UNLOCKING THE ARMOUR: ENABLING INTRUSION DETECTION AND ANALYSIS OF ENCRYPTED TRAFFIC STREAMS

Barry Irwin

Computer Science Department, Rhodes University

b.irwin@ru.ac.za

Rhodes University

Box 94

Grahamstown

6140

South Africa

ABSTRACT

In the interests of maintaining end to end security, increasing volumes of information are being encrypted while in transit. Many organisations and users will make use of secure encrypted protocols for information interchange given an option. The very security that is provided by these transport protocols, such as IPSEC, HTTPS and SSH also acts against the security monitoring of an organisation's traffic. Intrusion detection systems are no longer easily able to inspect the payload of encrypted protocols. Similarly these protocols can potentially be difficult for security and network administrators to debug, validate and analyse.

This paper discusses the need for a means of a trusted third party being able to unpack encrypted data traversing a network and a proposes an architecture which would enable this to be achieved through the extraction and sharing of the appropriate encipherment tokens, based on the assumption that an organisation has legitimate access to one side of a communication entering or exiting its network. This problem also has particular relevance to honey-net research and for investigators trying to perform real-time monitoring of an intruder which is making use of such a protected protocol. A proof of concept implementation of the proposed architecture is also discussed.

KEY WORDS

Encrypted Traffic, Intrusion Detection, Traffic Analysis, Cryptography, Traffic Decryption

1 INTRODUCTION

The increase in the use of cryptographic protocols such as Secure Shell (SSH) [1] and IPSEC [2] has led to a problem with traditional forensic and monitoring methods. Intrusion detection systems rely on being able to access the data payloads of packets in transit in order to be able to ascertain whether the payload is malicious. In addition Systems Administrators have traditionally also made extensive use of tools such as `tcpdump` for the monitoring and debugging of network connections. While the existing tools are still able to monitor the flow of packets, they are not able to perform any content inspection due to the encrypted nature of the payload.

The increased prevalence of Virtual Private Networks (VPN) connectivity and for non-VPN traffic the use of encrypted transports such as SSH and Secure Sockets Layer (SSL) have rendered many traditional network debugging and monitoring tools useless due to the encryption used for payload protection. All these protocols have been engineered to be resistant to attack and provide strong encryption and have undergone extensive public scrutiny. Many users however have a legitimate need access to the plaintext payloads of these protocols as part of normal debugging and security monitoring procedures [3].

Access to this information may also be required by Law enforcement, either for real-time wiretaps or for post processing of captured data. In South Africa the Electronic Communications and Transactions (ECT) [4] and Regulation of Interception of Communications and Provision of Communication-related Information (RICPIC) [5] Acts govern access to traffic and the related use of cryptography. Hence in order to comply wholly with these acts, newer tools and methods are required. The current state of affairs would leave most organisations unable to comply with the requirements stipulated in these acts.

Forensic analysis of data collected by researchers from Honey-net systems is an important part of the research process in identifying and analysing new attacks on Internet systems. Additionally, operators of live Honey-pot systems have a need to be able to monitor communications to and from their systems – something easy to perform in the days when telnet was prevalent, but near impossible with new and increasingly prevalent technologies like SSH. The ability to be able to watch intruders without them being aware is of paramount importance in being able to understand their operations, activities tools, and exploit techniques. Stoll describes this ability to have been pivotal in the investigations carried out (using hardware intercepts rather than software systems as proposed in this paper) in *The Cuckoo's Egg* [6].

Current solutions are very limited in their domain, flexibility and capabilities, requiring customized kernel modules which allow duplication of data streams for snooping such as the Honeynet projects' as Sebek [7]. The alternative to this is that in order to gain access to plain-text traffic the encrypted tunnel or protocol is terminated on a front-end device (for example IPSEC and HTTPS respectively). This is not always suitable or desirable as it provides a compromise of the end-to-end security expected by users of a service. It may also not be desirable to have plaintext traffic flowing within a network DMZ, as compromise of a server within this region could result in unfettered access to the plaintext traffic.

