

Global Information Assurance Certification Paper

Copyright SANS Institute Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering "Security Essentials: Network, Endpoint, and Cloud (Security 401)" at http://www.giac.org/registration/gsec

Shell Scripting for Reconnaissance and Incident Response

GIAC (GSEC) Gold Certification

Author: Mark Dalton Gray, markdaltongray@gmail.com Advisor: Mark Stingley Accepted: 01/15/2019

Abstract

It has been said that scripting is a process with three distinct phases that include: identification of a problem and solution, implementation, and maintenance. By applying an analytical mindset, anyone can create reusable scripts that are easily maintainable for the purpose of automating redundant and tedious tasks of a daily workflow. This paper serves as an introduction to the common structure and the various uses of shell scripts and methods for observing script execution, how shells operate, and how commands are found and executed. Additionally, this paper also covers how to apply functions, and control structure and variables to increase readability and maintainability of scripts. Best practices for system and network reconnaissance, as well as incident response, are provided; the examples of employment demonstrate the utilization of shell scripting as an alternative to applying similar functionality in more intricate programming languages.

1. Introduction

The Linux command-line interface (CLI) is one of the most powerful tools at the disposal of a security professional and yet, to many it remains a mystery and source of intimidation. For the uninitiated, this is reasonable due to the reliance on simple, intuitive graphical interfaces to which many users and professionals alike are accustomed. The CLI is just a black box with white text that is far less appealing and less forgiving. Fortuitously, learning the foundational usage of the CLI requires minimum time and effort. Not to mention, when necessary, the CLI serves to provide an eclectic range of versatile tools that are both powerful and inherent to the Linux operating system.

It is not uncommon for a security professional to be in a position where there is limited or no access to commercial security tools or a situation where they are unable to install a higher-level programming language. "No programming language is perfect. There is not even a single best language; there are only languages well suited or perhaps poorly suited for particular purposes" (Herbert Mayer, 1989). There are a multitude of instances where this predicament may present itself. For example, being in an environment where there is limited ability to install third-party software, budget constraints limit the purchasing power of commercial products, or on-site at a customer location where tools are absent. Establishing security framework or performing tasks to harden security is nearly impossible without proper tooling. When expensive commercials tools are not at your disposal, one must improvise, adapt, and overcome; a process that can be catalyzed with the aid of Linux command execution and scripting. "Shell scripting hearkens back to the classical Unix philosophy of breaking complex projects into simpler subtasks, of chaining together components and utilities" (Mendel Cooper, 2014).

Natively, Linux has an abundance of powerful commands that can be wielded for a spectrum of purposes. These commands can be harnessed and automated into reusable tools that can provide an immeasurable number of desired results. The tools are also automatable for the purpose of eliminating redundancy in realms such as lessening security voids. Foreseeably, commercial products will continue to reign supreme, but in situations that are constrained by the concern of organic functionality, consistency, and usability, shell scripting is vital.

Throughout this paper, bash will be the used and preferred shell. Please note that some of the sample implementation of bash scripting could be applied in a more efficient manner in the event of varying environments, user privilege, use case, etc.

2. What is a shell?

1.1. Overview

When operating on Linux (or Unix) machines, there are four common shell types that you will encounter: Shell Command Language(sh), Bourne Again Shell(bash), C Shell (csh), and KornShell(ksh). Without exploring the intricates of POSIX conformant and specifications, each shell is unique in the respect of feature sets and shell syntax. In most modern operating systems, "sh" is symbolically linked to bash (or dash) which results in its designation of the default shell type. In the event that a user wishes to modify that innate designation, Linux offers users the ability to install and make default "csh" or "ksh" shells.

1.2. Shell Basics

When operating a graphical user interface (GUI), a terminal emulator must be used to interact with the shell. There are many different emulators, and like each shell, they each offer a distinct set of features for the user. The default terminal emulator will be used throughout the paper for sample depictions.

2.1.1. Determining the Default Shell

Determining which shell is presented to the user upon logging into a machine via SSH or using a terminal emulator can be accomplished in multiple ways. A simple method for determining this is to investigate the "/etc/passwd" file; within this file each line represents a user.

The last field included in each of the lines defines which shell will be utilized as user logon actions are

hacks4snacks@athena:-\$ cat /etc/passwd | grep hacks4snacks
performed. hacks4snacks:x:1000:1000:Mark Gray,,,:/home/hacks4snacks:/bin/bash

Figure 1. Viewing the default shell by looking at the "/etc/passwd" file

The login program sets an environment variable called "SHELL" that can be queried in order to determine the default shell. As aforementioned, the results of the previous query should yield a result that matches the user's entry in the "/etc/passwd" file.

hacks4snacks@athena:-\$ echo \$SHELL /bin/bash

2.1.2. Configuring the Shell

Shells can be configured and customized with the utilization of "startup scripts." "Startup scripts" are files that define aliases, and source/set variables. Each shell has a fundamental set of "startup scripts" that it will execute. Of these "startup scripts", there are two broad categories that are referenced: "user" and "system" files. User files exist within a user's home directory and only affect shells owned/spawned by that user. The system files are typically located in the "/etc" directory and affect the shell of every user. When bash is invoked as a login shell, the system "/etc/profile", user specific "~/.bash profile", and "~/.profile" scripts are executed. If the shell is interactive, then subsequently, the user "~/.bashrc" file will be reviewed. Bash also comprises scripts that contain specific configurations for log-in and log-out procedures; these reside in the "~/.bash login" and "~/.bash logout" files.

