Global Information Assurance Certification Paper

IDS: File Integrity Checking

GIAC (GSNA) Gold Certification

Author: Lawrence Grim, ldgrim@gmail.com
Advisor: Robert Vandenbrink

Accepted: July 31, 2014

Abstract

The paper covers three, open-sourced, GNU General Public Licensed (GPL) file integrity checking packages. The three applications are Another File Integrity Checker (AFICK), Tripwire and Advanced Intrusion Detection Environment (AIDE). All three applications have similar methods of comparing baseline and current file signatures, but differ on execution and support. A short summary of the requirements for each package, a discussion of the capabilities, functions, features and limitations are provided. The comparison metrics for the three files integrity checkers are intended to be a deployment decision matrix. The paper will provide criteria for deciding which package meets an operational need.

Keywords: IDS, HIDS, File Integrity, Virus Detection, Computer Security

1. Introduction

The file integrity checking application is a host-based intrusion detection software. Host-based monitoring applications are "particularly effective at detecting insider misuse because of the target data source's proximity to the authenticated user" (Proctor, 2001, p. 50). A compromise of a system is often accompanied by alteration of files on the system. A file integrity checker periodically validates existing file criteria against a known, stored value to detect changes or modifications (SANS, 2009). The idea of monitoring file usage was proposed in a seminal paper by James Anderson (Anderson, 1980). Rather than just the focus on tracking and controlling access to files, in the early 1990s, Gene Kim and Eugene Spafford focused on monitoring added, deleted or changed files (Kim & Spafford, 1994). Their program, Tripwire, initially generated file signatures and periodically compared that historical baseline against current status. This was the anomaly detection intrusion detection model versus the misuse detection model (Jones & Sielken, 1999).

2. Application Evaluation

This paper outlines general limitations to each of the three open-source file checking applications, with comparison of the installation and operation of the applications. The applications will be evaluated based on five criteria:

- Effectiveness and Efficiency
- Response and Timeliness
- Operational Requirements
- Deployment Needs and Capabilities
- Security

2.1. Limitations to File Integrity Checking

The use of file integrity checking programs for intrusion detection and auditing have several limitations. These include file signature bypass, static anomaly detection, operating system environment restrictions and lack of change details. All three, open-

source file checking applications evaluated have these limitations. As such, limitations can be discussed generically for these three open-source file integrity checking applications.

2.1.1. File Signature Bypass

The use of signatures, with a known generation method, can be defeated when a compromise file is used to evade detection. Once a way has been found to avoid detection by a security tool's fingerprint designation, it can avoid detection by that software at all sites (Forrest, Perelson, Allen, & Cherukuri, 1992). The normal file integrity checking application may use a weak cryptographic algorithm. Legacy algorithms such as MD5 were found to be flawed and easy to attack. An MD5 hash value or digest can be duplicated from two different messages. This collision or duplicate hash value can be generated for an MD5 hash digest in less than a minute with a regular notebook computer (Klima, 2006). Using the default MD5 signature generation, an evasion tool (Stripwire) was developed to specifically create files that evade the normal Tripwire detection (Kaminsky, 2004). Having a wide variety of simultaneous cryptographic generation algorithms can help to detect evasion through signature weaknesses. Each of the open-source file integrity checking applications are either restricted to a subset of cryptographic algorithms or deploy with a restricted set of signature generation choices.

2.1.2. Static Anomaly Detection

These three file integrity applications note that something has happened. However, none of them note how a file was changed nor accurately when a change was made. One of the faults with comparing benchmarked and current file characteristics is the idea of a static anomaly detection. This method ignores activities within the intervening timeframe where an undetected, but modified file would be executed. In fact, a file could be changed, executed and be restored between integrity checks and go unnoticed. Ideally, file change detection should occur when the binary or other code is being executed (Nicholes, 2004). Expanding beyond just a historical file hash or signature, there have been a variety of security controls recommended to validate or permit applications and code to execute on an information system. These include "Digital Signature, Code

Signing, Watermarking,…" (Saha & Negatu, 2010, p. 1). Expanding to use these techniques increases the security of the system. None of the three open source file integrity checking programs went beyond historical hashes at pre-determined intervals.

2.1.3. Restricted Operating Environment

The memory space allocated to the guest system by the host system, such as /dev/kmem, would be unevaluated by a file integrity checking but would be a prime avenue for exploitation of the guest system. These constantly, dynamically changing files could not be compared with a file baseline. The file integrity checking application on the guest or host system would be focused on detecting such dynamic changes to the virtual machine's files. "Checking the integrity of a program binary on disk … does not ensure that the corresponding in memory image of that program has not been modified" (Garfinkel & Rosenblum, 2003, p. 199).