In terms of decrypting captured traffic with an enciphered payload, `tcpdump` currently offers a method for decapsulating, and decrypting IPSEC ESP (Encapsulated Security Payload) [8] traffic, through the provision of the key used to encipher the traffic stream by an administrator. Recovering the key used differs depending on the operating system or device used for the tunnel termination. Currently there are no solutions available for recovering traffic from SSH and TLS sessions without modification to the servers. There are a number of commercial products available which offer the ability to decrypt SSL encapsulated HTTP traffic through access to the private keys.

This architecture is designed based on the premise that it will be operated by a party who is privy to the secured communication (either as initiator or recipient), and thus has a legitimate and lawful right to access the plain-text data extracted form the secure channel. This data is already being utilised by their systems in plain-text form, the difference being that it is not in a readily available format outside the application runtime environment. For the purposes of this proposed architectural view, this deals with the legal issues surrounding wiretapping, and other electronic monitoring.

2 PROPOSED ARCHITECTURE

The proposed architecture for enabling near-real-time decryption and analysis of enciphered traffic for Intrusion Detection or Forensic purposes involves the development and placement of a cipher key vaulting system, and optional real-time Decryption service server as shown in Figure 1 below. These are linked to data collection agents on the systems for which traffic is to be analysed.

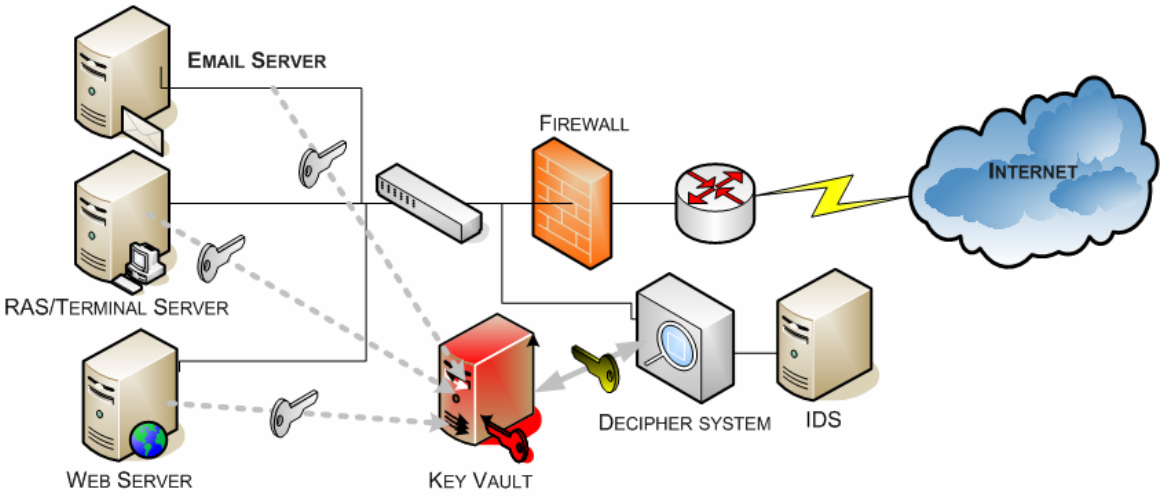

Figure 1: Proposed Collection and Decipherment Architecture

While this architecture is primarily aimed at providing a means for enabling IDS systems to analyze enciphered traffic, and to allow for real-time or near-real time monitoring of encrypted streams on honey-net systems, the architecture should be scalable to a number of different scenarios, including post incident traffic analysis, and day-to-day access to encrypted streams and tunnels for maintenance, monitoring and debugging purposes.

2.1 Modes of Operation

It is anticipated that the proposed architecture will be implemented in one of two operational modes. The most common mode of operation is likely to be in offline or passive mode, with the online/active mode only being used for the monitoring of specific traffic or for debugging purposes, although with suitable refinement, the latter mode should be able to become the standard operating mode in some scenarios.

2.1.1 Offline/Passive Mode

In offline mode the cipher keys are recovered and stored on the central vault server, rather than direct deployment to the decipherment system. When traffic captures requiring decryption need to be processed, the appropriate keys are recovered from the vault server, and loaded onto the

decryption server, which will then further process the data producing appropriate plaintext output for further analysis.

