“Secure Thyself”: Securing Individual Peers in Collaborative Peer-to-Peer Environments

Ankur Gupta1 and Lalit K. Awasthi2

1 Computer Science & Engineering, Model Institute of Engineering & Technology, Jammu, Jammu & Kashmir, India
2 Computer Science & Engineering, National Institute of Technology, Hamirpur, Himachal Pradesh, India

Abstract - P2P networks and the computations that they enable hold great potential in designing the next generation of very-large scale distributed applications. However, the P2P phenomenon has largely left untouched large organizations and business which have stringent security requirements and are uncomfortable with the anonymity, lack of centralized control and censorship, which are the norm in P2P systems. Hence, there is an urgent need to address the security concerns in deploying P2P systems which can leverage the under-utilized resources in millions of organizations across the world. This paper proposes a novel containment-based security model for cycle-stealing P2P applications, based on the Secure Linux (SE Linux) Operating System, which alleviates existing security concerns, allowing peers to host untrusted or even hostile applications. The model is based on securing individual peers rather than computing trust/reputation metrics for the thousands of peers in a typical P2P network, which involves significant computational overheads.

Keywords: Peer-to-Peer Computing, Peer-to-Peer Security, Containment-Based Security Model, Secure Linux

1 Introduction

P2P networks are self-organizing in nature, extremely fault-tolerant and designed to provide acceptable connectivity and performance in the face of a highly-transient population of nodes. Their desirable characteristics have resulted in several P2P-based applications being proposed/built for a wide-variety of domains such as distributed computing (SETI@HOME [1], Condor [2], Avaki [3]), file/information sharing (kazaa [4], facebook [5]), collaboration (Groove [6]) and middleware/platforms (JXTA [7]) enabling further development and deployment of P2P applications. However, business organizations have largely been left untouched by the P2P phenomenon, citing lack of security as the biggest concern for staying away from potentially beneficial collaborations, sharing of information and other possible financial benefits that cross-organizational P2P interactions could enable. The lack of centralized control, censorship and anonymity offered by P2P systems, coupled with the potential for malicious activity, could compromise the confidential data of an organization, besides putting its compute resources at risk. However, the potential benefits of enabling cross-organization P2P interactions [8], necessitates that the research community attempt to mitigate the security threats posed by the P2P system model.

Several schemes for ensuring the security of peers participating in P2P networks have been proposed. Most of the research has focused on security of content-sharing P2P applications/networks, while P2P applications which involving remote task execution have not received the desired attention. A majority of these schemes rely on trust and reputation [10-12] based mechanisms in attempting to isolate untrusted/malicious peers from the rest of the peers. However, such schemes require computing trust values for each peer interaction and then communicating the computed values to all peers in the network. Clearly, such schemes are not fool-proof. Besides forged identities, groups of malicious peers can act together to circumvent the trust management schemes. Moreover, the computational overheads required for such schemes render them ineffective when trying to ensure bullet-proof security for participating peers, a major requirement for all organizations. Other schemes [13-16] incorporating elements of access control, authentication/authorization and encryption have also been proposed, which rely on introducing centralized security servers in an otherwise decentralized P2P environment. This introduces a single-point-of-failure in the P2P network, besides constituting a performance bottleneck. Also, in resource-exchanging P2P environments, once the remote code is resident on the host-peer these approaches are silent on containing the potential damage that can occur. What is really needed is to secure individual peers while dealing with untrusted peers and reduce or mitigate the impact of hosting
malicious remote code, especially for cycle-stealing P2P applications, which rely on harvesting idle compute cycles.

This research paper proposes a simple containment-based security model for cycle-stealing P2P applications, which creates a virtual sandbox for remote code, thereby mitigating the security threat posed by any malicious peers. The model uses the fine-grained privileges and Rule-Based Access Control mechanisms provided by Secure Linux [9] and deploys a custom-built Remote Application Monitor (RAM) to create a secure computing environment for cycle-stealing P2P applications, without having to pay the overheads of managing trust and reputation values. This work is a part of a bigger effort to enable secure P2P interactions across organizations and some of the issues addressed are due to specific requirements of the architecture proposed in [8] by the authors.

