Detection of Complex Cyber Attacks [*]

Ian Gregorio- de Souza and Vincent H. Berk and Annarita Giani and George Bakos and Marion Bates and George Cybenko and Doug Madory

Thayer School of Engineering, Dartmouth College, Hanover, NH 03755 USA
firstname.lastname@Dartmouth.EDU

ABSTRACT

One significant drawback to currently available security products is their inabilty to correlate diverse sensor input. For instance, by only using network intrusion detection data, a root kit installed through a weak username-password combination may go unnoticed. Similarly, an administrator may never make the link between deteriorating response times from the database server and an attacker exfiltrating trusted data, if these facts aren't presented together.

Current Security Information Management Systems (SIMS) can collect and represent diverse data but lack sufficient correlation algorithms. By using a Process Query System, we were able to quickly bring together data flowing from many sources, including NIDS, HIDS, server logs, CPU load and memory usage, etc. We constructed PQS models that describe dynamic behavior of complicated attacks and failures, allowing us to detect and differentiate simultaneous sophisticated attacks on a target network.

In this paper, we discuss the benefits of implementing such a multistage cyber attack detection system using PQS. We focus on how data from multiple sources can be combined and used to detect and track comprehensive network security events that go unnoticed using conventional tools.

Keywords: Cyber Attacks, Process Query Systems

1. INTRODUCTION

Network security analysts are overwhelmed daily with large volumes of information regarding attacks on hosts residing on a given network. This information, usually in the form of alerts from intrusion detection systems such as Snort[1] and Dragon,[2] contains many false positives and reports of benign activity triggered by Internet "background noise" [†]. However, evidence of serious successful attacks launched against hosts on the network exist in this same set of security alerts. The analyst is therefore required to manually sift through all received alerts in order to identify truly malicious activity. Moreover, cyber attacks that involve more than a single step are reported as separate events by conventional intrusion detection systems.

We define *complex cyber attacks* as a sequence of malicious activity directed at hosts on a network, performed in multiple stages, where evidence of the attack steps are distributed across various data sources and occur in an unknown order. These types of attacks cannot be detected using solely one source of network security data, and usually require the analyst to go through multiple system logs, firewall logs and other sources of system audit information in order to reveal what occurred.

Conventional network security applications fail to provide the security analyst with a central repository and correlated view of attack data as observed by different systems, processes, etc. In situations where the analyst oversees an enterprise-sized network, critical information can be easily overlooked due to massive overload of data from deployed security applications. Security Information Management Systems such as NetForensics,[3] OSSIM[4] and Arcsight[5] address the data aggregation and retrieval but not the correlation challenge, without which many attacks are indiscernable from background noise and legitimate activity. Process Query Systems (PQS) have proved to be useful in the fusion and correlation of malicious network activity records and can be tuned to recognize the sequence of security state transitions that occur during complex cyber attacks.

The next section of this paper gives a brief overview of the PQS and how it is applied to the problem of detecting complex cyber attacks. The section after that will discuss specific complex cyber attack examples and the PQS sensors and models used to detect them. Finally, we present results of our experiments and the benefits of using PQS.

[*]Supported under DOJ Award No. 2000-DT-CX-K001 and DTO Award No. F30602-03-C-0248. Points of view in this document are those of the author(s) and do not necessarily represent the offi cial position of DOJ and DTO.

[†](http://www.switch.ch/security/services/IBN/)

