MANAGEMENT, PROCESSING AND ANALYSIS OF CRYPTOGRAPHIC NETWORK PROTOCOLS

1 INTRODUCTION

This paper describes research in the field of cryptographic protocols, currently being performed in the REMOVED. While this research is not entirely novel, as it makes use of elements from existing research in the detection of encrypted applications [2], we consider a generic solution to the given problem of analysing encrypted traffic with the intention of later extension to provide support for multiple protocols and performing further analysis than application detection. This paper provides a starting point for further research into the evaluation and analysis of supposedly secure applications and suggests an outline for the development of a framework which could perform this.

Cryptographic protocols are a vital component of information security [9] as a means of securing modern networks against would-be attackers by providing data integrity, encryption and authentication to network traffic at the transport layer [12]. Sensitive information, such as banking details, that transverses networks will most likely do so through an encrypted tunnel provided by the cryptographic protocol; it is thus imperative that both the protocol itself is secure and the applications use of the protocol is correct and sensible. A recent paper by Lee *et al.* shows that in a study of over 19000 web servers, 98.36% of the servers provided support for TLS and 97.92% provided support for SSLv3.0 and 85.37% provided support for SSLv2.0 [4]. These statistics serve to show the prevelance of SSL/TLS and the need to support these protocols.

We now present cases for the need for such research and the development of a framework that allows for the decryption of encrypted traffic.

HTTPS has become prevalent as a means to communicate with a web server securely; however if an attacker were to use HTTPS as a means to perform an attack, it becomes difficult to detect such an attack due to the encrypted nature of the traffic. It would be useful if a system existed to decrypt this traffic and then perform analysis. This is highlighted by work done by Marklinspike [5] in developing a tool, SSLStripper, that removes the secure components of a connection allowing for a new form of MITM (man in the middle) attack where the user believes that his connection is secured (using HTTPS) but in reality messages are passed through HTTP, and are intercepted by a third-party. Furthermore the SANS institute announced “Increasingly Sophisticated Web Site Attacks That Exploit Browser Vulnerabilities - Especially On Trusted Web Sites” as the top security menace in the “Top Ten Cyber Security Menaces for 2008” with “Web Application Security Exploits” in 8th position [7].

Wang *et al.* [14] comment that in the long term, software development cannot afford to consider implementing security only after the application has been developed or late in the development cycle as irreparable security compromises may already exist and that attempts to correct them would require significant resources. Further we consider that security is one of the core metrics in McCall’s Software Quality Checklist [1]. However, software development is notorious for being over budget and far exceeding its expected completion date; as a result we often find that security is left until late in the development cycle and sometimes even after the application has been built [14]. Often this causes poorly implemented security and this only serves to degrade the quality of the system built as it provides the user with a false sense of security; further an insecure application that passes and receives sensitive information is as equally unusable as an application that fails to meet its specifications in terms of correctness [14]. We could argue that the reason why security is not part of many development cycles in earlier stages is due to the difficulty and tedium of checking the correctness of security [10, 11]. To put this in context, if we consider that between January 2004 and December 2008, there have been 26139 reported security vunerablities [6]. It would be useful if there existed a framework that decrypted data and then provided some analysis on issues pertaining to the implemented security.

The remainder of this paper will consider the construction of such a framework, with the following sections. Section 2 contains related work, while Section 3 provides a brief of overhead of the architecture involved for SSL. Section 5 details the approach to be taken and Section 4 highlights the expected goals on completion.

2 RELATED WORK

The analysis of cryptographic protocols is a subject that has been extensively researched with algebraic models to provide descriptions of protocols and techniques such as, BAN Logic and Running Mode Analysis, to provide formalizations to determine whether a protocol is correct in terms of achieving its goals of authentication and data integrity. Research into security and software development is also widely available for discussion of implementations of security into development lifecycles and evaluation of implemented security mechanisms. Furthermore there are a number of systems which, to some degree, provide some of the features already discussed. In this section of the paper we discuss some of the related work around existing software that could possibly be used to perform the functionality outlined in Section 1. We will consider some research that may be beneficial when considering such a framework.