The rest of the paper is organized as follows: section 2 provides an overview of various flavors of Secure Operating Systems, which form a basis for the proposed solution. Section 3 discusses the system model of the proposed solution, while section 4 provides the implementation details on Secure Linux. Section 5 analyses the effectiveness of the proposed solution and finally section 6 concludes the paper.

2 Secure Operating Systems

Several Operating Systems [17-19] now provide inbuilt security mechanisms based on virtualization [20], fine-grained privileges, role-based access control and concepts of containment and compartmentalization. The basic idea is to allow the system to host a wide-variety of users/applications, requiring access to specific resources, while insulating them from the other users/applications on the system, thereby creating a “sandbox” effect. Moreover, virtualization has been employed by many hardware vendors to create logical partitions on the same computer system. Some of these solutions even allow for isolation of physical faults in these partitions, thereby ensuring that the other partitions continue to operate normally. However, these features are primarily designed for specific applications installed on the system and not for the varied applications that could be uploaded to a remote peer for execution. Moreover, these security features are typically available only on the high-end servers, not the typical systems you would see in a P2P network. A security framework needs to be more generalized and work with all possible hardware configurations. Hence, we decided to use SE Linux for its easy availability (open-source). However, our implementation can be applied to any of the operating systems discussed above.

3 System Model

For P2P systems, we need to provide security at two levels; security of shared content, with multiple remote peers accessing the data and and when multiple remote peers/applications make use of the idle CPU cycles and disk space on the host peer. Any security model would also need to be flexible and customizable, since different organizations would have different security policies. We also introduce a Remote Application Monitoring (RAM) module, which shall monitor the application for any suspicious activity like increased/heavy CPU utilization, increased outbound/inbound network traffic, increased disk space usage, increased memory utilization etc. If the application exceeds its pre-defined quotas (quotas could be negotiated based on the local peers characteristics and resource availability or via an organization-wide policy), the RAM can promptly terminate the application. By using the access control features provided by SE Linux in conjunction with our custom application monitor, the local peer can be secured and malicious remote code can be contained without causing serious damage within the P2P network. The proposed containment based solution shall be implemented on individual peers hosting remote applications and shall work as follows:

a. Create an area reserved for remote P2P applications.

b. Create a security policy for Secure Linux, which shall control access to the resources of the local peer. This policy is pre-configured.

c. Allow remote applications/code to reside in the reserved area and use idle compute and system resources, as per defined security policy. SE Linux security features shall ensure that access to critical resources is denied to any malicious code.

d. Configure the Remote Application Monitor (RAM) with pre-defined thresholds or quotas. These quotas provide the upper limit on peer resource usage and shall be based on the peer’s local requirements and resource availability.

e. Monitor the resource usage of the remote application and take corrective action, if usage crosses thresholds, say CPU utilization crosses a particular threshold, or the application stays active after its allotted time is over.

f. Monitor the application for any potentially malicious activity; say a spike in the incoming/outbound network traffic etc. and terminate the application if needed.

Fig. 1 provides an overview of the system model for the proposed solution.
4 Implementation

The implementation of the system has two components; the security policy for individual peers to securely host remote applications and a custom-built Remote Application Monitor (RAM). We have used the SE Linux features to define a security policy suited to meet the security requirements for cycle-stealing P2P applications. Although the system is still under development, some aspects of the system have been realized and tested. Due to the lack of user support for SE Linux, writing security policies is a hit-and-trial approach at best. Hence, a lot more testing shall be needed to fine-tune the system and make it work in a live environment.