This mode requires that the network traffic data is collected and stored prior to decipherment and analysis. An example of this could be the debugging of a higher level protocol over an IPSEC Virtual Private Network (VPN). A traffic capture could be stored, and then deciphered with the appropriate key resulting in un-encrypted IP datagrams which can be analyzed using standard existent packet and protocol analysis tools. Post incident analysis of incident related traffic would fall into this category.

2.1.2 Online/Active Mode

An alternative is to use online mode of operation. Rather than the keys only being stored on the vault server, keys can be passed onto the decipherment server in near-real time. This would allow for real-time decipherment of the traffic. It is unlikely that all traffic would be deciphered, and rather only certain sessions or classes of traffic specifically selected for handling. This mode is what is likely to be used for real-time monitoring of communications between a honey-pot host or network and the Internet at large. Suitable precautions need to put in place to protect the transmission of key data to the decipherment server, and in particular the plaintext traffic that is then passed to the Intrusion Detection Server as compromise of either of these systems, or of another privy to the communication channel could potentially allow an attacker easy access to the confidential clear text data stream.

2.2 Data Collection

This system requires the installation of cipher key collection agents on the target server systems. These agents can take the form of stand-alone applications (such as in the case of collecting IPSEC cipher keys on FreeBSD platforms) or as an integrated patch to appropriate services (such as to the OpenSSH Secure Shell Daemon - sshd). These agents are responsible for the provision of the appropriate cryptographic key tokens to the vault server and ultimately the decipherment service. Alternately where the server side software can not be patched and an agent cannot be constructed to otherwise gain access to the internal cryptographic information, the existing current solution of a cryptographic proxy placed inline and used to broker the encipher connection thereby gaining access to the key can be integrated into the architecture, but is less than ideal.

Some cryptographic information can be recovered by purpose built agents. Two common examples where agents could be employed are HTTPS/SSL communications and certain IPSEC implementations. In the case of the former, an agent with access to the private portion of the key contained in the standard X.509 certificate would be able to snoop and decrypt the initial key negotiation between the web server and client web browsers. From these data packets, the nonce value selected by the client can be recovered and ultimately used to recover plaintext for the remainder of the client's session. In the case of IPSEC, on some platforms the admin user is able to easily access the encryption and hashing keys use for a particular IPSEC Security Association (SA). In some cases this is as simple as parsing the output of an existing system command. Methods can also be put in place to recover the key when the SA expires and a new one is negotiated. These methods can be regarded as non-intrusive, as no actual system code needs to be modified and are the preferable method of collecting data. This may however not be desirable, particularly in the situation where a honey-pot system is being run and these agents could potentially be identified and scare an intruder off, effectively rendering the honey-pot system ineffective. Methods for hiding these processes are however beyond the scope of this paper.

Where stand-alone agents cannot be used, due to the cryptographic information not being readily available, a more intrusive approach may be required. This approach involves the direct integration of both a complete extraction and reporting agent into existing application source, or

just a stub agent responsible for extracting the cipher key from within the running process. The latter would be coupled with a stand-alone reporting software agent which would facilitate the reporting and upload of the key to the vault server. This method is however only available for server systems for which the source code is readily available – fortunately software of this type constitutes a large percentage of the services running on Internet servers. Closed-source systems could potentially still be accessed in this manner through the construction of appropriately enhanced versions of shared libraries, or even modification of underlying system cryptographic libraries. This approach is likely to be preferable for systems which require covert monitoring of communications, as other than the transmission of the key messages, the embedded agent is concealed.

2.3 Time keeping

All systems involved in the cipher recovery system are required to maintain accurately synchronised system clocks. This is best achieved using a local Network Time Protocol (NTP) source which itself is being synchronised from an external source such as dial-up to an atomic clock source, or via GPS or similar accurate broadcast signal. This external time source reduces the risk of corruption of system times by an attacker through the manipulation of incoming datagrams from public time servers. Such a service should also be protected internally from spoofing or other manipulation attacks by making use of the inherent protocol security methods such as cryptographic identification of peers.

