A Challenging but Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL

Gregory V. Bard[*]

April 4, 2006

Abstract

The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols are widely used for securing communication over the Internet. When utilizing block ciphers for encryption, the SSL and TLS standards mandate the use of the cipher block chaining (CBC) mode of encryption which requires an initialization vector (IV) in order to encrypt. Although the initial IV used by SSL is a (pseudo)random string which is generated and shared during the initial handshake phase, subsequent IVs used by SSL are chosen in a deterministic, predictable pattern; in particular, the IV of a message is taken to be the final ciphertext block of the immediately-preceding message.

We show that this introduces a vulnerability in SSL which (potentially) enables recovery of low-entropy strings such as can be guessed from a likely set of 2–100 options. Also, we argue that the one-channel nature of web proxies, anonymizers or Virtual Private Networks (VPNs), which result in all Internet traffic from one machine traveling over the same SSL channel, provides a feasible "point of entry" for this attack via some corrupted content (which the user must be induced to view); moreover, we show that the location of target data among block boundaries can have a profound impact on the number of required guesses to recover that data, especially in the low-entropy case.

While this attack is not trivial to implement, it is feasible and is intended only to show that the vulnerability is in fact exploitable. The attack in this paper is an application of the blockwise-adaptive chosen-plaintext attack paradigm. This is the only known feasible attack to use this paradigm with a reasonable probability of success. However, we demonstrate that this model is not only of theoretical interest, and highlight flaws in CBC which should be avoided in other applications, where structurally similar but more dangerous vulnerabilities might exist.

Keywords: Blockwise Adaptive, Chosen Plaintext Attack (CPA), Secure Sockets Layer (SSL), Transport Layer Security (TLS), Cryptanalysis, HTTP-proxy, Initialization Vectors (IV), Cipher Block Chaining (CBC), Virtual Private Networks (VPN).

1 Introduction

This paper outlines a vulnerability of SSL 3.0 and TLS 1.0, by means of a feasible blockwise-adaptive chosen-plaintext attack (BACPA). The attack proceeds in two major steps: First, by spoofing one packet, the version of the protocol is forced below SSL 3.0 during the negotiation stage, which in turn results in the same key being used in each direction of the encrypted traffic for the entire session. Second, after valuable low-entropy data has been transmitted by the target, (e.g. a stock from a list of 2–100 companies) the adversary inserts plaintext into the communications stream by inducing the target to view data designed to contain particular byte sequences. Based on the ciphertext values of these known plaintexts, the low-entropy data can be guessed. The probability of success depends on the type of data being targeted, with examples

[*]`gregory.bard@ieee.org`. Ph.D. Candidate, Dept. of Applied Mathematics, University of Maryland, and Lecturer, Dept. of Computer Science, American University.

given below. Furthermore, it is shown that the position of this target data within block-boundaries *directly and significantly* impacts the number of guesses required to recover that data.

The vulnerability described here was closed in TLS 1.1 and OpenSSL after 0.9.6d. There may still be SSL servers in use that operate with version 3.0. Nonetheless, the main purpose of this paper is not to claim the existence of a major threat to computer security, but rather to disprove the myth that blockwise-adaptive chosen-plaintext attacks are totally infeasible and thus of theoretical interest only. Furthermore, we prove that it is easier to guess the values of low-entropy data, when it is divided across block boundaries.[1] It is hoped this paper will highlight some of the disadvantages of Cipher Block Chaining; emphasize the importance of using distinct keys for each direction of communications; and illustrate the importance of using random numbers as initialization vectors.

1.1 Blockwise-Adaptive Chosen-Plaintext

BACPA was simultaneously discovered in 2001 by Bellare, Kohno, and Namprempre [4] and by Joux, Martinet, and Valette [17]. The BACPA differs from classical chosen plaintext attack (now termed "message-wise" or MCPA) in essentially one detail. In MCPA, the attacker can generate arbitrary messages for the target to encrypt, as part of a sequence of messages. In BACPA, the attacker can generate arbitrary blocks for the target to encrypt, inserted as part of an existing message (i.e., a sequence of blocks). However, four years after its discovery, BACPA has received very little notice outside the cryptographic community, despite several additional papers within it [12, 6, 13].

In Bellare, Kohno, and Namprempre [4], in addition to outlining a theoretical model of potential adversary capabilities expanded upon in later papers [12, 6, 13], a general attack on the Secure Shell (SSH) was loosely described (first found by [7]). That attack had a success probability[2] of $2^{-12.5}$ or 6.9×10^{-4}. This paper outlines an attack with a success probability that can under certain circumstances approach 50%. We are aware of no other attack scenario under the BACPA model, against an existing or proposed protocol.

1.2 The Versions of the SSL Protocol

The Secure Socket Layer (SSL) is currently one of the most widely-used methods for securing communication over the Internet (see [22, Chap. 19] for an excellent overview of SSL). The blockwise-adaptive chosen-plaintext attack outlined here applies to all versions of SSL, except the most recent (i.e., version 3.0) [14]. However, in Section 5, we show that targets running SSL 3.0 (by far the most deployed version) can be induced to run an earlier version, by spoofing one packet, and thus be made vulnerable to this attack. Meanwhile, the replacement for SSL, namely TLS 1.0 [8] is also affected if and only if the configuration allows for compatibility with legacy versions of SSL. In this case, the attack proceeds identically to SSL 3.0.

Versions of OpenSSL since 0.9.6d[25] have closed this vulnerability, as has TLS 1.1 [9], which is still in draft status as of the writing of this document. All SSL deployments other than OpenSSL, are believed to be currently vulnerable to this attack if they allow clients to use versions below 3.0 (as is required by the protocol specification).

