CERIAS Tech Report 2002-16

CS 490: WIRELESS SECURITY INDEPENDENT STUDY – FINAL REPORT

By Patrick Fitzgerald

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907

CS 490: Wireless Security Independent Study – Final Report

Patrick Fitzgerald
Purdue University

April 30, 2002

1 Abstract

This paper presents the purpose, goals, accomplishments, and design details of this CS 490 project: design and implementation of improved security measures for wireless networks.

2 Introduction

Wireless networking using commodity 802.11b hardware provides a convenient, simple method for rapidly building a local area network in practically any location. Unfortunately, it is quite common for security to be overlooked in exchange for convenience. While provisions built into the protocol such as MAC authentication and WEP do provide some measures for authentication and encryption, these measures do not offer significant security for anything but the most trivial communications. Analysis of WEP has shown that the security it provides is minimal.[1] open source project called "Air Snort" claims that after gathering 100 megabytes of data, the encryption key can be guessed.[2] Other tools are available to force access into MAC authenticated networks.[3] Clearly, the tools inherently provided by 802.11b for encryption and authentication are not sufficient.

Other cryptographic protocols are available for adding security to inherently insecure networks. The IP Security Protocol (IPSEC) offers many strong cryptographic measures which would be useful in wireless environments. However, a full IPSEC implementation is very resource intensive and would be ineffective on handheld network devices due to its size. Additionally, the complexity of IPSEC yields much greater likelihood of insecurity in implementation. IPSEC also incurs penalties of both speed and bandwidth. Furthermore, the available implementations of IPSEC for common platforms are expensive to license, incomplete, and often incompatible with each other. SSL tunnelling of all data can be more lightweight than IPSEC, but requires either extensive modification to the operating system or foreknowledge on both the wireless client and the router of all protocols which will be used in communication.

In addition, it is the social responsibility of the owners of the wireless network to ensure that only valid users may access the network. An attacker finding an insecure wireless network in a public area is likely to do two things. First, he will attempt to exploit the data and systems on the network. Second, he will attempt to use the wireless network, any host on the wireless network, and any connection between the wireless network and other networks to mount attacks on other networks and hosts. The attacker can anonymously connect to the wireless network, then launch attacks which appear to be coming from the legitimate owner's systems. Using the MAC address as an authentication token has proven a faulty approach. Furthermore, even if MAC authentication were strong enough, the nature of wireless networks makes theft of authentic hardware much more likely. It is therefore critical to ensure the strong authentication and identification of users– not hardware.

In this project, I propose a software remedy to this problem. The widespread availability of HTTPS enabled browsers allows for strong authentication of the users and a secure channel for transfer of a unique session key for each user. IP payload data is encrypted using the Blowfish cipher[1] in cipher feedback mode. The implementation is fast, efficent, and does not incur any bandwidth overhead.

3 Purpose of this Study

The purpose of this study was to gain knowledge about security software engineering in a networked environment. I have studied a wide range of subjects in order to successfully complete this project. Some of the areas which have been covered are:

- Common cryptographic algorithms
- Secure software development
- IP firewalls
- 802.11b wireless networking
- key management
- SSL certificate management

My primary accomplishment in this project was the implementation of a lightweight mechanism for improving the security of a wireless local area network. The software produced in the course of this project provides the following functionality:

- Client authentication of router.

[1]Any block cipher can be used– Blowfish was chosen for its speed and small memory footprint.

- Router authentication of client.
- Packet level encryption of all IP payload data on the network.

The software developed in the course of this project is not intended to replace other security protocols but rather to provide additional security to a public local area network. It should also be noted that because of its design, there may be other possible uses of the software

4 Design Goals

- All payload data on the network must be encrypted.
- All clients must be able to authenticate the router.
- All clients must be authenticated by the router.
- Implementation of the client software must be lightweight and fast enough to be usable on a PDA.
- The entire system must be implementable in software only and must not require any additional modifications to commodity 802.11b hardware.
- The client software should be portable to many platforms.
- The server software should be portable, at least, to Linux and FreeBSD platforms.

5 Assumptions

- A secure web browser platform supporting HTTPS is available on the client.
- A secure web server platform supporting HTTPS is available on the server.
- Username/password pair is sufficient for authentication of a user.
- Client and server system software is not vulnerable to other attacks.
- The software is not required to provide protection from attacks against IP implementations.
- Client systems will not need to communicate securely with each other.

6 Requirements Analysis

6.1 Authentication

- The client must be able to strongly authenticate the server.
- The server must be able to strongly authenticate the client.

