# *** Offensive Security Revised *** A Analytical Overview of Digital Crime, Data Extraction, and Effective Penetration Testing

***Written and Researched by Julian Perep***

*4/1/2012*

# Contents:

# **Part I:** Abstract

Cybercrime is growing at a rate beyond comprehension. There is an ongoing effort on the part of security professionals, major corporations, and the US government to effectively defend against cyber threats. However, the skill set of the opposition is quickly surpassing the progress of the aforementioned entities, and the resulting disparity is rapidly evolving into an epidemic. The evident failure of defensive tactics made way for a new approach, which entails simulating the offender's strategy, formally known as offensive security, or ethical hacking. My research proposes that offensive security should be implemented with greater regard to stealth, attributed to the fact that digital intruders typically do not make much noise upon network infiltration. I conducted my research through reading literature relevant to offensive security, including papers accredited by the IEEE and books written by the foremost experts on the topic. I also took a practical approach by implementing researched techniques on a dummy target in order to provide proof of concept. My findings revealed that network scanning is the most popularly employed technique by penetration testers when attempting to identify a target system's vulnerabilities. However, if executed recklessly, such scans may leave behind a wealth of digital evidence. Seasoned intruders are able to avoid scanning altogether, alternatively relying on thorough reconnaissance and their assumed knowledge of the target's infrastructure. Furthermore, it is imperative that attack simulations be geared towards accurately mimicking the intruder's process, rather than achieving the general end result paying little attention to the preceding groundwork. Ultimately, acquiring a better understanding of the criminal hacker's subtlety and stealth will in turn allow security professionals worldwide to develop tools and strategies which more effectively mitigate the success rates of network and system intrusions.

## **Part II:** A Brief Recap of Cybercrime Detection

Having nearly completed the cybercrime detection course, I've been empowered with an entirely new way of thinking in regard to investigating digital anomalies and the presence of malware. Should the situation ever arise where I'm assigned the rigorous task of assessing the intrusion of a live Windows or Linux machine, I will know the proper protocol to delineate the source of the problem. The use of any forensic tools should always be preceded by a logging of the date and start time of the investigation, as well as a full memory dump of the subject system. A key component of being able to extract accurate volatile data from a subject system is using binaries from a trustworthy source, rather than the statically compiled binaries on an infected victim machine.

The golden rule in volatile data extraction, and data extraction period, is that one should never trust the subject operating system. Suppose a Windows XP (SP 2) machine has been compromised by a rootkit, which replaced the XP desktop interface with one that mimics that of Windows 7; also, suppose that several of the tools native to the XP command prompt have been compromised by the rootkit. A novice investigator may be easily fooled and assume the operating system is Windows 7, and may verify this using the **ver** command on the machine's corrupted command prompt. The modified command may relay that the operating system is Windows 7, although it is most definitely XP underneath the Win7 GUI.

Although the preceding example is not as threatening to the success of an investigation as other scenarios, it surely demonstrates the importance of using trusted binaries. More importantly, when gathering specific information, such as open ports and current network connections, it's more effective to use multiple tools which extract the same type of data. One

tool may catch what another tool misses, as evident in the differences between **w** and **who** in the UNIX environment, or **quser** and **psloggedon** in the Windows world.

During live volatile data extraction, the purpose is to salvage as much information as possible from the DRAM, for the information stored in DRAM cannot be restored for further analysis following a system shutdown or reboot. Among essential volatile data is open ports, open files, network connections, logged on users, running processes, open applications, command history (more common in responses to compromised UNIX/Linux machines), and quite possibly web browsing history. A digital analyst should acquire information regarding the previously mentioned categories during multiple stages of the live response. A seasoned analyst may even have automated scripts do the work while he or she manually assesses the state of the subject system. Upon completion of the live response, the investigator may proceed to tying everything together and identifying the structure and overall effect of the malware.

Once volatile data extraction has been completed, it's on to the static analysis of the memory dump. The beauty of conducting a memory dump prior to live analysis is evident in the untainted body of the system's data, just as it was at the peak of its corruption and before the investigator leaves digital footprints from the execution of various commands. Among the most important data contained in a memory dump is the information about processes that were running during the start of the live response. Malware, more often than not, cannot compromise a target without the help of a running process, whether malicious code latches onto a currently existing process, or whether the process itself is a malicious executable.

Suspicious processes are typically identified by external information, such as the user which spawned the process, the relevant files, if any, the initial execution time, and any child

processes which the parent may have spawned. By acquiring the username linked to the process, investigative resources may be reallocated toward learning more about a particular user. A skilled digital forensics analyst may ask the question, was the user even logged on at the time the process began executing? And if they were, what other processes was the user running, and did he or she have any open files? Any network connections associated with the process must also be investigated, as well as what modules and services were linked to the process. If any atypical services are identified, the logical action is to determine what ports they map to. Thorough analysis of open ports may help identify the point of intrusion. In the case of relevant modules, the location of the module must be found such that the source code may be analyzed for any recent modifications. The list of useful information from a memory dump is quite extensive, but I'll have to cut it short for brevity.