1.3 Initialization Vectors

Our attack relies on the fact that SSL currently mandates the use of a weak variant of the cipher block chaining (CBC) mode of encryption [22, Chap. 4]. CBC mode requires a one-block initialization vector (IV) for each message that is encrypted. In "standard" cryptographic usage of CBC, a fresh, random IV is chosen for each message. In SSL, however, only the *initial* IV is chosen in a (pseudo)random manner; IVs for subsequent messages are simply taken to be the final block of the ciphertext corresponding to the immediately-preceding message. (This process is called "chaining the IVs.") In particular, an attacker may know *in advance* the IV that is going to be used to encrypt the next message. We show that this enables

[1]We believe that this has not been shown previously in the published literature.

[2]The success probability is stated to be equivalent to waiting for a collision on approximately 25 bits. See the last sentence of Section 3 of [4].

an attacker mounting a chosen-plaintext attack to validate a guess as to the value of a particular plaintext block.

1.4 Consequences

Since the adversary can validate a guess as to the value of a particular plaintext block, this attack violates the theoretical standard of Left-Or-Right Indistinguishability, whereby no polynomial time adversary can be able to distinguish between the encryption of two messages of his/her own choice, given the ciphertext, (with non-negligible advantage) [16, 3]. On the more practical side, it allows an attacker to possibly determine a low-entropy string by repeatedly guessing all possible values for this string until the correct one is identified.

Examples of such low-entropy information (2–100 choices) include names of stocks, cities, users, or even PINs that have been previously encrypted (or, for example, knowing if a stock order is buy, sell, or stop-loss could be valuable information by itself). Given the use of SSL for transmitting exactly this sort of data, we believe this represents a potentially serious (but challenging) attack which should be addressed by any security group selecting an SSL implementation.

1.5 Related Work

Essentially this method has been used previously to attack SSH [4, 7]. In fact, the flaw attacked there is identical to the flaw attacked here (namely, setting IVs in a predictable way). Little discussion of feasibility or point-of-entry was given, and the probability of success was very low as already stated. On the other hand, that discovery gave birth to the blockwise-adaptive chosen-plaintext world.

Due to the similar structure of SSL and SSH, the related vulnerability in SSL was discovered soon after, independently by Moeller [21] and the author of this paper, in late 2002—about 8 months after the publications of the two original blockwise papers. Moeller's work identifies the attack but does not show how it could be exploited. Moreover, this paper elaborates upon the attack by showing how low-entropy data, in particular, is easy to recover, and provides mechanisms by which to execute the attack. We believe our discussion of the means for executing the attack on SSL (for example, our suggestion that the single channel nature of VPNs, proxies, or anonymizers could easily provide a "point of entry" for chosen-plaintext attacks) is of independent interest. Finally, the fact that the vulnerability is present in almost all commonly used SSL implementations (subject to the version spoofing trick in Section 5) even following recent work demonstrating similar attacks in SSH only further indicates the need to publicize these attacks within the security community.

The changes to TLS between versions 1.0 and 1.1 were made partially in response to this class of vulnerability[3], as well as that of Vaudenay[4] [9].

1.6 Organization

This paper is structured as follows. Following a discussion of related work, we provide a "high-level," cryptographic description of our attack in Section 2. Section 3 focuses on "low-level" details of the attack, and shows that the attack as outlined can actually be implemented against SSL/TLS. As mentioned earlier, the attack allows an adversary to confirm guesses as to the value of a plaintext block at a rate of one guess per block of chosen plaintext. Thus, low-entropy strings are particularly vulnerable. We show in Section 6 how our attack can be used to recover even moderately-long target strings when these strings are "split" across block boundaries (which is expected to occur frequently in practice). In Section 4 we discuss the feasibility of our attack when SSL is used within a web browser (as is almost always the case). Although the vulnerability we expose here is not trivial to exploit, the reasons enumerated in Section 4 indicate that the attack does represent a potential concern. In particular, Section 4 highlights the mechanisms by which this attack can take place. Moreover, this attack — which requires the adversary to convince a user to

[3]This vulnerability is called CBCATT in some TLS documentation [21], see also Section 6.2.3.2 of TLS 1.0 and 1.1 [8, 9].

[4]Another interesting attack on the use of CBC in SSL, discovered by Vaudenay, relates to the padding of messages, and is totally unrelated to the vulnerability in this paper [24].

view content of his/her own creation — is likely to be easier than an attack which requires the adversary to convince a user to install Trojan Horse software at the operating system level (say, to "sniff" the user's password via keystroke capture). We conclude the paper in Section 7 with some discussions of the remedies of the vulnerability exploited here, and Section 7.4 talks about how the attack must change to accommodate SSL 3.0 or TLS 1.0, where distinct keys are used for each direction of communication.

2 High-Level Outline of The Attack

We begin by briefly highlighting the minimal aspects of SSL needed to understand our attack at a high level. A more detailed treatment of the attack (and hence of SSL) is given in Section 3. A good survey of the SSL protocol is given in [22, Chap. 19].

The SSL protocol begins with a handshaking stage during which the parties agree on a protocol version (see Section 5), select cryptographic and (optionally) compression algorithms, perform optional authentication steps, and use public-key mechanisms to share secrets. The shared secrets, which include symmetric keys and IVs for each direction of communication[5], can then be used for symmetric-key encryption and message authentication. We note that while messages may optionally be compressed before encryption, few SSL implementations currently do so [22, Chap. 19], [14].

