Threat Hunting using GRR Rapid Response

Hussein Rasheed [1], Ali Hadi [2], Mariam Khader [3]
Computer Science Department
Princess Sumaya University for Technology
Amman, Jordan
[1] hussein_rasheed@yahoo.com
[2] a.hadi@psut.edu.jo
[3] mkhader@psut.edu.jo

***Abstract*— Cybercrimes have evolved, and their tactics and techniques are increasingly changing with an alerting pace. This calls for a change in the mindset used to implement security measures, by adopting the approach of continuously and constantly looking for attacks that pass through the deployed security solutions. This approach of searching through the networks for any evidence on threat activity, rather waiting for a breach notification is referred to as cyber threat hunting. This paper discusses the deployment of threat hunting process using GRR Rapid Response. Two experiments were conducted, in which, both remote code execution, client side exploits are tested, and successful exploitation was used to configure a backdoor to the victim's system to achieve persistence. The experiments show that threat hunting can be achieved by the study of the monitored system's normal patterns of behavior, which will help identify the indications and thresholds that can be used in threat hunting.**

Keywords*— *IoC; GRR Rapid Response; Response; Threat Hunting; Threat Intelligence;

I. INTRODUCTION

Combating cybercrime today cannot be by relying on traditional protective perimeter technologies including firewalls, network IDSs/IPSs and anti-viruses, which are immune against basic attacks. This is mainly due to two facts; technology and infrastructure of services are dramatically changing, plus cybercrimes have evolved and their tactics' sophistication is increasingly changing with an alerting pace [1]. One of the main problems to be considered is that threat might already exist undetected in organizations' networks, with those organizations under a total belief that their highly paid-for security measures are perfect and impenetrable [2]. Such hidden threats are considered advanced targeted threats or Advanced Persistent Threats (APTs), because of their ability to work around security controls, and hide themselves in locations such as memory where they cannot be detected easily [3]. Measuring such problem is complicated, because most organizations do not have the needed visibility into their environments and networks despite the deployed traditional security solutions. This lack of visibility make it difficult, to define what constitutes to be a normal behavior threshold, which leads to inability to detect abnormal activities that need further investigation [4].

The solution of such threats is continuous looking for attacks that pass through the deployed security solutions, by searching through the networks for any evidence on threat activity, rather waiting for a breach notification. This process is referred to as "cyber threat hunting". Although hunting is a relatively new term, it has been informally and partially performed for quite some time by security practitioners [2]. Threat hunting can be defined as "the act of aggressively intercepting, tracking and eliminating cyber adversaries as early as possible in the Cyber Kill Chain [5]." The earlier a threat is detected within the chain, faster response and recovery of operations and services can be achieved. Threat hunting is categorized under the active defense category, and is considered one of the most effective components of intelligence. With the term emerging however, more focus is being practiced on the analysis and automation aspects of the hunting process targeting faster and more accurate results. This becomes an essential corner stone in the security policy of any organization, taking into consideration that attackers depend on frequent hits to achieve occasional success.

The exploit, or an attack on the system, takes advantage of existing vulnerabilities and weakness on the system (An operating system or an application) to achieve the desired action of obtaining illegal access, denial of service or any other malicious activity. Once exploitation is achieved, post exploitation activities can be started, in which the value of the compromised system is determined, and means of control of the system are configured for later use. Maintaining means of control on the compromised system to be used later, which is referred to as Persistence, is used by attackers to maintain their gained access. This approach is performed by attackers especially if the exploitation process may not be reproducible. Persistence mechanisms used include modifying registry keys to run an installed backdoor on startup, or activating a Windows remote administration service. Such changes can be used as a fingerprint to identify malicious activity in the threat hunting process.

