CURTAILING WEB DEFACEMENT USING A READ-ONLY STRATEGY

Andrew Cooks **Martin S Olivier**

Information and Computer Security Architectures Research Group
Department of Computer Science, University of Pretoria

ABSTRACT

Web defacement is the act of altering the contents of a web site with malicious intent, sometimes associated with vandalism and sabotage.

Web defacement results in extreme embarrassment to the web site owner, regardless of the commercial interest in the web site. However, persons and companies who are targets of web defacement, often have substantial interest in maintaining the professional image and integrity of the web site.

While proper security management and appropriate security tools can be used to avoid this embarrassment, the complexity inherent in current systems implies that loopholes often exist.

This paper proposes a solution that uses read-only media to address the problem of web-defacement. In order to be trustworthy, such a system should have no persistent writable media — otherwise a cracker could potentially change the system to serve web pages from this medium, rather than from the read-only medium.

In addition, the volatile memory needs to be reset frequently to minimise attacks targeted at this potentially weak spot. Such resetting should not compromise availability or auditability.

The solution proposed in this paper uses read-only DVDs and custom live CD versions of the Gentoo Linux distribution, to achieve trustworthy, read-only media. The simplicity provided by the use of read-only media achieves a level of trust that is hard to achieve in any other way.

KEY WORDS

Web defacement, read-only servers, integrity

This material is based upon work supported by the National Research Foundation under Grant number 2054024 as well as by Telkom and IST through THRIP. Any opinion, findings and conclusions or recommendations expressed in this material are those of the author(s) and therefore the NRF, Telkom and IST do not accept any liability thereto.

1 INTRODUCTION

The problem of protecting web servers from crackers and vandals has received substantial attention as the commercial interests in the web have grown. This problem is the general and familiar security problem of protecting confidentiality, integrity and availability [1].

All three aspects need to be maintained and improved in order to build the multi-billion dollar facet of trade known as e-commerce [2]. Despite the broad understanding of the problem and numerous guidelines, there are few ways to enforce, or guarantee these security aspects.

Prevention of web defacement is a practical facet of the problem of maintaining web server software and content integrity. This paper forms part of the internationally ongoing research to address this situation [3]. Web defacement differs from other forms of system breaches in two ways. Firstly, such defacements are typically very visible to the outside world. Secondly, such defacements are often used to use an organisation's server to distribute a hate speech message, or some other message that the organisation does not want to be associated with. Hence, such a defacement could cause considerable embarrassment to an organisation and be the cause of a significant loss of credibility. The nature of such defacements is clear from sites that contain mirror images of some defacements — even though their lists of are not up to date (see, for example, attrition.org [4, 5] and Project Gamma [6]). The fact that the problem remains, is clear from a report that about 35 South African sites were defaced on a single day (16 May 2004) [7].

Methods discussed in this paper attempt to improve integrity [8], by ensuring that neither web content, nor the system configuration can be altered on the web server. We achieve this by using known-good media and frequently rebooting the system resetting it to the trusted state.

The specific technique that is focused on, involves the use of custom live CDs for the system state and web content. A live CD is a fully functional operating system installation that runs directly from a CD or DVD, instead of a hard drive. We discuss the implementation details and mechanical process of building these CDs in Section 4. The prototype described in this paper uses Linux, but, in principle, such a system could be built using any of the popular operating systems.

Resetting the server to the trusted state by rebooting it will heavily impact on the availability of the web content and poses a new problem. Techniques to overcome web server availability problems are able to achieve reasonable web server availability and are well understood. This paper will discuss these failover techniques briefly for background purposes.

The rest of the paper is organised as follows: Section 2 discusses different functional approaches to protect integrity. Section 3 outlines the read-only strategy in an integrity context. Section 4 focuses on the implementation details of a read-only web server. Drawbacks and novelties of this approach are discussed in Section 5. Section 6 proposes future work and is followed by a conclusion in Section 7.

2 BACKGROUND

Integrity, in general, has received a large amount of academic attention, including functional [9] and nonfunctional [10] approaches. This paper explores a functional approach to integrity, focussing on a read-only strategy to curtail web defacement.

Indeed, the vast area of research covering integrity has a large overlap with popular topics such as watermarking, fault tolerance, firewalls and hashing algorithms [10].

Watermarking in its most common form, is the idea of embedding information in an image that would uniquely identify it [11]. The applications of this technology include identification of images for copyright purposes, as well as detecting modification of images. It is safe to conclude that watermarking plays a significant role in the integrity aspect of security.

