Incident identification through outlier analysis

Joshua Lewis
Incident Identification Through Long Tail Analysis

GIAC (GCUX) Gold Certification

Author: Joshua Lewis, Joshua.d.Lewis@gmail.com
Advisor: Rob VandenBrink
Accepted: February 1st, 2016

Template Version September 2014

Abstract

Incident identification is one of the most challenging steps in the incident response process. If an incident cannot be detected, it cannot be contained. Long tail analysis is an effective incident detection method to supplement existing incident detection techniques. The long tail analysis framework utilizes a simple assumption that compromised hosts represent a subset of the enterprise asset population. In a homogenous environment, where all assets are nearly identical, differences between systems can be used to identify potentially compromised assets. Realistically, most organizations have a heterogeneous environment and struggle with configuration drift, resulting in a limited ability to easily compare systems. This research extends the Kansa Windows PowerShell incident response framework to Linux operating systems, and utilizes long tail analysis to detect outliers across homogenous and heterogeneous environments.
1. Background

Standardization of enterprise IT infrastructure provides significant benefits such as a consistent user experience, reduced maintenance effort, and lower support costs. Utilizing an enterprise IT architecture to obtain standardization, business use cases can be consistently translated into reusable implementation patterns. Unfortunately, many enterprise IT teams struggle to find the resources necessary to build a comprehensive architecture, keep pace with the relentless technology evolution, remain connected to constantly changing business use cases, and ensure consistent adoption of the technology standards. Additional IT resources or cloud based deployment models, such as Platform as a Service, may partially alleviate some of the standardization challenges. However, the reality is that a significant number of heterogeneous systems will continue to exist for the foreseeable future.

The lack of standardization of the IT environment not only complicates administration, it impairs the enterprise security posture. One of the most significant side effects of a non-standard IT environment is a reduced ability to detect security incidents. In a homogenous environment, anomaly analysis tools can be reliably deployed to monitor deviations from the baseline. In a heterogeneous environment, there is no baseline or consistent normal state, causing anomaly analysis tools to produce significant false positives.

Beyond anomaly analysis tools, the operating system security tool ecosystem generally consists of three categories of capabilities: configuration management, file integrity monitoring, and signature based detection. Configuration management tools such as Puppet and Chef have been developed to assist with the application of IT and security standards. These tools apply standards based on an established baseline and can eliminate configuration drift. However, some organizations may not have the necessary financial resources or understanding of each business system to identify the implications of the configuration changes. The second operating system security capability category focuses on file integrity monitoring. File integrity monitoring solutions such as AIDE and Tripwire monitor critical system configuration files and binaries. These tools provide the ability to alert on sophisticated zero-day exploits or stealthy configuration changes by comparing the operational state to a known checksum baseline. However,
implementing file integrity monitoring post system deployment in a non-standard environment may result in the whitelisting of potentially malicious configurations or binaries. The third operating system security capability category focuses on signature detection. Signature based detection identifies malicious actions by recognizing a set of previously seen characteristics or patterns that were determined to be malicious. Signature based detection is typically very reliable and can help classify an incident. However, signature based detection is easily foiled, does not alert on patterns that have not been previously analyzed, and is difficult to obtain a comprehensive signature coverage for all malicious operating system and application permutations in a heterogeneous environment.

Configuration management, file integrity monitoring, and signature detection tools provide a substantial ability to detect sophisticated incidents. The use of these capabilities in non-standard environments reduces the effectiveness and can be difficult to administrate. However, long tail analysis can be utilized to establish a baseline for heterogeneous environments and verify the baseline for homogeneous environments. With an accurate baseline, anomaly analysis tools can be reliably utilized to supplement the existing incident detection approaches. Long tail analysis operates on the premise that malicious configurations or binaries should be less frequent than benign configurations or binaries. Therefore, forensic artifacts can be collected across the enterprise environment, sorted based on the number of occurrences, and analyzed to determine outliers that may indicate an incident. Long tail analysis is an effective method to determine outliers, however a security analyst must have a firm grasp of the forensic artifacts that are being collected to truly confirm if the outlier is a false positive or an incident. The identification of outliers through long tail is helpful for organizations that do not have a strong configuration management capability, and can identify a malicious state that has been whitelisted. From an attacker’s standpoint, long tail analysis can be thwarted by compromising large portions of the environment, but this will result in an increased footprint and greater chance of detection through other incident identification techniques.

Long tail anomaly analysis is a good supplement, not a replacement, for existing incident identification approaches. The Kansa Incident Response Framework (Hull,

Joshua Lewis, Joshua.D.Lewis@gmail.com
2014) pioneered the use of long tail analysis to identify outliers. This framework uses a series of PowerShell scripts to collect forensic artifacts from Windows operating systems. Each artifact can be collected in parallel leveraging PowerShell remoting, which enables simultaneous artifact collection across massive host populations. After the artifacts are collected, the results of each artifact are merged together and can be opened in Microsoft Excel or imported into a database for long tail sorting and outlier identification. At the time this research was conducted, the author was unaware of any open source or commercial products that provided a similar capability for Linux or Unix operating systems.

The goal of this research is to champion the use of long tail analysis as a simple and effective incident detection and investigation prioritization technique. Additionally, this research extends the Kansa Incident Response Framework capabilities to Linux and Unix operating systems. Specifically, this research is laser focused on parallel OS and application forensic artifact collection, parsing, and outlier identification for Red Hat and Cent OS. Red Hat and Cent OS were chosen based on their prominent enterprise market share and popularity. However, this framework can be easily ported to other Linux or Unix distributions. Additionally, network based long tail analysis will not be covered within the scope of this research.

2. Long tail analysis framework

2.1. Artifact collection phase: conceptual design

The long tail analysis framework is divided into two phases, the artifact collection (section 2.1) and the artifact analysis phase (section 2.2). Each phase of the framework is described in the context of a conceptual design to introduce the high level processes and considerations. Beyond the conceptual design, proof-of-concept scripts are included appendix 5.1.2 - 5.1.5. Reference Figure 1 for a graphical view of the overall long tail analysis framework.
The artifact collection phase is supported by two processes, the artifact extraction (2.1.1) and the artifact gathering process (2.1.2). These processes are designed to extract forensic artifacts from a population of remote systems and gather the extracted artifacts from each remote host for analysis on a central server.

### 2.1.1. Artifact extraction process

The first step in the artifact extraction process is to determine what forensic artifacts should be collected. The initial selection of forensic artifacts should be based on the operating systems and applications (e.g. Sendmail, Apache, Bind, etc.) used in the organization. Within each operating system and application, artifacts should be identified that indicate exploitation, persistence, or privilege escalation. Some forensic artifacts may not be suitable for long tail analysis (e.g. the analysis of logic within a script). Therefore, long tail analysis should supplement, not replace, existing incident detection techniques. Additionally, long tail analysis is not intended for incident investigation, and should not be utilized to collect evidence such as memory or disk images. An example of forensic artifacts for Red Hat and Cent OS operating systems are listed in Table 1. Appendix 5.1.1 provides an explanation for each artifact listed in Table 1, to include commands utilized to extract the artifact, and what to look for in the output.