2.1.4. Lack of Change Detail

While important to identify changes to files in a timely manner, these file integrity checkers do not provide information on how the file had changed. Just like an unnecessary medical test might force unintended reactions, the modification alert from a file integrity checker serves to identify the detection point instead of the chain of events that caused the change. (King & Chen, 2005).

2.2. Environment

A 64-bit base system, hosting three virtual machines, was built using Ubuntu 12.04 LTS. The default kernel for Ubuntu can be used with either a guest or host system with the Xen virtual machine (Xen, 2014). The host evaluation system was built within a firewalled, protected network. Because Xen virtual manager works best in a graphical interface, the X11 X-Windows interface was used to access the headless host server. The X11 client used was a NoMachine version 3 client, which now not offered on the current site, but available through the Wayback Archive (NoMachine, 2009). Access to the graphical user interface on the host machine was necessary both for using the Xen virtual machine manager, but also to gather information from each of the virtual machines, each hosting different file integrity checkers. Three virtual machines, running 64-bit Ubuntu 12.04, were setup as guests, each with a single CPU, 1GB memory, 8GB

IDE hard disk, cdrom, network interface, mouse, sound card and serial port. The initial specifications of the bare bones Ubuntu LINUX virtual machines were captured and were used as the baseline for disk usage during the application evaluations.

2.3. Comparison

Each of the three file integrity checking programs were installed in their own virtual system, as noted in the baseline comparison system. Points of comparison were gathered during the installation that included cryptographic signature support, installation resources, detection of changes, performance during operations, and deployment considerations.

2.3.1. Comparison of Hash Algorithms for File Signatures

As shown below, each of the three open source file integrity checking programs supported a variety of message digest or hash algorithms for file signatures, with AIDE have the widest choices.

	md5	sha1	sha256	sha512	rmd160	tiger	whirlpool	gost	crc32	haval	Citation
AFICK	■	■	■	■							Gerbier, 2004
Tripwire	■	■							■	■	Mir, 2000
AIDE	■	■	■	■	■	■	■	■	■	■	von Haugwitz, 2013

Table 1: Supported Cryptographic Signatures

2.3.2. Basic Installation Notes

After the installation, the space used for the application as well as the database was noted, as shown below. Both AIDE and Tripwire increased the threat attack surface from a basic Ubuntu LAMP server by requiring a mail server (PostFix). Both of these two file integrity checking applications required the mail server. Without any further guidance, the installation of the packages left the system with a mail server. Subsequent changes to the PostFix installation would be required to make it a client-only mail system. The third file integrity checking program, AFICK, did not email any reports and did not require either a host or client mail capability.

	AFICK	Tripwire	AIDE
Space (1K blocks)	42,284	176,380	14,780

Database Size	2,650,332	3,050,628	4,656,697
Database Location	/var/lib/afick/afick	/var/lib/tripwire/<servername>	/var/lib/aide
Files Indexed	16271	64,008	Unknown
cronjob created	Yes, daily	No	Yes, daily
Mail Server (PostFix)	No	Yes	Yes

Table 2: Initialization Requirements

2.3.3. Post-Installation Patching

The guest operating system and installed IDS package were started a month earlier. As a test of the performance of the file integrity checking applications, a significant amount of identical changes had to be made to each system. Since that initial build and software installation, there were packages upgraded and available for installation. For the PostFix mail server IDS (Tripwire/AIDE), there were 174 packages, while the Perl-based AFICK had only 171 packages upgraded. Applying this variety and volume of patches served as a reasonable benchmark for measurements of the file integrity checking applications.