The SSL standard allows for symmetric-key encryption using either block ciphers or stream ciphers. Most implementations utilize block ciphers, and the vulnerability in this paper applies only when block ciphers are used. A block cipher is a keyed, invertible permutation over strings of some fixed length called *blocks*; DES, for example, operates on 64-bit blocks. We represent application of a block cipher using key *sk* to block X by writing $F_{sk}(X)$. To encrypt messages longer than one block in length, a *mode of encryption* must be used. SSL mandates the cipher block chaining (CBC) mode, which encrypts a message $P = P_1, \ldots, P_\ell$ (where the length of each P_i is the block-length of the cipher) as follows: given some IV denoted C_0, compute $C_1, \ldots, C_\ell$ sequentially via:

$$C_i = F_{sk}(P_i \oplus C_{i-1}).$$

In the general case of CBC, the resulting ciphertext is usually taken to be $C_0, \ldots, C_\ell$ although if the receiver already knows C_0 then it need not be transmitted. To decrypt, the receiver computes P_i for $i = 1$ to ℓ via:

$$P_i = F_{sk}^{-1}(C_i) \oplus C_{i-1}.$$

We note that it is considered "standard" security practice to choose a new, random IV for every message that is encrypted. However, the above definition of CBC does not force this to be the case. As we have mentioned already, SSL chooses all but the initial IV by setting it equal to the final ciphertext block of the preceding encrypted message; this is referred to as "chaining IVs across messages". (Thus, continuing the above example, the IV used for the next message would simply be C_ℓ). SSL chooses the initial IV in a pseudorandom fashion which is not important for the purposes of the present attack.

The attack. Suppose an adversary who can mount a chosen-plaintext attack wants to verify a guess as to whether a particular plaintext block has a particular value. Specifically (continuing the above example), say an adversary who has observed the ciphertext $C_0, \ldots, C_\ell$ wants to determine whether plaintext block P_j is equal to some string P^*. Note that the adversary knows the IV (i.e., C_ℓ) that will be used when encrypting the next message. Consider now what happens if the adversary causes the sender to encrypt a message P' whose initial block P'_1 is equal to $C_{j-1} \oplus C_\ell \oplus P^*$. The first ciphertext block C'_1 is then computed as:

$$\begin{aligned} C'_1 &= F_{sk}(P'_1 \oplus C_\ell) \\ &= F_{sk}(C_{j-1} \oplus C_\ell \oplus P^* \oplus C_\ell) \\ &= F_{sk}(P^* \oplus C_{j-1}). \end{aligned}$$

[5]More specifically, there is one key for each direction in version 3.0, and one key for both directions in earlier versions.

However, we also know that $C_j = F_{sk}(P_j \oplus C_{j-1})$. This implies that $C'_1 = C_j$ if and only if $P_j = P^*$, since F_{sk} is a permutation. In this way, an adversary can verify a guess P^* for the value of any plaintext block P_j. In particular, if the adversary knows that P_j is one of two possible values then the adversary can determine the actual value by executing the above attack a single time. Similarly, if the attacker knows that P_j is one of N possible values then by repeating the above attack $N/2$ times (on average) the adversary can determine the actual value of P_j. This already violates the standard notions of security for encryption (in terms of Left-Or-Right Indistinguishability, as mentioned in the Introduction on page 3 [16, 3]). Moreover, this implies that an attack of this form can be used to determine the value of a low-entropy string in its entirety. (Note that, in practice, the block of plaintext containing the user's data also likely contains additional information such as headers, etc. However, it is also likely that this additional information is known to the attacker; for example, if the information is fixed padding then an adversary can learn the format of this data from the web-page source code. We discuss this further in Section 4).

Attack requirements. Focusing specifically on the case of an adversary trying to recover a user's targeted low-entropy data, we briefly highlight the requirements needed for the above-described attack to succeed; in Section 4 we discuss in more detail how these requirements are typically met in practice. First, the attacker must know which plaintext block j contains the desired information. All this means, however, is that the adversary knows the format of the HTTPS transmission being targeted. Second, the adversary must know C_{j-1}. However, since the ciphertext travels over the Internet (in the clear!), this is not expected to be difficult. (In fact, if it is assumed difficult to obtain this information then there is little reason to use encryption in the first place). Third, the adversary must know the value of the IV that is going to be used for the next message. However, we have noted already that because of the way SSL computes IVs, an attacker would actually obtain this information from the last ciphertext block of the previous message. Finally, the adversary must be able to insert a plaintext block of its choice into the first block of the next message to be transmitted. This is the most challenging part of the above attack.

The scenario. The mechanism by which the adversary executes the attack is summarized as follows. First, we assume that the user is connected to an HTTP-proxy, web anonymizing service, or virtual private network. This guarantees that all Internet traffic will be routed over one SSL tunnel with a single secret key. Second, we assume the attacker can setup a site under his/her control, and create content for the user to receive. In particular, streaming audio has features, which will be described below in Section 4, that make it desirable for use. Third, a "reflector" or mechanism for observing the target's ciphertexts is set up. This is needed in all forms of cryptanalysis, so we presume it is available. Fourth, we assume that the user can be induced into receiving this content. This can be from an inviting email which persuades the user to go to the site, or perhaps an email that simply causes the content to be played upon opening, via HTML encoded requests for streaming audio in the email body. We fully acknowledge that these assumptions are non-trivial but they demonstrate that a chosen-plaintext attack is feasible.

3 Attacking SSL

Here, we simply note that there is nothing in the structure of SSL (such as extraneous headers or formatting information) which prevents the attack of the previous section from succeeding.