Joshua Lewis, Joshua.D.Lewis@gmail.com
Table 1: Example forensic artifacts for long tail analysis on Red Hat and CentOS

<table>
<thead>
<tr>
<th>Forensic artifact</th>
<th>Category of actions detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of listening ports</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of running processes</td>
<td>Exploitation, persistence</td>
</tr>
<tr>
<td>List of services and service state</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of kernel modules</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of setuid and setgid files</td>
<td>Privilege escalation</td>
</tr>
<tr>
<td>List of open but unlinked files</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of regular files in the /dev directory</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of files that start with dots or spaces</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of directories that start with dots</td>
<td>Persistence</td>
</tr>
<tr>
<td>List and hash of files in bin directories</td>
<td>Privilege escalation, persistence</td>
</tr>
<tr>
<td>List of immutable files</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of crontab entries</td>
<td>Persistence</td>
</tr>
<tr>
<td>List and hash of scripts executed by run-parts</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of users allowed to make changes to cron</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of user accounts</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of user accounts with null passwords</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of uid 0 user accounts</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of SSH authorized_keys for the root account</td>
<td>Persistence</td>
</tr>
<tr>
<td>Hash of sudoers file</td>
<td>Persistence, privilege escalation</td>
</tr>
<tr>
<td>Pam configuration</td>
<td>Persistence</td>
</tr>
<tr>
<td>SSHd configuration</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of .rhost files</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of hosts.equiv files</td>
<td>Persistence</td>
</tr>
<tr>
<td>List of X*.hosts files</td>
<td>Persistence</td>
</tr>
<tr>
<td>List default run level and init entries</td>
<td>Persistence</td>
</tr>
<tr>
<td>List and hash of run level 3&amp;5 startup and kill scripts</td>
<td>Persistence</td>
</tr>
</tbody>
</table>

Joshua Lewis, Joshua.D.Lewis@gmail.com
After the forensic artifacts have been selected for the long tail analysis investigation, the next step is to ensure that the output of these artifacts can be trusted. Attackers have been known to replace operating system binaries that are used for incident identification (e.g. the `ps` command) or modify binary shared system libraries to hide their malicious artifacts. Rootkits that replace binaries or system libraries can be subverted by temporarily copying statically or dynamically linked binaries from the analysis server to the remote endpoint to run the artifact extraction commands. These statically or dynamically linked binaries are intended to be used solely by the artifact extraction script, do not overwrite the native operating system binaries, and are deleted after the results are extracted. Statically linked binaries can be created using the `gcc --static` flag (Wright, 2010). However statically linked binaries are operating system and kernel version specific, and would require that each binary be statically compiled with each operating system and kernel version in an enterprise (Pogue, Altheide, & Haverkos, 2008). Depending on the operating system and kernel version standardization in the environment, two options can be utilized. The first option is to statically compile all forensic artifact extraction binaries for each operating system and kernel version used in the enterprise environment. This option may be feasible for homogenous environments, and can provide a higher level of assurance. The second option is to compile dynamically linked forensic artifact extraction binaries for each operating system within the enterprise environment. This option may be more feasible for heterogeneous environments, but provides a lower level of assurance since the shared libraries on the system being analyzed will be used in conjunction with the trusted binary. However, both options provide a higher level of assurance than utilizing the binaries and the shared libraries on the systems being analyzed. For example, the binaries need to extract the forensic artifacts from Table 1 are listed in Table 2.

<table>
<thead>
<tr>
<th>Hash of rc.local file</th>
<th>Persistence</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM package verification</td>
<td>Exploitation, persistence, privilege escalation</td>
</tr>
</tbody>
</table>

Joshua Lewis, Joshua.D.Lewis@gmail.com
Table 2: Example binaries for forensic artifact extraction

<table>
<thead>
<tr>
<th>Binaries need to extract forensic artifacts from Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>netstat</td>
</tr>
<tr>
<td>egrep</td>
</tr>
<tr>
<td>sed</td>
</tr>
<tr>
<td>ps</td>
</tr>
<tr>
<td>chkconfig</td>
</tr>
<tr>
<td>hostname</td>
</tr>
<tr>
<td>find</td>
</tr>
<tr>
<td>grep</td>
</tr>
<tr>
<td>sh</td>
</tr>
<tr>
<td>tar</td>
</tr>
</tbody>
</table>

After the trusted dynamically or statically linked binaries have been used to obtain the artifacts, the last step in the artifact extraction process is to parse the results. The comparison of the forensic artifacts across multiple hosts requires the output files to utilize a consistent format. Leverage native command output options or tools, such as grep and awk, to remove unnecessary fields and headers that will cause long tail calculation errors. Additionally, utilize tools such as sed and tr to store the output in a consistent delimited format, such as a CSV, to easily compare the results and enable integration with other tools. Finally, append the hostname and artifact extraction run time to each line in the results. The hostname and run time columns can be ignored during the long tail calculations, but they can be used to quickly trace an outlier back to the original host that generated the artifact.

Based on the artifact selection, use of trusted binaries, and artifact parsing logic, the final step in this process is to automate the artifact extraction. Leverage a script to execute the trusted binaries, parse the output, and write the results of each artifact to a local or remote file. Writing files to local storage may overwrite forensic evidence in unallocated space. Consider writing the extracted artifact files to a remote share or server. Reference appendix 5.1.2 for an example artifact extraction script.

Joshua Lewis, Joshua.D.Lewis@gmail.com
2.1.2. Artifact gathering process

The second process within the artifact collection phase is the artifact gathering process. The artifact gathering process sets up the authentication to the remote endpoint, establishes the investigation methodology, and outlines the logic to automate the process with a script.

The first step in the artifact gathering process is to setup the authentication for the remote endpoints. Copy and remote command execution operations are orchestrated by SCP and SSH with strong public and private key authentication. The SCP and SSH public and private keys can be created using the ssh-keygen command. During the key generation process, ensure that a password is used to encrypt the private key at rest. After the ssh-keygen command completes, edit the public key to add restrictions for how the key can be used. For example, the public key will not be used for X11 forwarding, so the “no-X11-forwarding” option can be set to prevent this activity. After the public key has the appropriate options set, copy the key into the root account authorized_keys file on each remote endpoint that will be evaluated with the long tail analysis framework.

Finally, leverage ssh-agent on the analysis server to load the decrypted private key in memory and automatically perform the authentication for each command. The ssh-agent program provides the ability to utilize encrypted private keys on disk, without continuously re-entering the password each time a command is executed. However, since ssh-agent stores a decrypted copy of the SSH private key in memory, ensure that the analysis server is a single role server that is appropriately hardened and has limited administrator access. Reference section 5.1.3 for an example implementation walkthrough for setting up SSH authentication and ssh-agent.