6.2 Encryption

- All IP payloads on the network must be encrypted.
- Key exchange must take place across the insecure network as well.
- An eavesdropper should not be able to gain significant advantage in defeating the encryption by monitoring any part of the traffic.
- There should be no significant cryptographic weakness in the key generation.
- The server should provide protection against attacks on the key generation algorithm.

6.3 Network

- The software must encrypt all IP payloads on the network after the key exchange.
- The software may ignore all non-IP traffic.
- The software may assume that all hosts on a specified subnet will participate in encrypted communications.

7 Architecture

7.1 Overview

The architecture of this project was designed to meet the requirements and goals listed above in a simple and straightforward manner, using open software and standard protocols wherever possible. This philosophy was adapted for two reasons. First, a complex system is much more difficult to verify than a simple one. The software consists of approximately 800 lines of C code and 200 lines of PHP. Secondly, use of proven libraries and standards decreases the risk of error in our own implementations of standard functions and increases the portability of the project. Rather than implementing an unportable kernel module, I developed a user-space packet encryption engine based on divert socket functionality. Because of this, the port to OS X was extremely simple and the port to Linux is expected to be less than twenty extra lines of code. All cryptographic functions are provided by the OpenSSL library, a well-known open source project.

The project uses HTTPS to authenticate the client and server, then transfers a key across the HTTPS connection. The key is generated by reading from the /dev/urandom device. Configurable time constraints prevent valid users from mounting attacks on the random number generator. The key is used for the duration of the session to encrypt IP traffic across the network using the Blowfish cipher in cipher feedback mode. Keys are transferred from the HTTPS server to the packet encryptor using a named pipe; they are transferred from the HTTPS browser to the packet encryptor via copy/paste buffer.

7.2 Authentication

- The client and server use HTTPS to authenticate each other mutually using certificates.
- The user is authenticated with a username/password combination.
- The web server stores session data (session key, username, IP address) in an SQL database.
- The web server sends the session key to the user via the HTTPS session.
- The web server then passes the IP address and the key to the gateway server daemon through a named pipe.
- When the user has finished using the encrypted connection, he/she may use a web interface to revoke his/her key. The web server then passes a key-revocation message to the packet encryptor, which revokes the key.

7.3 Key Exchange

- A session key is given to the client via HTTPS after both client and server have authenticated.
- This key is used for symmetric encryption on all subsequent communications.
- The server ensures that each user may only request one key per IP at a time and limits key request rates.

7.4 Encryption

- The payload of the IP datagrams sent to the network by both client and server software is encrypted using the Blowfish cipher in cipher feedback mode.
- The payload is encrypted and decrypted using blowfish in CFB64 mode as implemented in the OpenSSL package, and as such requires the openssl library to be installed. The initializing vector is computed as follows:
  $iv = K(\text{source address, dest address})$
  $iv' = K(iv \otimes \text{length, id, flags, offset, ttl, proto})$
  iv' is used as the initializing vector. (See figure 1.)
- Late in the semester, it was determined that this method exhibits a weakness when packets are sent with identical IP headers. This weakness has not been fixed in the software due to time limitations, but an outline of the weakness two possible solutions are given below.
  - Because identical IP headers will generate identical initializing vectors, it is possible to determine the first 64 bits of the IP payload data if the IP headers are identical.

- An attacker can exploit this weakness by sending additional forged packets with identical headers through the server. This reveals the first 64 bits of the packet, which can be used to generated another forged packet which reveals the next 64 bits and so on.
- This attack requires the attacker to have connections on both sides of the server and send $(size/64) * size$ additional data for each packet to be decrypted.
- This attack can only reveals data sent from server to client. An attacker cannot forge duplicates of packets sent from client to server.
- The attack can be mitigated by randomly modifying certain parts of the header. Filling TOS and TTL fields with random bits in the IP header before encryption on the server will not affect packet delivery and require the attacker to send $size * 65535$ additional data for each packet for a 63% probability of decrypting the first 64 bits of the packet payload. Assuming a minimum packet size of at least 300 bytes, this attack would require sending over twenty gigabits of additional information to have a 95% probability of decrypting a full packet.
- For packets which are not fragmented, the identification and fragment offset fields can also be used. This increases the available entropy to 2^{31}. Recovering these packets would require approximately two hundred days on an eleven megabit network.
- The weakness could also be fixed by using a modified version of cipher block chaining (CBC) mode adapted for variable-length data. The cipher would function in CBC mode as long as 64 bit blocks were available, then encrypt the remaining data by XOR with the re-encrypted last block of ciphertext.[2] *i.e.* $c_n = p_n \otimes K(c_{n-1})$

* The client software decrypts the payload of all IP datagrams with a destination address matching that of the client using the session key, excepting HTTPS communication with the authenticating web server.