SSL sits between the Application and Transport layers, and so acts like a Session Layer in the OSI model. As such, SSL receives plaintext from the Application Layer as raw data. This plaintext is fragmented into blocks of length less than or equal to 2^{14} bytes. These blocks are optionally (but rarely) compressed[6] and are then processed and sent as follows:

- **Unencrypted Portion**:
  - message type (one byte);

[6]The attacks in this paper do not apply directly when compression is used; however, we have already noted that compression is rarely used.

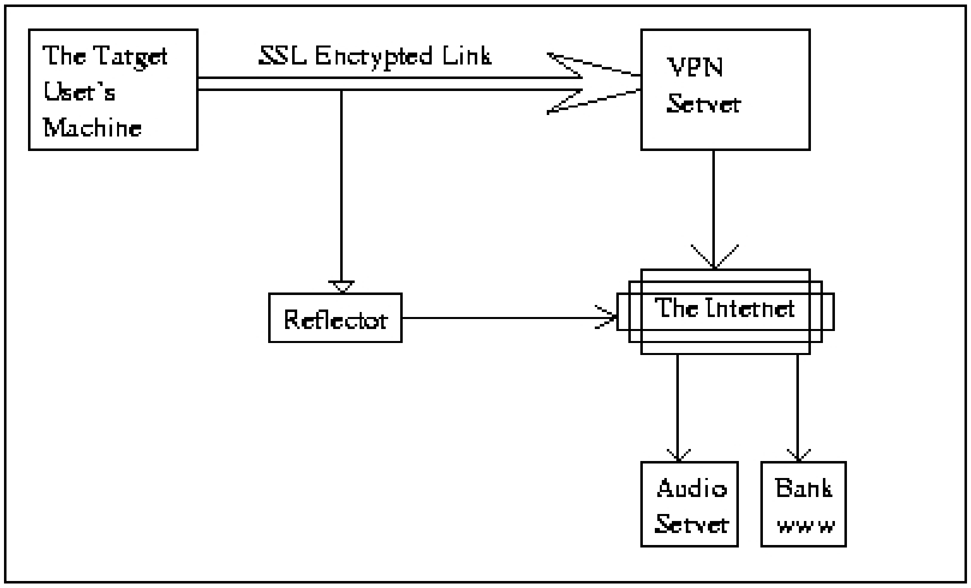

Figure 1: The data-flow of the SSL attack scenario.

  - major/minor version number (two bytes);
  - length counter (two bytes);
- **Encrypted Portion**:
  - plaintext fragment (arbitrary length $\leq 2^{14}$ bytes);
  - message authentication code (typically 20 bytes);
  - padding (0 to 7 bytes; ensures that the plaintext length is a multiple of the block length)
  - padding length (one byte)

We stress that *the first block of the plaintext is indeed the first block to be encrypted.* In particular, the header information that is prepended to the eventual transmission (i.e., the message type, major/minor version number, etc...). is *not* encrypted. Thus, as long as the adversary can set the first block of the plaintext fragment to some desired value (as discussed in the previous section), that block will be encrypted first and the attack will succeed.

We note that in SSH, some header data is prepended to the plaintext *before* encryption. This makes an attack such as the one outlined here more difficult in the context of SSH [4, 7], since the adversary no longer has complete control over the first block of the plaintext that is eventually encrypted. Although it may be possible to work around these constraints (see [4]), the attack is much more difficult against SSH than it is against SSL (demonstrated by a success rate of $2^{-12.5}$ as compared to rates as high as 2^{-1} under certain circumstances).

4 Feasibility of Implementing the Attack

Several challenges must be surmounted before an adversary can successfully perform the attack that has been outlined here. The necessary steps to meet these challenges are listed below.

Using one common key. First and foremost, the data to be learned and the test cases must be encrypted with the same key. Furthermore, since the key for the block cipher is chosen at random each time an SSL socket is created, the data to be learned and the test cases must be transmitted through the same tunnel. Luckily enough, when a web-tunneling service like an HTTP-proxy, or a VPN is used, all the data travels over one SSL link (and thus uses only one secret key in versions before SSL 3.0!)

Learning the plaintext format. Despite the length of a plaintext message, there are times when only a very small sequence of bytes is of critical importance. For example, a stock choice, a destination city for a geospatial inquiry, a user name, or something where only 2–100 choices are considered likely, and known in advance (i.e., a "low-entropy string" in our terminology).

We have mentioned earlier that the adversary must somehow know which block of the plaintext contains the data of interest. Note, however, this is easily done by reading the source files for the pages that are used in sending the data. Discerning the format merely requires knowledge of the HTTP, HTML, and CGI protocols, and perhaps Javascript. Commonly available browsers have a "show page source" command, which displays the page's HTML source code. Both the "form elements" which compile the user's data, as well as the optional Javascript code which would verify its format, would thus be available to an attacker. While Javascript can alter HTML code, for example, to add hidden form elements, these would be visible in the Javascript code. The attacker need only read this code and the format is trivially derived.

Ensuring that the adversary's chosen plaintext block is encrypted first. It is essential that the chosen plaintext, namely $P_i = P^* \oplus C_\ell \oplus C_{j-1}$, be the first encrypted block of the SSL datagram inside which it is found. However, this is easy to ensure, as we argue now.

Data is submitted to the SSL layer in the form of application-level messages, which are first aggregated into blocks of (at most) 2^{14} bytes in length. SSL does not respect message boundaries. If more than 2^{14} bytes are submitted, additional blocks are created; if several application level messages are submitted, they are concatenated in the buffer. However, in the absence of these two conditions, the data is encrypted and transmitted to the TCP layer immediately. Therefore, short messages, like streaming-audio packets, would be encrypted and transmitted immediately. The structure of SSL packets guarantees, in the absence of concatenation, that the first block of the application message will be the first encrypted block of the SSL datagram.