1.3. Input and Output

Through the use of various commands and tools, there will be instances that necessitate the alternation of how input is supplied to programs. A common occurrence of this is providing the output of one program as input into another, which is often referred to a "piping" or "chaining" commands. Other instances can include using the contents of a file as input for a program, recording commands or saving outputs to a file.



2.1.3. File Descriptors

File descriptors are merely abstractions to resources on a host. On Linux systems, everything is treated as a file, which to a user or programmer translates to the existence of a standard set of functions that can be used to interact with resources on a host. Resources can include, but not limited to a file, socket, or device. To view file descriptors on Linux, the command "*lsof*" can be used.

File Descriptor	Name	Default "File"
0	Standard in (STDIN)	Keyboard
1	Standard out (STDOUT)	Terminal display
2	Standard error (STDERR)	Terminal display

Figure 5. Default file descriptor table

2.1.4. Chaining Commands

An additional form of redirection also exists within the shell known as chaining commands or piping commands. When chaining commands the first command will execute and generate output that is then used as input for the following command. Chaining commands are often useful when modifying command outputs and performing tasks that are dependent on the completion and output of another.



Figure 6. Piping the output of the cat command to serve and input for the grep command

3. Keeping it Organized

1.4. Variables

Variables are an important piece of scripting because it allows the temporary storage of information within the shell that can be referenced by other commands within the script. There are two types of variables: environmental and user. Environmental variables are used to track system-specific information such as the name of the user logged in, the search path used by the shell to find programs, and the name of the system. User variables are set by the user to reference information within a script or command. Variables can be a string of up to twenty letters, digits, or the underscore character. Additionally, within a shell script when using functions, there are another two types of variables. These Mark D. Gray, markdaltongray@gmail.com

are known as global and local. Global variables can be used anywhere within the script while local variables only exist within a function. Examples of these variables will be demonstrated in another section.

```
#!/bin/env bash
#Testing variables
days=2
man=mark
echo "The $man hasn't been outside in $days."
```

```
hacks4snacks@athena:-/dev$ echo $PATH
/home/hacks4snacks/dev:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin
```

Figure 7. Environment variable

1.5. Aliases

Aliases are an effective method for assigning a name to a collection of commands or just a single command. The ".bashrc" file, which is standard for the bash shell, includes some useful aliases by default. The general purpose of aliases is to make complex tasks more straightforward by invoking a simpler command that executes a number of complex commands.

```
# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval
"$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'
    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
    fi
# some more ls aliases
alias ll='ls -alF'
```

1.6. Control Structures

Control structures are used within a script to perform logical decisions in which a condition is tested, and depending on the result of the test, either performs a subsequent action or exists. The bash shell supports "if" and "switch" (case) conditional statements. Conditional statements work conjunctively with the decision control statements in order to determine whether or not the action will be executed.

3.1.1. IF Statement

The IF statement is one of the simplest decision statement expressions used in any scripting or programming language, although the syntax may vary. The typical formats of the IF statement include, if, elif, and else. This allows for multiple conditions to be tested within a single statement.

```
#!/bin/env bash
#set value of first argument
arg=$1
if [[ $arg == 1 ]]; then
    echo true
elif [[ $arg == 2 ]]; then
    echo false
else
else
fi
```

3.1.2. Switch (case) Statement

Switch (case) statements are much like the IF statements in the regard of logical decisions, however, switches are more commonly used to test multiple conditions that are determined by the value of a single variable. An example of this is the processing of command line arguments while using an IF statement will have the same outcome, it is much cleaner using a switch statement.

#!/bin/env bash

```
#set value of first argument
arg=$1
case $arg in
```

```
0)
    echo false
    ;;
1)
    echo true
    ;;
*)
    echo unknown
;;
```

esac

3.1.3. Loops

Loops with bash are extremely useful for running a series of commands continuously until a specific situation is reached; loops are often used for automating repetitive tasks. Bash supports multiple looping constructs which are until, while, and for. Typically, until and while loops are used when the amount of times the loop needs to run is unknown and for loops are used when the amount of iterations is known.

```
#!/bin/env bash
 #while loop
 arg=no
 while [ $arg != "yes" ]
do
     echo "are you ready to continue? yes or no"
     read arg
 done
 echo "Going to continue"
 #!/bin/env bash
 #until loop
 arg=no
 until [ $arg == "yes" ]
 do
     echo "are you ready to continue? yes or no"
 Mark D. Gray, markdaltongray@gmail.com
```

© 2019 The SANS Institute

read arg

```
done
echo "Going to continue"
```

```
#!/bin/env bash
#for loop
for num in {1..10};
do
        echo $num
done
```

1.7. Functions

Functions are a method of breaking a programming problem into smaller individual problems which facilitate the creation of reusable blocks of code that assist in making scripts more modular. In bash, once a function is created, it can be referred to anywhere else in the script without the need to rewrite it.