Once the authentication for the copy and remote command execution is setup, the next step is to determine the investigation methodology. The long tail analysis framework can be executed based on a targeted or macro methodology. Running the long tail analysis framework against a targeted set of similar assets (e.g. Web servers) provides the ability to spot service specific logic outliers. For example, a running Apache service may have a high rate of occurrence across the macro enterprise environment, however the Apache service should be disabled on DNS servers as part of the configuration hardening process. A low rate of occurrence of running Apache services

Joshua Lewis, Joshua.D.Lewis@gmail.com
Incident Identification Through Outlier Analysis

Joshua Lewis, Joshua.D.Lewis@gmail.com
2.2. Artifact analysis phase: conceptual design

The second phase of the long tail analysis framework focuses on analyzing the results of the extracted artifacts gathered from each of the remote endpoints. Using the analysis server, the first step in this process is to concatenate like artifacts (e.g. host1.localdomain_01-18-2016.08.25_kernelModules.csv, host2.localdomain_01-18-2016.08.25_kernelModules.csv, etc.) from the results of each host into a combined file (e.g. currentKernelModulesCombined.csv). Using the combined file, display the appropriate columns that should be used for the long tail calculation. The rate of occurrence of an artifact is calculated based on the full contents of a line. Each extracted artifact line contains fields that should not be displayed for the long tail calculation (e.g. hostname, date, etc.), and are intended to provide the ability to trace an outlier back to a host for further investigation. After the appropriate artifacts are displayed, the data should be grouped based on duplicate lines. Next, the number of lines within the duplicate grouping is counted, the duplicated lines are removed, and unique lines are prefixed by the count of the lines within each duplicate group. Finally, the data can be sorted in ascending order to display the least frequent occurrences first. Since the combined artifact file is in a consistently delimited format (e.g. CSV), the rate of occurrence calculation can be performed with a script or using pivot tables in Microsoft Excel. Reference section 5.1.5 for an example script that implements the long tail analysis process.

After the artifact rate of occurrence has been calculated, the final step in the analysis phase is to interpret the results. Artifacts that have a low rate of occurrence do not immediately indicate a compromise. A low artifact rate of occurrence can stem from a small sample size that is not representative of the larger asset population, a configuration error, a one-off system, or a compromise. Long tail analysis is helpful to quickly spot outliers, where a large number of artifacts are present (e.g. kernel modules). However, long tail analysis merely helps focus investigation efforts, and an analyst needs to understand the underlying artifacts to differentiate a false positive from a compromise. Reference section 5.1.1 for walk through of common Red Hat and Cent OS artifact explanations.

Joshua Lewis, Joshua.D.Lewis@gmail.com
2.3. Long tail analysis limitations

The long tail analysis framework adds a significant capability to supplement existing incident detection methods. However, long tail analysis is not well suited for some types of artifacts such as script logic. Additionally, long tail analysis will not detect rootkits that load a kernel module to mask their existence from the forensic artifact collection tools used in this framework. The benefits and limitations of this framework should be considered based on the needs of the enterprise and the incident detection strategy.

3. Future research

The long tail analysis framework presented in this research establishes a parallel outlier analysis capability for Red Hat and Cent OS operating systems that can be scaled to massive enterprise asset populations. The example implementation scripts in appendix 5.1.2 - 5.1.5 make extensive use of binary path variables to limit the effort required to port this capability to other Linux or Unix variants. Additionally, each forensic artifact is captured in a CSV delimited format and enables data to easily be imported into a database for scalability and increased usability with a web application. Future research should focus on enhancing the long tail analysis framework by adding the capability to analyze multiple Linux or Unix distributions, integrating the capability with a database, and using the the output from multiple long tail analysis investigations to build a historical analysis capability to spot deviations over time.
4. References


https://github.com/davehull/Kansa

Retrieved from http://csweb.cs.wfu.edu/~torgerse/Kokua/SGI/007-2862-005/sgi_html/ch05.html


Red Hat. (n.d.). 13.2.4 Using Key-Based Authentication. Retrieved from redhat.com:


Joshua Lewis, Joshua.D.Lewis@gmail.com


Joshua Lewis, Joshua.D.Lewis@gmail.com


5. Appendix

5.1.1. Artifact explanation

The artifacts captured in the table below are derived from the authors previous experience responding to incidents. These artifacts are not meant to capture all relevant forensic data points, but are provided as a sample for the types of analysis that should be conducted. Organizational specific artifacts can easily be added into the long tail analysis example implementation scripts.

<table>
<thead>
<tr>
<th>Artifact Category</th>
<th>Artifact</th>
<th>Command</th>
<th>Artifact Rationale and Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td>List listening ports</td>
<td>netstat -ln</td>
<td>Listening network sockets that are outside the normal parameters for the asset function (e.g. HTTP listening on a DNS server) may indicate a configuration error or the presence of backdoor utilities. Additionally, obscure listening ports (e.g. 4444) may indicate unauthorized or malicious software. netstat displays the listening sockets (-l) based on port number (-n).</td>
</tr>
<tr>
<td>Processes</td>
<td>List running processes</td>
<td>ps –Ao “comm,pid,ppid,start,user”</td>
<td>Processes can be used to find malicious programs or administration activities. Process anomalies can be discovered by looking for processes that utilize tricky process names (e.g. kthread vs. kthreadd), processes started in the wrong user context (e.g. init started by any user except root), uncommon processes (e.g. rktdaemon), and process with the wrong parent process ID (e.g. init started by the sendmail process). ps displays all processes (-A) with a format (-o) of command, process ID, parent process ID, time process was started, and user (“comm,pid,ppid,start,user”).</td>
</tr>
</tbody>
</table>
| Services | List services and state (on/off) | chkconfig --list | Services are used to run scripts and applications in the background, and can be utilized by attackers for persistence. Run level 3 (full multiuser with networking) and run level 5 (full multiuser with networking and X windows) are the most commonly used run levels. chkconfig lists the services and the service status (on/off) for each run level (--list). Reference the start and stop scripts for additional information on the functionality of each service (e.g. head –n 15 /etc/rc3.d/S24nfslock).

| Kernel modules | List loaded kernel modules | lsmod | Kernel modules provide the Linux kernel with a modular design and enable the kernel to be dynamically or manually modified at runtime by loading and unloading modules (Red Hat, n.d.). Attackers can leverage kernel modules to perform low level rootkit functionality or obtain low level access (e.g. access to all memory). The lsmod command lists the loaded kernel modules and what each kernel module is used by.

| Files | List setuid and setgid files | find / -perm -2000 –o –perm -4000 | setuid and setgid files are permissions granted to executables that allow executable owners or groups to temporarily run privileged functions as unprivileged users. Attackers commonly upload setuid or setgid files to escalate privileges or modify existing binaries for stealthy persistence. The find command is used to locate files in the root directory (/) with the setuid (-perm -4000) or (-o) setgid (-perm -2000).