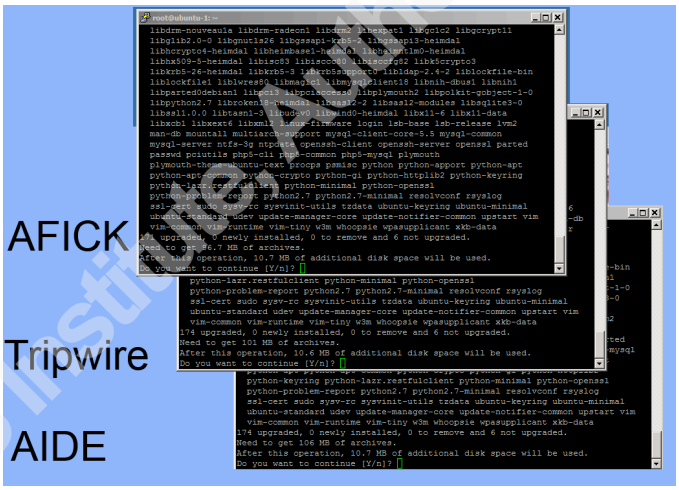

Figure 1: First Patch Session

Execution of the upgrade and after a system reboot, disk usage was measured and resulted in the following changes in disk utilization:

Changes	AFICK	Tripwire	AIDE
/	+180,640	+185,480	+190,168
/dev	n/c	n/c	n/c
/run	n/c	n/c	n/c
/boot	+12	+12	+12

Table 3: Disk Space After Significant Patching

The changes to the file system noted after the patching verified that significant changes occurred on the system because of the upgrades.

2.3.4. Large File Change Performance Metrics

With the amount of changes generated by the package upgrade/update, the performance of the three IDS file checking applications was measured during the system scan and re-indexing. This data gathering was done with the `XenState` Perl script, ran on the hosted domain (Lim, 2009). As noted, the Perl-script based AFICK used generally less CPU resources than the next resource intensive file integrity checking program, AIDE. Tripwire generally used a higher level of CPU cycles throughout the process than the other two programs.

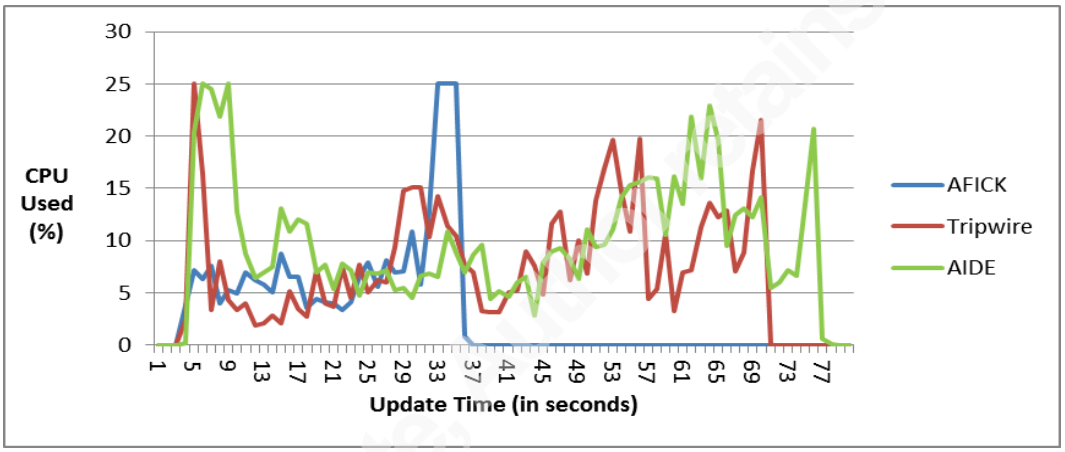

Figure 2: CPU Utilization During Update

Running the update to the three IDS file integrity checking routines was monitored and compared after the significant patch upgrade. Metrics from before and after the update were gathered including disk utilization and file integrity database size. After the upgrades, file change detection metrics were noted.

		AFICK	Tripwire	AIDE
1K Blk	Disk space (old) 1K	1,528,092	1,667,104	1,510,680
	Disk space (new) 1K	1,531,872	1,668,264	1,525,488
	Disk space (change) 1K	3,780	1,160	14,808
bytes	Database Size (old) bytes	2,650,332	3,050,628	4,656,697
	Database Size (new) bytes	2,666,624	3,075,396	4,707,548
	Database Size (change) bytes	16,292	24,768	50,851
	Ratio Files/Size	1:1	2.3 : 1	1:1
	Files Scanned	16,431	64,398	59586
	Files Changed	4676	17787	16603
	Files New	223	510	521
	Files Deleted	63	120	127
	Files Other	106	17157	16603
	Execution Time	37 sec	71 sec	78 sec
	Files/Second Scanned	~440	~915	~760

Table 4: File Integrity Check after Update

Results, both in database size and overall disk space usage clearly show AIDE using the most disk space, while Tripwire created the most compact or efficient database. Execution, based on files scanned per second, shows the optimization of the compiled programs (Tripwire and AIDE) over the Perl-script based AFICK.