3.1.4. Creating a Function

There are three syntaxes that can be used to create functions in bash.

```
function reboot_message(){
    echo "Please reboot at your earliest convenience"
    return 0
}
function reboot_message {
    echo "Please reboot at your earliest convenience"
    return 0
}
reboot_message(){
    echo "Please reboot at your earliest convenience"
    return 0}
```

```
Mark D. Gray, markdaltongray@gmail.com
```

0

While all three are valid, the reboot_message() {} more closely resembles how functions are defined in other programming languages.

3.1.5. Using Functions

After a function has been created, it can be executed by simply referring to it. Important to note that a function must be created before it is able to be called upon, bash does not read the entire script then execute functions, scripts proceed linearly from top to bottom.

4. Reconnaissance

1.8. Network Reconnaissance

Now that the basic functionality of the bash shell has been covered, we can move on to using the shell and CLI utilities to collect information about different networks you may encounter. Tools used in this section include, but are not limited to, Nmap, Ndiff, Whois, and Dig. These utilities may or may not be installed on the system depending on the distribution of Linux being used.

4.1.1. NMAP Target Enumeration

NMAP, short for Network Mapper, is a powerful utility that can be used for vulnerability scanning and network discovery. NMAP provides a plethora of features and options that assist in numerous activities such as scanning a network, scanning a specific host for information such as open ports and OS type, and as mentioned vulnerability scanning. Due to the complexity of NMAP, only the common tasks and utilization are going to be covered.

The basic syntax for NMAP is provided below; the -sn argument tells NMAP to use the ICMP protocol to determine if hosts in the target range are reachable and it disables port scanning. The -v argument activates the verbose output and –reason will print information as to why it has determined a certain result about a host.

nmap -sn {OPTS} [host address | domain name | CIDR netmask | IP Range]

nmap -sn -v -reason 172.21.0.0/24

mgray@athena:-/dev\$ nmap -sn -v --reason 172.21.0.0/24
Starting Nmap 7.60 (https://nmap.org) at 2018-12-27 14:52 EST
Initiating Ping Scan at 14:52
Scanning 256 hosts [2 ports/host]
Completed Ping Scan at 14:52, 2.11s elapsed (256 total hosts)
Initiating Parallel DNS resolution of 256 hosts. at 14:52
Completed Parallel DNS resolution of 256 hosts. at 14:52, 0.07s elapsed
Nmap scan report for 172.21.0.0 [host down, received net-unreach]
Nmap scan report for _gateway (172.21.0.1)
Host is up, received syn-ack (0.0024s latency).

Common NMAP host discovery options include the following:

Discovery Option	Description
-PE	This tells Nmap to use ICMP
	echo requests, which is the
St	packet that's sent when you
S	ping a host.
-PP	This tells Nmap to use
Sh	timestamp requests. Hosts
	that respond to timestamp
	requests are usually
<u>9</u>	reported as findings in
0	penetration tests. Often,
	default and weakly
	configured cryptographic
	libraries use system time to
	generate the cryptographic
	primitives.
- PM	uses ICMP netmask requests;
	these ICMP packets were
	originally implemented so
	that network engineers could
	query a host for information

Mark D. Gray, markdaltongray@gmail.com

	about its network
	configuration.
-PS	TCP SYN flag scan: This
	option sends SYN packets to
	a host and determines
	whether they are actually on
	the network by interpreting
	the response or the lack
	thereof.
-PA	TCP ACK flag scan: This
	option tells Nmap to send
103	TCP ACK flags to the target
S	to determine whether it is
	alive and responding to
S	packets. Machines on a
	network will often try to
	strictly respect the TCP
2	protocol standard and
0	respond to packets with the
	ACK flag set by sending a
	packet with the REST packet.
-PO	IP protocol ping: This
	option enumerates the
	protocols supported by a
	target host, by listening
	for TCP packets with the
	REST flag set, since live
	hosts will often respond
	this way to invalid packets



1 3

```
with arbitrary identifiers set for the protocol number.
```

4.1.2. Arping for Host Discovery

Arping is a utility that facilitates the crafting of packets which include ICMP and ARP for the intent of sending them to random hosts on a local network. This makes for a simple method for enumerating live hosts. Below are some simple examples of *arping* command utilizations.

arping [IP Address]

arping -c3 172.21.0.1

<pre>mgray@athena:-/dev\$ arping -c3 172.21.0.1</pre>	
ARPING 172.21.0.1 from 172.21.0.35 ens33	
Unicast reply from 172.21.0.1	2.395ms
Unicast reply from 172.21.0.1	2.814ms
Unicast reply from 172.21.0.1	1.940ms
Sent 3 probes (1 broadcast(s))	
Received 3 response(s)	

In the output above, the *arping* command sent three packets to the 172.21.0.1 IP address and received three responses, which is a strong indication that this host exists and is active on the network. However, it is important to note that there is no guarantee that this information is accurate due to the utilization of insecure protocols.

Other standard options for *arping* include the following:

Command Option	Description
-c COUNT	This means only send COUNT
	number of requests.
-d	This finds duplicate
	replies. This option is
	great as a monitoring tool.
	It will be able to pick up
	if anyone on your network is
	spoofing the MAC address of

	another host; attackers
	often do this to initiate
	man-in-the-middle attacks.
-i	This is the interface. Don't
	try to autonomously find the
	interface; use the one
	supplied
-р	This turns on promiscuous
	mode for the specified
	interface and allows you to
All P	specify MAC addresses other
1103	than your own as the source,
Si	that is, MAC spoofing.
-r	This displays raw output and
5	means only the MAC and IP
A CO	addresses are displayed for
	each reply.