Threat hunting goes through a number of stages: starting with planning, where assets of importance are identified, and the teams responsible for the different threat hunting tasks are assigned. The patterns of the chosen assets are also learned, by monitoring the proper segments of network, and by ensuring that systems and users activities are logged at the appropriate level of details (avoiding too much details leading to processing overhead and too little details causing lack of visibility). Pursuing and detection come afterwards, where adversaries are pursued using one of two approaches: inspecting for known threats by relying on existing indicators of Compromise (IoCs), such as signatures of known attacks. The other approach is detecting unknown threats, which

978-1-5386-0527-1/17 $31.00 © 2017 IEEE
DOI 10.1109/ICTCS.2017.22

requires defining the baseline of what is considered normal activity enabling the ability to detect deviations and, with high possibility, unknown threats. The next stage is responding, which is the active stage of the hunting process, and that includes using the needed and efficient means to detect and stop the attack at the earliest stage possible (e.g. the reconnaissance stage). Response would be more intense if an attack has already succeeded or is in progress. In such cases, tracking all pathways to determine the scope of impact, data loss and egress points is needed, along with locating and locking out the attacker erasing any back doors. Reporting comes at the end to document the threat details, tools and logs used, affected systems details, lessons learned and any other information that could add to the intelligence knowledge base. There are usually two ways to conduct threat hunting. The first is by continuously hunting for known IoCs and threats, and for new unknown threats and anomalies. The second is by performing on-demand hunting focusing on a certain IoC or behavior pattern. A concept that is becoming more popular in addressing the problems of collection, sharing, and storage in the Threat Intelligence area is the Threat Library or Threat Intelligence Platform. It mainly focuses on collecting intelligence from commercial tools feeds and Open Source Intelligence (OS-INT), storing it securely, facilitating easy and flexible access to it, and enabling integration with defensive tools [6]. Of the sources of collected intelligence, a number of open source projects are beginning to gain popularity in this area through their ability to add great value to intelligence collected from commercial sources. One example of those projects is GRR Rapid Response from Google.

GRR Rapid Response is an open source incident response framework, which is written in Python and aims to performing scalable remote live forensics on hosts running different OSs. GRR does not seem to have any other competition now (besides a few pricey options). It adopts the server-client model, where agents are installed on machines (clients) to be able later to communicate with the GRR server to be authenticated and assigned a unique client ID. Once this setup is up and running, server can send requests to clients collecting information, and clients send back the responses to those requests [7]. GRR, with its built-in configuration, can collect a wide variety of information that is considered vital in detecting any suspicious activity from the requested number of clients. Such information includes registry keys values, history files of browsers, files systems metadata, files and folders, network connections, memory contents and many other pieces of information. GRR can also be used to quickly collect forensics artifacts such as logs, configured services, cron jobs, user accounts, and much more information that can help during security investigations. This is achieved through GRR's ability to define customizable artifacts with varying location and format across systems [8]. GRR has a many features that make it competitive in the area of threat hunting such as: scalability, remote live forensics, secure communication, and others [9]. Such features are achieved through GRR's artifacts, flows, hunts and cron jobs. Artifacts are common pieces of information that security analysts frequently collect during a security investigation, such as the collection of event logs, user accounts, services configuration, or searching for files with certain extensions or registry keys with certain properties. Flows are GRR's way to collect the artifacts from a single client in an asynchronous way, where a request with an assigned ID is sent to the client of interest to collect the desired artifact data, then the flow is serialized and saved to the GRR

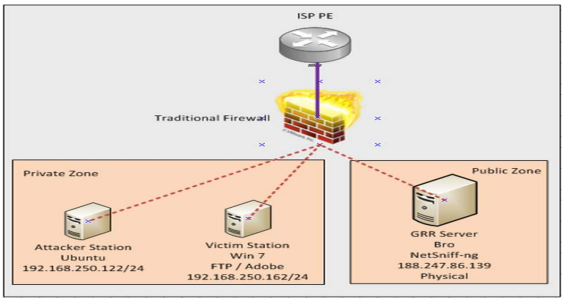

Figure 1: Experiment Setup Topology

data store until the server received a response from the client. When a flow is sent to many clients, it is called a hunt, and GRR cron jobs are used to schedule hunts.

The rest of this paper is organized as follows: The technical details of the testing environment and tools are illustrated in Section II, followed by the details of experiments explained in Section III, and a discussion of the results in Section IV. Section V concludes this paper.

