redirect host), and Snort rule 394 (ICMP-T3 destination unreachable). Both may (but don't necessarily have to) be generated by a reachable router closeby. When the source IP on any of these two specific Snort observations is from the local network then the observation may grow the score in the track. The score starts out at 0.2 and grows to 0.9 when the destination IP is the same for all observations, otherwise to 0.7, according to the following formula:

$$S_{\mathcal{M}}(T_n) = \begin{cases} \frac{1}{2} \times (0.9 + S_{\mathcal{M}}(T_{n-1})) & : \text{ iff destination IP is the same} \\ \frac{1}{2} \times (0.7 + S_{\mathcal{M}}(T_{n-1})) & : \text{ otherwise} \end{cases}$$

where the scores never really grow to 1.0, mostly because this model can never be really certain of what is going on. It may merely be the “best” explanation of what is happening, due to the lack of a better hypothesis.

3.3. Level-2 Models

Level-2 models refers to a category of models that have both sensor observations, as well as the output of other models available for evaluation. This means that the output of the models discussed so far may be used as input to these level-2 models. The output of the above models includes: track score, model name, source and destination IP (where appropriate), and source and destination port and protocol (where appropriate). These higher level models are often very interesting because they deal with complex behavior. For instance, the two models discussed in this section are designed to track the higher level behavior of single hosts (host-state), and to track the progression of a multistage attack through a network (attack-state).

Consider Figure 5 where a level-2 PQS tracker is added to the architecture. Outputs from the level-1 tracker are used as input observations to the level-2 tracker, together with sensor observations. In practice there is often no particular need for a second PQS core, instead it is possible to loop the output from one core back into it's own input, effectively combining both level-1 and level-2 models in the same tracking engine.

Based on level-1 output, or direct sensor observations the status of an individual host can be tracked. Consider the model in Figure 6-a where a host is in one of seven states, three of which are considered “trusted”, and the others are “hostile”. Once a host is in a hostile state it cannot move back to a trusted state automatically. Notice how the “compromised” state is essentially a placeholder, waiting for further information on what activity the host will be undertaking, while hostile. The depiction of this model omits the specific sensor observations that are associated with each state, or state transition. However, assigning these observations is intuitive; examples of “recon” are the results of the *low&slow* scan model, a DIB:s observation, or a Snort scan alert. This model also shows that there is not necessarily a structured set of state transitions

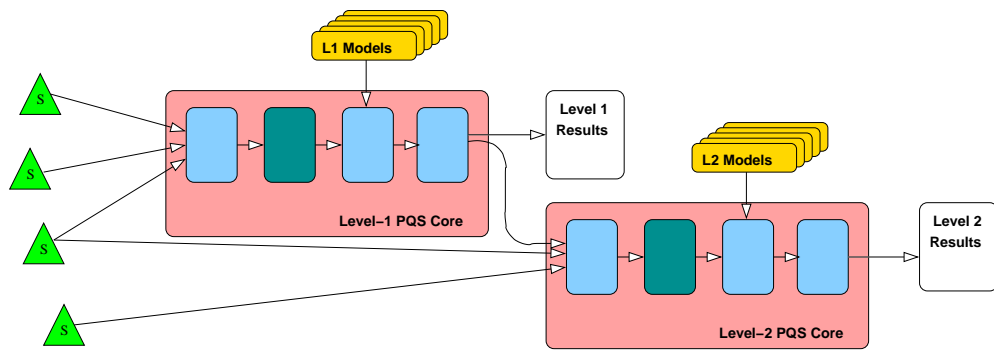


Figure 5. Toplevel view of a level-1 and a level-2 PQS tracker.