Concatenation only occurs when two messages arrive at the SSL layer simultaneously. However, since the audio source is under the control of the adversary, the timing could be adjusted to guarantee that two packets never arrive during the required interval. Of course, packets could arrive from some other user activity or application. These timing failures can be avoided probabilistically by repeating each guess several times. Finally it should be noted that in the event concatenation does occur, if the audio packet is the first packet, with other packets appended, no disadvantage is caused for the attack. Nonetheless, we acknowledge the timing difficulties of the attack.

Encrypting Chosen Plaintext. The attacker must force the encryption of particular data in a chosen plaintext attack, in this case based upon the previous ciphertext block, the guess and the ciphertext previous to the data to be guessed. Once this block is calculated, it must be inserted into the "plaintext stream."

Since our scenario of a user connecting via an HTTP-proxy or a VPN involves *all network traffic flowing to the target machine, being encrypted by the same key all in one cipher-stream,* the attacker need only cause the chosen plaintext data to appear in an application-level datagram . However, there are several other considerations.

- **The data must be tolerated.** The data must not result in a crashed application, a dropped connection or error messages that would alert the user. In particular, sending arbitrary bytes to an application via a datagram would almost surely violate that application protocol's datagram formatting rules. For this reason, the data must "blend in" to the data that would have been otherwise sent.[7]
- **The data must start the datagram.** Once the immediately previous ciphertext block is known, the very next plaintext block must contain the attacker's plaintext. There must be no blocks in between, otherwise our previous formulas fail. Therefore, it is essential that the test block (first block of new plaintext) be the very first block of the datagram.

[7]Special thanks to the anonymous referee of XXXX (removed for blinding) who pointed out several challenges in this regard.

- **The application datagram must be header-free.** If the application protocol requires a header on the datagram with any particular structure, it is unlikely that the attacking plaintext will conform to its standard. Therefore combining the previous two requirements, it becomes clear that an application with no datagram header would be useful (though a trailer would be of no hindrance).
- **The application must anticipate many datagrams.** Since only one guess can be validated per packet, the application must anticipate a moderate number of packets (on the order of 1–50 for a guess among a set of 2–100 choices).

Given these four requirements, one clear choice is streaming audio. Naturally, there may be others, but audio streams have several advantages. First, they consist of a large number of small packets. Second, pauses or interruptions in service occur often, and are not surprising. This includes even dropped connections. Third, when a stream pauses or stutters, normally a short burst of static occurs before the continuation of the music. Thus a misplaced sequence of a few bits at the start of a datagram will make some random noise, but for an extremely short time for the human ear. If this is preceded by a pause, the noise would be not only tolerated, but anticipated! Fourth, streaming audio uses UDP, which avoids any hassle of sequence numbers, acknowledgments or sliding windows. Fifth, since UDP is being used, the applications are highly tolerant to lost, delayed, or reordered packets, making them error-tolerant in general. Other options might include streaming video, voice-over-IP, et cetera.

Providing feedback to the Audio Generator. Note that the adversary needs feedback from the reflector to the streaming site (in particular, to inform it of the ciphertext blocks C_{j-1}, C_j, C_ℓ) in order to perform the attack. It is not expected to be difficult for an adversary's reflector to obtain these ciphertexts (after all, they are traveling on the Internet), but the information must somehow be communicated to the streaming audio site.

There are two principal avenues through which this communication could occur. First, if the attack is being performed in a wireless environment, setting a wireless device to monitor/promiscuous mode is attractive. However, many systems use WEP (Wired Equivalent Privacy) though certainly not all (including, for example, the University of Maryland and American University campus networks, which are totally unencrypted).

Alternatively, if the adversary has access to the same Ethernet network on his/her own machine that the user is connected to, then the following can be executed. Perhaps a subordinate employee will use his/her private laptop with tcpdump and a special feedback script acting as a reflector, to capture his/her employer's SSL traffic. The audio site would only need to know where the feedback daemon is, receive the ciphertexts, and use it to compute new plaintexts. The search for the daemon could be through the use of a throw-away DNS address, or by hard-coding the IP address.

On the one hand this could be detected by an Intrusion Detection System as unauthorized traffic, but on the other, steganography could be used to embed the data in legitimate communications. Alternatively, since few Intrusion Detection Systems would be this sensitive, the reflector could simply pass the packets back in the clear.

Synchronization. The demands of synchronization are important since there can be no transmitted data between the "last ciphertext block" and the encrypted attacking plaintext. There may be many ways to achieve this, but the audio server could signal that it is ready by triggering a 1–2 second pause in the audio (which occurs naturally from time to time because of congestion on the network). The reflector responds with the most recently transmitted ciphertext, which hopefully travels with minimal delay to the audio server. Upon receipt of this, the audio server transmits the stream once again, however the first 8 or 16 bytes of the packets (one encrypted word) is not audio data, but the chosen-plaintext. If no data was transmitted on the tunnel in this time, then this stage of the attack succeeds and one guess can be checked. If data was transmitted on the tunnel in this time, then the plaintext loses its meaning in this context, but no harm is done and the attack can be attempted again. Since the Poisson Distribution is known to be a good model of user packet generation during web-browsing and other non-streaming Internet tasks, large gaps between packets will be expected with moderate frequency.

The user's patience must be a factor. If we assume 15 minutes of listening, with a single gap of 1–2 seconds duration each minute, then there are 15 gaps. If we tolerate a $\frac{1}{3}$ chance of success (meaning a 33.3% probability that the tunnel is unused during the pause) then 5 guesses can be made before the user gets bored or irritated by the static. Even if there are 100 options in the data-set to be selected from, there is a 5% chance of success (assuming equiprobable choices, otherwise higher if the probability distribution is known). If there are as few as 10 options, then the chance of success is one-half or higher, rather acceptable in most cases.