2.3.5. Protection of Binaries

All three file checking programs protected their own binaries after the baseline scan of the system determined the hash signatures. For example, the first action that AFICK completes after installation is calculation of the file signature for the program. This is forced during the installation of the software and the MD5 signature is used during subsequent executions as well as incorporated in the file integrity checking report.

```
# Hash database created successfully. 16271 files entered.
# ###############################################################
# MD5 hash of /var/lib/afick/afick => w7CCKrRrSS52LDtUH6N7bw
# user time : 4.8; system time : 1; real time : 8
```

Figure 3: AFICK Binary Calculations

During the installation of TripWire, the installation script asked for two passphrases, both a minimum of eight characters in length. These passphrases were necessary for any policy changes, to reload or initialize the product and other house-keeping tasks. The site passphrase protected configuration and policy files, while the local passphrase protected databases and report files (Natarajan, 2008). These passphrases subsequently prevented changes to Tripwire's configuration and policy files.

```
----------[snip]----------
The Tripwire site and local passphrases are used to sign a variety of files, such as the
configuration, policy, and database files.
Passphrases should be at least 8 characters in length and contain both letters and
numbers.
See the Tripwire manual for more information.
-------------------------------------------------------------------------------
--
The Tripwire site and local passphrases are used to sign a variety of files, such as the
configuration,policy, and database files.
Passphrases should be at least 8 characters in length and contain both letters and
numbers.
See the Tripwire manual for more information.
-----------------------------------------
```

Figure 4: Tripwire Passphrase Protection

All three file integrity checking programs have checksum values for the downloadable source files, whether that is Perl scripts or application code. But for the AIDE file integrity checking software, it is signed and stored in the Ubuntu repository, automatically verified during the download and installation. The AFICK

author has provided a copy of the application that runs off of a CDROM to avoid having the application changed.

While all three programs check their binaries during the standard file system verification, AIDE starts out more secure because of the restricted and controlled source of the application. TripWire locks up the configuration and policy files with passphrases created during installation. AFICK, though, provides the highest level of program binary protection from start through execution by having the capability to run the verified application executable from a CDROM.

2.3.6. Evaluation of Reporting

Both Tripwire and AIDE generate email reporting on the results of a file integrity check of the system. This is convenient as the report can be sent both internally to a local user (such as root) or externally through the required mail server/client. Both Tripwire and AIDE emails highlighted the files whose current file hash differed from the baseline hash digest. The AIDE file integrity checking application generated a voluminous log, which was difficult to read. To get a readable report of a Tripwire session, there is a command that will generate a text report.

```
root@ubuntu-2:/# /usr/sbin/twprint --print-report --twrfile
/var/lib/tripwire/report/ubuntu-2.grim.enterprises-20140413-170159.twr >/root/2014-04-
13.txt
```

Figure 5: Generating a Tripwire Report

The easiest reporting output was from the AFICK file integrity checking program in that it generated a file with comma-separated-values (CSV) formatted data. This was easy to export and process in a standard spreadsheet program such as Excel.

2.3.7. Configuration Restrictions

Looking at the reports for all three applications, the identification of files was based on the default configuration, not fine-tuning of the parameters for the setup. However, some comment is necessary for the configuration file contents. AFICK uses suffixes to exclude files from comparison checks, such as shown below:

```
# text files
exclude_suffix := log LOG html htm HTM
txt TXT xml
# help files
exclude_suffix := hlp pod chm
# old files
exclude_suffix := tmp old bak
# fonts
exclude_suffix := fon ttf TTF
# images
exclude_suffix := bmp BMP jpg JPG gif
png ico
# audio
exclude_suffix := wav WAV mp3 avi
```

Figure 6: AFICK File Exclusion Parameters

The default setup for AFICK predefines very small or limited number of files and directories, such as /bin, /boot, /etc, /lib and /root. Tripwire and AIDE are almost equal in their default system coverage setup in their configuration files. Further comparison of the output from each of the three file integrity checking products could develop a tuned configuration to support this Ubuntu-base LAMP (Linux/Apache/MySQL/PHP) installation. However, this paper does not need to level-set or balance the file integrity checking coverage as the comparison has been made with files scanned per second and database density metrics.