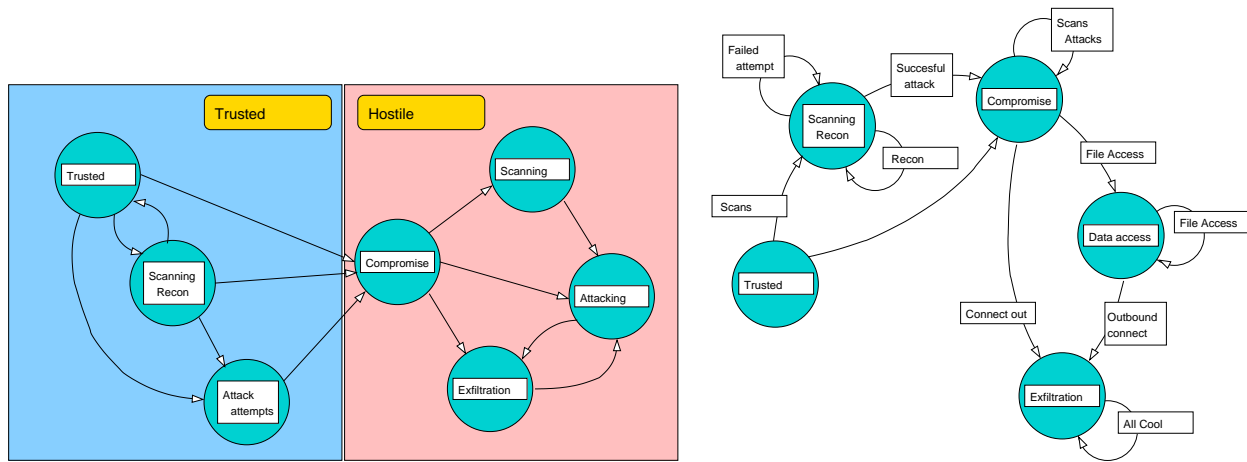


Figure 6. a: State machine for an individual host.

b: Level-2 model tracking steps in a multistage attack.

that transfer a host from a fully trusted state into a compromised state; depending on received observations it is very well possible that a host is considered compromised without going through the recon or attack attempt states.

Figure 6-b shows another level-2 model that tracks the progression of a complicated, multistage attack. The challenge here is to correlate the activities of various hosts together and identify which steps are related. Consider the following two level-1 observation sequences concerning hosts **A**, **B**, and **C**:

- | | | |
|----------|------------------------|----------|
| A | scans | B |
| A | attacks (failed) | B |
| A | scans | C |
| A | attacks (successfully) | C |

and:

- | | | |
|----------|------------------------|----------|
| A | scans | B |
| C | attacks (failed) | B |
| A | scans | B |
| C | attacks (successfully) | B |

In the first case it is clear that host **A** is being very hostile, and there will be little doubt that all four steps are related and part of the same multistage attack. It is unlikely that **A**'s attack on **B** is unrelated to its attack on **C**, although both hypotheses will be generated by the PQS. It is expected that the first hypothesis will ultimately dominate because all steps originated from the same aggressor.

The second sequence, however, is not as clear cut. Although host **A** is doing the scanning, it is host **C** that actually performs the attacks. The PQS will generate two hypotheses (one assuming that all steps are related, the other assuming that the scanning is independent from the attacks), each of which has a separate set of reasons why it is a likely hypothesis.

For example, it could be argued that the target is **B**, and that the attacker is using multiple systems to attempt a compromise. On the other hand **A** and **C** are separate machines acting independently. If, later on, evidence comes in that **A** is actually scanning the entire subnet, whereas **C** has only attempted to attack one system, then it becomes more likely that their actions are, in fact, independent, although the other scenario cannot be ruled out. In the end, however, the human analyst will probably care more about the fact that **B** was compromised than whether **A** and **C** are one and the same aggressor.

4. RESULTS

Process Query System technology is still in its infancy, however its use as a powerful and flexible data processing system has been proven by various applications. The goal of the PQS-Net system was to build a system that can assist intrusion analysts, not replace them. Data is conveniently collected from all over the network through the PQS sensor interface and evaluated by one or multiple models. So far the models have reflected relatively straightforward relationships between events, however, these relationships are the same as the associations made by analysts. Based on these associations the analyst will often search for more information or evidence by collecting logs, watching traffic, or browsing websites on global Internet trends. Collecting data is probably the single most time consuming task for any analyst. The PQS-Net system effectively eliminates the need for the human to retrieve the data, look at it, and decide what other information to look for.

Since all of the basic network security sensors are present (IDS monitoring, firewall logs, ...) the PQS-Net system doesn't miss any intrusions per se. Therefore, performance should not be measured as a factor of the false positives and false negatives, a more accurate measure is to evaluate if the associations made by the models give an accurate top-down list of important relationships. Meaning, is the system properly telling the analyst what is "important", and what is "unimportant" in the flow of observations from the sensors. The problem is that "importance" is a very subjective term because it depends not only on network data, but also on the specific purpose of the systems, and the value placed on them by the organization. For example, a bank will place a very high value on the system that holds the account information. When the bank gets attacked, the analyst will like to see an intrusion of the account information computer at the top of the list, preferably not cluttered with alerts about scanned teller workstations. This is what the PQS-Net system does.