File analysis plays a major role in determining how the malicious code was transmitted to the target. Malicious code does not magically appear on a machine, for it usually has to be transmitted to the machine as a file. It may be a file downloaded from a web browser, an email attachment, or a shared file over the network. An important component of static file analysis, file descriptors are essentially indexes for entries in a kernel-resident data structure containing detailed information regarding an open file. Analysis of descriptors may lead a seasoned investigator to delve further into an unfamiliar process associated with a file. Perhaps a service which mapped to port 6485 was running under the questionable process. A deeper look at the nature of the service may reveal several discrepancies and atypical actions on behalf of the service. Consider an instance where the Windows svchost.exe may have suspicious characteristics. Files are a great source of information, whether they are the means by which the malicious code was implanted in the machine, or if they are legitimate files affected by the

infection. The beauty of file analysis is that it serves as another investigative gateway in case the initial process analysis leads to a dead end. Process dumps may not always disclose information about mysterious related files, thus it may be advantageous to work backwards, starting with the static file analysis and leading to related processes.

The Windows registry, a hierarchical database which includes all the Windows configurations and settings, may also be of use in determining the presence of malware on a system. It's essentially the equivalent of the UNIX directory tree structure, with the exception that UNIX-based operating systems are completely open source. The Windows registry proves most useful in acquiring information related to users, such as the contents of the NTUSER.DAT file. Suppose the user associated with the execution of virus code was logged on to the system at the execution start time, and prior investigative efforts revealed that this user was disgruntled and unhappy with their employer. The investigator may then direct his or her efforts to learn as much about the user as possible, including user permissions, policies, account information, account activity, logon times, and other process the user invoked. User information extraction is among a myriad of categories where the registry may be of assistance in malware forensics.

The use of scripts can be more than helpful when conducting a live system investigation. A script essentially automates command execution and retrieves the most valuable information with great haste, in comparison to typing each of the commands one by one, which can be a very strenuous and time consuming process. Each command is usually enabled with specific flags conducive to the exact purpose of the command. For example, **ps** relays a raw list of running processes paired with their corresponding process PIDs, whereas **ps –aux** involves a much more detailed process description, including related user information, which is undoubtedly of more importance with respect to malware forensics. The state of a victim machine constantly changes,

thus it is of great necessity to capture state at multiple stages of the live response, which is where scripts come as a golden resource. The outputs of the scripts are typically redirected to files, which the investigator resorts to during static analysis of the target, of course succeeding the live response. Once the appropriate files are ready for analysis, the investigator may analyze the files individually, scoping out any potential anomalies; furthermore, it may be beneficial to conduct a differential analysis of the relevant files, using the **diff** command, such that the change in the state of the system over the course of the live response may be taken into consideration as well. It is important to note that the **diff** command should most always be enabled with the **–iw** parameter; this will allow for execution of the command with no regard to spacing or character case, thus filtering out several redundancies. Batch scripts are often implemented during live responses to Windows machines, whereas Perl and Python scripts are more popular in the UNIX/Linux world.

Before concluding the general recap, it is important to note that the investigator must always assume the worst case scenario. Assume that the victim system has been hit hard by a rootkit that dug its way deep into the kernel, such that nature of the system calls has been modified. A rootkit's presence on the system is typically stationery, thus the information related to the rootkit's infestation may not be as volatile as that of a response involving the one-and-done execution of malicious code. For example, assume the download of an untrusted PDF file from a web browser spawned a process that deletes a system's volumes and corrupts the partitions. It is of extreme essence to collect as much information as possible before the goal of the malicious process is achieved and any live trace of the process is irretrievable. The investigative efforts may not be as successful if all that's left to analyze are the remnants of the process, or in this case, nothing but an empty hard drive. Even more troubling, the process may

be configured to not only destroy the local drive, but any mounted drives as well, which may put the investigator's mounted devices at risk. In the case of rootkits, a quick full memory dump of the system may end up being just as useful as any intelligence gathered from live response. However, assuming the worst case scenario, a skilled designer of rootkits may have configured the rootkit to relinquish its hold on the victim system upon detection, for fear of being traced to the source IP. Some rootkits may also be paired with time bombs that spawn a process similar to the one spawned by the aforementioned PDF attachment.

Having learned the specifics of cybercrime forensics and investigative procedures, I've worked hard to familiarize myself even further with different tools, their advantages, and their caveats. I will now move onto a discussion of volatile data extraction from a live Linux system.

## **Part III:** An Overview of Volatile Data Extraction from a Live Linux System

The importance of effectively extracting volatile data from a live Linux system cannot be understated. Volatility, in this sense, can be held in the same context as fragility. Suppose a freight carrier is transporting fragile valuables made of glass and one of the workers drops one of the concealed packages causing the item to break. It's possible that the item in question may be repaired, but the item may never again be analyzed in its original state. There's always going to be the shards of glass that got away. The same concept applies to volatile data. Once the data is "dropped", it may be very difficult, and in most cases, impossible to recover. Whether working with a Windows or Linux system, volatile system data carries the most weight in a digital forensics investigation.

Contrary to the Windows hosts of operating systems, Linux is open source and comes with a much wider variety of useful native tools. Furthermore, Windows stores the majority of its configurations and settings in a hierarchical database known as the Windows registry, whereas Linux stores such information in a tree of directories stemming from the root. Anyone that's used Windows for the better portion of their life and successfully transitioned to Linux should undoubtedly be able to appreciate the simplicity and ingenuity of its design, as well as the power of the tools included in the standard UNIX /bin/ (binaries) sub-root directory. The well-structured design of the Linux operating system allows for smooth exfiltration of volatile data, for at times the same process on a live Windows system ends up being twice as rigorous.