4.1.3. NDIFF for Identifying Network Changes

The Ndiff utility is simply a tool that is used to compare the results of NMAP scans. The Ndiff utility functions by reading two separate NMAP result files, compares the differences (if any), and then output the differences to a separate file. The can be helpful in identifying hosts that have been added or removed from the network.

#!/bin/env bash

TARGETS="172.21.0.1/24" OPTIONS="-v -T4 -F -sV" date=\$(date +%F)

Mark D. Gray, markdaltongray@gmail.com

1 5

```
cd /opt/nmap_diff
nmap $OPTIONS $TARGETS -oA scan-$date > /dev/null
    if [ -e scan-prev.xml ]; then
        ndiff scan-prev.xml scan-$date.xml > diff-$date
        echo "*** NDIFF RESULTS ***"
        cat diff-$date
        echo
        fi
echo "*** NMAP RESULTS ***"
cat scan-$date.nmap
        ln -sf scan-$date.xml scan-prev.xml
```

4.1.4. Dig for DNS Server Interrogating

DNS servers are entrusted to provide associations between IP addresses that are used by computers and the human-readable domain names It is not uncommon for organizations to use multiple domains and subdomains for a single IP address. So, what this means is that DNS potentially contain valuable information about an organization's public footprint and potentially expose an attack surface.

The Dig CLI tool is an all in one utility that can provide the essential need to know for a given domain or domains in relation to an IP address. The *dig* utility emulates browser and other network application queries when interacting with DNS servers. The command syntax for *dig* is similar to the *whois* utility:

dig [domain name]
dig microsoft.com

ngray@athena:-/dev\$ dig microsoft.com <>>> DiG 9.11.3-lubuntul.3-Ubuntu <<>> microsoft.com global options: +cmd ;; ;; Got answer: ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36870 ;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1 ;; OPT PSEUDOSECTION: EDNS: version: 0, flags:; udp: 65494 ;; QUESTION SECTION: ;microsoft.com. ΤN A ;; ANSWER SECTION: microsoft.com. 1097 A 40.112.72.205 40.113.200.201 microsoft.com. microsoft.com. 1097 microsoft.com. 1097 IN А 104.215.148.63 microsoft.com. 13.77.161.179 1097 IN

This is the output for the previous command, the IP addresses in the "ANSWER SECTION" are the IP addresses that belong to Microsoft.

The *dig* utility can also drill down on specific types of records, for example, *dig* can be used to only find mail exchange records (MX records).

dig microsoft.com MX

```
mgray@athena:-/dev$ dig microsoft.com MX
; <<>> DiG 9.11.3-1ubuntu1.3-Ubuntu <<>> microsoft.com MX
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 465
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;microsoft.com.
                                        МΧ
;; ANSWER SECTION:
                        676
microsoft.com.
                                IN
                                        МΧ
                                                 10 microsoft-com.mail.protecti
on.outlook.com.
```

Below are some common record types that the *dig* utility can lookup:

Record Type	Description
A	This is address record and
	holds the IPs associated
	with the queried domain.

Mark D. Gray, markdaltongray@gmail.com

This is the IP Version 6

	address record.
CNAME	This is the canonical name
	record, which will return
	the domain names for which
	the specified domain is a
	canonical record. This is
	like asking dig whether the
	supplied domain is a
	nickname for another, or
	more precisely, whether the
	given domain name uses the
	IP address of another
	domain, and dig returns
	these domains.
MX	This is the mail exchange
MX	This is the mail exchange record and lists the
MX	This is the mail exchange record and lists the addresses that are
MX	This is the mail exchange record and lists the addresses that are associated with the supplied
MX	This is the mail exchange record and lists the addresses that are associated with the supplied domain as message transfer
MX	This is the mail exchange record and lists the addresses that are associated with the supplied domain as message transfer agents. You would use this
MX	This is the mail exchange record and lists the addresses that are associated with the supplied domain as message transfer agents. You would use this to find the mail domains for
MX	This is the mail exchange record and lists the addresses that are associated with the supplied domain as message transfer agents. You would use this to find the mail domains for a given domain.
MX	This is the mail exchange record and lists the addresses that are associated with the supplied domain as message transfer agents. You would use this to find the mail domains for a given domain. This is for pointer records,
MX	This is the mail exchange record and lists the addresses that are associated with the supplied domain as message transfer agents. You would use this to find the mail domains for a given domain. This is for pointer records, which are often used in
MX PTR	This is the mail exchange record and lists the addresses that are associated with the supplied domain as message transfer agents. You would use this to find the mail domains for a given domain. This is for pointer records, which are often used in reverse DNS lookups.
MX PTR SOA	This is the mail exchange record and lists the addresses that are associated with the supplied domain as message transfer agents. You would use this to find the mail domains for a given domain. This is for pointer records, which are often used in reverse DNS lookups. This is the start of
MX PTR SOA	This is the mail exchange record and lists the addresses that are associated with the supplied domain as message transfer agents. You would use this to find the mail domains for a given domain. This is for pointer records, which are often used in reverse DNS lookups. This is the start of authority/zone record, which

Mark D. Gray, markdaltongray@gmail.com

AAAA

8

	to the primary domain server
	"authoritative" for the
	supplied domain.
	* O.
	00
AXFR	This is for the authority
	zone transfer, which asks a
	given domain name server to
	return all records related
	to a given domain. Modern
	DNS servers should not have
	this option enabled remotely
103	as it presents considerable
S	information about disclosure
	vulnerabilities, primary
Sr	internal address, and
0	enables denial of service
	attacks.