| Open but unlinked files | lsol +L1 | Each file within an operating system should normally have one or more links. In the past, attackers have started a program then immediately deleted the binary from disk to make the recovery of the original binary more difficult. However, unlinked binaries can be recovered from the /proc virtual file system. lsol lists open files with a link count of zero (+L1). | (SANS Institute, n.d.) |
| Regular files in the /dev directory | find /dev –type f | Character and block files are typically found in the /dev directory (Pomeranz, 2015). A limited number of regular files (-type f) normally exist in the /dev directory. Regular files in the /dev directory should be fairly consistent between systems. The find command is used to locate regular files (-type f) in the dev directory (/dev). |
| Files that start with dots or spaces | find / ( -name "*" –o –name ".*" ) (SANS Institute, n.d.) | Attackers commonly utilize file names that start with a space or a dot for obfuscation. Files that start with a space may allow a file to co-exist and blend in with a similar named file. Similarly, files that start with a dot are hidden from the default ls command (utilize the –a flag to display all entries). The find command locates all files in the root directory (/) with a name that begins with a space (-name "*") or (-o) a name that begins with a dot (-name ".*"). |
| Directories that start with dots | find / -type d –name \* | Attackers commonly utilize directories that start with a dot for obfuscation. Directories that start with a dot are hidden from the default ls command (utilize the –a flag to display all entries). The find command locates all files in the root directory (/) that are directories (-type d) and begin with a dot (\*). |
| List and hash of files in the bin directories | sha256sum /bin/* /sbin/* /usr/bin/* | The /bin, /sbin, and /usr/bin directories contain core binaries and libraries. These files should be relatively consistent across systems (e.g. files are in these directories and the file contents). In the past, attackers have trojaned commonly used binaries (e.g. ls) or added additional binaries to these directories. The sha256sum command creates a SHA 256 hash of each file in the /bin/*, /sbin*, and /usr/bin/* directories. This can be utilized to find unauthorized or trojanized binaries. |

Joshua Lewis, Joshua.D.Lewis@gmail.com
In previous incidents, attackers have set the immutable file permission to prevent a malicious file from being deleted by administrators that are unfamiliar with immutable files. Even with root permissions, files that have the immutable bit set cannot be deleted without removing the immutable bit. Immutable permissions occur infrequently in a normal operating system build. Immutable files should be considered anomalous unless an organization uses this as a common system administration technique. The lsattr command displays a list of file attributes, recursively (-R) for the root directory (/). Files that have the immutable permission set are listed as “----i----------”.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>List crontab entries</th>
<th>Cron is a method to schedule commands and scripts to run at periodic intervals (Red Hat, n.d.). Attackers commonly use cron for persistence. The cat command is used to read the crontab (/etc/crontab) and all shell scripts located in the cron.d (/etc/cron.d/*) directory. This can be utilized to find unauthorized cron jobs or shell scripts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>List and hash scripts</td>
<td>sha256sum /etc/cron.daily/* /etc/cron.hourly/* /etc/cron.monthly/*</td>
<td>Within the crontab, Red Hat includes a “run-parts” cron job that executes any shell script under the /etc/cron.daily, /etc/cron.weekly, and /etc/cron.monthly directories (Red Hat, n.d.). The sha256sum command creates a SHA 256 hash of each file within the /etc/cron.daily/<em>, /etc/cron.weekly/</em>, and /etc/cron.monthly/* directories. This can be utilized to find unauthorized cron shell scripts.</td>
</tr>
<tr>
<td>List users allowed to make</td>
<td>cat /etc/cron.allow</td>
<td>If a cron.allow file exists, then users who are allowed to run cron jobs must be listed in this file. Users that are not in the cron.allow file are not allowed to run cron jobs. If a cron.allow file does not exist, Linux and Unix operating systems look for a cron.deny file. Any user listed in the cron.deny file is not</td>
</tr>
<tr>
<td>changes to cron</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
allowed to run cron jobs. Users not listed in the cron.deny file are permitted to run cron jobs. If a cron.allow or cron.deny file does not exist, then all users are allowed to run cron jobs by default (Red Hat, n.d.). The presence of cron.allow, cron.deny or entries within each of these files may indicate anomalous activity. The cat command is used to read the contents of the /etc/cron.allow file if it exists. This can be utilized to find unauthorized cron.allow files or entries.