II. TESTING ENVIRONMENT

This section describes the details of the machines used and the topology adopted for the testing environment. The following is breakdown of the machine, its role, and OS and software used, the IP address for each machine is presented in Table 1:

1. GRR Server, which was used to collect and analyze clients' artifacts (Ubuntu 14.04.4 LTS Desktop x86_64 bit, GRR Server 3.0.0.7, Python 2.7, Exim MTA)
2. Victim, which is a host vulnerable software used for experiments (Windows 7 64-bit, GRR Client 3.0.0.7, Adobe version 8, PCMAN's FTP Server 2.0.7)
3. Attacker, which perform exploitation and post exploitations attacks (Ubuntu 14.04.4 LTS Desktop x86_64 bit, Python 2.7, Metasploit Framework)

As depicted in Figure 1, the topology of the implementation setup consists of two zones separated by a traditional firewall; the published zone used for hosting the GRR Server with Internet access for updates, and send alerts, and the private zone to host the internal stations. Both the Victim and Attacker Stations will be located in the private zone. Rules added to the firewall include: opening inbound and outbound rules between the GRR Server and the GRR Client (Victim Station), and opening outbound rules from the GRR Server towards the Internet.

Table 1: Machine IP Addresses

Machine	IP Address
GRR Server	188.247.86.138/27
Victim	192.168.250.162/24
Attacker	192.168.250.122/24

III. EXPERIMENTS

This section discusses the details of the experiments conducted. The first experiment covers a remote memory corruption exploit executed against a vulnerable FTP server application, and the second covers a client side exploit using a vulnerable Adobe application with configuring a persistence mechanism. Each experiment consists of two parts: The first part illustrates the steps, tools, and applications used to accomplish the exploit, post exploitation and persistence mechanism. The second part explains the deployment of GRR artifacts, flows, and hunts to detect the exploits executed in the first part.

3.1 Experiment One: GRR Threat Hunt of a Memory Corruption Exploit with a Reverse TCP Shell Payload

This experiment targeted the FTP server application, which is vulnerable to remote code execution. The used payload was a reverse TCP shell that results in establishing a reverse TCP connection from the victim machine to the attacker's machine. GRR ability to detect the exploit is then demonstrated by network connections and memory scan, showing both on-demand and continuous collection of GRR data for the network connections artifact.

The executed buffer overflow exploit resulted in two main events on the Victim Station that can be considered as IoCs. The first is the large amount of NOPs and garbage data (in this case "\x42") that got placed in the Victim Station's Memory, and which was used to cause the buffer overflow. The second is the resulting TCP connection from the Victim Station towards the Attacker Station. Both of the two events are unusual events for the operation of an FTP Server, and hence can be probed to give an indication of a possible threat. When it comes to memory, GRR Rapid Response provides a number of flows under Memory Category such as AnalyzeClientMemory, DownloadMemoryImage, Memory Collector, etc. of which, ScanMemory flow provides the functionality of scanning the client's memory for a pattern or a regex, returning the offset of the first hit or all hits (as set in the artifact configuration). This makes ScanMemory suitable for looking for the junk characters and NOPs injected by the exploit performed in the previous section. As for Network category, only one flow currently exists which is the Netstat flow. This flow can collect the connections of the client and store them in the GRR Server data store. This flow can also be useful since the resulting reverse TCP shell connection of the exploit under study shows in the client's network connections, and hence can be collected by this flow.

3.1.1 GRR NetStat Hunt

Once GRR flows of use to the case under study are determined, two steps must be taken: the first is to configure a GRR cron job that schedules the hunt for collecting the needed data and artifacts from the clients under monitoring (usually, a labeled group of clients), as per the schedule that satisfies the monitoring needs. In this experiment and for NetStat flow, the details of the created cron job are Flow (Network\Netstat), Output Plugin (None), Rule Type (Clients with Label, Label: Monitor), Description (NETSTAT) and Periodicity (5m).

Once the cron job is created and enabled, periodic hunts can be seen running as per the selected schedule and set configuration, and retrieved results can be viewed for each completed hunt. As for the configured periodicity time, the less the time between the collected data, the better when it comes to detection rate, but that will create a huge amount of data, so a compromise needs to be done depending on the criticality and resources of systems monitored. The second step of using GRR is to figure out the way the retrieved data is exported from GRR data store and analyzed. With the version of GRR used in this experiment, the following options are available:

- Viewing the results of the triggered hunts using User Interface (UI), which provides the option to view the html rendered results, or view and save the CSV output (when choosing the CSV Output Plugin when configuring the flows or hunts parameters).
- Using the GRR Console grr_export command to export the CSV output (when choosing the CSV Output Plugin when configuring the flows parameters). This only works with GRR flows and not hunts, due to the different format of hunts results (called Collections) which requires advanced and complex manipulation of data.
- Using GRR provided APIs that uses GET and POST methods to perform GRR's functionality available through the UI and Console.