المنارة للاستشارات

www.manaraa.com

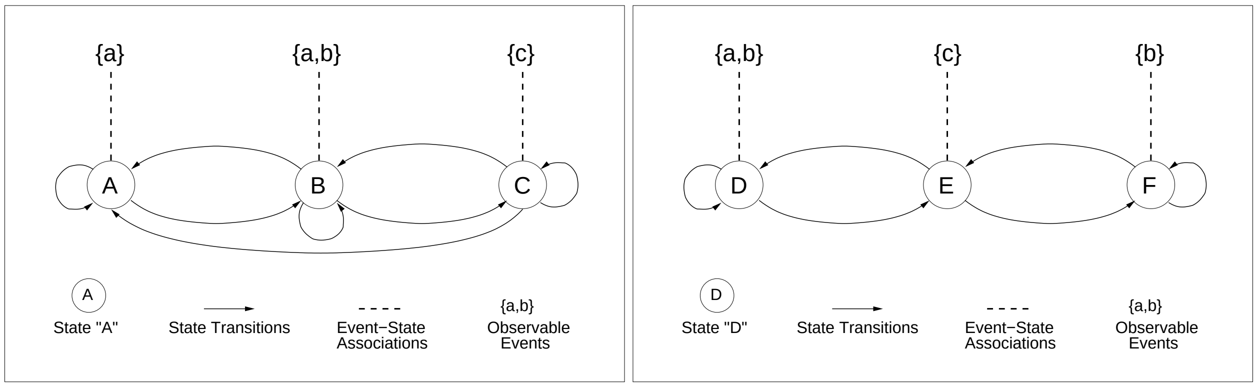

Figure 1. Two process models, $\mathcal{M}_1$ an $\mathcal{M}_2$.

2. BACKGROUND

2.1. 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 form hypotheses regarding the observed environment, based on the process queries given by the user. Figure 1 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 observableevents $\Sigma_1 = \{a, b, c\}$, 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:

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

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 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:

المنارة للاستشارات
www.manaraa.com

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

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 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 simple 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.

The big benefits of a PQS are 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, and submitted to the PQS. The PQS is now ready to track processes occurring in a dynamic environment and continuously present *the best possible explaination of the observed events* to the user.

2.2. PQSNet

PQSNet is a system that applies the PQS concept and architecture to the network security domain.[6] PQSNet utilizes sensors and subscribers built around common network security applications such as IDS (Snort, Dragon), firewalls, file system integrity monitors (Tripwire,[7] Samhain[8]), and system and/or web server logs. Alerts and logs from these security applications are parsed into observations which are then fed into a PQS and handled by the process models. Observations encompass the essential information provided to security analysts by the various applications. Observation information therefore allows expert knowledge to be applied via process models.

PQSNet models that are implemented as finite state machines represent the different stages and steps in an attack as finite states. This allows the model to use observables such as Snort alerts, for instance, to trigger transitions between states. For example, in the very general PQSNet model shown in Figure 2, the states of the model are:

- Start ($\mathcal{S}_0$) - nothing malicious or suspicious detected
- Recon ($\mathcal{S}_1$) - reconnaissance activity detected (e.g. portscan)

www.manaraa.com

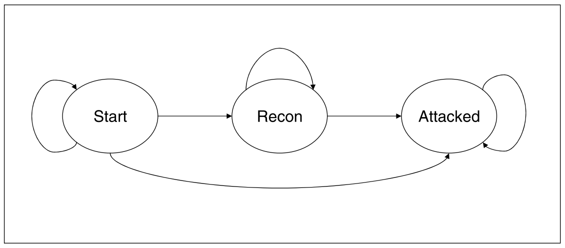

Figure 2. A General PQSNet Model

- Attacked ($\mathcal{S}_2$) - host has been attacked (e.g. buffer overflow exploit)

A host can transition from $\mathcal{S}_0$ to $\mathcal{S}_1$ when an IDS alert, indicating that some form of reconnaissance has occurred, is observed. A transition from $\mathcal{S}_1$ to $\mathcal{S}_2$ can occur when alerts of incoming attacks or attack responses are observed. The observations that trigger these transitions are provided from the various PQSNet sensors.

Since PQS provides support for the tiering of models, PQSNet models can easily abstract simpler malicious activity to lower level models. This promotes writing simpler higher level models for detecting complex attacks. For instance, if the model described in Figure 2 detects exploit attempts on hosts, the output of this model can be fed into a higher level PQS. This input can be teamed with output from another lower level model that detects suspicious data upload, for instance, to find a host that may be leaking valuable data as a result of a compromise. Such tiering of models will be shown in the models used in this paper for detecting complex attacks. Models discussed in this paper will be tiered, finite state machine models.

2.2.1. PQSNet Sensors

PQSNet has a growing list of sensors based on common host and network security applications. The sensors relevant to work presented in this paper are:

- The *Snort* and *Dragon sensors* reports alerts from Snort and Dragon respectively. These applications are both signature-matching Intrusion Detection Systems. The sensor's observations give information on source and destination IPs, ports, and which rule or signature was violated.