The value of accurate timing information in the post incident analysis of an intrusion incident has been well established, and systems for accurate time synchronisation particularly with reference to Intrusion detection and data collection systems has been discussed in [8]. Precise timing information will allow for the accurate identification and correlation of information from multiple sources, and the identification of specific streams of data to be analysed even where there are high traffic loads. In the case of the proposed system, accurate timing is needed in order to accurately and uniquely identifies the connection for which a cipher key is retrieved and stored.

2.4 Message Transport and Communication

Agents need to be able to communicate the cipher key information to a store for archiving. Two aspects however need to be borne in mind. The actual means of transmission should cause as little interference as possible to the host system. Secondly, for agents that are integrated into existing systems at the source level, the impact of large blocks of code on their overall primary function should also be considered, particularly where public-key cryptography is involved. The key store can either be local to the system being monitored, or preferably a secured remote system. Remote system communications should preferably be over a dedicated serial or LAN link to minimise the possibility of an intruder gaining access to the recovered cipher keys, and thereby being able to further compromise monitored systems and communications. Use of a dedicated link – particularly serial – is also less likely to arouse suspicion of being monitored on honey-pot systems. In the event of transmission over a non dedicated link encryption should be regarded as mandatory for protection of the messages sent. Depending on the perceived impact of messages being compromised, either the entire message payload, or just the cipher key can be encrypted. Given the overall small size of the message payload, little is likely to be gained from only encrypting the key.

The actual transport to be used for the transmission of data over an IP network is currently unspecified, with merits for the use of both UDP and TCP transports. The choice of protocol is currently dependant on the requirements, and the volume of traffic to be processes. UDP transports are simple to implement at the agent level, and are quick to send, but comes at the ‘cost’ of non guaranteed delivery, and no receipt of confirmation at the transport level. TCP based transports on

the other hand are more reliable; provide receipt confirmation and retransmission, at the price of increase overhead establishing a connection. This however can be amortised over a large number of messages that can be sent over a single connection in a given period of time – given that the agent establishes a persistent connection to the vault server. With regard to visibility of the transmissions a UDP based solution is likely to be less noticeable to an intruder than a persistent TCP connection to another host – unless suitably disguised.

The message format proposed is illustrated in Figure 2 below. The first two lines provide enough information to uniquely identify each connection, with an associated cipher key. The portion shown in bold is what can be regarded as being confidential. The final portion of the message format is intended to provide a means to extend the message with protocol specific information. For example, given a message reporting the cipher key for an IPSEC tunnel, this region could be used to hold the Security Association (SA) number for the particular tunnel, as well as the HMAC hashing key, the latter of which could again be regarded as confidential and enciphered. The vault server would be required to potentially implement specific handlers for each protocol message. It is anticipated that some form of XML mark-up with preferably with suitable extensions for handling the encryption could be utilised to facilitate easy processing.

```
<Protocol> <timestamp>
<SourceIP> <SourcePort> <DestinationIP> <DestinationPort>
<key> <CipherProtocol>
<additional Protocol Specific Information>
```

Figure 2: Proposed Agent-Vault server Message format

In all cases encryption of the messages is recommended, and it would make sense to utilise public-key ciphers as this would entail only having the public portions of the key available to the agents. This would negate the problem of the key to decrypt communications being compromised in the event of a monitored system, and agent being compromised by an attacker, should symmetric algorithms be used. The asymmetric nature of public-key systems ensures that only an authorised party can access the transmissions. It is assumed that suitable measures are put in place for the protection of the private key component.

On the message being received by a store (local or remote) the message can either just be stored as is or can be further processed. The issues surrounding message storage are discussed in the next section.

2.5 Message Storage and Vaulting

In almost all deployment scenarios it would be desirable to protect the recovered cipher keys in a secure manner and to limit access to these keys. Existing well proven public-key encryption protocols allow for this to happen, with the messages containing the cipher keys being encrypted with the public portion of a key, thus providing an effective one-way trapdoor for the data that can only be unlocked in a authorized instance.