<table>
<thead>
<tr>
<th>List users denied from making changes to cron</th>
<th>cat /etc/cron.deny</th>
</tr>
</thead>
<tbody>
<tr>
<td>If a cron.allow file exists, then users who are allowed to run cron jobs must be listed in this file. Users that are not in the cron.allow file are not allowed to run cron jobs. If a cron.allow file does not exist, Linux and Unix operating systems look for a cron.deny file. Any user listed in the cron.deny file is not allowed to run cron jobs. Users not listed in the cron.deny file are permitted to run cron jobs. If a cron.allow or cron.deny file does not exist, then all users are allowed to run cron jobs by default (Red Hat, n.d.). The presence of cron.allow, cron.deny or entries within each of these files may indicate anomalous activity. The cat command is used to read the contents of the /etc/cron.deny file if it exists. This can be utilized to find unauthorized cron.deny files or entries.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authentication</th>
<th>List user accounts</th>
<th>cat /etc/passwd</th>
</tr>
</thead>
<tbody>
<tr>
<td>User accounts provide access to an operating system or application. In previous incidents, attackers have created additional user accounts for persistent access. The first column in the /etc/passwd file contains a list of all operating system user accounts. The cat command is used to read the password (/etc/passwd) file to obtain the current list of user accounts. This can be used to find unauthorized user accounts.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incident Identification Through Outlier Analysis</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td><strong>List user accounts with a null password</strong></td>
<td>awk -F: '($2 == &quot;&quot;) {print $1}' /etc/shadow (Murdoch, 2014)</td>
<td></td>
</tr>
<tr>
<td>Blank (null) account passwords should not normally occur within an enterprise managed operating system. The presence of a user account with a null password may indicate anomalous activity and should be investigated. The awk command is used to read the /etc/shadow file (/etc/shadow), by separating the fields by a colon (-F:), and print the username in field one ({{print $1}}) where the password in field two is blank {{2== &quot;&quot;}}. A password field with a value of &quot;*&quot; indicates an account is disabled, and a password field with a value of &quot;!!&quot; indicates that a password has not been set and the account cannot be used to login.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SSH authorized_keys for the root account</strong></td>
<td>cat /root/.ssh/authorized_keys</td>
<td></td>
</tr>
<tr>
<td>The /&lt;user account&gt;/.ssh/authorized_keys file lists public keys that are allowed to utilize SSH on an operating system (Red Hat, n.d.). In previous incidents, attackers have added their own public keys to the authorized_keys files for persistent access. The cat command is utilized to list the contents for the authorized SSH public keys for the root account (/root/.ssh/authorized_keys). This can be utilized to find unauthorized public keys that are allowed to authentication to the operating system.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Hash of sudoers file</strong></td>
<td>sha256sum /etc/sudoers</td>
<td></td>
</tr>
<tr>
<td>Sudo provides the ability to delegate the use of privileged commands to unprivileged users without the use of the root account (Red Hat, n.d.). The /etc/sudoers configuration file is used to determine the privileged commands that can be ran by users. In an enterprise environment, this file tends to be consistent across related systems or business units. Attackers could utilize the sudoers file to maintain privileged access with an unprivileged account. Deviations in the sudoers files could indicate malicious activity and should be investigated. The sha256sum command is used to create a SHA 256 hash of the</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List of .rhost files</td>
<td>find / -name .rhosts</td>
<td>The <code>.rhosts</code> file is used for insecure host/IP based authentication (Pomeranz, 2015). <code>.rhosts</code> files should not be used in an enterprise environment. Attackers may setup rhost authentication to hide in plain sight, or leverage a compromised machine with rhost authentication to move laterally across the network. The find command is used to locate <code>.rhosts</code> files in the root directory <code>/</code>. This can be utilized to find unauthorized <code>.rhosts</code> files.</td>
</tr>
<tr>
<td>List of hosts.equiv files</td>
<td>find /etc -name hosts.equiv (Murdoch, 2014)</td>
<td>The <code>hosts.equiv</code> file allows users and hosts to utilize legacy commands such as <code>rcp</code>, <code>rsh</code>, and <code>rlogin</code> (Kerrisk, 2015). The legacy <code>r*</code> commands do not utilize encryption. The presence of the <code>hosts.equiv</code> file may indicate a configuration error or an attacker attempting to hide in plain sight. The find command is utilized to locate <code>hosts.equiv</code> files in the <code>/etc</code> directory (<code>/etc</code>). This can be utilized to find unauthorized <code>hosts.equiv</code> files.</td>
</tr>
<tr>
<td>List of X0.hosts files</td>
<td>find /etc -name X*.hosts (Murdoch, 2014)</td>
<td>The <code>X*.hosts</code> file is used for X server host authentication (Katherine &amp; Joseph, 2001). Host based authentication can by trivially defeated using DNS poisoning. Attackers may setup <code>X*.hosts</code> authentication to hide in plain sight, or leverage a compromised machine with <code>X*.hosts</code> to move laterally across the network. Alternative methods should be used to remotely access X serveries, such as X11 forwarding over SSH. The find command is utilized to locate <code>X*.hosts</code> files in the <code>/etc</code> directory (<code>/etc</code>). This can be utilized to find unauthorized <code>X*.hosts</code> files.</td>
</tr>
<tr>
<td>Task</td>
<td>Description</td>
<td>Command</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>Startup</strong></td>
<td>List the default runlevel</td>
<td><code>cat /etc/inittab</code></td>
</tr>
<tr>
<td></td>
<td>List and hash of run level 3&amp;5 startup and kill scripts</td>
<td><code>sha256sum /etc/rc3.d/*</code>&lt;br&gt;<code>sha256sum /etc/rc5.d/*</code></td>
</tr>
<tr>
<td><strong>DNS</strong></td>
<td>List the local DNS host file entries</td>
<td><code>cat /etc/hosts</code>&lt;br&gt;(Zeltser, 2015)</td>
</tr>
<tr>
<td></td>
<td>List DNS servers used</td>
<td><code>cat /etc/resolv.conf</code></td>
</tr>
</tbody>
</table>
used to read the contents of the resolv.conf file (/etc/resolv.conf). This can be used to identify the use of unauthorized DNS servers.

| Packages | Verify RPM packages | rpm –Va (SANS Institute, n.d.) | The rpm package manager can be used to verify an installed package with information about the package from the rpm database. Specifically, rpm can be used to verify package file attributes (size, hash, permission, type, owner, group), package dependencies, and run a verification script (rpm.org, n.d.). The rpm command can be ran with the verification option (-V) for all packages (-a). This can be utilized to find package files that have changed since they were built. However, this may output a large number of changes that may not be malicious. Each change should be verified on a case by case basis. |
5.1.2. Artifact extraction script

The artifact extraction script automates the extraction of the defined long tail analysis artifacts listed in section 5.1.1. This script should be copied over to the remote endpoint that is being analyzed and ran locally. Additional organizational specific application or operating system specific artifacts can easily be added to this script.

#!/bin/bash
# Copyright Joshua Lewis, 2016.
# Version 1.0
# Permission to use and distribute this script permitted as long as the copyright is preserved and referenced
# Use of this script at your own risk, no warranty is implied

#Define path to trusted binaries
BINPATH="/root/analysis/tools"
#Define path for output
OUTPATH="/root/analysis/results"
#Set date and time the analysis was ran
DATETIME=$(date +%m-%d-%Y.%H.%M)

#Make directory results
mkdir $OUTPATH

########################### Ports ###########################
$BINPATH/echo "### Capture listening ports "
$BINPATH/netstat -ln | $BINPATH/egrep -v '^unix|Active|Proto' | $BINPATH/sed -e 's@^@"$(($BINPATH/hostname))"@' | $BINPATH/awk '{ print $5,$2,$1}' | $BINPATH/tr -s '' ',' > $OUTPATH/$(($BINPATH/hostname))_$(echo $DATETIME)_netstat.csv

########################### Process ###########################
$BINPATH/echo "### Capture running processes "
$BINPATH/ps -Ao "comm,pid,ppid,start,user" | $BINPATH/grep -v ^COMMAND | $BINPATH/sed -e 's@^@"$(($BINPATH/hostname))"@' | $BINPATH/awk '{ print $2,$3,$4,$5,$6,$1}' | $BINPATH/tr -s '' ',' > $OUTPATH/$(($BINPATH/hostname))_$(echo $DATETIME)_processListing.csv

########################### Services ###########################
$BINPATH/echo "### Capture run level 3 & 5 services "
$BINPATH/chkconfig --list | $BINPATH/awk '{print $1,$5,$7}' | $BINPATH/sed -e 's@^@"$(hostname)"@' | $BINPATH/tr -s '' ',' > $OUTPATH/$(($BINPATH/hostname))_$(echo $DATETIME)_runLevel3And5Services.csv

########################### Kernel modules ###########################
$BINPATH/echo "### Capture kernel modules "
$BINPATH/lsmod | $BINPATH/awk '{print $1,$2}' | $BINPATH/tr -s '' ',' | $BINPATH/sed -e 's@^@"$(($BINPATH/hostname))"@' > $OUTPATH/$(($BINPATH/hostname))_$(echo $DATETIME)_kernelModules.csv

########################### Files ###########################
$BINPATH/echo "### Capture setuid and setgid files "

Incident Identification Through Outlier Analysis

Joshua Lewis, Joshua.D.Lewis@gmail.com
Joshua Lewis, Joshua.D.Lewis@gmail.com
Incident Identification Through Outlier Analysis