Another functionality of *dig* is the option to only return important data, this is accomplished by using the +*short* option. This will only return data that is important to the supplied request (such as IPs when querying a domain).

dig microsoft.com +short

```
mgray@athena:-/dev$ dig miscrosoft.com +short
104.215.95.187
52.164.206.56
```

When using this method, *dig* can be used with pipes and for loops to elevate the need to filter or parse out irrelevant information.

```
for ip in $(dig microsoft.com +short); do whois $ip; done
```

4.1.5. Whois Servers

Whois servers contain information about IP addresses, domain names, and other network addressing relevant information that certain organizations are responsible for or are strictly associated with. When querying a Whois server, a request for information is sent to the server using an application call Whois. Interrogating Whois servers from the CLI is executed by utilizing the *whois* command. The *whois* command offers numerous options that can be specified when using the tool. The basic functionality of Whois includes the return of a set of attributes associated with an IP address, the collection of attributes is called a "whois record". Looking up a record using an IP address is done with the following syntax:

whois [IP address]

An example of retrieving the Whois record for an OpenDNS server address:

whois 208.67.222.222

Mark D. Gray, markdaltongray@gmail.com

2 0

mgray@athena:-\$ whois 208.67.222.222		
# # ARIN WHOIS dat # available at: # # If vou see ina	ta and services are subject to the Terms of Use https://www.arin.net/whois_tou.html accuracies in the results, please report at	
<pre># https://www.ar</pre>	rin.net/resources/whois_reporting/index.html	
# # Copyright 1997 #	7-2018, American Registry for Internet Numbers, Ltd.	
NetRange: CIDR: NetName: NetHandle: Parent: NetType: OriginAS: Organization: RegDate: Updated: Ref:	208.67.216.0 - 208.67.223.255 208.67.216.0/21 OPENDNS-NET-1 NET-208-67-216-0-1 NET208 (NET-208-0-0-0-0) Direct Assignment AS36692 OpenDNS, LLC (OPEND-2) 2006-06-06 2012-03-02 https://rdap.arin.net/registry/ip/208.67.216.0	
OrgName: OrgId: Address: City: StateProv: PostalCode: Country: RegDate: Updated: Comment: Comment: Comment: Comment: Pef.	OpenDNS, LLC OPEND-2 145 Bluxome st. San Francisco CA 94107 US 2008-02-26 2018-10-11 http://www.opendns.com/ Use OpenDNS to make your Internet faster, safer, and smarter. DNS Servers: 208.67.222.222 208.67.220.220 IPv4/IPv6 Peering: peering@opendns.com bttps://rdap.arin_pet/registry/opEND-2	

This is the output of the whois command; it is an object consisting of multiple associated attributes that are in key-value pairs.

As previously mentioned, the *whois* tool offers numerous options. Aside from looking up information associated with an IP address, *whois* can be given other attribute parameters to lookup information associated with an organization, an email address, or the maintainer. This action is referred to as a reverse lookup. The following command is an example of this action:

whois -i [attribute name] [value]

whois -i mnt-by YAHOO-MNT

Additional attributes that can be used for inverse queries include the following:

Attribute	Description
admin-c	NIC-handle or person
person	NIC-handle or person
nsserver	Domain or address prefix or
	range or a simple address
sub-dom	Domain
upd-to	Email
local-as	Autonomous system number

Other usage includes looking up domain names, and by taking advantage of chaining commands only the useful information from the *whois*, object can be returned.

whois [domain name]

whois microsoft.com | grep "Name Server" | cut -d: -f2

The command above queries the Whois server for the domain name "microsoft.com" then feed the output to *grep* which looks for lines that match "Name Server", finally the output of *grep* is passed to the *cut* command that looks for ":" as a field delimiter that prints the second field, which in this case are the nameservers used for the domain "microsoft.com".

5. Linux Incident Response

Intrusion detection is a reactive measure that seeks to identify and mitigate ongoing attacks, while an incident response is an organized approach to addressing and managing a security breach or incident. The purpose of the next section is to break down what is known as "live forensic" actions. Using bash, common CLI tools are going to be used to collect user, system, and network information that could be used in an investigation.

Mark D. Gray, markdaltongray@gmail.com

2

1.9. User artifacts

User artifacts in this scenario consist of: logged in users, remote user logins, failed logins, local user accounts, local groups, sudo access, account UIDs, open files by user, orphan files, and potentially duplicated user IDs.

5.1.1. User Artifact Investigation

Viewing logged in users:

Command	Description
W	This command shows who is
	logged on and what they are
	doing.
lastlog	Reports the most recent
5	login of all users or of a
	given user
cat /etc/passwd	The passwd file is used to
C C	keep track of every
	registered user that has
	access to the system.
cat /etc/shadow	The shadow file is a system
2	file that stores user
	encrypted passwords.
cat /etc/group	The group file stores group
	information and defines user
	groups.
cat /etc/sudoers	The sudoers file contains
	information about user and
	group privileges.

egrep ":0+" /etc/passwd	Using regular expression				
	grep to find account with a				
	UID of zero in the				
	/etc/passwd file.				
lsof -u	List files opened by a user.				
cat /root/.bash_history	View the bash history of the				
	root user.				