As is the protocol with any live response, the first actions of the responder should be a quick logging of the time and date via the use of a trusted binary, succeeded by a full memory dump of the subject system. In Linux, this is accomplished with the **dd** command. Windows

platforms typically require special, non-native tools to accomplish the same task, reinforcing the overall simplicity of the Linux OS. After a trusted suite of binaries, such as Helix or Volatility, is mounted onto the subject Linux system, it's important to obtain as much intelligence as possible regarding all running processes. The **pcat** tool, part of the Helix suite, is unique in gathering the memory contents of running processes, for it does not interrupt the process itself. This is highly advantageous with consideration to the state of the system.

Tools that are used to extract volatile data from a Windows OS typically aren't well-known by the average Windows user, whereas the corresponding Linux tools are likely more than familiar to even a novice Linux user. Linux data extraction does not often employ a GUI with its native tools, for Linux is mainly driven by tools that only support CLI. By opening up a trusted shell from the forensic toolkit, the examiner can then refer back to the basic commands of any Linux OS to retrieve as much information as possible about the subject system. The **hostname** command reveals the name with which the system uses to communicate with other systems, and the **ifconfig** command paired with the **a** flag relays important information regarding the machine's network settings, including whether or not the local network sniffer is set to promiscuous mode.

Analysis of the system's users should follow the acquisition of the basic system data. **W** and **who** are commonly known commands in the UNIX environment, both of which play a major role in identifying users on a live Linux system. Perhaps one of the users listed in the output of the preceding commands could be tied to a malicious runaway process identified in the process analysis portion of the live response. Upon identification of the users logged onto the subject system, it's important that the network statistics be analyzed with great haste, and by multiple same-purpose binaries for later cross-referencing. For example, the **route** command, as well as

the **netstat** command with the **r** and **n** flags enabled display the routing table for the subject system, but it is highly encouraged to verify/cross-examine the output of both commands. Network connections can be retrieved via the **ss** command line utility. Analysis of the arp cache, generated by the **arp** command with the **a** flag enabled allows the investigator to view the MAC address of any machines that the subject system may have communicated with on the subnet. Should the contents of the arp cache go unseen, the potential success of the investigation consequently suffers a potentially huge blow. Consider the arp table as one of those shards of glass that should never get away.

Once most of the basic and general information is retrieved from the system, it's important to piece together the data acquired from each category and paint a digital picture of how the malware is operating on the system. Tying it all together begins with a look back at the processes in the context of associated users and open files. Perhaps one of the suspicious processes was spawned by a currently logged on user, and the user has several files open that map to ports which are linked to suspicious services. Nefarious activity such as the afore mentioned instance can be spotted with the help of utilities such as **lsof**, also native to most Linux systems. With no parameters enabled, this tool simply lists all the open files on the system, a list which is typically quite extensive. However, there are of course multiple parameters available to condense the list to open files associated with a particular process or port, which in turn allows for further analysis of the accomplices (the ports, their services, and other processes). Not even criminal hackers underestimate the power of **lsof**.

The open-source structure of the Linux OS makes it moderately less difficult to identify a potential rootkit infestation, given that the investigator knows exactly where to look. The ambition of any diligent designer of rootkits is to hit home, or in other words, hit the Linux

kernel, the almighty interface/mediator between the software and hardware components. The keen forensic examiner would of course examine the currently loaded modules via the **lsmod** command, which would bring to light any modules that may have been compromised. Rootkits are surreptitious in nature largely in part because they do not generate any sizeable processes or files, but rather replace/modify existing components of an OS. Rootkits are essentially revered as the most powerful backdoors in the hacking community.

An acquisition of the command history via the .bash_history file is also considered important volatile information, although slightly lower in priority level. This comes in handy when the malware in question allowed the perpetrator remote and privileged access to the subject system, wherein the hacker used the system's terminal as if he or she was the user at the desk. Also important in identifying anomalous system behavior is the contents of the /var/spool/cron/ and /etc/cron.daily files/directories, wherein any scheduled system tasks are logged. This is advantageous in the case where the malware in question simply plants malicious code to execute at specifically designated times.

As mentioned in Part II, the use of automated scripts can be quite advantageous with regard to acquiring as much information possible in as little time as possible. The Perl programming language, most popular among system administrators, comes equipped with a large number of modules and built-in functions that may help investigators of malware infestations automate the recording of sensitive information. The vast number of modules related to network programming may assist in the creation of a useful script to monitor ongoing network activity and capture network traffic, additionally writing the captured information to an encrypted file on the investigator's mounted device. To provide some context to this, assume that an attacker has successfully executed a denial of service attack on all the machines connected to a dummy

switch, wherein the attacker flooded the switch resulting in reduced speed of data transfer over the network. Furthermore, assume that the DOS attack was a scapegoat for an ulterior attack specifically intended for just one of the machines connected to the switch. Upon responding to the infection of the target machine, the investigator may want to continually monitor the ever-changing state of the network to which the target is connected. Perhaps, the later analysis of the file containing all the network activity reveals a remote IP probing the subject switch with absurdly large packets, which would have likely caused the switch to function like a hub, broadcasting all network traffic publicly over the subnet. The attacker may have planned his or her attack such that the flooding was done in five minute intervals. While the investigator proceeds with his or her live response, the continuously running Perl script simultaneously captures other volatile information. If the investigator conducted static TCP dumps, he or she may have missed the sequences during which the attacker swarmed the system with oversized packets. Network traffic can be considered a pretty indispensable shard of glass.