- The *Web log sensor* reports information from Apache, IIS and SSL error logs. Observations from this sensor contain information about suspicious URLs that have succeeded or failed, as well as web server errors.

- The *NetFlow sensor* reports statistics on network flow and is based on netflowd.[9] The goal of the NetFlow sensor is to classify a communication between two or more machines in terms of volume of data transfer, length of the transmission and number of packets involved. The sensor monitors network packets and aggregates them in groups of the same source IP, destination IP, source port, and destination port. As packets in the same flow arrive, the number of bytes of new packets are recorded. The flow is considered closed if there is no transmission for more than a specified time threshold. At that point, the number of packets and total duration of the flow is computed. These quantities are then classified as shown in Table 1. These classifications are crucial to building models that reflect how data is moving

- The *Samhain sensor* reports alerts from the Samhain host file integrity monitor. Samhain frequently checks the critical system files on a host for additions, modifications and deletions. Any changes are immediately logged locally or reported to a remote log server. Samhain sensor observations include timestamps of changes, filenames, violation type, and changes in the system kernel. Observations from this sensor allows correlation of network and host based activity.

www.manaraa.com

Volume	Packets	Duration	Balance	Percentage
0: 0 1: 1-128b Tiny 2: 128b-1Kb Small 3: 1Kb-100Kb Med. 4: $>$ 100Kb large	1: one packet 2: two packets 3: 3-9 4: 10-99 5: 100-999 6: $>$ 1000	0: $<$ 1 s Zero 1: 1-10 s Very Small 2: 10-100 s Small 3: 100-1000 s Medium 4: 1000-10000 s Large 5: 10000-100000 s Very Large 6: $>$ 100000 s Very Very Large	1: IN 2: OUT 3: PAR	N

Table 1. NetFlow sensor classifi cations

3. EXPERIMENTAL SETUP

The network testbed used for experiments presented in this paper was entirely software-emulated. This virtual network resides on one physical host running Linux; the router and the other network hosts are all virtual machines on that same physical host (via User Mode Linux(UML),[10] or Qemu[11]). A significant advantage of this configuration lies in the flexibility of adding single hosts or entire subnets. The addition of a virtual host to the testbed simply requires duplicating the base OS file, modifying configuration files on the host and launching the virtual machine as an application. The new virtual host is easily networked to the existing testbed by initiating a new network interface via *ifconfig*. Software network interfaces can be created, enabled, or disabled on the fly. The size and configuration of the network is therefore not limited by physical space or the number of open network ports. The virtual routers can also support a large number of network interfaces.

Although the physical host has a limited amount of memory and disk space, UML allows virtual machines to share a single "backing" file system mounted via NFS, so the only disk space required is for a COW[‡] file. The COW file solely stores modifications to the backing file and thererfore tends to stay small, even on a heavily modified UML. Memory and CPU utilization depends on what applications run within the virtual hosts. Since the virutal machines run as user-space applications on the host, they can be "nice'd" and otherwise throttled by the physical host.

Network topology for this setup is likewise software-emulated, but is transparent for purposes of traffic sensing. An emulated hub/switch (in hub mode, here) handles the bridging of traffic from the host's physical network interface to the VMs' emulated "tap" interfaces. Bridges (provided by the bridge-utils package, and supported by modern kernels) are the metaphorical "cabling" of the network. As with a real phyiscal network, sniffing can be done at any point and on any interface within the virtual network.

In the setup shown in Figure 3, the UML acting as router and firewall has the only real, routable IP address on the network. It performs masquerading and/or NAT to allow the Internet to reach the DMZ webserver. Internal hosts and DMZ hosts can all communicate with each other via the router's internal interfaces, and internal hosts can reach the Internet via the router. The Internet cannot reach internal hosts directly. This realistically represents a typical network setup.