Fault tolerance is usually not studied from a security perspective. However, considering that it is primarily a mechanism to improve availability, it is clearly relevant to the field of security. In certain cases it can also be seen from an integrity perspective, as we show in Section 3.

A firewall is, simply stated, a device that serves as a controlled point of entry to computing resources [12], in a network context. Firewalls, broadly speaking, are involved in protecting integrity, by preventing unauthorised hosts from accessing certain hosts, or parts of the network in a certain way. This could be used to ensure that only web traffic is allowed to reach the web server from the external network, which adds another layer of security.

Hashing algorithms, also called message-digest algorithms, are designed to compute a fingerprint on a specific data set. A good algorithm has two important properties [13, 14]:

1. Given a specific hash value and hashing algorithm, it should be difficult to craft a data set that has this specific hash value.

2. Given a specific hashing algorithm, it should be difficult to produce two data sets having the same hash value.

These properties ensure that message digests can be used to determine integrity of a data set. This can be done by computing a digest of the data and comparing it to a previously computed digest of the known-good data. If the digests differ, then the data sets are not equivalent.

Another strategy to protect web content integrity involves placing the content on a secure file server that only allows read access [15, 16, 17]. This solution makes use of access control mechanisms that revoke write access for anyone who is accessing the data via the web server. While such a solution has merit, it assumes that the access control mechanism cannot be bypassed. Given the long history of security problems [18, 19] with Remote Procedure Call (RPC) services [20] (which are commonly used in networked file systems), implies that this approach is not as secure as it could be. This threat becomes even more serious where an organisation is not in a position to apply the frequent patches required to keep a machine secure. Hence, different options, such as read-only media, should be considered.

The idea of using live CDs to ensure the read-only state and integrity of the system is not entirely new, but due to the practical nature of this approach, it has not received much academic attention to date. Related work has been done by system administrators and integrators in a non-academic context [15, 21, 17].

This specific read-only strategy is also popular with firewall administrators and in widespread use. One source lists six different live CD based Linux distributions, specially designed for use as firewalls [22].

3 READ-ONLY STRATEGY

The system described in this paper is based on three premises:

1. If a web server is a read-only system, it will be significantly harder to deface it than when it is not.

2. Basing a system on read-only media is one of the simplest strategies to make the system read-only.

3. Many web servers can use such a read-only strategy.

The first of these premises — if a web server is a read-only system, it will be significantly harder to deface it — is obvious. We contend that any access control strategy that can be used in a system that does not enforce a read-only strategy can also be used on a system that does. Making a system read-only simply ensures that, if these access controls are somehow bypassed, a cracker cannot change the contents the server serves. Note that there must still be some way in which pages are updated. The proposed system will use a process to prepare the read-only media in an isolated environment that can be made significantly more secure than a web server for use by the public typically can be made. However, since such an environment may still be compromised, we do not claim that a read-only strategy makes defacement impossible. Given the isolated production environment and possibility to review read-only content before publication, we do claim that this can be used to create a system that is significantly harder to deface.

The second premise — basing a system on read-only media is one of the simplest strategies to make the system read-only — is also obvious, but hides a potential threat. It is clear that hardware limitations make it impossible to change read-only media. This also applies to media, such as DVD-R and CD-ROM discs, that can be written once and only read after that. Even rewritable media, such as CD-RW, DVD-RW, DVD+RW and DVD-RAM discs are effectively (and simply) confirmed read-only media if no writer for such a disc is installed on the server. The use of write-once-read-many media, such as DVD-R and CD-R, threfore ensures a level of trustworthiness and auditability that is beyond that of other read-only strategies.

The potential threat that could be overlooked when using read-only media is the fact that even a system based on read-only media needs some volatile storage (in RAM, for example). If that volatile storage is compromised, it will still be possible for the server to be cracked so that it serves unintended content. Such a compromise can, however, not be made persistent and a simple reboot of the system will restore it to its original state. We will consider the implications of frequently rebooting the system below.