For the continuous way of threat hunting, the third option comes into play along with GRR cron jobs. That helps for automating the export of GRR data. Relying on APIs that can get the results of running flows and hunts in JSON format, and that can post commands to create new flows and hunts with certain parameters, noting that the APIs available still do not cover all the UI and console tasks, and new APIs are added per release. Selecting the API approach and looking up the available APIs, two APIs were found to be useful in achieving the objective of detecting abnormal network connections, through analyzing the Netstat hunts launched by the scheduled cron jobs. The two APIs are:

- The GET /api/hunts API, which returns a list of all available hunts ordered by date and time of starting the hunt (with the newest at top) in JSON format. Searching through the results of this API for the last run hunt with the NETSTAT Description, will result in finding the last Netstat hunt performed on the Monitor labeled clients, which once found, its hunt ID can be read from hunt properties in the same JSON output. This traversing through the JSON formatted results (along with calling the APIs) is accomplished through Python, which enables access to JSON fields using methods of json library. One note to be considered when dealing with GRR's resulted JSON responses, is that the JSON response body starts with a magic prefix line (consisting of extra closing brackets) that must be stripped before feeding the rest of the response body to a JSON parser. This line is added at the beginning of the file to prevent against Cross Site Script Inclusion (XSSI) attacks.
- The GET /api/hunts/[hunt id]/results API, which returns the results of the specified hunt (with the hunt ID passed after searching through /api/hunts API result). The results of this hunt are the network connections of the clients being hunted, which can be inspected for unusual connections. In the experiment at hand, the expected "normal"

ESTABLISHED connections are the ones targeting the FTP Server (on TCP port number 21). That in mind, it is safe to assume that any other ESTABLISHED connections on the Victim Station might be suspicious and might need further investigation. The JSON response of this API is searched for connections with the status of "ESTABLISHED", and when found, those connections are inspected to check the local port number, to see if it is equal to 21.

The Python script for calling the APIs and analyzing the JSON responses is added to the cron tab of the GRR Server and scheduled to run every 5 minutes to sync with the GRR Hunt Cron Jobs, reading latest hunts once they are available. Once the exploit is executed, alert emails are received. The alert email contains the details on the local IP address and port number of the Victim Station, along with the remote IP address and port number of the Attacker Stations. The on-demand way of threat hunting can be achieved by combing options 1 and 2. GRR flows can be configured using GRR UI by choosing the client (the Victim Station) and then choosing to start a new flow. An output plugin (CSVOutputPlugin in this case) can be chosen to determine the appropriate output format of the flow results. The results of this flow can afterwards be viewed and downloaded from the UI of GRR. For a number of launched flows against a certain client, analyzing the results can be a bit cumbersome when tried from the UI. Therefore, the grr_export can be used to export the CSV output files from the GRR data store into the analyzing machine's file system. After that, a python script is used to analyze the CSV output files and look for suspicious network connections.

3.1.2 GRR ScanMemory Hunt

The buffer overflow targeting the PCMAN's FTP Server is accomplished through sending a large number of characters (usually the ASCII code of a large number of 'A's, 'B's…). A large number of NOPs, along with the shellcode through input variables that are not verified by the target application. All of these components will reside in the stack segment of memory once exploit succeeds. The existence of such large number of garbage data in memory can at sometimes be an indication of a threat, although special care must be taken when dealing with this IoC, since such characters (NOPs, 'A's, 'B's …) typically do exist in modules loaded into memory in the normal course of any client's operation. Therefore, the number of characters to consider as a threshold should be chosen based on a number of scans of the clients' memory, that will help define the threshold of number of characters, above which examining the clients of memory might be needed.