4.1 The SE Linux Security Policy

This section provides details of the policy designed to secure individual peers, enabling them to host untrusted remote applications. To host the remote application a user “remote_user” is created on the host machine. The security policies are defined in the context of the “remote_user” and the remote application “remote_app”, which shall make use of the resources of the host peer. Figs. 2. and 3. provide indicative security policies on Secure Linux. The syntax and semantics for writing security policies for SE Linux are available at [9] along with sample policies. It is assumed that the relevant P2P middleware shall place the remote application in the remote_user’s home directory and assign the ownership of the remote application to “remote_user”. Moreover, for RAM, the executable name is assumed to be fixed (remote_app) at present. It is possible to generate configuration policies at runtime at a later stage to cater to different applications.
type remote_user_t, domain, userdomain, unpriv_userdomain;

# Grant permissions within the domain.
general_domain_access(remote_user_t);

defines the domain remote_user_t. Allows the remote_user_t domain to send signals to processes running in unpriviledged domains such as remote_user_t. Allows remote_user_t to run ps and see processes in the unprivileged user domains. Also allow remote_user_t to access the attributes of any terminal device.

full_user_role(remote_user)
allow remote_user_t unpriv_userdomain:process signal_perms;
can_ps(remote_user_t, unpriv_userdomain)
allow remote_user_t { ttyfile ptyfile tty_device_t }:chr_file getattr;

don’t allow access to root directory (ls, cd etc.)
deny unpriv_userdomain sysadm_home_dir_t:dir { getattr search };  

Don’t grant the privilege of setting/changing the gid or owner
deny remote_user_t self:capability { setgid chown fowner }
audit remote_user_t self:capability { sys_nice fsetid };

# Type for home directory.

file_type_auto_trans(privhome, remote_user_home_dir_t, remote_user_home_t)

Set the default home directory for new created files. This prevents access to the /root dir

tmp_domain(remote_user, `user_tmpfile')

Give permissions to create, access and remove files in home directory.

file_type_auto_trans(remote_user_t, remote_user_home_dir_t, remote_user_home_t)
allow remote_user_t remote_user_home_t:dir_file_class_set { relabelfrom relabelto };  

# Allow remote_user to bind to sockets in /tmp directory
allow remote_user_t remote_user_tmp_t:unix_stream_socket name_bind;
# Security Policy for a remote executable

# Types for the files created by remote app
type remote_app_file_t, file_type;

# Allow remote_app to create files and directories of type remote_app_file_t
create_dir_file(remote_app_t, remote_app_file_t)

# Allow user domains to read files and directories these types
r_dir_file(userdomain, { remote_app_file_t })

can_exec(remote_app_t, { remote_app_exec_t bin_t })

# Allow network access
can_network(remote_app_t)

# Allow Socket Operations
can_unix_send( { remote_app_t sysadm_t }, { remote_app_t sysadm_t } )
allow remote_app_t self:unix_dgram_socket create_socket_perms;
can_unix_send(remote_app_t)

can_unix_connect(innd_t, self)

# Allow PIPE operations and binding to a socket
allow remote_app_t self:fifo_file rw_file_perms;
allow remote_app_t innd_port_t:tcp_socket name_bind;
allow remote_app_t innd_var_run_t:sock_file create_file_perms;

deny remote_app_t self:capability { dac_override kill setgid setuid };
deny remote_app_t self:process setsched;

deny remote_app_t { bin_t sbin_t }:dir search;
deny remote_app_t { bin_t sbin_t }:file { getattr read ioctl };
deny remote_app_t { proc_t etc_runtime_t }:file { getattr read };
deny remote_app_t urandom_device_t:chr_file read;

4.2 The Remote Application Monitor

The RAM is a collection of scripts which uses the psmon [21] freeware tool to enforce certain quotas like max application instances, time to live for the remote application, max CPU utilization, max memory utilization etc. psmon has the ability to slay the processes which exceed the configured limits. Other quotas like max network connections are implemented using scripts based on the output of the netstat command on linux. Monitoring is done on % CPU utilization, number of instances of remote application, disk usage, number of network connections, network usage and time-based usage (CPU time). The RAM allows the user to specify quotas on these parameters and if the application exceeds its quota, it is terminated by RAM.