Summary

The requirements listed in this section are by no means trivial. The timing and other network packet structure considerations will result in some guesses failing to serve their purpose. But we believe that we have demonstrated the potential feasibility of this sort of attack. This further demonstrates that BACPA is of more than purely theoretical interest.

5 Version Negotiation

The BACPA part of the attack works on versions of SSL until version 3.0, and not later. However, version 3.0 is by far more common, so it would be attractive to find a way to force communicants to use a previous version. Appendix E of the SSL 3.0 specification clearly states that the protocol is backwards compatible with version 2.0. The version to be used is selected during the handshaking protocol (See Section 5.5 of that document [14]).

The handshaking protocol begins every SSL session and starts with a "client hello" message. In this message are various fields for client capabilities, such as cipher suites and compression methods, but also the highest supported version number. Moreover, this packet has a highly predictable format.

An adversary on the same wireless or Ethernet subnet merely need listen for the above packet, and while it is still underway, transmit random bits into the medium to disrupt the packet. The disruption will result in exponential back off by the sender, in which case the attacker transmits a datagram with the same information, including spoofing the IP and MAC addresses of the target, but with a lower version number (2.0). This can be easily done with an embedded controller that identifies and records the transmitted data and transmits the scrambling signal during the data-link layer trailer (or some other point late in the datagram.[8]) Since the lower of the version number supported by the client and the server is selected for communication, version 2.0 will be chosen.

In particular, we suggest one alternate scenario. Two users are in adjacent hotel rooms, connecting with laptops over a wireless Ethernet (unencrypted, as nearly all public wireless networks are). The target user connects to his/her main office via a VPN or anonymizing tunnel. The wireless network first provides a mechanism for the version spoofing described above. The attacker need only fake the target's MAC (Medium Access Control) Layer address, and the wireless hub will interpret the attacker's packets as coming from the target. (Since they are in adjacent rooms, surely the same hub is in use). Once the financial transaction is complete, the attacker can attempt to trigger the remainder of the attack as in the remainder of the paper.

6 Recovering Financial PINs

Here we show how even a moderately-small entropy string (10,000 choices) can be easy to recover due to segmentation that occurs when the target data falls on a block boundary. (We assume throughout this section that the data surrounding the target information is known; see above). Since it is now demonstrated that the adversary has the ability to verify guesses of plaintext blocks, one can imagine that an adversary can attempt to guess the value of a valuable target low-entropy string either by exhaustive search (in the case

[8]We have used this technique in a highly related but different situation.

of Personal Identification Numbers or PINs) or by more efficient context-specific schemes (using dictionary based attacks on passwords).

If we assume for simplicity that the data is chosen uniformly from a space of size S, then the expected number of guesses needed before determining the string is $S/2$. (Note that in the case of passwords chosen by a user, the actual entropy is likely to be much lower than would be indicated by the length of the password alone. In particular, an 8-character password typically has entropy much lower than 64 bits). For example, a 4-digit PIN can be determined with (on average) 5,000 guesses. If only 100 guesses can be made, the probability of success is 2%, which is low, but this certainly represents a feasible attack.

Split Target Data: However, assume 3-DES is being used as the block-cipher in the SSL transaction[9]. Then the plaintext blocks are 64 bits or 8 bytes. There is a 12.5% probability that the four bytes of pin-data will be divided exactly in the middle of a block boundary.[10] *Since each block can be guessed independently*, 50 guesses are required for the 100 options on the left, and 50 guesses are required for the 100 options on the right. This use of 100 guesses results in a 25% chance of success, since both halves must be correct—rather than a mere 2% above. Alternatively, fewer guesses can be made if a lower probability of success is tolerated. Even a 3-1 or 1-3 split results in a major distortion of the number of required guesses. And there is a 37.5% chance that such a split (1-3, 2-2 or 3-1) will occur. See Appendix B for a discussion of applying this technique to passwords.

7 Solutions

It has been noted that TLS 1.1 and OpenSSL (after 0.9.6d) are not vulnerable to this attack, for reasons detailed below. Finally, one can explore why SSL 3.0 and TLS 1.0 are not vulnerable to this attack scenario, unless the version number can be lowered.

7.1 TLS 1.1 and Explicit IVs

The TLS 1.1 protocol [9] fixes this vulnerability by using explicit IVs. That is to say, each message has one more ciphertext block than plaintext blocks. This first ciphertext block is the IV, determined as a (pseudo)random number. As we have stated several times already, this is the accepted way to encrypt using CBC. Since the attacker does not know this value in advance, this attack cannot be executed. Using the formula for CBC, it is easy to see that having an Explicit IV is identical to adding an additional plaintext block of all zeroes to the start of each message, which the receiver knows to discard, and chaining the IVs from message to message [21].

If generating truly random bits is a concern (say, for reasons of efficiency), it is easy to generate a pseudorandom IV in any of a number of ways. For example, instead of simply using C_ℓ (i.e., the last block of the preceding ciphertext) as the IV, the protocol could use $H(C_\ell|sk)$ where sk is the shared secret key used for encryption and H is a cryptographic hash function.