4. ATTACK SCENARIOS

4.1. Remote Shell Through Web Application Exploit

Insecure code in web applications often creates an entry point for an attacker in an otherwise secure network. Since public web servers are accessible from the Internet, they are a top choice for initial exploit attempts, attacks and stepping stones to other attacks. In this scenario, an attacker discovers a vulnerable PHP[§] web application by performing a web vulnerability scan on a web server. The attacker proceeds to exploit the poorly written PHP code using a custom PHP *passthru()* string. The exploit shovels a remote shell from the web server back to the attacker on an unusual port. Once the attacker gets access to the host, he escalates his privileges on the host, and downloads and installs a rootkit. The attacker then uses the compromised host to scan the local network for other possible targets.

[‡]Copy On Write
[§]Hypertext Preprocessor

www.manaraa.com

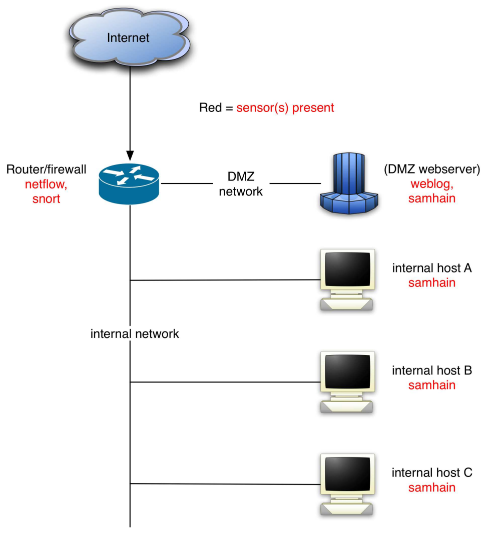

Figure 3. Experimental network setup

Sensors

- Snort
- Samhain
- Web Log

Models

Detecting the sequence of steps in the above described attack with a finite state machine process model would require a highly complex state machine. In order for the PQSNet model to detect more than just this ordered sequence, a generic model that handles all possible state transitions must be implemented. In such a complex attack, the high number of possible states creates a highly intricate web of potential state transitions. Figure 4 shows the state diagram of a general model that can be used to detect this complex attack. The *System Tampering* state in this diagram represents a number of state which have been abstracted for clarity. This single state encompasses various states that a specific host may be in after being comprised. These include, but are not limited to:

- Privilege escalation
- Creation of rogue accounts
- Installation of rootkits
- Replacing system binaries and/or libraries with trojanned or backdoor versions, etc

Adding each of these independent states individually makes the model more complex to implement.

The tiering support in PQS allows us to write simpler models for detecting this cyber attack sequence and similar complex attacks. In this case, we created three simple lower tier models (Figure 5a-b) to detect (a) network based reconnaissance, attack attempts or compromises, (b) data uploads and downloads and (c) malicious system events. The model for detecting data transfers (Figure 5b) uses observations from the NetFlow sensor to determine an upload or download based largely on the balance and parity fields. The NetFlow observations make it possible to differentiate between large data flows which possibly represent the transfer of files, and smaller data flows that may result from merely browsing to a web site. The system tampering model (Figure 5c) uses Samhain observations from various hosts to determine if any