Although the custom security policy on SE Linux would deny access to critical system resources, it does not prevent the remote application from initiating malicious activity, like Distributed-Denial-of-Service (DDoS) attacks on other peers in the P2P network by pumping in invalid packets/queries. Also, SE Linux is unable to specify the quantum of resource usage, a requirement for collaborative P2P environments, where resources are exchanged frequently. Hence, we need the RAM to strictly enforce resource usage limits for the peer hosting the remote application, besides monitoring the number of network connections established by the remote application and the traffic generated. It is planned to extend
the RAM to monitor several other application specific parameters in future.

5 Analysis

Fig. 4 provides details of the responses by the proposed security framework to some potentially malicious activity by a sample application.

Fig. 4 Possible malicious behavior by remote hosted application and the responses of the CBSM

<table>
<thead>
<tr>
<th>S.No</th>
<th>Malicious Activity</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Establishing Multiple Network Connections and generating traffic above threshold</td>
<td>RAM Terminates the Application</td>
</tr>
<tr>
<td>2.</td>
<td>Attempt to access files in a directory outside reserved area</td>
<td>Access declined by SE Linux</td>
</tr>
<tr>
<td>3.</td>
<td>Attempt to fork repeatedly resulting in multiple instances of the remote application</td>
<td>RAM terminates application after it crosses num_instances quota for remote app.</td>
</tr>
<tr>
<td>4.</td>
<td>Attempt to generate network traffic and propagate Distributed-Denial-of-Service Attacks</td>
<td>RAM terminates the application after monitoring network traffic generated by it.</td>
</tr>
<tr>
<td>5.</td>
<td>Application performs compute-intensive tasks leading to high CPU utilization</td>
<td>RAM terminates the application after it crosses CPU utilization quota.</td>
</tr>
<tr>
<td>6.</td>
<td>Attempt to execute system calls with admin privileges</td>
<td>SE Linux Security Policy denies access to the application.</td>
</tr>
</tbody>
</table>

The overheads introduced by the Remote Application Monitor (RAM) are incurred only when the remote application is being run. Although, SE Linux does introduce some overheads in making access decisions at run-time, several optimizations like caching access decisions for future use help to keep the application performance impact to negligible levels. A complete analysis of performance overheads of SE Linux can be found in [22]. It is evident that by combining the rule-based access control mechanisms provided by SE Linux along with the custom Remote Application Monitor (RAM), the security of the peer hosting remote work is enhanced significantly. By providing more flexible security configuration settings, it is possible to safely enable peer interactions across organizations and host untrusted remote applications.

6 Conclusions and Future Work

Our research shows that by using standard off-the-shelf components combined with a custom application monitor, it is possible to build a secure environment for cycle-stealing P2P applications. Very little attention of the research community has been focused on this important area. Hence, this research is significant, since it allows individual peers to be secured, allowing them to host any remote application, without putting its resources/confidential information at risk. Moreover, the proposed scheme does not require any expensive trust/reputation management schemes to ensure security. It is hoped that this research can form the basis for more comprehensive security mechanisms, promoting the adoption/deployment of P2P applications across organizations, helping realize the enormous potential of cross-organization P2P interactions.

Future work shall involve identifying as many scenarios indicating malicious activity by the remote applications and tweaking the system to provide fool-proof security. The detailed analysis of the effectiveness of the proposed model and its performance overheads shall be available shortly when the system is tested extensively using several flavors of malicious code and compared with existing security approaches.

7 References

[1] SETI@HOME: Website [http://setiathome.berkeley.edu](http://setiathome.berkeley.edu)


[19] VMWare Website: http://www.vmware.com


[21] Process Table Monitoring Script; psmon website: http://www.psmon.com