7.2 Single Block Nonces

This solution is mentioned because it can be used in other applications that use CBC, to protect them from BACPA. It also helps to explain the solution used by the OpenSSL community. As was noted previously, the adversary's guess must be the first block of each message. Introducing a one block nonce—which would always be discarded before reading the message—would make this impossible. The nonce does not even need to be random. Suppose the nonce was always the all-zero string, and the previous message ended on block C_i. Then the adversary will submit a block P_{i+2} based on a guess G for block g. This will be output in block

[9] Recall that AES was not available when SSL 3.0 was originally specified.

[10] One must assume that enough traffic has gone by that the "indentation" or "offset" within the stream of the valuable data is unpredictable, and therefore can be treated as a uniformly distributed random variable.

$i+2$, where $i+1$ is the nonce. Therefore the submitted plaintext, from the attack formula given previously would be

$$P_{i+2} = C_{i+1} \oplus C_{g-1} \oplus G$$

But C_{i+1} has not been transmitted yet. It is equal to

$$C_{i+1} = F_{sk}(C_i \oplus 0000 \cdots 0)$$

Therefore, C_{i+1} can only be determined if $F_{sk}(C_i)$ has been calculated before, or the adversary guesses the output of $F_{sk}(C_i)$. Since $F_{sk}(\cdot)$ is a function family believed to be pseudorandom, this guess will be correct only with negligible probability. Likewise, since the space of all possible ciphertexts is 2^{64} or 2^{128}, the probability of a repetition is low. As mentioned earlier, in the special case of the all zero plaintext, this is identical to explicit IVs. The TLS specification for version 1.1 allows for this solution, in addition to Explicit IVs, but strongly recommends that the nonce be pseudorandom, generated for each message independently [9].

7.3 OpenSSL and the Empty Message

A slight variation of the above is used by OpenSSL after version 0.9.6d [25]. An empty message, which consists of no plaintext, but only padding and a hash, is prepended to each set of messages before encryption. The "extra parts", namely the padding and hash, are encrypted, and so form the throw-away blocks similar to the nonce above. Since the adversary's chosen blocks are no longer the first to be encrypted, this attack becomes impossible. What is interesting about this solution is that it does not require any major changes at all to the SSL standard, and fulfills the present definition of SSL as written. (And this further implies only sites, not customers, need change their software). The only point to mention is that some SSL clients will declare an error if an empty message is received. This error message need only be suppressed, which is a minor change.

7.4 The Impact of Directional Keys

When the SSL 3.0 specification was released [14], the keys for each direction of communication were no longer identical. In regards to this attack, TLS 1.0 behaves identical to SSL 3.0. Naturally, data encrypted in distinct keys is fully unrelated and incomparable. Therefore, this scenario would fail, because the outgoing data which contains the PIN would be encrypted in a different key than the audio coming in. One approach would be to try to force the connection to a lower version number, as described in Section 5. This method seems relatively straight-forward and is feasible. However, academically speaking, it is interesting to explore if the attack can occur without lowering the version number.

One might suspect a threat still exists, because it was for this class of attacks, (and the attack by Vaudenay [24]) that changes were made between TLS 1.0 and TLS 1.1, as explained later. To adapt this attack to mono-directional keys, one must induce the target to send data of the attacker's choice. One option is to put a process on the target machine, but then the keystrokes could be read directly, and there is no reason for a cryptographic attack. However, perhaps some strange arrangement of error messages or data forwarded from other users could cause this to occur. Users are well-known for coming up with unusual uses of software, and some application designer could use SSL in a way not yet foreseen. As encryption becomes feasible even on tiny mobile devices like sensors, one observes that protocols are often used in ways and in environments that were not originally anticipated. Therefore, the fact that we cannot think of a mechanism by which the target can be induced to encrypt this choice data, should not be taken as a proof that no such mechanism exists. Otherwise there would have been no need to make changes between TLS 1.0 and TLS 1.1, which the TLS committee indeed made[11].

[11]See also Section 6.2.3.2 of TLS 1.0 and 1.1 [8, 9], and see [21, 24].

7.5 Compression

Note that an immediate way to prevent the attack suggested here is to turn compression on (as we have noted, an attack of the sort suggested here is much more difficult — if not impossible — if compression is used). However, this requires that peers only communicate with others who also use compression (or else an adversary connecting to the honest party could mount a "chosen-protocol attack" in which they claim to be unable to use compression) which would anyway limit inter-operability with deployed versions of SSL.

Furthermore, this solution only requires that the client or server be configured to reject uncompressed sessions during negotiation (or else an adversary connecting to the honest party could mount the "chosen-protocol attack" explained before). However, until all distributed SSL clients have compression included, this could block numerous customers from E-commerce sites.

8 Conclusions

The attack presented here is not so easy that it can be done on the spur of the moment by the typical hacker. However, while the attack is challenging to carry out, particularly with regard to timing, the success probability and relatively low numbers of datagrams required should be sufficient to motivate the SSL community to migrate away from TLS 1.0 and SSL, to OpenSSL after 0.9.6d, or TLS 1.1 when it is finally released.

We are eager to demonstrate the existence of this feasible, though challenging, attack on a real-world protocol which corresponds to the theoretical definition of blockwise-adaptive chosen-plaintext attack, for the four reasons outlined previously, as well as for pedagogical purposes. First, to prove that BACPA is not a sterile model, and practical attacks can be modeled with it. Second, to show that Cipher Block Chaining has disadvantages, and should in some cases be deprecated in favor of modern modes of encryption. Third, to demonstrate that multiple channels of data should not be encrypted in the same key. Fourth, to highlight the potentially critical vulnerability of low-entropy data spaced across block boundaries, when each block can be guessed and verified independently.