2.1 Running Mode Analysis

Running Mode Analysis is a technique for the formal analysis of cryptographic protocols. It makes use of conclusions derived from model checking. The central component of Running Mode Analysis involves creating a system including an attacker, a protocol and two parties attempting communication and then discovering all of the possible modes the system can enter. For example, in a three-principal security system there are seven running modes; if we can show that these seven modes do not exist then the protocol is deemed to be safe within the system. When working with complex protocols, such as SSL, it is a matter of decomposing the more complex protocol into a number of smaller protocols and then performing Running Mode Analysis on each of the simpler protocols. This sort of analysis is often done by hand and provides an interesting means of the verification of the correctness of a protocol. In a by paper Zhang and Liu [15], running mode analysis is performed on the SSL Handshake protocol. While it may not be important to perform such an analysis, as such research already exisits; it's important to understand that many protocols are fundamentally flawed and identification of such flaws when providing analysis of application security would be a useful addition.

2.2 Practices in SSL/TLS

It has already been mentioned that cryptographic protocols are a popular method of securing web servers. We need to consider that simply providing support for cryptographic protocols is not sufficient to provide adequate security. Lee *et al.* [4]produce a tool, the PSST (probing SSL Security Tool), to perform analysis of over 19000 web servers employing SSL/TLS. They conclude from their results that in 2006, 85.37% of the over 19000 web servers still provided support for SSLv2.0, a fundamentally flawed protocol due to weakness to Man in the Middle (MITM) attacks, while 66.55% of servers still supported DES-40 encryption even though the US export laws limiting the key length of DES to 40 bits is no longer in effect. It is unwise to still provide support for SSLv.2.0 as its well documented that MITM attacks can force the adoption of a weak encryption protocol like DES-40 creating a large and exploitable vulnerability for brute force attacks. While adaption of new algorithms such as AES, is prevalent, the rate at which old standards are no longer being supported is not sufficiently rapid; it is, therefore, important that these issues are highlighted when performing analysis of a systems security.

2.3 Detection of encrypted applications

The use of libraries such as openSSL provides a means to add encryption to generic traffic; this creates a problem for the analysis of network traffic as the traffic is now encrypted. For example, most common torrent clients provide a means to encrypt traffic or by means of using an encrypted tunnel provided through SSH as a means to avoid the content blocking of p2p applications. This makes it difficult to block or limit certain types of traffic which may be the goal of a network administrator. Bernaille and Teixeira [2] suggest a system for the early recognition of encrypted applications is outlined and developed with a high degree of success in terms of identification of applications within an SSL connection. They take the approach of using specific parts of the TCP payload to identify the SSL connection by studying said traffic in detail and then producing patterns to be used in detection methods. A similar methodology of analysing the TCP payloads could be incorporated into the research topic.

2.4 Related tools

A number of tools exist that provide means to analyse SSL; these include SSLDump and SSLSniffer. SSLdump [8] is an SSL/TLS network protocol analyzer which identifies TCP connections on the chosen network interface and attempts to interpret them as SSL/TLS traffic. When it identifies SSL/TLS traffic it decodes the records and displays them in a textual form to stdout. If given the cryptographic keys involved it can be used to decrypt the traffic passing through. SSLSniffer [3] provides similar functionality as SSLDump with the exception that it can act as a SSLv3/TLS and SSLv2 proxy server. The issue with these sorts of tools is two-fold, they don't provide any security analysis and further they are protocol specific.

3 ARCHITECTURE OF SSL/TLS

It is important to understand the underlying architecture for each of cryptographic protocols for implementation is intended. We will consider the architecture of TLS focusing solely on the Handshake Phase, as it is the most significant to the development of the framework. Extension to other protocols would require similar understanding. Firstly, we consider some of the goals of SSL/TLS as these goals dictate the structure of TLS [12]. TLS aims to provide a secure connection between two parties with interoperability, extensibility, allowing for incorporation of encryption algorithms or hashing functions and efficiency provided by caching. We will consider the basic architecture of TLS as it is very similar to the architecture of SSLv3.0. For our purposes, we need only to consider the Handshake phase of SSL.