In addition to Perl, and other scripting languages such as BASH and Python, SED and AWK may be of quite some use during the post-live response static analysis portion of the investigation. SED and AWK are characterized as stream editors, wherein the use of regular expressions is implemented to filter out all the content of a stream that doesn't match the patterns specified in the body of the program. Both SED and AWK automatically iterate through a file, or whichever alternative stream is specified within the header of the program, and relays any information that an investigator instructed the program to extract. Where SED is used primarily as a stream editor and nothing else, AWK is utilized as more of an organizer of information. AWK manipulates data with unparalleled ease, and may come in handy when an investigator wishes to organize the data they've previously collected. The use of SED typically precedes any

potential use of AWK. SED may be of most use when analyzing a subject system's log files, such as cron.daily. A keen investigator may already know what they're looking for and may want to go for the quick find. The investigator may conjure up a quick SED program to extract services running on a very specific range of ports, or perhaps a pattern to gather usernames and/or password hashes in the shadow file. If the investigator does not stumble upon the intended discoveries, then it's on to the rigorous task of sorting through the rest of the file contents manually. A useful AWK program may create an alphabetical word index, which specifies the line number on which each word occurs. Although it may sound complex in nature, such a program may range between ten to fifteen lines of actual code, depending on the efficiency of the individual coding the program. The caliber of Perl, Python, SED, AWK, and BASH should never be underestimated.

There are a myriad of data categories which each play a role in piecing together the picture of the malware present on the subject system. Too much attention to detail may allow even the keenest of investigators to neglect the obvious. Hardware, mounts, and shares on the subject Linux system all are just as volatile, but not as prevalently examined as the obvious network statistics and running process information. Extraction of fragile data from a Linux OS is comparable to a lawyer trying a case he or she should easily win. The facts are there, the pieces of the puzzle are all around, and not much is hidden, but it's easy to lack attention where it is most needed. That is the beauty of volatile data extraction from Linux. There is so much to be seen, and yet the most important pieces are hidden away in the deepest bowels of the root subdirectories. It's one massive scavenger hunt. I will now move on to the bulk of the paper, which encompasses offensive security with regard to the criminal process.

## **Part IV:** An Analysis of Effective Penetration Testing with Regard to Stealth

Criminal hackers pose an imminent threat to information security, spanning from small scale denial of service attacks to large scale infiltrations of corporate networks. The goal of the criminal hacker is to ultimately gain administrative access to the target, wherein the access if manifested to accomplish any number of nefarious tasks. Combating this threat has quickly moved up the priority list of major corporations and the US Government. Common defense mechanisms such as state of the art firewalls, intrusion detection systems, and advanced password protection schemes lack effectiveness in foiling the truly skilled attackers, for such mechanisms are a mere speed bump en route to the inevitable compromise of the target system's integrity. Ethical hacking entails a more proactive approach to combating infiltrators. It is essentially a simulation of the offender's tactics when attempting to gain unauthorized entry into a system (Bryan Smith). Even the most skilled hackers must rely on weaknesses in the target, thus the main goal of an ethical hack is to identify said weaknesses and configure patches.

Although an improvement from passive security, ethical hacking is vexed by one fatal flaw. Ethical hacking is characterized by consent of the employing entity allowing the ethical hacker to conduct a penetration test (Palmer), thus the ethical hacker typically does not worry about being detected when executing their infiltration of the target. Contrary to ethical hackers, criminal hackers implement special tactics to avoid making noise, such that they may not be easily detected over the network. I believe that ethical hackers should tone down the messy brute force approach and conduct their tests as a criminal hacker would – with great regard to stealth and minimizing the amount of digital evidence left behind. This revised approach may reveal the less obvious vulnerabilities emboweled deep within the network system, or perhaps an obvious vulnerability in the company's public web page. I will now discuss an overview of the

penetration testing process, followed by the practical methods involved in accomplishing more surreptitious system intrusions.

Offensive security encompasses four phases: reconnaissance, scanning, exploitation, and maintaining access (Engebretson). The reconnaissance phase is the groundwork for a successful hack, gathering the gritty details of the decided target. A successful reconnaissance phase typically results in the acquisition of an extensive list of IP addresses, such as the target's name servers, web servers, mail servers, and the jackpot NFS (Network File System) server, wherein the ethical hacker chooses a target IP suitable to the nature of the intended attack (Engebretson). The scanning portion of offensive security revolves around the utilization of various tools to diagnose the personality of the target, for which the main purpose is to determine vulnerabilities and potential access points. The exploitation phase is the actual attack against the vulnerabilities identified during the scanning phase. The exploitation is delivered in the form of a payload. A payload may bind itself to the target, meaning that the attacker invokes a listening port on the target system through which the payload will be delivered; however, a payload may also be reversed, such that the attacker poses as somewhat of a client, where the target adapts the personality of a server – the attacker establishes a listening port on their machine and then waits for the target to connect to the attacker's listening port per request. The maintaining access phase entails the creation of tunnels and/or backdoors, such that the intruder may return to the target at his or her leisure. The ethical hacker does not actually do any damage, unless encouraged to do so, during the last two phases of a penetration test, for the sole purpose is merely to provide proof of concept. The next few paragraphs will focus on the tradeoff of excessive scanning.