The third premise — that many web servers can use such a read-only strategy — is, firstly, based on the observation that in practice many web servers are used to serve static content, and the proposed solution clearly applies to such servers. In addition, in the case of dynamic content, such as e-commerce sites, the bulk of the content on the web server is also often static and the dynamic content (such as stock levels) is retrieved from a back-end database. If this database is breached, extensive damage can still be done. The following factors should, however, be taken into account:

1. The fact that such a database can be placed in the back-end makes it less exposed and somewhat harder to compromise from the outside world than the web server itself;

2. While still potentially extremely serious, a compromised database will probably be less visible than a defaced web site — and hence may be less embarrassing to the victim;

3. Many database items — such as product descriptions — may themselves be served from a database built on read-only media; and

4. The use of a read-only web server, in the worst case, does not impact on the security of the database behind it.

Hence, it is indeed meaningful to use a read-only strategy in many environments.

Building live CDs has the added benefit that content changes are forced to go through change control. Each CD serves as a complete checkpoint of the state of the web server, which simplifies quality assurance testing. Testing can be done on any convenient machine, be it a dedicated web server or not, since the complete web server installation is now tangible and portable.

Given that content changes go through a change control system that can associate a specific checkpoint with a specific change, such live CDs can also be used for backup purposes.

4 IMPLEMENTATION DETAILS

This section discusses the implementation details of the proposed solution. It is divided into subsections covering the architecture of the solution, the mechanical process of building a live CD, the implementation details of the live CD and considerations in the operation and maintenance of the system, such as logging.

4.1 Architecture

The solution consists of three distinct components:

1. A secure and audited system, used for building the live CD

2. A load balancer and/or failover system

3. Two, or more web servers

A relationship diagram between these components is shown in Figure 1.

The practical implementation of this system makes use of an audited and secured building host that contains the master copy of the web content and possibly the CD and/or DVD writer for the production of the live CDs.

The integrity of this machine is of paramount importance, since this will be the easiest point of entry to the system. According to Stein [23], the most overlooked threats are from people who can be considered as insiders or have legitimate access to the system.

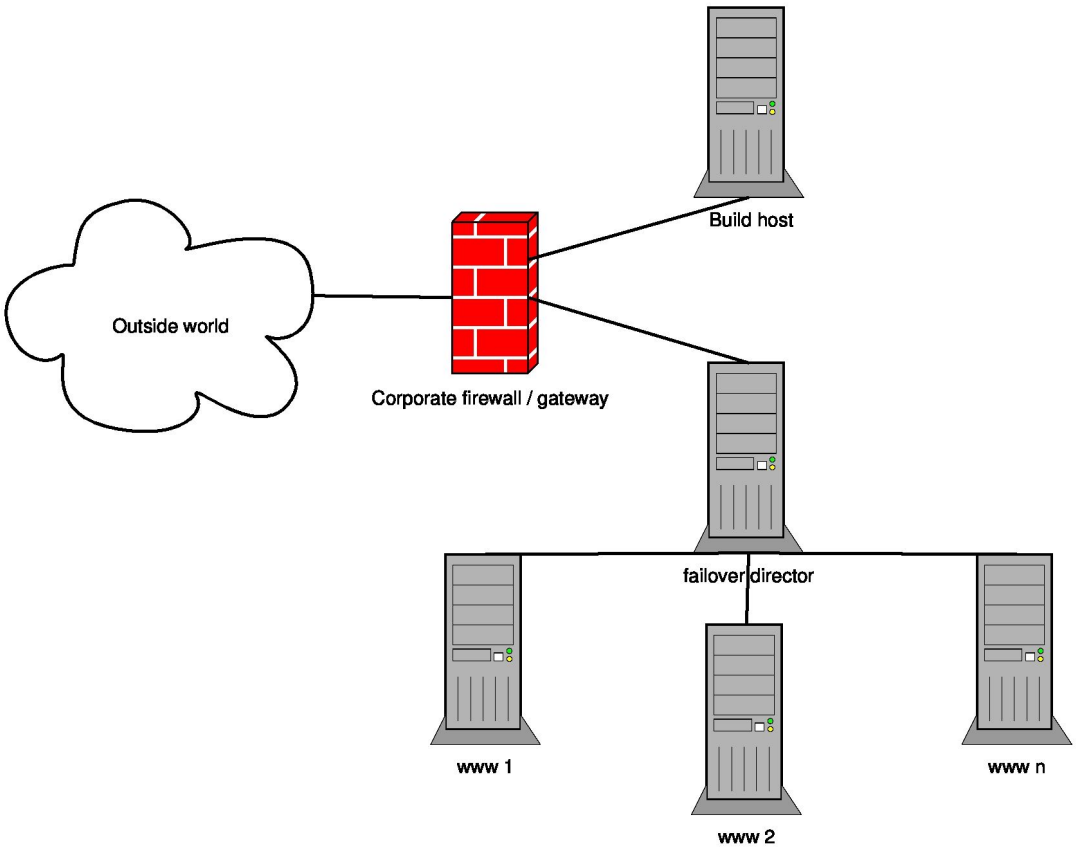

Figure 1: Relationship diagram of a live CD based infrastructure

Quality assurance tests on the live CD can easily be done offline, since it can safely be assumed that the content of the write-once-read-many media will not and cannot change again, by definition.