3.1 The Handshake

During this phase decisions are made as to what cryptographic parameters are to be used for the actual TLS connection. This include deciding on the protocol version, selecting a cipher suite and performing some secret key exchange.

The client sends a client hello message to the server. The server then possibly responds with a server hello message. If there is no response then a fatal error occurs and the connection is closed. These hello messages establish: the protocol version to be used, session ID, cipher suite to be used, compression algorithm to use, clientHello.random and ServerHello.random. The actual key exchange may consist of up to four messages containing: the Server Certificate, the Client Certificate, the Server Key Exchange and the Client Key Exchange. If the Server Certificate is to be authenticated it is sent after the hello messages phase. Following that the Server Key Exchange message may be sent if necessary. If the server passes the authentication, it may request the Client Certificate (if the client has one and if it is required by the cipher suite). The server then sends a Hello Done message back to the client indicating the end of the Hello Message part of the handshake is complete. The server then waits for a for a client response. If the certificate request message was sent then the client needs to respond with a certificate. The client will then send its Client Key Exchange message with the contents dependant on the public key encryption algorithm chosen. After the exchanges have taken place a Change Cipher Suite Message is sent from the client to server. The client then sends new messages containing the new algorithms and keys. The server responds by sending a Change Cipher Suite Message back with the new keys and algorithms. The handshake is then complete [12].

4 RESEARCH OUTCOMES

The authors intend to develop a framework that could be used to evaluate the correct implementation of security protocols in software and other analysis of encrypted network traffic. In this regard the framework needs to be able to decrypt traffic that has been encrypted by a specific algorithm, further it needs to be able to determine which algorithm has been selected to provide encryption; in the case of SSL/TLS this is a case of inspecting packets sent during the SSL handshake. Once plain-text has been obtained the developer can inspect the payload of the messages being sent was and can use this to perform a form of manual debugging. Seeing as a number of security related parameters can be derived from examining the SSL handshake , it would be useful to alert the user to possible security issues such as the use of keys generated on the Debian platform during the Debian/OpenSSL security breach [13] or suggesting that certain cryptographic algorithm be removed from the cryptographic algorithms supported during negotiation in the case of SSL/TLS. For example 40-bit DES is considered to be extremely insecure as it is weak to brute-force attacks or even suggesting that support of SSLv2.0 is a security risk as SSLv2.0 is well-documented as a flawed protocol. It would also be useful to check the entropy of the cryptographic keys used. Once the system has been developed some form of assessment needs to be performed on the usefulness of such a system; this could be performed by distributing the system to a number of users and collecting feedback or by developing a number of test applications with glaring security flaws and then evaluating the output produced. In this way we can further determine if the system is of any practical use to developers.

5 APPROACH TO RESEARCH

Firstly, we assume legitimate access to the data or network connection and that the private keys used are available. As this system would be used in a legitimate context, there is no reason for the private keys to not be available for use; attempting to recover private keys is outside of the scope of the research context of this paper. The objective is to produce a suitable application framework that permits easy recovery and secure storage of cryptographic keys; including appropriate tools to decapsulate traffic and to decrypt live packet streams or precaptured traffic contained in PCAP files. The authors propose the development of a system to capture packets, filtering for TCP packets only (or to parse a dump files for TCP packets only) mostly likely written as a simple C++ application and making use of *libpcap* or *WinPcap* (implementation dependant). This application then removes the headers of the packet and considers the SSL/TLS handshake that has occurred, so as to recover the public key and cipher suites used. The resultant processing will then be able to provide a clear-text stream which can be used for further analysis. The framework should be implemented for protocols that use the standardized hybrid cryptographic protection system such as IPSec, TLS, SSL 3.0 and SSHv2. An issue of concern is the recovery of the nonce, which could either be retrieved by changing the server applications or more practically by having another trusted system holding a second copy of the private key. An investigation as to how to sensibly store cryptographic keys is also required as they form a central component of this system. Figure 1 illustrates this in diagramatic form.