* The client software encrypts the payload of all IP datagrams with a destination address other than the client's using the session key, excepting HTTPS communication with the authenticating web server.

* The server software decrypts the payload of all IP datagrams with a source address in the encrypted subnet and a destination address outside the encrypted subnet using the stored session key matching the source address, excepting HTTPS communication with the authenticating web server.

* The server software encrypts the payload of all IP datagrams with a destination address in the encrypted subnet and a source address outside the

[2]This method requires use of a cipher where $K(K(m))$ is not significantly weaker than $K(m)$.

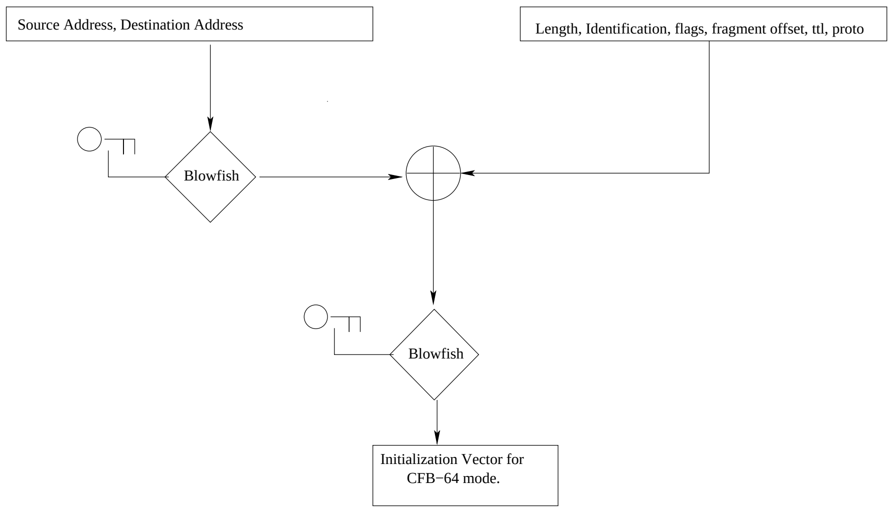
Source Address, Destination Address
Length, Identification, flags, fragment offset, ttl, proto
Blowfish
Blowfish
Initialization Vector for CFB−64 mode.

Figure 1: Generation of IV for CFB64 mode.

encrypted subnet using the stored session key matching the destination address, excepting HTTPS communication with the authenticating web server.

7.5 Packet capture and manipulation

- Packets are captured for the client and router packet encryption processes by the divert socket mechanism under FreeBSD and Mac OS X. A port using libipq under Linux is currently in development.
- Firewall rules must be added on the client and server to divert all traffic to the packet encryptor except for HTTPS traffic to the authentication server.
- Packets are then anaylzed by the encryptor to determine their eligibility for encryption and decryption according to the rules listed above.

8 Observations and Conclusion

At the CERIAS symposium, I had the opportunity to discuss my project with several members of the security community. The response I had was overwhelming. Sysadmins are currently piling IPSEC on top of LEAP on top of WEP and changing keys every nine minutes, trying to solve the wireless security problem by adding more complex structures. Clearly, there is much concern in the community about providing secure wireless access for business use. Unfortunately, complexity has never been a friend of security. The more complex a system is, the more likely it is to exhibit trivial weaknesses hidden within the complexity.

Offering free software which solves the problems of wireless networks in eight hundred lines of portable C seems unrealistic. I do not claim that this project is a panacea for wireless network security. Further research and investigation is stil needed within this project, particularly in the way initializing vectors are created and used. Wireless network security remains an open problem, despite my best attempts this semester. To address all concerns, it would be necessary to have pervasive influence in the protocol design process. As an undergraduate student, I certainly do not have the widespread influence in the network industry that IEEE does! Perhaps the greatest value of this project lies in causing others to question the validity of the current standards process.

This project falls short of its lofty goal of replacing WEP because it is an afterthought. I have "fixed" the broken standard by applying a patch. This approach is also fundamentally flawed. The only reasonable way to ensure security of a system is to design the system with security in mind. This project, and others like it, should be seen only as stop-gap measures designed to keep a broken system functional until it can be replaced. It is my sincere hope that the industry will utilize better practices in the future. This seems unlikely, however, until stronger market pressures develop supporting secure protocol design.[3] The widespread availability of inexpensive 802.11b devices ensures the necessity of stop-gap measures for the time being.

I intend to continue development of this project beyond the proof-of-concept phase. I plan to add features permitting better, more automated, key management; communication between clients; automated key revokation using a heartbeat protocol; graphical user interface for the client software; administrator-configurable cipher suites; and many other enhancements to the functionality of the software. It will also be necessary to modify the way initialization vectors are generated, especially on the server. Because the software functions on IP datagrams, it not restricted to wireless use only. On a wired network, it could easily be modified to provide VPN and secure IP tunneling functionality, with the proper configuration. Additional study is needed to address security concerns associated with this use before it can be approved for this use. The source code of this project will therefore be opened to the public to allow open analysis and further collaboration with the security community at large.