The buffer overflow string usually contains two parts. |The first are the characters used to cause the overflow of the destined variable, which in our case consists of huge number of 'B's (more than 2000 'B's), and the second consists of NOP instructions which are placed just before the shell code to ensure that the executed jump (JMP ESP) lands before the shell code. While the number of the filling characters can be large extending to thousands, the number of used NOPs might not be as large and might not even exceed a hundred instructions. GRR includes the ScanMemory flow that can scan the client's memory looking for hits on certain characters, which makes it suitable for the purposes of this experiment, as it can be used to look for a sequential number of NOP (\x90), "A" (\x41), "B" (\x42) characters. Starting with the NOP, and performing ScanMemory for 4 NOP instructions "\x90\x90\x90\x90" against a normal client shows 10001 hits. Continuously increasing the number of sequential NOPs to search for decreases the resulting hits, however, even with up to 512 NOPs set, the number of hits goes up to 884.

Analysis of the existence of other characters like 'B's in memory shows that they have a much smaller size of sequential characters in memory under normal conditions, as when setting the size of matched string to 256, the number of hits goes down to 0. The analysis of a number of clients' memory spaces to find out the normal threshold of the size of sequential NOPs, 'A's and 'B's, shows that it is hard to define such a number to be generalized among all clients, because of the variations among different clients (that are running same OS and program set). The analysis (which is conducted on three stations running Windows7 and basic programs) however shows, that the NOP usually has the largest sliding size (exceeding 512 sequential NOPs), followed by the 'A' (around 300 sequential A's), and with rest of characters ranging from 64 to 256 characters. Based on the analysis of normal memory contents, and knowing the exploit code, a ScanMemory hunt to search for sequential B's (\x42) is to be used. Because of this, flow has the option of using an EmailOutputPlugin which can send an email for each hit found, that could be used for the alerting part once a match on the characters is found. A GRR cron job with the following details is scheduled: Flow (Memory\ScanMemory), Regex (256 * \x42), Output Plugin ("Send an email for each result"), Rule Type (Clients with Label \ Label: Monitor), Description (MEMSCAN) and Periodicity (10m). Once the cron job is created and enabled, periodic hunts can be seen running as per the selected schedule and set configuration. Once the exploit is executed, alert emails are received. The alert email contains a link to the results page in GRR UI, which shows the offset address of the hit on the sequence of 'B's found and length of sequence found (2068 in this case).

3.2 Experiment Two: GRR Threat Hunt of a Persistence Mechanism Achieved by a Client Side Exploit

In this experiment, a client side exploit is conducted using a vulnerability in Adobe Reader, which is used to deliver a Meterpreter payload that is used to perform a post exploitation persistence mechanism. Meterpreter is an advanced payload which uses in-memory DLL injection stagers and is which is extended over the network at runtime. It performs the intended communication over a stager socket, to be able to provide a client-side Ruby API that features command history, tab completion, channels, and more [11]. GRR ability to detect the post exploitation action is then demonstrated by the use of an artifact that helps deal with clients' registry. Since compromising a network perimeter has become more difficult today with the focus paid by security teams on strengthening the perimeters of their networks, and the huge advancements in perimeter security solutions, client side attacks gained a huge momentum because they target users of the network, causing the threat to come from inside instead of the traditional direction of attacks. Client side attacks can be very dangerous due to a number of facts. One is that they have high success ratio, due to both the huge surface attack represented by the

large number of applications used by the end users of networks, and the drastically increasing number of exploitable vulnerabilities for those applications.

Another factor contributing to the criticality of client side attacks is that they hard to detect because they bypass security boundaries imposed by firewalls, intrusion detection engines and other perimeter devices. Users who have access to the network, systems, data, Internet and assets anytime and with high privileges most of the time represent a very high value for attackers if compromised. This fact, in addition to the fact that securing the users' environment is far more complex than securing servers environment, helps add to the criticality and exposure of client side attacks. The details of the exploitation, post exploitation and the hunting process are explained in the following two subsections.