Joshua Lewis, Joshua.D.Lewis@gmail.com
5.1.3. **Setup public and private key authentication**

Public and private keys provide a strong authentication mechanism to authenticate to a remote endpoint. Building and using the keys can be setup on the analysis server using the following steps:

- **Step 1**: Generate public and private keys
  - Run `ssh-keygen -t rsa -b 4096`
  - Ensure a bit length of 4096+ is utilized

  ![Generate public and private keys](image)

  *Figure 2 - Generate public and private keys*

- **Step 2**: Restrict how the public key can be utilized (Pomeranz, 2015)
  - `no-pty`: prevents an interactive shell
  - `no-port-forwarding`: prevents TCP port forwarding
  - `no-X11-forwarding`: prevents forwarding of the X11 protocol

Joshua Lewis, Joshua.D.Lewis@gmail.com
• **Step 3:** Copy the public key to the authorized_keys file for each remote host to be evaluated with the long tail analysis framework
  - scp /home/sans/.ssh/frequencyAnalysisKeys.pub root@webserver:/root/.ssh
  - ssh root@webserver “cat /root/.ssh/frequencyAnalysisKeys.pub >> /root/.ssh/authorized_keys”

• **Step 4:** Make a copy of the private key and rename as id_rsa
  - cp frequencyAnalysisKeys id_rsa

• **Step 5:** Setup ssh-agent
  - ssh and scp natively communicate with ssh-agent through the SSH_AUTH_SOCK variable (Pomeranz, 2015)
- Utilize the script below to run ssh-agent to output the shellcode for the SSH_AUTH.SOCK and SSH_AGENT_PID environment variables to a file, then run ssh-add to load the encrypted private keys in memory.
- Run this script each time the analysis server is restarted

```bash
#!/bin/bash
#This script re-sets the ssh-agent environment variables and loads the encrypted private keys into memory.
#RUN THIS SCRIPT AFTER EACH RESTART

#Script derived from SANS 506 Securing Linux/Unix by Hal Pomeranz

#remove previous versions of the ssh-agent file
rm -f $HOME/.ssh/ssh-agent

#run the ssh-agent command to get shellcode the SSH_AUTH.SOCK and SSH_AGENT_PID, then output this to a file
ssh-agent | grep -v echo > $HOME/.ssh/ssh-agent

#set the permissions on the ssh-agent file
chmod 700 $HOME/.ssh/ssh-agent

#source this file so that the environment variables remain available
$HOME/.ssh/ssh-agent

#force ssh-agent to load the private keys in memory; will be prompted to enter passwords for encrypted private keys
ssh-add

- **Step 6**: Run updateSSH.sh script to update the ssh-agent file created in step 5

- **Step 7**: Source ssh-agent file in analysis server scripts that require ssh-agent authentication

Figure 6 - Update ssh-agent file to be sourced in scripts

Joshua Lewis, Joshua.D.Lewis@gmail.com
5.1.4. Gather artifacts script

The gather artifacts script automates the parallel execution of the remote artifact extraction script and the parallel collection of the artifacts extracted from the remote endpoints. Parallel SSH and SCP are achieved using the parallel-ssh (Chun & McNabb, 2012) program. Download parallel-ssh from https://code.google.com/p/parallel-ssh/, and place the unzipped folder in the current working directory defined in the script variables. Additionally, ensure that the trusted binaries and artifact extraction script is placed in an analysis.tgz tarball in the current working directory (reference section 5.1.6 for additional details). Finally, ensure that a host.txt file exists in the current working directory that lists each remote endpoint that should be analyzed (format the file: root@<insert host name>, one entry per line).

#!/bin/bash
# Copyright Joshua Lewis, January 2016.
# Version 1.0
# Permission to use and distribute this script permitted as long as the
copyright is preserved and referenced
# Use of this script at your own risk, no warrany is implied

################### Setup variables ###################
#Define path to local binaries
LOCALPSCP=/home/sans/Desktop/pssh-2.3.1/bin/pscp
LOCALPSSH=/home/sans/Desktop/pssh-2.3.1/bin/pssh
LOCALSSH=/usr/bin/ssh
LOCALMKDIR=/bin/mkdir
LOCALTAR=/bin/tar
LOCALRM=/bin/rm
LOCALECHO=/bin/echo
LOCALLS=/bin/ls
LOCALMV=/bin/mv
LOCALGREP=/bin/grep
LOCALAWK=/bin/awk
LOCALSORT=/bin/sort
LOCALUNIQ=/usr/bin/uniq

Joshua Lewis, Joshua.D.Lewis@gmail.com
LOCALHEAD=/usr/bin/head
LOCALTEE=/usr/bin/tee
LOCALIFCONFIG=/sbin/ifconfig
LOCALCUT=/bin/cut
LOCALSSHKEYSCAN=/usr/bin/ssh-keyscan
LOCALCHOWN=/bin/chown

#Define working directories and file variables
LOCALWORKINGDIR=/home/sans/Desktop
SCRIPTANDTOOLSTAR=analysis.tgz
TARGETS=hosts.txt
FREQUENCYANALYSISKOWNHOST=frequencyAnalysisKnownHost
FREQUENCYANALYSISRUNLOG=frequencyAnalysisRunLog.txt
ARTIFACTCOLLECTIONSCRIPT=artifactExtraction.sh
LOCALRESULTSDIR=/home/sans/Desktop/results
LOCALHISTORICALDIR=/home/sans/Desktop/results/historical
LOCALCURRENTDIR=/home/sans/Desktop/results/current
REMOTEROOTDIR=/root
REMOTEWORKINGDIR=/root/analysis

#Error handling to make sure the hosts.txt file exists
if [ ! -f "$LOCALWORKINGDIR/$TARGETS" ]; then
  $LOCALECHO "## <error>"
  $LOCALECHO "host.txt file not present in $LOCALWORKINGDIR. Create this file and add \"roo@host\" entries, one per line"
  $LOCALECHO "## </error>"
  exit 255
fi

#Define network interface used to connect to remote machines
LOCALNETIF=eth3

#Define other runtime variables
DATETIME=$(date +%m-%d-%Y.%H.%M)
REMOTERESULTSTAR="${DATETIME}_results.tgz"
CURRENTUSER=$(whoami)

################### Setup SSH agent ###################
#Sourcing the ssh-agent file. This file contains the SSH_AUTH_SOCK variable
to allow ssh-agent to pass the keys that are loaded in memory.
$HOME/.ssh/ssh-agent

################### Clean up local results directory to prep for
# historical analysis #################################
# Check for previous host evaluation results, move any previous results
to <local working directory>/results/historical/<host>/<last run time of analysis>
$LOCALMKDIR $LOCALRESULTSDIR 2>/dev/null
$LOCALMKDIR $LOCALCURRENTDIR 2>/dev/null
$LOCALCHOWN $LOCALRESULTSDIR

# Check for previous host evaluation results, move any previous results
to <local working directory>/results/historical/<host>/<last run time of analysis>
$LOCALMKDIR $LOCALRESULTSDIR 2>/dev/null
$LOCALMKDIR $LOCALCURRENTDIR 2>/dev/null
$LOCALCHOWN $LOCALRESULTSDIR

if [[ ${$LOCALCALLS} | ${$LOCALGREP .csv} ]]; then
  while ${$LOCALCALLS} *.csv &>/dev/null; do
    HOSTLASTRUN=${$LOCALCALLS} | ${$LOCALGREP .csv} | ${$LOCALAWK -F \" _" '
      (print $1) | ${$LOCALHEAD -1}
      (print $2) | ${$LOCALHEAD -1}
      $LOCALALL | $LOCALHISTORICALDIR 2>/dev/null

Joshua Lewis, Joshua.D.Lewis@gmail.com
# Copy tar artifact collection and tools to remote host