The scanning of a target system includes two types of scans: vulnerability scans and network scans (Engebretson). Members of the hacking community may argue that they are

synonymous terms; but the fact of the matter is that network scanning is more of a general analysis of the network system which the target IP is connected to, whereas vulnerability scanning is a more comprehensive and detailed analysis of the target's possible weaknesses, such as poorly configured services running on open ports. The network scanning typically precedes the vulnerability scan (Engebretson).

When attempting to acquire a network map of the target, the first tool that should come to mind is Nmap, which fittingly stands for network map. Nmap is a command line utility native to most Linux systems, which takes an IP address as a command line argument alongside specific flags which in turn characterize the nature of the scan (Lyon). For example, a TCP port scan is indicated by the **-sT** flag, and a SYN scan is indicated by the **-sS** flag (Lyon). Nmap, although revered as the most powerful network scanning utility, is not the only freely available network scanner. Several other tools such as NetSonar, SATAN, and Trinux perform the same actions as Nmap (Bryan Smith); it's safe to assume most scanning methods (TCP, SYN, UDP, etc.) can be held within the context of all the previously mentioned network scanning utilities. However, examples throughout the continuation of the paper, per relevance to network scanning, will be demonstrated with regard to the specifics of Nmap.

Nmap is a powerful utility from the perspective of the ethical hacker in that it allows the ethical hacker to see what is visible to the outside world (Bryan Smith). However, from the perspective of the intruder, Nmap is merely a noisemaker if misused, for it establishes a blatant presence on the target system and may be easily detected. Nmap relies on connecting to the target host, thus relinquishing all hopes of remaining completely undetected since the target will recognize the probes sent from the attacker's IP. In reference to network connections, Locard's

exchange principle states that any peer to peer communication, such as the case with Nmap, will result in digital residuals transferred between the peers, thus amassing digital evidence (Carvey).

Although the evidence is left on the target, this does not necessarily imply that the system administrator of the target system will be able to quickly detect the scan, nor have the tools to detect the scan readily available. In the instance of TCP port scans, the system administrator may use the native **tcpdump** tool to capture network traffic over the transmission control protocol. A TCP port scan ($ **nmap –sT –P** [IP]) conducted from a remote IP may be made visible via the command **tcpdump –n tcp**.

Criminal hackers know better than to blatantly probe the target without an invisibility cloak, even when operating from a remote zombie machine. If a criminal hacker were to use Nmap, they'd likely choose the alternative SYN scan. The reason behind this is because the speed of a SYN scan shatters the speed of any other type of scan (Engebretson). The SYN scan will complete what is known as a two-way handshake, wherein the attacker's machine sends an SYN packet to the target and the target replies with an SYN/ACK packet; however, a full-blown TCP port scan conducts a three-way handshake, wherein the attacker replies once more with an ACK packet, thus establishing a full temporary connection (Engebretson). Although the SYN scan is much faster than the aforementioned TCP scan, the stealth with which it is executed is not a significant upgrade. An execution of **tcpdump** enabled with specific parameters will still be able to capture network traffic produced by an SYN scan. For example, **tcpdump -nnvv -i br0 'tcp[tcp-syn] & (tcp-syn)' != 0** run on the command line is configured to detect the two-way handshake. The capture of the network traffic may also be configured to keep a lookout for a specific protocol in particular, such as UDP (user datagram protocol). This can be done by

specifying the acronym of the protocol as the last command line argument, preceded by two backslashes with no space separation (i.e. $ tcpdump … \\udp).

Where **tcpdump** may lack, another network packet sniffer known as Wireshark may be able to overcompensate, and vice versa. Wireshark is a network sniffer, which comes native to the Backtrack Linux operating system (Ubuntu-based). It is a GUI driver utility, which supports the capture of network traffic over almost all protocols (Ranganath). PortSentry, which comes native to several Redhat Linux distributions (but special dependencies may be installed such that it may operate on a variety of Linux distributions), is very similar to Wireshark and TCPDUMP. It can be configured to route port scans to a dead host such that the target system will appear to be non-existent, and the attacking IP address may be added to the hosts.deny file in the /etc/ directory, meaning that any further attempt by the attacker to remotely connect to the target will be denied and logged (Mourani). In this aspect, PortSentry may be considered among the elite tools used to evade port scans altogether, since it may permanently ban the attacking IP from gaining access to the system. Upon implementation of PortSentry, configured with the blacklisting option, a pop-up will appear on the target machine which prompts the user to choose whether they'd like to add the IP to etc/hosts.deny or whitelist the IP. Upon blacklisting, the remote IP will not be able to detect any open ports on the target nor perform the **ping** command to diagnose the target's network connectivity.

In addition to the previously mentioned methods of port scan detections, ChaosTables is software that may be installed on any Linux distribution, given that the proper dependencies are also installed. In relation to a firewall, one may think of ChaosTables as a firewall log, which uses configuration files latched onto a process running in the background (Engelhardt). The process is a somewhat similar to the **tcpdump** command, but is more of a custom made version.