Should a local message story be used on the monitored system, this should be protected from compromise should an attacker be successful in compromise of the host system. Messages could potentially be encrypted a second time with a machine specific key ($KeyM_{public}$), again with the private portion ($KeyM_{private}$) suitably secured. A remote repository could follow a similar method. However for the stores to be useful the information inside should be quickly searchable and the recovered cipher keys rapidly accessible, especially if being used for real-time decipherment and monitoring of traffic.

The proposed solution to the first problem is to create a suitable key vault. The suggested architecture has the Vault server not having any external or internal network connection other then when required. Connection should be via dedicated serial links or similar connections. Great care should be taken to ensure that the Vault server cannot be compromised, as this could potentially result in a compromise of all traffic. The preferable form of storage would allow for clear-text storage of the non-confidential values contained in the agent messages. This would allow rapid searching based on public data fields such as IP address or Source and destination ports. The confidential components such the recovered cipher key and algorithm can be protected in the data store again via asymmetric encryption. The most suitable format for this would be some kind of relational database system. This data store would also need to be regularly maintained to remove information that has expired, as the majority of the stored cipher keys would have only a limited temporal window of usefulness. The data retention policy for this store would be dictated by the specifics of the application for which it was being used, and by organisation and operational constraints. It is envisaged that local system stored would be imported into the key vault on a batch basis.

One potential means for adding greater security to the data contained within the key vault would be to make use of a fragmented private key component. This feature is available in most PGP implementations, based on the Blakely-Shamir Key Splitting Algorithm [9]. Keys protected in such a manner allow for distribution of a private key ($KeyA_{private}$) as a number of key fragments among *n* parties ($KeyA_{frag1}$, $KeyA_{frag2}$ … $KeyA_{fragn}$) with *m* parties required to be present in order to constitute a successful reconstruction of the private key. The algorithm allows for *m* to be in the range 2 to *n*. Use of such a multi-partite private key would ensure that multiple parties were required to be present for the decipherment and reduce the risk of collusion, or compromise of the public component of the key. A further enhancement to this would be that the cipher key and other confidential information is encrypted with the public component ($KeyA_{public}$) corresponding to the fragmented private key ($KeyA_{private}$). The message as a whole would be protected by a key pair (KeyM) specific to the system being monitored, or alternately a dedicated key pair for the vault system (KeyV). The protection of the confidential information by a dedicated key-pair ensures that the information never exists as plaintext, even during processing, thus further guarding in the event of a vault server compromise. This would be particularly important when confidential information such as collected on e-commerce or online banking servers could potentially be recovered. In the event of a cipher key needing to be retrieved, the appropriate parties would need to concur that this was a legitimate extraction and proceed with the extraction. This model is obviously not suitable for use with near-real time application of this overall architecture.

2.6 Other Security Considerations

A number of security issues need to be considered during the implementation and operation of a snooping and decryption system such as the one described in this paper. While the primary goal of such a system is to enhance the overall security of a network of monitored systems, like the majority of security tool, if used improperly the consequences can be dire. Great care should be taken in securing all monitored systems, and especially the Vault and decryption systems to the best current standards. This may in some cases, such as honey-nets be counter-intuitive to the purpose of the honey pot systems. This is particularly problematic where a honey-pot host is deployed on a live production DMZ. In such a case the potential for improper use of deciphered data needs to be evaluated.

The security of the decrypted data also needs to be ensured, and it is recommended that the IDS system processing this data has no external network communications, to prevent possible leakage and compromise of this data. The keys needed to produce such plain-text data streams also need to be properly protected as discussed above to mitigate the risks and potential for improper use.

Analysis of the communications traffic between agents and the vault server, or vault server and decryption server can potentially also lead to security problems. While this traffic can be protected using cryptographic methods as discussed above, an intruder will still be able to potentially determine that there is some unknown communication going on. Solutions for mitigating this are varied, and range from the ultra-paranoid option of using stenographic hiding techniques under the guise of legitimate protocol traffic, to just accepting that this is a possibility, but that the average intruder and particularly 'script kiddies' will be either unaware of the existence of traffic, or unable to determine its purpose.