Unfortunately, read-only media does not guarantee complete integrity of the system, because certain parts of the filesystem are used for storing the system state and need to be writable during normal operation. To accommodate those parts of the system which require write access, a memory based filesystem is used, as discussed in section 4.3.

Since it is impossible to guarantee integrity of the web server system state under these conditions, this state will have to be reset to the trusted state, much like one would occasionally reboot a personal computer to achieve a known-good state. By using multiple web servers with a load-balancing failover system that ensures availability, machines can be rebooted at interleaved times. This ensures integrity, without compromising on availability, as discussed in section 4.4.

4.2 Live CD build process

There are multiple ways to build a live CD. One approach would be to take any existing live CD, copy it to hard drive and modify it to suit a particular purpose, then write it back to CD. This approach is one of trial and error and would be difficult to automate.

Another approach would be to compile every package from source, placing the executables, libraries and configuration files in the required directory structure and finally writing this to CD. This is a laborious and error prone process that should not be repeated without good reason to do so. Instead, the process should be automated as far as possible to rule out inconsistencies in the build process and insure repeatable results.

We opted to use the build tool for Gentoo Linux live CDs, called Catalyst [24], and used it to generate custom live CDs pre-configured with static web content included. Catalyst provides a convenient build tool that produces repeatable results and automates most of the manual work involved in creating live CDs, without compromising on the customisability of the end result.

The steps to build a live CD with Catalyst are as follows:

1. Configure Catalyst and determine the environment and compiler settings that will be used during step 3, the compilation phase.

2. Create *a spec* file to determine which packages to include on the CD and create a ”snapshot” to determine the versions of these packages to include.

3. Compile the packages, starting by compiling the compiler and build tools that will be used to compile the rest of the live CD packages.

4. Move the executables, supporting libraries and configuration files to the desired locations and construct a standard CD-ROM filesystem (ISO 9660), that can be written to CD.

The end result is a live CD that contains the Gentoo Linux meta-distribution, but the application of a live CD as a web server is not specific to Gentoo Linux.

4.3 Implementation details of the live CD

As mentioned, a live CD is a complete system installation that runs directly from the CD, without having to be installed on a hard drive. Live CDs are typically associated with the free, Unix-like operating systems such as GNU/Linux and the BSDs. For the rest of this discussion, a Linux distribution conforming to the Linux Standards Base (LSB) [25] is assumed.

During the normal operation of any operating system, certain parts of the filesystem need to be writable. The reason for this is that information about the state of the system is stored on the filesystem. This includes simple temporary files that indicate that a specific program is running, as well as named pipes and sockets. For this reason, a live CD contains a special file that contains another filesystem within the CD's filesystem.

This special file usually contains a compressed filesystem that is decompressed to random access memory (RAM) at runtime. Numerous different compressed and uncompressed file systems could be used and no clear standard has emerged at this time. The specific filesystem used will play a role in the performance of the web server, but should not have a significant impact on the maintenance aspect of the larger live CD system. We used *zisofs* [26] in the proof of concept implementation and will assume that *zisofs* is used for this discussion.

A *zisofs* is a compressed, read-only filesystem which dynamically decompresses each file on access at runtime, circumventing the need to keep the whole decompressed filesystem in memory. This accounts for the read-only part of the filesystem, but still leaves the writable part of the filesystem to implement.

For the writable part, a RAM based filesystem is used. It might be possible to use persistent media under certain conditions, but for the purposes of this discussion, no persistent storage can be allowed. This filesystem will store the runtime information and state of the system, including device nodes (/dev), lock files (/var) and temporary files (/tmp).

By rebooting the machine, all information on the RAM filesystem is discarded, thus resetting the machine to the known-good state.

4.4 Operation of the live CD

Operating without any persistent storage poses a unique set of problems to system administrators. Basic functions such as logging have to be adapted in order to have a workable system.