The process refers to the configuration files, which contain “tables”, or rows, of instructions. An example of an instruction to match any remote scans which pose as normal TCP scans would look like this:

**-p tcp ! --syn -m conntrack --ctstate INVALID.**

It would be of no use to have a log of only one type of detected scan. ChaosTables can be manipulated to capture any number of scans, such as an SYN scan, as is the case with the following instruction:

**-A INPUT -p tcp -m portscan --synscan -j LOG --log-prefix "[SYNSCAN] ";.**

It may take an experienced network administrator to understand the set of rules that must be abided by when dealing with ChaosTables. Obviously the unusual syntax is a major caveat of the software, which is a major reason why ChaosTables has been outdated and surpassed by firewalls, which employ similar techniques and configurations. ChaosTables doesn’t offer anything in addition to a simple notice of a port scan’s occurrence on a target machine.

Vulnerability scanning, although different from network scanning by definition, employs tactics quite similar to a network scan. Consider network scanning in relation to spying on an individual from afar, attempting to gage the individual’s personality type. Using that context, consider the vulnerability scan as the spy blatantly going to the individual and inquiring about his or her weaknesses, as well as how to expose them. A keen criminal hacker would wisely disregard such a blatant approach, for he or she would use their knowledge of the situation in order to determine what may or may not be vulnerable. For example, a criminal hacker’s intended target may be a web server. From this fact, the criminal and ethical hacker alike may make the assumption that port 80 is open and is likely running an http service. Additionally,

suppose that another target machine is a mail server. Once more, the criminal and ethical hacker alike may safely assume that an SMTP daemon/service is running on one of the open ports. I will discuss more involving web servers and mail servers in the paragraphs encompassing the reconnaissance phase of a penetration test.

Ethical hackers are always encouraged to use network scanning to scope out what attributes of the system are subject to easy discovery, but in order to more accurately emulate an offender's groundwork, greater respect must be paid to the reconnaissance phase. This phase is most difficult to imitate, granted that the ethical hacker is privy to information about the target which is not typically made public, let alone available to a criminal hacker. In some cases, a major corporation's greatest vulnerabilities may not have anything to do with the system's infrastructure or anything network-related for that matter. For example, suppose a company such as Fudge Inc. has a web site which contains information about the company history, and this web site provides email contact information in case a potential customer has any inquiries. A criminal hacker may learn a lot from such an email address. Assume the email address is in the form of jsmith@fudge.com , where jsmith is the owner of the email account. Perhaps jsmith is also the username with which the employee logs on to the system. This means that the criminal hacker may conduct a brute force password attack in an attempt to gain unauthorized entry into the system. One of the most powerful brute force attack tools is John the Ripper. John the Ripper is a password cracker which uses a word dictionary, usually a text file(s) created by the attacker or user of the tool, to manufacture as many possible alphanumerical password combinations until the password is unlocked or the possibilities are exhausted (Engebretson). An important caveat to note is that John the Ripper is only as good as the password dictionary file it utilizes. A keen criminal hacker will redirect multiple extensive password dictionary files to John the Ripper.

The beauty of brute force password attacks is that they are virtually undetectable right off the bat. Some systems enable defense mechanisms that bar users from logging onto the system after x number of failed logins; nonetheless, there are a large number of companies that neglect to employ such tactics. Unsuccessful brute force password attacks are not unfamiliar to criminal and ethical hackers alike. Alternatively, there are plenty more paths to a successful hack. Referring back to the previous example involving the email address acquired from Fudge Inc., the criminal hacker may send an email with a malicious attachment from a dummy gmail or yahoo account such that the target's mail scanner would detect the malicious attachment, prevent it from reaching its target address, and spit back an email reply regarding the details of the email's disposal. Such information may include any antivirus software that the mail server uses, mail virus scanners, email header information, and in some cases the name of the mail server, as well as the names of any daemons and protocols which the mail server uses (Engebretson).

The ethical hacker should always begin their penetration test pretending they know next to nothing about the target. Ethical hackers may be eager to show off their technical savvy in how quickly they're able to commandeer a system, but they must keep the main goal in mind at all times. The real challenge for the ethical hacker is understanding that the reconnaissance phase is the only phase that is virtually impossible to detect, thus criminal hackers strive to obtain as much information as possible from this phase before they begin their actual pursuit of administrative access to the target system. Any good spy will want to learn as much about their target before getting to the dirty work, for the same concept applies to hacking. The idea of effective ethical hacking is to remain as silent as possible, and there is nothing more silent than harmless and likely undetectable research about the target. As per the slogan of the Backtrack Linux OS, "the quieter you become, the more you are able to hear" (Engebretson).