```bash
$LOCALMKDIR $LOCALHISTORICALDIR/$($LOCALMkdir $HOSTLASTRUN) 2>/dev/null
$LOCALMKDP $LOCALHISTORICALDIR/$($LOCALMkdir $HOSTLASTRUN)*/$DATELASTRUN 2>/dev/null
$LOCALMV $($LOCALMkdir $HOSTLASTRUN)*/$.csv
$LOCALHISTORICALDIR/$HOSTLASTRUN/$DATELASTRUN
```
done

```bash
fi
```

# Copy the artifact collection results back to the local host and cleanup

```bash
$LOCALMKDIR $LOCALHISTORICALDIR 2>/dev/null
```

Joshua Lewis, Joshua.D.Lewis@gmail.com
Incident Identification Through Outlier Analysis

5.1.5. Long tail analysis script

The long tail analysis script automates the parsing and rate of occurrence calculation for the collected artifacts from remote endpoints. This script is ran on the analysis server after all of the artifact gathering scripts have completed.

#!/bin/bash
# Copyright Joshua Lewis, 2016.
# Version 1.0
# Permission to use and distribute this script permitted as long as the copyright is preserved and referenced
# Use of this script at your own risk, no warranty is implied

# Setup variables
# Define path to local binaries
LOCALECHO=/bin/echo
LOCALCLEAR=/usr/bin/clear
LOCALAWK=/bin/awk
LOCALCAT=/bin/cat
LOCALMKDIR=/bin/mkdir
LOCALSORT=/bin/sort
LOCALUNIQ=/usr/bin/uniq

# Define working directories

Joshua Lewis, Joshua.D.Lewis@gmail.com
Incident Identification Through Outlier Analysis

Joshua Lewis, Joshua.D.Lewis@gmail.com
#Find the frequency for listening ports
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTSDIR/*netstat*.csv >
$LOCALFREQUENCYDIR/currentNetstatCombined.csv
$LOCALAWK -F ', ' '{print $1,$2}'
$LOCALFREQUENCYDIR/currentNetstatCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n > $LOCALFREQUENCYDIR/netstatFrequency
less $LOCALFREQUENCYDIR/netstatFrequency
;;
2) #TODO Capture the PPID in this as well so that process name, user and PPID can help identify start anomalies for processes trying to hide in plane site
#Find the frequency for running processes
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTSDIR/*processListing*.csv >
$LOCALFREQUENCYDIR/currentProcessListingCombined.csv
$LOCALAWK -F ', ' '{print $1,$5}'
$LOCALFREQUENCYDIR/currentProcessListingCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n > $LOCALFREQUENCYDIR/processListingFrequency
less $LOCALFREQUENCYDIR/processListingFrequency
;;
3) #Find the frequency for run level 3 & 5 services
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTSDIR/*runLevel3And5Services*.csv >
$LOCALFREQUENCYDIR/currentRunLevel3And5ServicesCombined.csv
$LOCALAWK -F ', ' '{print $1,$2,$3}'
$LOCALFREQUENCYDIR/currentRunLevel3And5ServicesCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n > $LOCALFREQUENCYDIR/runLevel3And5ServicesFrequency
less $LOCALFREQUENCYDIR/runLevel3And5ServicesFrequency
;;
4) #Find the frequency for kernel modules
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTSDIR/*kernelModules*.csv >
$LOCALFREQUENCYDIR/currentKernelModulesCombined.csv
$LOCALAWK -F ', ' '{print $1,$2}'
$LOCALFREQUENCYDIR/currentKernelModulesCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n > $LOCALFREQUENCYDIR/kernelModulesFrequency
less $LOCALFREQUENCYDIR/kernelModulesFrequency
;;
5) #Find the frequency for setuid and setgid files
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTSDIR/*setUidAndSetGidFiles*.csv >
$LOCALFREQUENCYDIR/currentsetUidAndSetGidFilesCombined.csv
$LOCALAWK -F ', ' '{print $1}'
$LOCALFREQUENCYDIR/currentsetUidAndSetGidFilesCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n > $LOCALFREQUENCYDIR/setUidAndSetGidFilesFrequency
less $LOCALFREQUENCYDIR/setUidAndSetGidFilesFrequency
;;
6)
#Find the frequency for open but unlinked files
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTS/\*open\_unlinked\_file*\.csv >
$LOCALFREQUENCY/CurrentOpenUnlinkedFilesCombined.csv
$LOCALAWK -F ', ' '{print $1,$4}'
$LOCALFREQUENCY/CurrentOpenUnlinkedFilesCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n >
$LOCALFREQUENCY/openUnlinkedFilesFrequency
less $LOCALFREQUENCY/openUnlinkedFilesFrequency

#Find the frequency for regular files in the /dev directory
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTS/\*regularFilesInDev*\.csv >
$LOCALFREQUENCY/CurrentregularFilesInDevCombined.csv
$LOCALAWK -F ', ' '{print $1}'
$LOCALFREQUENCY/CurrentregularFilesInDevCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n >
$LOCALFREQUENCY/regFilesInDevFrequency
less $LOCALFREQUENCY/regFilesInDevFrequency

#Find the frequency for files that start with a dots or spaces
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTS/\*dotAndSpaceFiles*\.csv >
$LOCALFREQUENCY/CurrentdotAndSpaceFilesCombined.csv
$LOCALAWK -F ', ' '{print $1}'
$LOCALFREQUENCY/CurrentdotAndSpaceFilesCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n >
$LOCALFREQUENCY/dotAndSpaceFilesFrequency
less $LOCALFREQUENCY/dotAndSpaceFilesFrequency

#Find the frequency for directories that start with a dots
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTS/\*dotDirectories*\.csv >
$LOCALFREQUENCY/CurrentdotDirectoriesCombined.csv
$LOCALAWK -F ', ' '{print $1}'
$LOCALFREQUENCY/CurrentdotDirectoriesCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n >
$LOCALFREQUENCY/dotDirectoriesFrequency
less $LOCALFREQUENCY/dotDirectoriesFrequency

#Find the frequency for list and hash of files in the bin directories
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTS/\*hashOfFilesInBinDirectories*\.csv >
$LOCALFREQUENCY/CurrenthashOfFilesInBinDirectoriesCombined.csv
$LOCALAWK -F ', ' '{print $1,$2}'
$LOCALFREQUENCY/CurrenthashOfFilesInBinDirectoriesCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n >
$LOCALFREQUENCY/hashOfFilesInBinDirectoriesFrequency
less $LOCALFREQUENCY/hashOfFilesInBinDirectoriesFrequency

#Find the frequency for list of immutable files

Joshua Lewis, Joshua.D.Lewis@gmail.com
Incident Identification Through Outlier Analysis

Joshua Lewis, Joshua.D.Lewis@gmail.com
Incident Identification Through Outlier Analysis

Joshua Lewis, Joshua.D.Lewis@gmail.com
22) #Find the frequency for the list of X0.hosts files
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTSDIR/*/X0.hostsFiles*.csv >
$LOCALFREQUENCYDIR/currentX0.hostsFilesCombined.csv
$LOCALAWK -F ',' '{print $1}'
$LOCALFREQUENCYDIR/currentX0.hostsFilesCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n > $LOCALFREQUENCYDIR/X0.hostsFilesFrequency
less $LOCALFREQUENCYDIR/X0.hostsFilesFrequency
;;

23) #Find the frequency for default runlevels
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTSDIR/*/inittab*.csv >
$LOCALFREQUENCYDIR/currentinittabCombined.csv
$LOCALAWK -F ',' '{print $1}'
$LOCALFREQUENCYDIR/currentinittabCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n > $LOCALFREQUENCYDIR/inittabFrequency
less $LOCALFREQUENCYDIR/inittabFrequency
;;

24) #Find the frequency for the list and hash of run level 3 & 5 startup and kill scripts
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTSDIR/*/runlevel3And5StartKillScripts*.csv >
$LOCALFREQUENCYDIR/currentrunlevel3And5StartKillScriptsCombined.csv
$LOCALAWK -F ',' '{print $1,$2}'
$LOCALFREQUENCYDIR/currentrunlevel3And5StartKillScriptsCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n > $LOCALFREQUENCYDIR/runlevel3And5StartKillScriptsFrequency
less $LOCALFREQUENCYDIR/runlevel3And5StartKillScriptsFrequency
;;

25) #Find the frequency for the host file entries
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTSDIR/*/etcHosts*.csv >
$LOCALFREQUENCYDIR/currentetcHostsCombined.csv
$LOCALAWK -F ',' '{print $1,$2}'
$LOCALFREQUENCYDIR/currentetcHostsCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n > $LOCALFREQUENCYDIR/etcHostsFrequency
less $LOCALFREQUENCYDIR/etcHostsFrequency
;;

26) #Find the frequency for the DNS servers used
$LOCALCLEAR
$LOCALCAT $LOCALCURRENTRESULTSDIR/*/resolv.confEntries*.csv >
$LOCALFREQUENCYDIR/currentresolv.confEntriesCombined.csv
$LOCALAWK -F ',' '{print $1,$2}'
$LOCALFREQUENCYDIR/currentresolv.confEntriesCombined.csv | $LOCALSORT | $LOCALUNIQ -c | $LOCALSORT -n > $LOCALFREQUENCYDIR/resolv.confEntriesFrequency
less $LOCALFREQUENCYDIR/resolv.confEntriesFrequency
;;

27) #Find the frequency for RPM package verification

Joshua Lewis, Joshua.D.Lewis@gmail.com
5.1.6. Putting it all together

- **Step 1**: Create a tar file called “analysis” that contains the trusted binaries and artifact extraction script that will be copied over to the remote endpoint
  - mkdir -p /home/sans/Desktop/analysis /home/sans/Desktop/analysis/tools
  - cd /home/sans/desktop
  - copy statically linked or trusted binaries and scripts into the analysis folder
    - /bin/netstat
    - /bin/egrep
    - /bin/sed
    - /bin/ps
    - /sbin/chkconfig
    - /bin/hostname
    - /bin/find
    - /bin/cat
    - /usr/bin/sha256sum
    - /sbin/lsmod
    - /usr/sbin/lssof
    - /bin/rpm
    - /usr/bin/lsattr
    - /bin/awk
    - /usr/bin/tr
    - /bin/grep
    - /bin/echo
    - /bin/sh
- /bin/sort
- /bin/tar
- /bin/rm
  - tar –czf analysis.tgz analysis

![Figure 8 - Long tail analysis framework environment setup](image)

- **Step 2:** Run the updateSSH script