2.3.8. Deployment Restrictions

All three programs are open source, so the deployment opportunities are based on the code type for each application. AFICK, using Perl, has deployment on a variety of platforms, including Windows, Linux, HP Unix AIX and Solaris (Gerber, 2013). The last stable version was version 3.4, dated August 23, 2013 (York, 2002). Being a Perl script, there is no compilation or other modifications for AFICK. This gives AFICK the widest supported base of operating systems of the three, open-source file integrity checking applications. The open source version of Tripwire has been last updated in November, 2011 (Itripn & Stephd, n.d). Open-source Tripwire was the most difficult installation of the three file integrity checking applications in the guest Ubuntu environment. Tripwire had to be installed from binaries, with several critical changes to the installation scripts (Tripwire-2.3.22, 2012). AIDE was an easy installation as it has been in the Ubuntu repository since 2012, but could have been compiled from the open source binaries (Canonical, 2014). For this paper's comparison, no AIDE binaries were compiled.

2.3.9. Application Update Comparisons

The Ubuntu-tailored AIDE packages install from the Ubuntu software repository and get automatic patches delivered with normal system maintenance. Neither of the other two packages (AFICK or Tripwire) update with the normal Ubuntu update methods. However, Tripwire can use updated external libraries with recompilation and reinstallation. AFICK, however, relies upon the vendor for updates, which have to be monitored and tracked separately. The AFICK developer does not use package managers such as **`apt`** or **`rpm`** for software updates, but provides complete Perl scripts as updates.

2.4. Criteria Comparison

Criteria	AFICK	Tripwire	AIDE
Date of Software	2013	2011	2012
Effectiveness and Efficiency			
File Scanning Speed	3	1	2
Scanning Rate	1	2	3
Database/File Ratio	2	1	3
Response and Timeliness			
CPU Load	1	2	3
Database Size (smaller=better)	2	1	3
Automatic Daily Background Job	1	2	1
Operational Requirements			
Multiple Platform Support	1	3	2
Installation Prerequisites	1	3	2
Ease of Reporting	1	3	2
Deployment Needs and Capabilities			
Ease of Installation	1	3	2
Degraded Attack Surface	1	3	3
Multiple Platforms	1	3	3
Automatic Upgrades	2	3	1
Security			
Message Digest Security	2	1	3
Protection of IDS Binaries	1	2	3
Protection of Integrity Database	1	2	1
Summary (lower is better)	21	35	37

Table 5: Three File Integrity Checker Comparison

3. Conclusion

AFICK has the internal security, database protection and density, as well as ease of use over the other two tested open source file integrity checking applications. For the

open source versions, AFICK also supports the largest number of operating systems. Unlike the other two file integrity checking applications, AFICK does not require a mail server to be installed on the host system that would increase the attack surface. Installing additional software applications opened up new vulnerability vectors. Decreasing the server's overall security for a security package seemed counter-intuitive. The one major advantage that Tripwire retains over the other two open source file integrity checking utilities is the potential upgrade into a commercial package with enterprise support. However, based on this paper's evaluation, the recommendation would be to use AFICK as the open source file integrity checking intrusion detection system.

4. References

Anderson, J.P. (1980). Computer Security Threat Monitoring and Surveillance [White Paper]. Fort Washington, PA: James P. Anderson Co. Retrieved from http://csrc.nist.gov/publications /history/ande80.pdf

Canonical (2014). "aide" package in Ubuntu [Wiki post]. On *Ubuntu Packages*. Retrieved from https://launchpad.net/ubuntu/+source/aide

Denning, D.E. (1987) An Intrusion-Detection Model. *IEEE Transactions on Software Engineering,* 13(2), pp. 222-232. doi: 10.1109/TSE.1987.232894

Forrest, S., Perelson, A., Allen, L. & Cherukuri, R. (1992). Self-nonself discrimination in a computer. *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pp 202-212. Retrieved from http://people.scs.carleton.ca/~soma/ biosec/ readings/forrest-virus.pdf

Forrester, R. & Dudzinksi, S. (2013). Open Source Tripwire [Software Project]. On *SourceForge*. Retrieved from http://sourceforge.net/projects/tripwire/

Garfinkel, T. & Rosenblum, M. (2003). A Virtual Machine Introspection Based Architecture for Intrusion Detection. *Proceedings of Network and Distributed Systems Security Symposium, 2003*, pp 191-206. Retrieved from http://citeseerx.ist.psu.edu/ viewdoc/download? doi=10.1.1.11.8367&rep=rep1&type=pdf

Gerbier, E. (2004). Manual page: afick.conf(5). Retrieved from https://web.archive.org/ web/ 20041027205459/http://afick.sourceforge.net/ afick.conf.5.html