www.manaraa.com

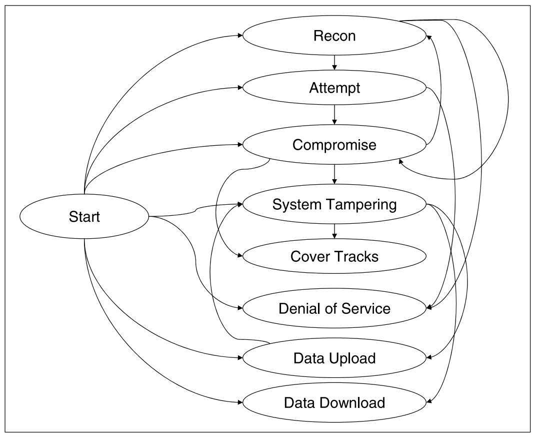

Figure 4. Generic Complex Model for Remote Shell Scenario

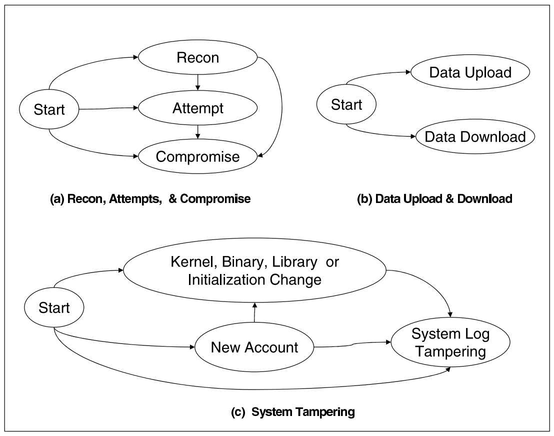

Figure 5. Simple Tier 1 Models for Remote Shell Scenario

tampering has occurred. Our second tier model, Figure 6a, then becomes more simplified than the model presented earlier in Figure 4.

With a lower level of abstraction present, it is easier to correlate sequences that represent an attack and disambiguate them from harmless events. For example, data upload and/or download may be normally benign traffic (dashed transition path). However, if data uploads or downloads occur after or between compromises and suspicious system changes, then these data transfers are definitely worth scrutinizing by the security analyst.

4.2. Phishing with Data Exfiltration

In a typical social engineering based phishing attack, a user receives a seemingly legitimate message from an established enterprise which directs the user to visit a web page and enter some personal information such as credit card or bank account numbers. Since the site looks legitimate, the user naively inserts the data without realizing that the page is a bogus one set up with the purpose of stealing sensitive information. The phishing attack discussed in this paper is a variant of the typical sequence:

1. An user (S) browses the web and visits a web page. and creates an account using the same username and password he uses for his account at work.
2. The attacker (A), who has control over the said website, retrieves username and password and uses that information to access (S)'s machine

المنارة للاستشارات
www.manaraa.com

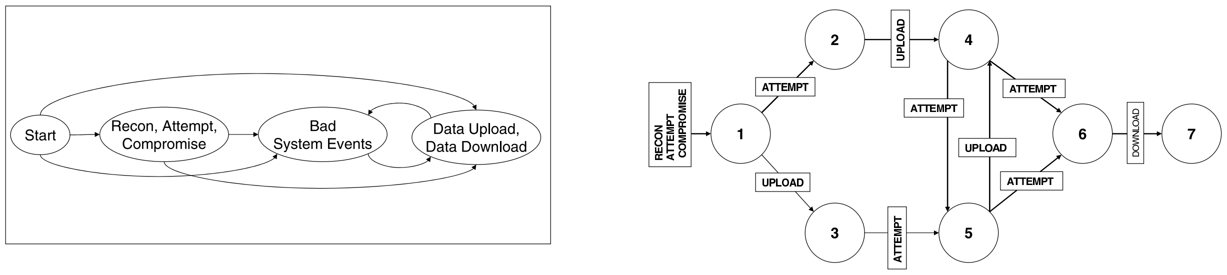

Figure 6. **(a)** Simple Tier 2 model for remote shell scenario. **(b)** Strict Phishing Model

3. The attacker uploads some code and launches an exploit against the target (T), which is on the same network as (S)
4. The attacker then downloads various data files from the target (T).

Sensors

- Snort
- Dragon
- NetFlow
- Web Log

Models

The detection of a complete phishing attack from security application data is difficult due to the social engineering aspect of the sequence. This first step is rather arbitrary and can only be detected if the web site the victim is lured to triggers a security alert. In our scenario, the victim is lured to a pornography site, which is easily picked up by the Dragon IDS. Assuming that the first step is detectable as a reconnaissance, attack attempt or compromise from an IDS sensor, we implemented models to detect the phishing attack sequence. These models are second tier models fed by output from the first tier models described in Figure 5a and Figure 5b.

We started with a very strict model that detects this particular attack sequence. This first model, shown in Figure 6b, follows the exact attack sequence as described. Obviously, any modification or rearrangement of the attack steps would be missed by this model. Hence, a similar but not identical attack would not be detected. We relaxed the transition requirements between states so that less specific alerts could cause state trasnsitions. As we make the state transition requirements less strict, our models are able detect a larger variety of attacks.