3.2.1 Exploitation Process

Adobe Reader and Acrobat 8.x has a vulnerability on one text field in the warning dialog triggered by the Launch File action, where no restriction exist on the contents of this field, which enables remote attackers to deceive users into executing an arbitrary code specified in the PDF document [12]. The first step in executing this exploit is to create the malicious PDF file. This is done using Metasploit Framework, and specifically the Adobe PDF Windows File Format module which embeds a Metasploit payload (EXE) into an existi¬ng PDF file. The payload used with this exploit is a Windows Meterpreter Reverse TCP (with LHOST set to use the IP address of the attacker machine and port set to 5555). After generating the malicious PDF file, the file could be transferred to the victim machine using different techniques outside the scope of this work

The second step is to prepare the Metasploit Multi-handler that will handle the reverse TCP connection initiated from the victim machine. Setting the payload and options for this tool to match the exploit's payload and options, then starting the handler makes the Attacker's Station ready for receiving incoming connections. Opening the malicious PDF file on the Victim Station will trigger establishing the reverse TCP Meterpreter connection towards the Attacker Station on TCP Port 5555. Having the Meterpreter connection opened to the Victim Station, persistence post exploit actions can be configured using the Meterpreter script persistence.rb, which will create a Meterpreter service to be available even if the Victim Station is rebooted. The persistence post exploitation step used is: "*run persistence –U –i 5 –p 5555 -r 192.168.250.122"*. This will cause the victim machine try to reconnect back to the attackers' machine five seconds after every reboot.

3*.2.2 GRR Threat Hunt*

The executed exploit resulted in adding an autorun registry key in the Victim Station, to leave a back door enabling the automatic start of the Meterpreter service at user login. Such a change on the Victim Station can be considered to be an indication of a possible threat, and hence can be added to the continuous checklist against clients.

GRR Rapid Response includes many ready artifacts that relate to retrieving registry keys values (such as Run and RunOne keys). Some of those registry related flows come with an email output plugin that can send an alert email if a value is found for any specific key. Registry Finder flow is one such flow that looks for values under registry keys set by the admin, and sends and email to the admin's email address once any value is found. This suits the experiment in hand, since the HKEY_USERS/%%users.sid%%/Software/Microsoft/Windows/CurrentVersion/Run Key contains no values for the Victim Station under normal operation. For clients with values under this registry key in normal operation, conditions can be added to the RegistryFinder flow to exclude those values. The details of the created cron job are as follow: Flow(Registry\Registry Finder), Output Plugin (EmailOutputPlugin), Rule Type (Clients with Label \ Label: Monitor), Description (REGFIND) and Periodicity (5m). Table 2 shows the properties of the resulting hunt. Once the cron job is created and enabled, periodic hunts can be seen running as per the selected schedule and set configuration. Once the exploit is executed, alert emails are received.

Table 2: GRR Hunt Experiements

Hunt Name	Description	Proporities
GRRScanMemoryHunt	Scan in memrory	• Flow: Memory\ScanMemory • Regex: 256 * \x42 • Output Plugin: "Send an email for each result" • Rule Type: Clients with Label \ Label: Monitor • Description: MEMSCAN • Periodicity: 10m
NetworkStatusHunt	View of the current network status	• Flow: Network\Netstat • Output Plugin: None • Rule Type: Clients with Label \ Label: Monitor • Description: NETSTAT • Periodicity: 5m
GRRRegisteryFinderHunt	Search for a key within windows registry	• Flow: Registry\Registry Finder • Output Plugin: EmailOutputPlugin • Rule Type: Clients with Label \ Label: Monitor • Description: REGFIND • Periodicity: 5m

IV. RESULTS AND DISCUSSION

The two executed experiments in this paper demonstrated a number of results that can be summarized by the following: In Experiment One, threat hunting of a remotely executed exploit using GRR Rapid Response was demonstrated. It was shown that threat hunting could be achieved by the study of the monitored systems normal patterns of behavior, based on which, IoCs could be determined and monitored to trigger alerts. One example on the IoCs used was the expected network connections, where the published service on the Victim Station required only port 21 to be opened and listening to incoming connections. This made any established network connections on ports other than port number 21 seem suspicious and consequently considered an IoC, which when used, could detect the reverse TCP shell payload of exploit performed against the Victim Station.

Another example on IoCs was demonstrated in Experiment One, which was the memory contents of the monitored system. This IoC helped detect the executed buffer overflow exploit through detecting the large number of junk characters injected into memory to cause the memory corruption attack, however, the analysis performed as part of the experiment to validate the threshold of alerting showed that it is difficult to determine

such a threshold and generalize it among clients, especially for the NOPs instruction. The threshold, therefore, should be carefully chosen and calibrated to avoid false positives, and this IoC should be used along other IoCs to help validate threats. In experiment Two, a client side exploit with a Meterpreter payload was used which helped establish a persistence mechanism, and GRR threat hunting of the installed backdoor was demonstrated using the detection of changed registry keys. Registry keys being an IoC could help detect any unauthorized change in the system state.