In the specific case of logging, a good option would be to log to a remote logging host [27]. Remote logging is good practice, even when live CDs are not used, as it provides consolidated logging information that simplifies the audit process.

Other tasks that system administrators consider part of their daily work are also affected. In all but the strictest environments, system administrators make routine changes to the system, for example the patching of security holes, without external intervention. This can no longer be the case when each live CD change has to be checked by a quality assurance panel and subjected to change control.

More research is needed to determine other challenges in maintaining a live CD based infrastructure.

5 DISCUSSION

It is a tedious process to build a live CD. Without good change control and quality assurance systems and policies, other read-only strategies can be used to achieve adequate security.

A second drawback is that typical write-once-read-many media have much higher access latency than ordinary hard drives. The largest part of the latency is due to the time it takes to spin the CD up to speed. This is a mechanical limitation, but the associated wait time is easily avoided by keeping the CD spinning at the correct speed. The conditions that the system will be performing under, is expected to require constant access to the media and will keep the CD rotating at the required speed, thus alleviating the spin-up problem.

CD latency will have low impact on web users who are using slow network connections, like a modem connection, since the CD access time will be a minor component of the total page load time. Aggressive caching also reduces the wait time dramatically, but is dependent on the amount of available RAM in the server.

Frequently resetting the system will also frequently render it unavailable to users. This is a dilemma in that it compromises availability of the machine, which in effect cancels the improvement in integrity. Fortunately the problem of availability can be solved by making use of failover systems. There are at least three different kinds of failover systems, including dedicated hardware, software that runs on the web server, or a software/hardware combination. The type of failover system used should not affect the outcome of the system as a whole, but this has not been researched in depth and is left to future work. The reliability of the failover system is also a concern, but beyond the scope of this project.

If we assume that a reliable failover system is used, it is possible to reboot the web servers one by one in an interleaved fashion, without affecting the availability of the system as a whole. The end-result is a system with somewhat reduced availability, but with substantially improved integrity protection.

Scalability of the system depends almost exclusively on the scalability of the failover system. For example, if five web servers are being used and one server is rebooted every 15 minutes, there will be at least four web servers available at any given time. The overhead of frequently having one server unavailable scales well as the number of servers increases.

The action of rebooting the machine can be scheduled to happen at any given time, however the schedule has to be planned when the live CD is built, since this is part of the permanent behaviour of the server. The problem that this poses is that different servers will need different schedules, which calls for different configuration files and ultimately, a unique live CD for each server. It might be possible that the server can be programmed to determine its schedule at runtime, by using information received from the network, but this has not been attempted.

6 FUTURE WORK

The work discussed here investigated some benefits of using a live CD based infrastructure with a strong focus on integrity. At the same time, several issues were uncovered that were not addressed in full. These issues are listed here for completeness and should receive attention in future research.

Different types of failover systems should be compared to determine the effect of frequently unavailable servers. In a live CD based infrastructure, one type of failover system might be more suitable than another.

Attention should also be given to auto-configuration of servers from the network. In some scenarios it might be feasible to use the Dynamic Host Configuration Protocol (DHCP) to configure servers. By using DHCP, identical live CDs can be used for each of the servers. However the security implications should be carefully considered.

Challenges in maintaining a certain infrastructure often emerge once it has been implemented and is in full scale use. Undoubtedly there are operational details not discussed here that will emerge in future and that should be discussed once they are known.

Live CD technology is a relatively recent development and there are many uses that could still be explored. This paper opens the exploration in the use of live CDs in a server environment, but only discusses the use in a web server context. We imagine a wider application of the technology, but leave the exploration to future work.

Possibly the largest drawback of the proposed solution, is that it does not provide an answer to automated attacks. Once an attacker has found a way to temporarily disable one web server, an automated attack could be launched that would disable each server as soon as it becomes available. This kind of attack is more traceable than a once-off intrusion and is not typically associated with opportunistic exploits.

Another aspect that was not studied in depth is the expected lifetime of CD media and drives. Further study is needed to determine the cost and manageability implications of this aspect.

7 CONCLUSION

Live CD technology is an emerging technology that is useful in a server scenario when integrity receives top priority. The added security layer of using read-only media provides a large deterrent to web vandals and should eliminate most attempts at web defacement.

While security holes could still allow an attacker to disable the server, attackers will have to develop new techniques that do not rely on writing to read-only parts of the filesystem or overwriting system binaries. This rules out the possibility of an attacker covertly penetrating the server and installing a backdoor or other malicious software.