5.1.2. Scripting User Artifact Collection

The commands above can easily be placed in a script that can be used in lieu of running each command individually. Now, a decision needs to be made on how the output should be formatted, should each command be in its own file? Should all outputs be in a single file, and if so, is there an efficient way to organize it?

For a single output file, a header can be created prior to adding additional command output. For example, the function below can be placed in a script and called to run prior to running a user artifact collection command.

```
header_split(){
    echo "------" >>
$OUTPUT
    echo "$@" >> $OUTPUT
    echo "$0UTPUT
    echo "------" >>
$OUTPUT
    }
```

In a script the "header_split" function would be executed as seen below (it is part of a larger function):

header split "Logged in Users"

\$W >> \$OUTPUT

```
Logged in Users

16:55:40 up 8:01, 1 user, load average: 0.38, 0.14, 0.04

USER TTY FROM LOGINE IDLE JCPU PCPU WHAT

mgray :0 :0 08:55 ?xdm? 1:57 0.00s /usr/lib/gdm3/gdm-x-session

on --session=ubuntu
```

```
© 2019 The SANS Institute
```

2 4

The output would be similar to the above, the write_header function creates the "Logged in User" header and below the \$W(variable for the w command) is ran beneath.

The entirety of the user artifact collection script can be found in appendix B.

1.10. Operating System artifacts

Operating system artifacts includes, but is not limited to filesystem information, scheduled jobs, determine system logging, uptime, disk usage and running processes.

Command	Description
lspci	List all PCI devices connected to
103	system.
lsb_release	Print distribution specific
	information.
uptime 🍼	Tell how long the system has been
NO NO	running.
df -h	Report file system disk space usage.
du -sh	Estimate file space usage.
cat /proc/cpuinfo	File that contains information about
	the CPUs on a system.
cat /proc/meminfo	File that contains information about
	the RAM on a system.
cat /proc/mounts	List of filesystems mounted to a
	system
cat /etc/fstab	System configuration file that
	contains information about major
	filesystem on the system.
dpkg -1	List Debian packages installed on
	system.

2 5

pstree -a	Display a	a tree	of	processes.

1.11. Network Activity

Network activity collection is vital to identifying the overall picture of a system and its health. The following commands will collect the essential network activity information necessary for subsequent analysis.

Command	Description		
iptables -L -n	Administrative tool for IPv4 packet		
	filtering and NAT.		
ip6tables -L -n	Administrative tool for IPv6 packet		
	filtering and NAT.		
route -n	Show the IP routing table.		
netstat -naovp	Print network connections, routing		
	tables, interface statistics,		
St	masquerade connections, and		
Sec. 1	multicast memberships		
arp -a	Show system ARP cache.		
ifconfig -a	Show all network interfaces on		
201	system.		
netstat -nap	Show listening ports.		
lsof -i	List processes listening on ports.		
lsof -nPi cut -f1 -d ''	List of open files, using the		
uniq tail -n +2	network.		

1.12. Finding Files

The following commands can be helpful in identifying when files were modified, who owns files, and files of interest.

Command	Description

2 6

find / -mtime -2 -ls	Find files modified
	in the last two days.
findtype -f -atime +30 -print	Find files older than
	30 days.
cp -R /var/log/* /media/logs	Copy system logs to
6	another medium.
lsattr -R / grep "\-i-"	Look for immutable
	files.
find / -xdev -type d \(-perm -0002 -a ! -	Look for world
perm -1000 \) -print	writable files.
find / -newermt 2018-12-17q	Look for files newer
	than specific date.
<pre>findtype f -size +100M -exec ls -lh {}</pre>	Find files over 100M
\;	and see what they
C. C.	are.

6. Conclusion

While the information covered in this paper hardly scratches the surface of the capabilities of bash, it should be an adequate start for creating Linux scripts that can assist in daily activates and provide some level of automation. Linux scripting, even basic scripts, can be used to create tools with ease and with the ubiquity of bash on Linux hosts, they are highly mobile. As previously stated, there are more advanced scripting languages available that can perform many of the same tasks and with better efficiency. The main takeaway from this paper is by leveraging shell scripting; simple tools can coalesce with other tools to create powerful scripts that can automate practically any task on a system that has bash.

References

- Kim, P. (2018). *The Hacker Playbook 3: Practical guide to penetration testing: Red Team edition*. Arlington (Virginia): Createspace.
- Murdoch, D. (2016). Blue team handbook: Incident response edition: A condensed field guide for the cyber security incident responder. United States: CreateSpace Independent Publishing.
- White, A., & Clark, B. (2017). BTFM: Blue team field manual. Columbia, SC: CreateSpace.
- Blum, R. H., & Bresnahan, C. (2015). *Linux Command Line and Shell Scripting Bible*. Somerset: Wiley.
- Taylor, D. (2017). Wicked cool shell scripts. San Francisco: No Starch Press.
- Clark, B. (2013). RTFM: Red team field manual. S.I.: S.n.
- Burtch, K. O. (2004). Linux shell scripting with Bash. Indianapolis, IN: Sams.
- Winterbottom, D. (n.d.). David Winterbottom. Retrieved from https://codeinthehole.com/