Several command line utilities native to most UNIX based operating such as **dig**, **nslookup**, **whois**, and **host** are essential to a successful reconnaissance phase. The **whois** utility takes a domain name as a command line argument and returns IP addresses and hostnames of a company's domain name system. The **host** command relays the IP address of the command line argument if the argument is a domain name, and conversely returns a domain name if the argument is an IP address. The **nslookup** tool returns the host names of name servers, as well as their corresponding IPs. Additionally, **dig** provides similar functionality to that of **nslookup**. Implementation of the aforementioned utilities does not necessarily indicate the likelihood of an attack from the perspective of the target. Most public domains blatantly allow the discovery of internal IPs by allowing DNS zone transfers. In the database of a company's DNS data, the "zone" is defined as the portion that is replicated and relayed to the client requesting the information via implementation of **nslookup** or **dig**. Suppose the output of the above commands is very vague and the attacker desires a little more knowledge, perhaps relating to open ports. The attacker may purposely attempt to **telnet** or **ssh** to the target in order to see if the remote target allows for an attempted connection. A successful **telnet** attempt should reveal that port 23 is a likely open port on the target machine, as is the case with port 22 in regard to **ssh**. One connection attempt with each of the aforementioned services is unlikely to stir suspicion among the target system.

Criminal hacker's always have in mind what kind of attack they wish to perpetrate against the target. For example, suppose a criminal hacker wishes to deface fudge.com's home page. The suitable target IP for this attack would be the IP of the fudge.com web server. A commonly utilized command line tool, **curl** enabled with the **I** flag, and the web server's IP

address or hostname as a command line argument, displays rather useful information about the target machine. For example:

```
perep@shell.cs.fsu.edu:~>curl -I www.fudge.com
HTTP/1.1 200 OK
Date: Tue, 10 Apr 2012 16:34:37 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Set-Cookie: bla=nse2bcmfitav741lqal8i0qtj7; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Connection: close
Content-Type: text/html
perep@shell.cs.fsu.edu:~>
```

The output has provided the potential attacker with the type of web server software which fudge.com uses. Additionally, **curl** relayed information regarding the operating system on which the web server is running. The criminal hacker may then redirect his or her focus towards known exploits in the web server software, as well as the operating system, which in this case is the CentOS Linux distribution. The beauty of researching known exploits once several qualities of the target system have been identified is that the independent research has nothing to do with the actual target, therefore allowing the attacker to freely plan and customize a fitting exploit with no danger of being traced or detected. The ethical hacker must realize that fancy scanning techniques and command line tools are but a small part of the criminal hacker's process, at least prior to the exploitation phase and the maintaining access phase.

Referring back to the disadvantages of port scanning and network scanning altogether, port scanning may be considered reconnaissance if used appropriately. There is little possibility

of being blacklisted from the system in conducting one quick port scan with Nmap. The intended effect is such that the port scan may be denied and the connection instantaneously closed. In some cases, the target system may use a firewall that relays a message back to the port scan's source IP which may include information about the firewall, as well as any related equipment. Assume fudge.com uses a Cisco Catalyst 6500 series firewall module, which is configured to block port scans. Upon the blocking of port scans, a poorly configured firewall module may output the name of the firewall service on the attacker's terminal, or perhaps anything that could indicate the company uses equipment manufactured by Cisco. This sets the stage for the criminal hacker to search for vulnerabilities in the subject firewall with the help of search engines. Ethical hackers should not be afraid to search the web, for there are many instances where they may find web pages that are a mecca for criminal hackers, an open insight into the world of underground criminal recon.

Proficient utilization of web search engines such as Bing and Google are also a large part of the hacking process. The implementation of search filters is quite beneficial as well, when attempting to obtain knowledge regarding something very specific about the target. Google's version of search filters come in the form of directives. Google directives are special search filters which abide by a very particular syntax in the form of [directive]:[specification] [search]. For example, suppose the hacker would like to browse the target's web page cache for contact information, the syntax would be as follows in the Google search bar: **cache:fudge.com contact information**. One may inquire upon the reasoning behind searching the cache, and not the current version of the web page. Perhaps the cache contains the contact information of users which are no longer employed by the company. A sloppy administrator may have forgotten to remove the user from the system, wherein the attacker may attempt to **ssh** to the target using the

vacant valid username in collaboration with a brute force password attack. Using a valid username to gain access to the target system may not only mitigate the possibility of detection, but it may direct the focus toward the actual known owner of the username should the failed logon attempts be detected. There's nothing more pleasing to a criminal hacker than multiple scapegoats; they buy the hacker time, which is among any hacker's most cherished commodities.

In this respect, the ethical hacker must understand how to manipulate the situation to his or her favor. Resorting back to the common knowledge that may be acquired during the beef of the reconnaissance phase, the ethical hacker should begin to ask questions, for which the answers will help optimize the stealth and efficiency of the attack. These are some questions that the ethical hacker should be able to answer upon completion of the reconnaissance phase:

- How can I direct the system administrator's focus elsewhere?
- What type of web server distribution is the target using?
- What are some known exploits regarding the target web server version?
- What information is made public about the target via web search?
- What is the alias and IP of the target's mail server?
- What are some possible usernames used to log on to the system?
- Does the target's DNS database allow zone transfers?
- What ports are likely open on the target machines?
- What firewalls may be enabled on the target system's network?

The preceding list of questions covers the basics, but the ethical hacker should never set a limit to how many questions he or she should be able to answer upon completion of the reconnaissance phase.