Integrity protection of read-only media is inherently greater than that which can easily be achieved through alternative techniques. It is easy to accept that CD-R media, by design, does not support modification and that a CD writer is needed to write to it. It is impossible to change the content of the media, using only ordinary CD reader devices.

Only one application of live CDs has been discussed in this paper. Different applications for live CDs exist and there are many aspects of this technology that requires further investigation.

References

[1] M Bishop. *Computer Security, Art and Science*. Addison Wesley, Boston, MA, USA, 2003.

[2] C Liu, J Marchewka, J Lu, and C-S Yu. Beyond concern — a privacy-trust-behavioral intention model of electronic commerce. *Information and Management*, January 2004. doi:10.1016/j.im.2004.01.003.

[3] H Nam, J Kim, S J Honga, and S Lee. Secure checkpointing. *Journal of Systems Architecture*, 48:237–254, March 2003. doi:10.1016/S1383-7621 (02) 00137-6.

[4] attrition.org. Attrition mirrored sites, May 2001. `http://www.attrition.org/mirror/attrition/`.

[5] attrition.org. Individual category — South Africa (za), May 2001. `http://www.attrition.org/mirror/attrition/za.html`.

[6] Project Gamma. Defaced web site archive. `http://defaced.projectgamma.com/`.

[7] Anonymous. Approximately 35 South African Web sites cracked simultaneously. *SA Computer Magazine*, 12(5):12, May/June 2004.

[8] B B Madan, K Goeva-Popstojanova, K Vaidyanathan, and K Trivedi. A method for modeling and quantifying the security attributes of intrusion tolerant systems. *Performance Evaluation*, 56:167–186, March 2004. doi:10.1016/j.peva.2003.07.008.

[9] J Jacob. The basic integrity theorem. In *Computer Security Foundations Workshop IV*, June 1991.

[10] S Foley. A nonfunctional approach to system integrity. *IEEE Journal on Selected Areas in Communications*, 21:36–43, January 2003.

[11] P Meerwald and S Pereira. Attacks, applications and evaluation of known watermarking algorithms with Checkmark. In *Proceedings of SPIE, Electronic Imaging, Security and Watermarking of Multimedia Contents IV*, volume 4675, January 2002.

[12] R Zalensky. Firewall technologies. *IEEE Potentials*, 21:24–29, March 2002.

[13] B Kaliski. The MD2 message-digest algorithm. RFC 1115, April 1992. `ftp://ftp.rfc-editor.org/in-notess/rfclll5.txt`.

[14] R Rivest. The MD5 message-digest algorithm. RFC 1321, April 1992. `ftp://ftp.rfc-editor.org/in-notess/rfcl321.txt`.

[15] M A Gips. Is your Web site a hacker's delight? http://www.securitymanagement.com/library/000713.html.

[16] Tcsecure. `http://www.tcs-sec.com/products/tcsecure/eserver/eserver.html`.

[17] Read-only Linux. `http: //www.ultimeth.com/linux/`.

[18] M Eisler. NFS version 2 and version 3 security issues and the NFS protocol's use of RPCSEC_GSS and Kerberos V5. RFC 2623, June 1999. ftp://ftp.rfc-editor.org/in-notes/rfc2623.txt.

[19] ... Security Bulletin MS03-026, Microsoft, September 2003. http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx.

[20] S M R Srinivasan. RPC: Remote Procedure Call protocol specification: Version 2. RFC 1831, August 1995. textttftp://ftp.rfc-editor.org/in-notes/rfc1831.txt.

[21] D S Rosenthal. The secret to strong security: less reliance on secrets. http://lockss.stanford.edu/lockssecurity.html.

[22] List of live cds. http://www.frozentech.com/content/livecd.php.

[23] L D Stein. *Web Security: A Step-by-Step Reference Guide*. Addison Wesley, Reading, MA, USA, 1998.

[24] D Robbins. Catalyst reference manual. texttthttp://www.gentoo.org/prof/en/releng/catalyst/reference.xml.

[25] Linux standard base. http://www.linuxbase.org.

[26] H P Anvin. http://www.kernel.org/pub/linux/utils/fs/zisofs/.

[27] Jaco Kroon and Martin S Olivier. An approach to build a fortified network logger that is resilient to cracking attempts. In *Information Security South Africa (ISSA2004)*, Midrand, South Africa, June/July 2004.