3 CASE STUDIES

The following sections provide brief case studies of commonly used encrypted protocols and a short discussion regarding how the cryptographic tokens could be extracted by an agent.

3.1 SSH

Considering the open-source OpenSSH implementation of this protocol, both the server and client components can be modified to provide the actual cryptographic token negotiated for the session. This ability is important as it would allow monitoring of both incoming connections (server) as well as connections made to an external server (client) commonly used for remote logins and secure file transfers In order to be compatible with both SSH version 1 [1] and version 2 [11] protocols, two options are available, the first is a man-in-the-middle attack which could be trivially implemented using tools such as Dug Song's dsniff [12] package. However this can potentially be quite easily detected, particularly in the case of a honey-net where an intruder's traffic needs to be monitored. The second option is to modify the SSH server in order to provide the ephemeral symmetric encryption key that is agreed upon during the negotiation part of the SSH algorithm. This method also allows for the recovery of the key when SSH version 2 is used as a Diffie-Hellman exchange is used, whereby the session key is not transmitted over the wire as in version 1 [11]. This is intended to either be logged to file or a secure device, or used in real-time for the monitoring of traffic flowing over the SSH connection. Sandstorm Communications' NetIntercept product [12] has similar functionality to that proposed above implemented, but the details have not been disclosed.

3.2 IPSEC

The use of IPSEC for the construction of secure Virtual Private Networks has increased dramatically in recent years, as the maturity of vendor implementations increases, along with the inter-vendor interoperability. Many network administrators make frequent use of tools such as tcpdump, or similar packet and traffic analysers for the routine debugging and configuration of network links. The increasing use of IPSEC can make this task much more difficult if not near impossible in some cases. Dependant on the platform and implementation used there may or may not be a way to easily monitor the plaintext traffic traversing the VPN. While tcpdump does offer a facility for the decryption and decapsulation of IPSEC ESP traffic, it still requires that the right cryptographic key be provided.

A tool needs to provide a means for extracting the specific encryption key and cryptographic algorithm for an IPSEC traffic flow (SPI value) used for ESP traffic [10]. This list needs to be periodically updated as keys are re-negotiated. These keys can be used to decrypt the packet payloads using standard symmetric cryptography. The process should be automated and as transparent as possible to the user. The ideal is that the solution will be able to produce a libpcap compatible output stream in order that any monitoring tools or Intrusion Detection systems be able to process the traffic as per normal, with the highest level of transparency possible. Currently the author session keys can only be recovered for one open source IPSEC stack implementations – the KAME stack for the BSD family of operating systems it is envisaged that the Free/SWAN stack for

Linux should be modifiable in a similar manner. Proprietary systems may prove more difficult to enable cipher key recovery for.

3.3 SSL/HTTPS

HTTPS and the underlying SSL protocol is most commonly used for secure web communications to e-commerce, online banking and other websites that require a communication channel where the identity of the remote server can be verified by a client. This protocol has also been in widespread use far longer than many of the other secure communications protocols. The secure nature of this communication prevents the use of traditional IDS methods for detecting and possibly reacting to incoming attacks or probes which take place over this channel. The most concerning of these is the potential for a new generation of Internet worms that may exploit web server weaknesses but over a secure channel.

A similar approach to that for the SSH protocol will be implemented, where the session key can be recovered through the decryption of the session initiation traffic using a copy of the web server's private key [14]. The session key can then be used to decrypt the actual flow to produce plaintext. One potential problem is that SSL sessions are generally relatively short-lived in nature in comparison to SSH and IPSEC sessions which can typically last for hours, resulting in quite a high workload for the collection agent, and particularly for the decryption service.

3.4 Other Protocols

A number of other protocols such as SMTP and IMAP make use of encryption through implementation of the TLS protocol [15, 16, and 17]. While these have not currently been investigated in detail, they have not been discounted, and it is hoped to provide a generic means of decrypting these protocols similar to that used for the processing of SSL protected HTTP Traffic in order for authorized parties to be able to analyze the traffic flows in plaintext.