Bash (Unix shell). (2018, December 23). Retrieved from https://en.wikipedia.org/wiki/Bash (Unix shell)

- 2005, P. D., & Publication date: 17 Oct 2005 ripe database. (n.d.). RIPE Whois Database Query Reference Manual. Retrieved from https://www.ripe.net/publications/docs/ripe-358#a1
- Mayer, H. G. (1989). Advanced C programming on the IBM PC. Blue Ridge Summit, PA: Windcrest.
- Sanders, C., Smith, J., & Bianco, D. J. (2014). *Applied network security monitoring: Collection, detection, and analysis.* Waltham, MA: Syngress.
- Shell Script For Collecting Information on the Linux Network Configuration. (n.d.). Retrieved from https://bash.cyberciti.biz/networking/shell-script-to-find-linux-network-configurations/

7. Appendix A – Environment Used for this Paper

1.13. Operating System

Ubuntu 64-bit 18.04.1 (Desktop ISO) Hypervisor: VMware Fusion 11.0.2 Mark D. Gray, markdaltongray@gmail.com

Hostname: Athena User: mgray

1.14. Packages Removed from System

Aisleriot Solitaire Amazon Cheese GNOME majo GNOME Mines GNOME Sudoku Rhythmbox Shotwell Simple Scan Videos

1.15. Additional packages installed on System

Most recent Ubuntu updates VMWare Tools preload curl gnome-tweak-tool nmap vim sublime-text git wireshark tshark

8. Appendix B – User Report Script

#!/bin/env bash

LSPCI=/usr/bin/lspci

Mark D. Gray, markdaltongray@gmail.com

2 9

```
LSB=/usr/bin/lsb release
W=/usr/bin/w
LASTLOG=/usr/bin/lastlog
CAT=/bin/cat
EGREP=/bin/egrep
LSOF=/usr/bin/lsof
DATE=/bin/date
HOSTNAME=/bin/hostname
UNAME=/bin/uname
FAILLOG=/usr/bin/faillog
## files ##
PASSWD="/etc/passwd"
SUDOERS="/etc/sudoers"
SHADOW="/etc/shadow"
GROUP="/etc/group"
ROOTHIST="/root/.bash_history"
## Output file ##
OUTPUT="user.$(date +'%m-%d-%y').info.txt"
root check(){
   local meid=$(id -u)
   if [ $meid -ne 0 ]; then
       echo "You must run this tool as root or sudo."
       exit 1
   fi
}
header_split(){
   echo "_____
$0UTPUT
   echo "$@" >> $OUTPUT
                    echo "-----
$0UTPUT
   }
```

Mark D. Gray, markdaltongray@gmail.com

```
user_info(){
    echo "* Hostname: $(hostname)" >$0UTPUT
    echo "* Run date and time: $(date)" >>$OUTPU1
    header_split "Linux Distro"
    echo "Linux kernel: $(uname -mrs)" >>$0UTPUT
    $LSB -a >> $OUTPUT
    header_split "Logged in Users
    $W >> $OUTPUT
    header_split "Remote User Logins"
    $LASTLOG >> $OUTPUT
    header_split "Failed Logins"
    $FAILLOG -a >> $OUTPUT
    header split "Local User Accounts"
    $CAT $PASSWD >> $OUTPUT
    $CAT $SHADOW >> $OUTPUT
    header_split "Local Groups"
    $CAT $GROUP >> $OUTPUT
    header_split "Root Bash History"
    $CAT $ROOTHIST >> $OUTPUT
    echo "The User Report Info Written To $OUTPUT."
    }
root check
user_info
```

```
Mark D. Gray, markdaltongray@gmail.com
```

9. Appendix C – Operating System Report Script

#!/bin/env bash

```
LSPCI=/usr/bin/lspci
LSB=/usr/bin/lsb release
UPTIME=/usr/bin/uptime
DISK USAGE=/bin/df
HOME SPACE=/usr/bin/du
## files ##
CPU="/proc/cpuinfo"
MEMORY="/proc/meminfo"
MOUNTS="/proc/mounts"
FSTAB="/etc/fstab"
## Output file ##
OUTPUT="system.$(date +'%m-%d-%y').info.txt"
root check(){
   local meid=$(id -u)
   if [ $meid -ne 0 ]; then
       echo "You must run this tool as root or sudo."
      exit 1
    fi
header split(){
   echo "-----
$0UTPUT
   echo "$@" >> $OUTPUT
   echo "-----
$OUTPUT
   }
system info(){
   echo "* Hostname: $(hostname)" >$OUTPUT
Mark D. Gray, markdaltongray@gmail.com
```

```
echo "* Run date and time: $(date)" >>$OUTPUT
```

```
header_split "Linux Distro"
echo "Linux kernel: $(uname -mrs)" >>$0UTPUT
$LSB -a >> $0UTPUT
```

```
header_split "PCI Devices"
${LSPCI} -v >> $OUTPUT
```

```
header_split "Disk Space Output"
${DISK_USAGE} -h >> $OUTPUT
```

```
header_split "Home Space Output"
${HOME_SPACE} -sh /home/* >> $OUTPUT
```

```
header_split "Host Uptime"
$UPTIME >> $OUTPUT
```

```
header_split "CPU Info"
cat $CPU >> $OUTPUT
```

```
header_split "Memory Info"
cat $MEMORY >> $OUTPUT
```

```
header_split "Mounts"
cat $MOUNTS >> $OUTPUT
```

```
header_split "FSTAB"
cat $FSTAB >> $OUTPUT
```

```
header_split "Installed Packages"
dpkg -l >> $OUTPUT
echo "The System Report Info Written To $OUTPUT."
}
```