```bash
[sans@localhost Desktop]$ ./updateSSH.sh
Enter passphrase for /home/sans/.ssh/id_rsa:
Identity added: /home/sans/.ssh/id_rsa (/home/sans/.ssh/id_rsa)
[sans@localhost Desktop]$ 
```

![Figure 9 - Update ssh-agent file](image)

Joshua Lewis, Joshua.D.Lewis@gmail.com
• **Step 3:** Test ssh-agent with a remote endpoint
  
  o Source the ssh-agent file in the current terminal window
    
    ▪ `. /home/sans/.ssh/ssh-agent`
  
  o `ssh root@webserver1 uptime`

![Figure 10 - Test ssh-agent authentication](image1)

• **Step 4:** Run the gather artifacts script

![Figure 11 - Run the gather artifacts script](image2)

• **Step 5:** Review the updated directory structure

Joshua Lewis, Joshua.D.Lewis@gmail.com
Figure 12: Updated directory structure

- **Step 6**: Run the long tail analysis script
Incident Identification Through Outlier Analysis 46

Joshua Lewis, Joshua.D.Lewis@gmail.com

- **Step 7**: Tracing an outlier back to a host
Figure 15 - Tracing an outlier back to a host

```bash
[sans@localhost frequency_analysis]$ pwd
/home/sans/Desktop/results/frequency_analysis
[sans@localhost frequency_analysis]$ grep 8080 currentNetstatCombined.csv
0.0.0.0:8080,tcp,localhost.localdomain
[sans@localhost frequency_analysis]$ 
```
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Chicago Spring 2020</td>
<td>Chicago, ILUS</td>
<td>Jun 01, 2020 - Jun 06, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS ICS Europe Summit &amp; Training 2020</td>
<td>Munich, DE</td>
<td>Jun 08, 2020 - Jun 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Budapest June 2020</td>
<td>Budapest, HU</td>
<td>Jun 08, 2020 - Jun 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Las Vegas Summer 2020</td>
<td>Las Vegas, NVUS</td>
<td>Jun 08, 2020 - Jun 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Zurich June 2020</td>
<td>Zurich, CH</td>
<td>Jun 15, 2020 - Jun 20, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS London June 2020</td>
<td>OnlineGB</td>
<td>Jun 01, 2020 - Jun 06, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>