4 HONEY-NETS AND HONEY-POT APPLICATIONS

The potential positive outcome for the application of a key recovery and inspection architecture such as that described in this paper could be of great benefit to security researchers and operators of honey-net systems. The prime goal of these systems is to actually be able to entrap an intruder for careful study. With the increased prevalence of remote access protocols making use of encryption – SSH in particular, the honey-net operator can now gain access to the plaintext intruder command stream as it is sent, a valuable tool which has not been readily able to be used recently since the majority of systems no longer make use of the plain-text Telnet protocol.

5 CONCLUSION

While this work is currently at an early stage of development and conceptualisation, it is hoped that an automated method of extracting relevant data for analysis can be developed. The perceived outcomes are a set of tools, and patched for the modification of system servers (where needed) to allow Intrusion Detection Systems, network and system administrators, honey-pot operators and forensic investigators to access encrypted protocols in simple manner for further analysis and monitoring. A the same time these tools should be reliant on privileged access to a system, and be relatively undetectable – particularly in the case of modified servers for use on honey-pot systems so as not to alert potential intruders to the fact they are being monitored.

The current state of this work is that the architectural design has been completed, and a preliminary assessment of the security risks introduced by the deployment of such architecture

assessed. Initial work has been done on the actual decipherment of both SSH and SSL traffic at the proof of concept level. Currently investigation has been limited to open source server implementations, although once the fundamental process for agent construction has been established, proprietary systems will be investigated.

6 REFERENCES

[1] T Ylonen. "*SSH---Secure login connections over the Internet*". In Proceedings of the Sixth USENIX Security Symposium, pages 37-42, July 1996.

[2] S. Kent, R. Atkinson (1998, November) "*Security Architecture for the Internet Protocol*" IETF RFC 2401

[3] V. Corey, C. Peterman, S. Shearin, M. S. Greenberg and J. Van Bokkelen (2002 December) "*Network Forensics Analysis*". IEEE INTERNET COMPUTING pages 60-66

[4] South African Government (2002) *Electronic Communications and Transactions Act*, No. 25 of 2002

[5] South African Government (2002) Regulation of Interception of Communications and Provision of Communication-related Information Act (No. 70 of 2002)

[6] C. Stoll (1989) *The Cuckoo's Egg*. Pocket Books.

[7] Honey net Project (2003, November) "*Know Your Enemy: Sebek - A kernel based data capture tool*". Available: http://www.honeynet.org/papers/sebek.pdf

[8] Forte D "THE 'ART' OF LOG CORRELATION: Tools and Techniques for Correlating Events and Log Files." In Proceedings of InfoSec South Africa 2004, June 2004.

[9] Schneier B "Applied Cryptography" Second Edition. Wiley 1996.

[10] S. Kent, R. Atkinson (1998, November) "*IP Encapsulating Security Payload (ESP)*" IETF RFC 2406

[11] T. Ylonen and C. Lonvick, Ed. (2004, June) "*SSH Protocol Architecture*" IETF Draft. Available: http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-16.txt

[12] D. Song (2002) Dsniff Toolkit. Available: http://www.monkey.org/~dugsong/dsniff.

[13] M Friedl, N Provos and W A. Simpson (2003, July) "*Diffie-Hellman Group Exchange for the SSH Transport Layer Protocol*" IETF Draft. Available: http://www.ietf.org/internet-drafts/draft-ietf-secsh-dh-group-exchange-04.txt

[14] Sandstorm Enterprises "*NetIntercept 2.0 White Paper*" Available: http://www.sandstorm.net/downloads/netintercept/ni-2-0-whitepaper.pdf

[15] E. Rescorla (2000, May) "*HTTP Over TLS*" IETF RFC 2818

[16] P. Hoffman (2002, February) "SMTP Service Extension for Secure SMTP over Transport Layer Security" IETF RFC 3207

[17] C. Newman (1999, June) "*Using TLS with IMAP, POP3 and ACAP*" IETF RFC 2595

[18] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen and T. Wright (2003, June) "*Transport Layer Security (TLS) Extensions*" IETF RFC 3546