GRR Rapid Response Project showed the ability to collect valuable forensics data from a number of clients, and to organize the collected data in a hierarchical versioned way to ease access and retrieval of information with specific time stamp and for a specific client when needed. This collection of data through the asynchronous server-client model enabled minimum utilization of resources on clients and on network. Although GRR is great tool for collecting forensics data from high number of clients, it still lacks the needed capabilities to perform analysis of the collected data, with the integration with analytics tools such as ElasticSearch and Google's BigQuery still on the development roadmap. This requires the use of external tools and scripts to make use of GRR's collected data, which was demonstrated in Experiment One through the use of Python. Two approaches were used in that section: one was adopting the automated hunting of threats using a Python script added to the cron tab, which used GRR APIs to check for triggered Netstat hunts, and the other was adopting the on-demand scan of clients which also used a Python script to export flows results and analyze them.

In the two experiments, the detection rate (periodicity) of GRR hunts was configured to be a number of minutes (5 to 10 minutes). This period leaves a window of 5-10 minutes for the attacker to perform the exploit and do the cleanup hiding any used IoCs. For critical systems, this period can be decreased noting that this will increase the size of data collected.

V. CONCLUSIONS

Threat hunting is an effective and critical security measure that is needed to minimize business impact of attacks, improving visibility of organizations' environments and weaknesses, and achieve proactive, accurate and early detection of threats. Proper and effective threat hunting requires a thorough study of the monitored systems and segments to define the normal behavior that will help filter out unexpected and malicious activities. This will help define IoCs for the systems, which can be monitored to trigger the needed action once met.

The threat hunting process requires the continuous monitoring of systems to help keep IoCs and thresholds of normal behavior updated and matching the changes in the monitored systems. GRR Rapid Response from Google provides a huge number of artifacts that it can collect from large number of clients in timely and distributed fashion, however, when it comes to exporting the collected data, the options available are still under development and might cause a bit of an obstacle for large scale threat hunting.

REFERENCES

[1] Macrae, A. (2013). Identifying threats in real time. Network Security, 2013(11), 5-8. doi:10.1016/s1353-4858(13)70119-3

[2] Lee R. & Lee R., The Who, What, Where, When, Why and How of Effective ... Retrieved August 29, 2016, from https://www.sans.org/reading-room/whitepapers/analyst/who-what-where-when-effective-threat-hunting-36785

[3] E. C.. Automating the Hunt for Hidden Threats (SANS Reading Room). Retrieved August 29, 2016, from https://www.sans.org/reading-room/whitepapers/analyst/automating-hunt-hidden-threats-36282

[4] Mawudor, B. G. (2013). Present Cyber Threat Management (Methodologies to Mitigate Evolving Cyber-Attacks). 2013 Eighth Asia Joint Conference on Information Security. doi:10.1109/asiajcis.2013.24

[5] Alonso, S. (2016). Cyber Threat Hunting (1): Intro. Retrieved August 29, 2016, from http://cyber-ir.com/2016/01/21/cyber-threat-hunting-1-intro

[6] Poputa-Clean, P., SANS - Automated Defense Using Threat Intelligence to ... Retrieved August 29, 2016, from https://www.sans.org/reading-room/whitepapers/threats/automated-defense-threat-intelligence-augment-35692

[7] G. C., & D. B.. Google/grr-doc. Retrieved August 29, 2016, from https://github.com/google/grr-doc/blob/master/faq.adoc

[8] Google/grr-doc. Retrieved August 29, 2016, from https://github.com/google/grr-doc

[9] Rekall at a glance. - Rekall Memory Forensic Framework. (n.d.). Retrieved August 29, 2016, from https://www.rekall-forensic.com/pages/at_a_glance.html

[10] Common Vulnerabilities and Exposures. Retrieved August 29, 2016, from http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-4730

[11] About the Metasploit Meterpreter. Retrieved August 29, 2016, from https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/

[12] Common Vulnerabilities and Exposures. Retrieved August 29, 2016, from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1240