Gerber, E. (2013). Another file integrity checker [Software Project). On *SourceForge*. Retrieved from http://sourceforge.net/projects/afick/

Itripn & Stephd (n.d.). Open Source Tripwire®. On *SourceForge*. Retrieved from http://sourceforge.net/projects/tripwire/files/tripwire-src/tripwire-2.4.2.2/

Jones, A. & Sielken, R. (1999), *Computer System Intrusion Detection: A Survey* [White paper]. doi: 10.1.1.24.7802&rep=rep1&type=pdf

Kaminsky, D. (2004) *MD5 to Be Considered Harmful Someday* [White paper]. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download? doi=10.1.1.58.8575& rep=rep1&type=pdf

Kim, G. (2008). Virtualisation: Seven steps to a secure virtual environment. *Network Security, 8*, pp. 14-18. doi:10.1016/S1353-4858(08)70098-9

Kim, G. & Spafford, E. (1994). *Writing, Supporting, and Evaluating Tripwire: A Publically Available Security Tool* [White Paper]. Retrieved from https://www.cerias.purdue.edu/assets/pdf/bibtex_archive/94-04.pdf

Klima, V. (2006). Tunnels in Hash Functions: MD5 Collisions Within a Minute (Report 2006/105) [White paper]. On *IACR Cryptology Eprint Archive*. Retrieved from http://eprint.iacr.org/2006/105.pdf

Lim, J. (2009, September 28). Monitoring and logging CPU utilization of virtual machines in XEN [Blog post]. On *PHP Everywhere*. Retrieved from http://phplens.com/phpeverywhere/ ?q=node/view/266

Mir, R. (2000) *IDS and Tripwire* [PowerPoint presentation]. Retrieved from http://www.iup.edu/WorkArea/DownloadAsset.aspx%3Fid%3D61177&sa=U&ei=jDJQU_DiOsKprAe9_YCIDQ&ved=0CCgQFjAA&sig2=Jli5kNV1QQlGGHGKEIQpgg&usg=AFQjCNHf9GhCN5nBpb-6-_Tyrrv6u88QuQ

Natarajan, R. (2008, December 8). Tripwire Tutorial: Linux Host Based Intrusion Detection System [Blog post]. On *The Geek Stuff*. Retrived from http://web.archive.org/ web/20081219185410/http: //www.thegeekstuff.com/2008/12/ tripwire-tutorial-linux-host-based-intrusion-detection-system/

Nicholes, M.. (2004). *Foundation for Secure Boot on Itanium 2 Systems*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.132.2897&rep=rep1&type=pdf

NoMachine (2009). *Download Get NX Software from No Machine*. Retrieved from http://web.archive.org/web/20091108233137/http://www.nomachine.com/download.php

Proctor, P.E. (2001). Host-Based Intrusion Detection Systems. In *The Practical Intrusion Detection Handbook* (pp. 49-77). Upper Saddle River, NJ: Prentice Hall.

Saha, S. & Negatu, A. (2010). Techniques for validation and controlled execution of processes, codes and data: A survey [White Paper]. In *Proceedings of 2010 International Conference on Security and Cryptography (SECCRYPT)*, pp. 1-9. Retrieved from http://ais.cs.memphis.edu/ files/papers/SECRYPT-Paper9.pdf

SANS Institute (2009). *Intrusion Detection FAQ: What is the role of a file integrity checker like Tripwire in intrusion detection?* Retrieved from http://web.archive.org/web/ 20091010234537/http://www.sans.org/security-resources/idfaq/integrity_checker.php

Tripwire-2.4.2.2 (2012, September 15). *On Beyond Linux® From Scratch*. Retrieved from http://web.archive.org/web/20120920005313/http://www.linuxfromscratch.org/blfs/view/svn/postlfs/tripwire.html

von Haugwitz, H. (2013). AIDE-Advanced Intrusion Detection Environment. On *SourceForge*. Retrieved from http://aide.sorceforge.net

von Haugwitz, H. & Van De Berg, R. (2010). AIDE [Software Project]. On *SourceForge*. Retrieved from http://sourceforge.net/projects/aide/

Xen (2014, March 17). On *Community Help Wiki, Ubuntu Documentation*. Retrieved from https://help.ubuntu.com/community/Xen

York, D. (2002). Frequently Asked Questions about AFICK. On *SourceForge*. Retrieved from http://afick.sourceforge.net/faq.html#installation0