The initial strict model starts with alerts coming from Snort or any other IDS that detects the fact that the user (S) visited a "bad" web site. An IDS alert (Policy violation) indicates the attacker (A) accessing the user's host from an untrusted host outside the local network. This is followed by a data download from the attack host and a compromise of the target host (T). A data upload finally shows massive data transfer from the target (T) to the attacker. The relaxed version of the model, Figure 7, shows how the transition triggers between the states of the model have been generalized to accommodate other occurrences that could constitute part of a phishing attack sequence.

5. RESULTS

We ran tests of our models using recorded datasets of the chosen attacks. Using these datasets allowed us to evaluate and tweak our PQSNet models repeatedly without executing the multistage attacks each time. Our models were able to succes-fully identify and correlate the different security alerts, events or observations of the attacks presented and simultaneously emphasize their importance over other processes present in everyday Internet background noise.

www.manaraa.com

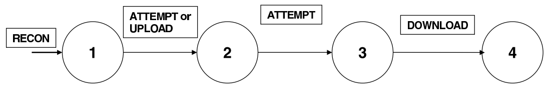

Figure 7. Relaxed Phishing Model

In both attack scenarios presented, we found it was relatively easier to write complex process models using the tiered model approach. The modularity attained with this approach provides a high level of convenience when modifying how observations and/or events deemed to be associated should be matched. Also, dependable process models which are known to be effective can be re-used when developing newer models.

In general, we observed that more false positives were generated when state transition requirements were relaxed. However, pruning and hypothesis scoring algorithms inherent in PQS allow really malicious activity to rise in the list of processes that need further attention. This is a major improvement over the large amount of false positives that are generated daily by conventional IDS such as Snort or Dragon.

Finally, the detection on complex attacks, as defined in this paper, depends heavily on the ability to report diverse, yet important, information such as malicious or suspicious network activity as well as unplanned changes on hosts filesystems. For instance, without the Samhain sensor, information about a rootkit installed on a recently exploited host will be missed and the IDS alert for the exploit may be dismissed as a false alarm. Consequently, having automated information about a circulating phishing email will reduce false positives generated by our phishing models. PQS (PQSNet) gives us the luxury of easily subscribing to data from various sources and using the valuable aggregation and correlation ability to identify or predict potential cyber attacks.

REFERENCES

1. "Snort - open source network intrusion detection system." Project website, 2006. Available at http://www.snort.org/.
2. "Enterasys Dragon 7 Network Intrusion Detection." Company website, 2006. See website at http://www.enterasys.com/products/ids/DSNSS7/.
3. "NetForensics security information management." Company website, 2006. http://www.netforensics.com/.
4. "OSSIM — Open Source Security Information Management." Project website, 2006. http://www.ossim.net.
5. "ArcSight security information managment." Company website, 2006. See website at http://www.arcsight.com/.
6. V. H. Berk and N. Fox, "Process query systems for network security monitoring," in *Proceedings of SPIE Vol. 5778 Defense and Security Symposium, Orlando, Florida*, April 2005.
7. "Arcsight security information managment." Company/Project website, 2006. See website at http://www.tripwire.com Open source available at:http://sourceforge.net/projects/tripwire/.
8. "Samhain — open source file integrity monitor and intrusion detection system." Project website, 2006. See website at http://la-samhna.de/samhain/.
9. "Cisco IOS netflow." Project Website, 2006. See website at http://www.cisco.com/en/US/products/.
10. "User mode linux (uml)." Project website, 2006. Available at http://user-mode-linux.sourceforge.net/index.html.
11. "Qemu - open source processor emulator." Project website, 2006. Available at http://fabrice.bellard.free.fr/qemu/.
12. V. Berk, W. Chung, V. Crespi, G. Cybenko, R. Gray, D. Hernando, G. Jiang, H. Li, and Y. Sheng, "Process query systems for surveillance and awareness," in *Proceedings of the 7th World Multifconference on Systems, Cybernetics and Informatics (SCI 2003)*, (Orlando, Florida), July 2003.
13. C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, Kluwer Academic, 1999. ISBN 0-7923-8609-4.
14. G. Cybenko, V. H. Berk, V. Crespi, R. S. Gray, and G. Jiang, "An overview of process query systems," in *Proceedings of SPIE Vol. 5403 Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense III , Orlando, Florida*, April 2004.

www.manaraa.com