Mark D. Gray, markdaltongray@gmail.com

root_check
system_info

10. Appendix D – Network Activity Report Script

#!/bin/env bash

IP4FW=/sbin/iptables
IP6FW=/sbin/ip6tables
LSPCI=/usr/bin/lspci
ROUTE=/sbin/route
NETSTAT=/bin/netstat
LSB=/usr/bin/lsb_release
IFCFG=/sbin/ifconfig
ARP=/usr/sbin/arp

files

DNSCLIENT="/etc/resolv.conf" DRVCONF="/etc/modprobe.conf" NETALIASCFC="/etc/sysconfig/network-scripts/ifcfg-eth?-range?" NETCFC="/etc/sysconfig/network-scripts/ifcfg-eth?" NETSTATICROUTECFC="/etc/sysconfig/network-scripts/route-eth?" SYSCTL="/etc/sysctl.conf"

Output file

OUTPUT="network.\$(date +'%m-%d-%y').info.txt"

```
root_check(){
```

```
local meid=$(id -u)
if [ $meid -ne 0 ];
then
echo "You must be root user to run this tool"
exit 1
fi
```

Mark D. Gray, markdaltongray@gmail.com

4

```
}
header_split(){
   echo "-----
$OUTPUT
  echo "$@" >> $OUTPUT
   echo "-----
$0UTPUT
   }
network_info(){
   echo "* Hostname: $(hostname)" >$OUTPUT
   echo "* Run date and time: $(date)" >>$OUTPUT
   header_split "Linux Distro"
   echo "Linux kernel: $(uname -mrs)" >>$0UTPUT
    $LSB -a >> $OUTPUT
    header split "IFCONFIG Output"
    ${IFCFG} -a >> $OUTPUT
   header_split "Kernel Routing Table"
    ${ROUTE} -n >> $OUTPUT
   header_split "DNS Client $DNSCLIENT Configuration"
   [ -f $DNSCLIENT ] && cat $DNSCLIENT >> $OUTPUT || echo "Error
$DNSCLIENT file not found." >> $OUTPUT
    header_split "IP4 Firewall Configuration"
    $IP4FW -L -n >> $OUTPUT
    header_split "IP6 Firewall Configuration"
    $IP6FW -L -n >> $OUTPUT
```

```
header_split "Network Stats"
$NETSTAT -s >> $OUTPUT
```

```
header_split "ARP Cache"
$ARP -a >> $OUTPUT
```

```
header_split "Network Tweaks via $SYSCTL"
  [ -f $SYSCTL ] && cat $SYSCTL >> $OUTPUT || echo "Error $SYSCTL
not found." >>$OUTPUT
```

echo "The Network Configuration Info Written To \$OUTPUT."
}

root_check
network_info

Mark D. Gray, markdaltongray@gmail.com

11. Appendix E - DNS Scripts

```
NMAP Reverse DNS lookup
#!/bin/env bash
#NMAP reverse DNS lookup
nmap -R -sL -Pn -dns-servers 172.21.0.82 172.21.0.0/24 | awk
'{if(($1" "$2" "$3)=="Nmap scan report")print$5" "$6}'
| sed 's/(//g' | sed 's/)//g' > nmap_rdns.txt
Bash domain name resolution
#!/bin/env bash
echo "Enter class C Range: 172.21.0"
read range
for ip in {1..254..1};do
    host $range.$ip | grep "name pointer" | cut -d" " -f5
done
```

```
DNS Reverse Lookup
```

```
#!/bin/env bash
```

```
for ip in {1..254..1}; do dig -x 172.21.0.$ip | grep $ip >> dns.txt;
done;
```

Bulk DNS lookup
#!/bin/env bash
domains="microsoft.com
sans.org
google.com
gmail.com
bing.com

facebook.com

12. Appendix F – Network Analysis Scripts

Find live hosts with NMAP

#!/bin/env bash

```
nmap -sP -n -oX out.xml 172.21.0.0/24 | grep "Nmap" | grep -v "https"
| grep -v "addresses"
| cut -d" " -f5 > live_hosts && rm out.xml
```

Ping sweep with bash

```
#!/bin/env bash
read -p "Enter the first 24bits of the IP range e.g. 172.21.0 : "
subnet
```

```
alive_ping()
{
        ping -c 1 $1 > /dev/null
        [ $? -eq 0 ] && echo "Host with IP: $i is up."
}
for i in $subnet.{1..254..1}
do
        alive_ping $i >> live_hosts & disown
dence
```

done

Identify top talkers after set number of packets.

#!/bin/env bash

```
Mark D. Gray, markdaltongray@gmail.com
```

Shell Scripting for Reconnaissance and Intrusion Detection 3 s, i cut . 8 sudo tcpdump -nn -c 350 | awk '{print \$3}' | cut -d. -f1-4 sort -n