In our test network, we have been able to successfully detect simulated attacks such as worm propagation, router failure, unauthorized insider document access, and multistage attacks from external hosts. This means that when we ran attack scripts while the PQS system was running and collecting observations from deployed sensors, the models representing these attacks came up with the highest scores and resulted in an alert level that passed a predefined threshold. Using a standardized dataset produced by Skaion Corporation,¹³ the system also produced pleasing results. The system was able to disambiguate the successful multistage attack from the failed attack attempts.

5. CONCLUSION

A Process Query System is a powerful paradigm for using process descriptions to detect processes occurring in a dynamic, continually changing environment where sensor input is often unreliable. The system can take input from arbitrary sensors and then form hypotheses regarding the observed environment, based on the process queries given by the user. Through experiences deploying PQS-Net, it has been proven that building the necessary infrastructure for a PQS to monitor a computer network is quick and painless. Sensor are easily installed and robust models can be quickly written.

The future focus of PQS development branches in a few directions:

- *Model building simplification.* At this point most models are written directly in Java code, PQML or XML. It is quite straightforward to envision graphical user interfaces that will make model building significantly easier. Code is often reused between different models, so using graphical "building blocks" to make new models would be natural for a PQS.

- *Model allotment to trackers.* Studies need to be completed on whether it is best for models to share a tracker and vie for observations, or for each model to have its own tracker. It seems that some models (low&slow for instance) require their own tracker because of specific tuning parameters that make the model work. However, in other cases the system performs better when the models compete for observations in the same tracker, thus forming many meaningful hypotheses.
- *Hypothesis space visualization.* Currently, a PQS user has no way to compare how similar two hypotheses are than to look back and forth at each track. A better way to present the hypotheses, showing how much information is shared and between them and what is different, would really take advantage of the underlying multiple hypothesis and structure and give the analyst the most complete view of the monitored environment.
- *When to draw hard conclusions.* Although the PQS-Net system was designed to assist analysts by showing event correlations in order of importance, it can be argued that in some cases an autonomic response would be desirable. In order for that to work the system must be able to draw hard conclusions and act on them. Hard conclusions are usually drawn when the score of a track goes above a certain threshold. If there are multiple competing models all claiming a different conclusion, The tuning of this parameter will depend highly on the specific circumstances.

Process Query Systems can be used, and have in fact been used to solve many problems in very diverse domains. So far, PQS has been applied to the fields of battle tank tracking, chemical plume detect, fish tracking, server farm monitoring, and, of course, network security. The potential for this technology is quite promising, and as PQS is applied to more domains, the knowledge gained can be shared between the different domains and applications to create new interesting and effective methods for detection, and even prediction, of processes in an event space.

REFERENCES

1. "Snort - the open source network intrusion detection system," project website, 2005. Available at <http://www.snort.org>.
2. "Enterasys dragon 7.0 network sensor," company website, Enterasys, 2005. Available at <http://www.enterasys.com/products/ids/DSNSS7>.
3. "Dshield - distributed intrusion detection system, the internet's early warning system and internet," project website, 2005. Available at <http://www.dshield.org>.
4. "Tripwire, inc. - tripwire is the leading provider of change auditing solution," company website, Tripwire, Inc., 2005. Available at <http://www.tripwire.com>.
5. R. S. Gray and V. H. Berk, "Rapid detection of worms using icmp-t3 analysis," *Proceedings of SPIE* **5403**, pp. 89–101, April 2004.
6. K. Lougheed and Y. Rekhter, "A border gateway protocol (bgp)," *Internet Engineering Task Force Network Working Group RFC 1105*, June 1989. Available at <http://www.ietf.org/rfc/rfc1105.txt?number=1105>.
7. D. McGrath, "Passive internet health monitoring with bgp," presentation, NANOG, October 2003. Available at <http://www.nanog.org/mtg-0310/mcgrath.html>.
8. S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security* **10**, pp. 105–136, 2002.
9. "The netfilter/iptables project homepage," project website, Netfilter, 2005. Available at <http://www.netfilter.org>.
10. "Samba - opening windows to a wider world," project website, 2005. Available at <http://www.samba.org>.
11. L. Spitzner, "Honeytokens: The other honeypot," *SecurityFocus InFocus*, July 2003. Available at <http://www.securityfocus.com/infocus/1713>.
12. H. Debar, D. Curry, and B. Feinstein, "The intrusion detection message exchange format," *Internet Engineering Task Force Intrusion Detection Exchange Internet-Draft*, January 27, 2005. Available at <http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-14.txt>.
13. "Skaion corporation - specializing in computer network security," company website, Skaion Corporation, 2005. Available at <http://www.skaion.com>.