The specifics of exploitation and maintaining access are beyond the scope of offensive security, for they do not play a large part in an ethical hack. Exploitation and maintaining access is typically where the compromise occurs (Engebretson). Once more, the goal of the ethical hacker is not to accomplish the compromise of a system, but rather to identify multiple vulnerabilities and volatile information that may lead a criminal hacker to the eventual successful unauthorized entry. The last two phases of the penetration testing process is usually where the criminal and ethical hackers kiss stealth goodbye. Going back to the spy example, if reconnaissance is just looking at the target and scanning is just talking to the target, then exploitation is punching the target in the face and maintaining access is handcuffing the target and throwing them into an escape vehicle. With respect to the importance of stealth, SQL injection is the safest way to avoid detection. Although there's an ongoing debate in the hacking community that SQL injection should be considered more along the lines of scanning, a successful execution of an SQL injection ultimately exploits sensitive information about the target, and sets up the framework for the attack.

SQL, meaning structured query language, revolves around search patterns that map to values which evaluate to true or false. An example of a basic SQL injection would be as follows, assuming a username and password prompt Web GUI:

| | |
|---|---|
| Username: | Jsmith (assuming that this is a valid username) |
| Password: | SELECT * FROM passwords WHERE pwd = ' 'or 1 = 1-- |

The above example demonstrates the most trivial case. The attacker would have to know name of the target's internal database which stores the passwords/hashes. If executed successfully, the result would be access to the user's account. From there, the attacker may attempt to extract as much insider information as possible before logging out. But how does an SQL injection work? The username field input queries the appropriate database containing usernames, and if the entered username matches a row in the internal database, the username search will evaluate to true. In the case of the password field, the expression mimics that of SQL syntax where the attacker inputs a pattern which will always evaluate to true. The "pwd" variable, representative of the variable containing the empty password string will likely evaluate to false, but when compared with the "1 = 1—" statement via the **or** operation, the entire pattern in turn evaluates to true, thus allowing for a successful login. The proof of this is as follows:

*The expression 1 = 1, represented by variable p is trivially true. The expression pwd = '', represented by variable q is trivially false. Assuming that p is true and q is false, true or false evaluates to true, thus p* v *q is trivially true.*

SQL injection, although subtle and stealthy, is fool's gold. The successful execution of any type of code injection requires a proficient understanding of the programming language used with the injection, as well as a mastery of logical expressions. When dealing with an experienced criminal hacker, SQL injections are not an obsolete possibility of attack method.

Acquiring the names of the internal storage databases of a target can be a painful and often discouraging process to the criminal and ethical hacker alike. One may use the **wget** command line utility paired with a web page URL to retrieve the source code of the

web page containing the form which the attacker would like to inject, typically a username and password prompt. The attacker may then use a SED script equipped with regular expressions to filter out potential database names from the source code. However, only a sloppy webmaster would allow for such sensitive information to be easily retrieved by a savvy criminal hacker. This would be an example of a vulnerability that the ethical hacker may consider looking for.

Delving into the maintaining access phase (just to give some closure), the ethical hacker may put themselves in the shoes of the attacker and ask the following questions:

- Where can I hide the shell code which enables a potential backdoor?
- What is the best way to erase digital evidence created from the previous phases?
- Are the conditions right for the enabling of a rootkit?
- What is the most efficient way to hide myself when logged on to the system?

The list of questions should go on until all possible questions have been exhausted.

With appropriate attention to stealth, it should no longer be a secret that most sensitive information which propagates a successful hack can be acquired through methods which cannot be detected via passive defense tactics. Even with an active defense approach, a paranoid potential target wouldn't know the first place to look. A good spy would never take action against a target unless they got to know as much about the target as possible, excluding communication with target, which is more relatable to the scanning phase of a penetration test. Ethical hackers should implement offensive security with one main goal in mind: A good penetration test is an invisible penetration test, one that the system administrator would not be able to detect even if he or she were looking for it. Attention to detail is of great importance, but

ethical hackers should not be afraid to pay some mind to even the most obvious things. Usually eager to employ their fancy tools and infiltration strategies, ethical hackers tend to neglect search engines and other information that can obtained by any novice who knows how to use a keyboard. The age of passive defense is quickly dissipating, for active defense is the only potentially effective defense against criminal hackers. Cybercrime has become a far more disruptive crime than anyone may have imagined years ago, and the demand for proficient security professionals is skyrocketing. It's the age of offensive security, and the ethical hackers must remain as silent as possible when putting offensive security to the test, for the quieter they remain, the more they will be able to hear.

# Part V: References

## Works Cited

Bryan Smith, David Yurcik, David Doss. "Ethical Hacking: A Security Justification Redux." 7 August 2002. *IEEE.org.* Web. 18 February 2012.

Carvey, Harlan. *Windows Forensic Analysis*. Burlington, MA: Syngress Publishing Inc., 2008. Book.

Engebretson, Patrick. *The Basics of Hacking and Penetration Testing* . Waltham, MA: Syngress, 2011. Book.

Engelhardt, Jan. "Detecting and Deceiving Network Scans." 13 May 2008. *http://jengelh.medozas.de.* Web. 3 April 2012.

Lyon, Gordon. *Nmap Network Scanning*. Sunnyvale, CA: Insecure.com LLC, 2008. Book.

Mourani, Gerhard. *Securing and Optimizing Linux*. 23 January 2000. Web. 4 April 2012.

Palmer, C.C. "Ethical Hacking." 13 April 2001. *IEEE.org.* Web. 22 February 2012.

Ranganath, Karthik. *The Backtrack Experience for Novice: An Introduction to White Hat Hacking*. Las Vegas, NV: Team NetNoblesand 3psilonlambda, 2011. Book.