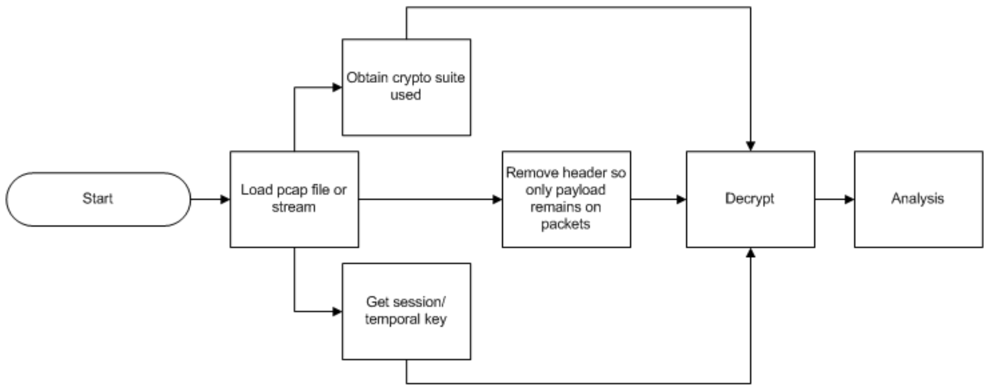
Start
Load pcap file or stream
Obtain crypto suite used
Get session/ temporal key
Remove header so only payload remains on packets
Decrypt
Analysis

Figure 1: Diagram of Proposed System Design

6 CONCLUSION

This paper has discussed the need for a framework to provide a means to decrypt encrypted traffic for debugging needs within software development and also as a means to provide analysis of encrypted traffic and has outlined the system to be developed. At this stage, the research is still in its formative stage. We expect that if the system is developed correctly, and is adopted as a component in system development, that it may provide a standard to ensure correctly implemented security systems and that it would also be useful to network analysts. The system could later be extended for development with other protocols, possibly including IPSec and SSH.

ACKNOWLEDGEMENT

The authors would like to acknowledge the support of REMOVED.

References

[1] Reesa E. Abrams. A checklist for developing software quality metrics. In *ACM 82: Proceedings of the ACM '82 conference*, pages 5–6, New

York, NY, USA, 1982. ACM.

[2] Laurent Bernaille and Renata Teixeira. Early recognition of encrypted applications. pages 165–175. 2007.

[3] Eu-Jin Goh. Sslsniffer. Online: `http://crypto.stanford.edu/~eujin/sslsniffer/index.html`.

[4] Homin K. Lee, Tal Malkin, and Erich Nahum. Cryptographic strength of ssl/tls servers: current and recent practices. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 83–92, New York, NY, USA, 2007. ACM.

[5] Marlinkspike. New tricks for defeating ssl inpractice. 2009.

[6] NIST. National vunerability database. Online: `http://nvd.nist.gov/`.

[7] Alan Paller. Top ten cyber security menaces for 2008. 2008.

[8] Eric Rescorla. Ssldump. Online: `http://www.rtfm.com/ssldump/`, 2005.

[9] Schneier. *Applied Cryptography.* Wiley and Sons, 1996.

[10] B. Schneier. Security in the real world: How to evaluate security. *Computer Security Journal, v 15*, pages 1–14, 1999.

[11] Bruce Schneier. Why cryptography is harder than it looks. Online: `http://www.schneier.com/essay-037.html`, 1997.

[12] C. Allen T. Dierks. The tls protocol version 1.0. *RFC Editor*, 1999.

[13] Debian Secuity Team. Debian security advisory:dsa-1571-1 openssl – predictable random number generator. Online: `http://www.debian.org/security/2008/dsa-1571`.

[14] Huaiqing Wang and Chen Wang. Taxonomy of security considerations and software quality. *Commun. ACM*, 46(6):75–78, 2003.

[15] Yuqing Zhang and Xiuying Liu. Running-mode analysis of the security socket layer protocol. *SIGOPS Oper. Syst. Rev.*, 38(2):34–40, 2004.