A Software availability

The software produced in the course of this project has been released under an Open Source license and is available at the project web page, http://wepless.sourceforge.net/. The implementation supports FreeBSD and MacOS X as of this paper, and will support Linux and OpenBSD very soon.

[3]It is clear that no amount of complaint from the security community will ever have near the persuasive strength of the almighty dollar.

B Schematic Diagrams

B.1 Server

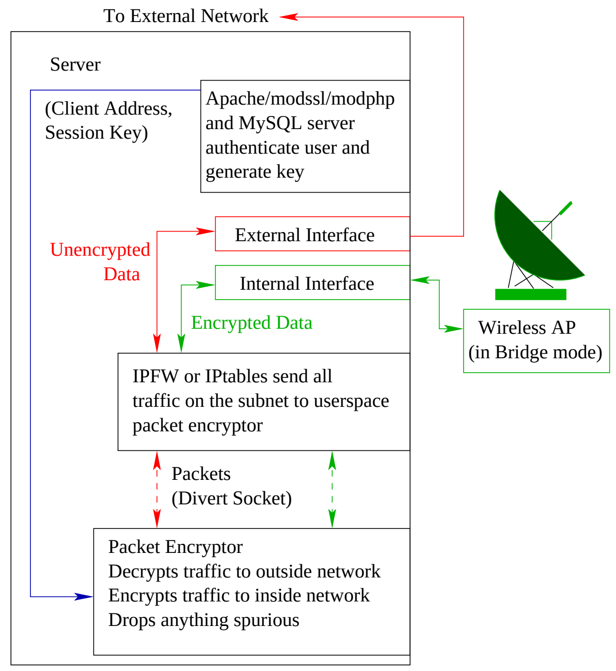
To External Network
Server
(Client Address, Session Key)
Apache/modssl/modphp and MySQL server authenticate user and generate key
External Interface
Unencrypted Data
Internal Interface
Encrypted Data
Wireless AP (in Bridge mode)
IPFW or IPtables send all traffic on the subnet to userspace packet encryptor
Packets (Divert Socket)
Packet Encryptor
Decrypts traffic to outside network
Encrypts traffic to inside network
Drops anything spurious

Figure 2. Server schematic diagram.

B.2 Client

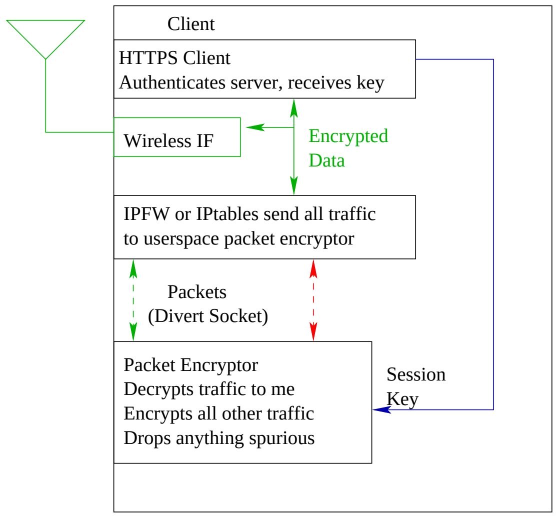

Figure 3. Client Schematic diagram

C Acknowledgements

I would like to acknowledge the guidance and input of my advisor Pascal Meunier in this project. Without his persistent suggestions that I should "do something about the wireless problem," I probably would never have attempted this project. Without his constant prodding to keep coding, I probably never would have finished the project. Without his help in understanding the complexity of the problem, I probably would have implemented a system as flawed as the one I wanted to replace.

I would also like to thank Dave Jacoby and his wife Kerry, for feeding me several hot meals during the course of this project. Too often, I have overlooked the importance of proper nutrition in software development.

Finally, I would like to thank my fianceé for putting up with my incessant ramblings about this project.

D References

1. The AirSnort project: http://airsnort.shmoo.com/
2. *Weaknesses in the Key Scheduling Algorithm of RC4*, Mantin, Shamir and Fluhrer. http://www.securityfocus.com/cgi-bin/library.pl?cat=154&offset=10
3. *ActiveSync, TCP/IP and 802.11b Wireless Vulnerabilites of WinCE-based PDAs*, Pascal Meunier, *et al.* (Publication pending.)
4. The unofficial 802.11 security web page: http://www.drizzle.com/~aboba/IEEE/