Moreover it is hoped that this work will take a step toward opening the dialog between protocol designers and theoretical cryptographers, and stimulate discussion between these two camps which are otherwise independent. Finally, there are other uses of CBC similar to that of SSL, and this attack shows that those applications should also use explicit IVs or another solution listed here (e.g. Datagram Transport Layer Security or DTLS [20]).

Acknowledgments

Thanks to Prof. Jonathan Katz for suggesting the problem, for helpful discussions, and for substantial help editing this document. Thanks also to Ruggero Morselli, Patrick Studdard, Radostina Koleva, Zhongchao Yu, Susan Schmoyer and Prof. Lawrence Washington (all of the University of Maryland), for proofreading early versions of this paper. Valuable feedback was received from Daniel Brown of CertiCom, Eran Tromer of the Weizmann Institute, Eric Rescorla of the Security Consulting Firm RTFM, Prof. Bodo Moeller of UC Berkeley, and Jack Lloyd of randombit.net, about an earlier paper outlining a different attack on this same SSL vulnerability [1]. This was truly a collaborative effort.

References

[1] G. Bard. "The Vulnerability of SSL to Chosen Plaintext Attack." E-print Technical Report 2004/111.

[2] M. Bellare, A. Boldyreva, L. Knudsen, and C. Namprempre. "On-line Ciphers, and the Hash-CBC Constructions." *Advances in Cryptology, Crypto 2001.*

[3] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation." *Proceedings of the IEEE Symposium on the Foundations of Computer Science*, 1997.

[4] M. Bellare, T. Kohno, and C. Namprempre. "Provably Fixing the SSH Binary Packet Protocol." *Proceedings of the Ninth ACM Conference on Computer and Communications Security*, 2002.

[5] M. Bellare and C. Namprempre. "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm." *Advances in Cryptology, Asiacrypt 2000.*

[6] A. Boldyreva, and N. Taesombut. "On-line Encryption Schemes: New Security Notions and Constructions." *Proceedings of the RSA Conference, Cryptographer's Track, 2004.*

[7] W. Dai. "An attack against SSH2 protocol," Feb 2002. Email to the ietf-ssh@netbsd.org email list.

[8] T. Dierks and C. Allen. "RFC 2246: The TLS Protocol, Version 1.0." *Internet Engineering Task Force*, 1999.

`http://www.ietf.org/rfc/rfc2246.txt`

[9] T. Dierks, and E. Rescorla. "The TLS Protocol, Version 1.1." *Internet Engineering Task Force*, 2005.

`http://www.ietf.org/internet-drafts/draft-ietf-tls-rfc2246-bis-11.txt`

[10] M. Dworkin. "NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation, Methods and Techniques." National Institute of Science and Technology: 2001.

[11] M. Dworkin. "NIST Special Publication 800-38B: Recommendation for Block Cipher Modes of Operation: The RMAC Authentication Mode, Methods and Techniques." National Institute of Science and Technology: 2002.

[12] P. Fouque, A. Joux, and G. Poupard. "Blockwise Adversarial Model for On-line Ciphers and Symmetric Encryption Schemes." *Advances in Cryptology, SAC 2004.*

[13] P. Fouque, G. Martinet, and G. Poupard. "Practical Symmetric On-Line Encryption." *Advances in Cryptology, FSE 2003.*

[14] A. Freier, P. Karlton, and P. Kocher. "The SSL Protocol, Version 3.0." *Transport Layer Security Working Group Internet Draft*, 1996.

[15] V. Gligor, and P. Donescu. "Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes." *2nd NIST Workshop on AES Modes of Operation.* 2001.

[16] S. Goldwasser and S. Micali. "Probabilistic Encryption." *Journal of Computer and System Sciences*, 1984.

[17] A. Joux, G. Martinet, and F. Valette. "Blockwise-Adaptive Attackers: Revisiting the (In)Security of Some Provably Secure Encryption Models: CBC, GEM, IACBC." *Advances in Cryptology, Crypto 2002.*

[18] L. Knudsen. "Block Chaining Modes of Operation." *Symmetric Key Block Cipher Modes of Operation Workshop*. 2000.

[19] H. Lipmaa, P. Rogaway, and D. Wagner. "Comments to NIST concerning AES Modes of Operation: CTR-Mode Encryption." *Symmetric Key Block Cipher Modes of Operation Workshop.* 2000.

[20] N. Modadugu, and E. Rescorla. "The Design and Implementation of Datagram TLS." *Proceedings of the Network Distributed System Security Conference 2004.*

[21] B. Moeller, “Security of CBC Ciphersuites in SSL (TLS Problems and Counter Measures).”

`http://www.openssl.org/~bodo/tls-cbc.txt`

[22] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World*, second edition, Prentice Hall, 2002.

[23] H. Krawczyk. “The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?).” *Advances in Cryptology, Crypto 2001.*

[24] S. Vaudenay, “Security Flaw Induced by CBC Padding Applications to SSL, IPSEC, WTLS, ...” *Advances in Cryptology, Eurocrypt 2002.*

[25] Various documents at the Open SSL web-site.

`http://www.openssl.org/`

A General Problems with CBC

The Cipher Block Chaining (CBC) mode of encryption was first proposed as a mode for DES, the Data Encryption Standard [3]. However, in the three decades that have passed since that time, much research has taken place in both adversarial modeling and modes of encryption [2, 10, 11, 13, 15, 18, 19].

A mode of encryption is an algorithm for defining how the block cipher will be used to produce ciphertexts when the plaintext is of length longer than one block. For example, CBC has the formula $C_i = F_{sk}(C_{i-1} \oplus P_i)$ where C_0 is an initialization vector, and Counter Mode (CTR) has the formula $C_i = F_{sk}(i + i_0) \oplus P_i$, where i_0 is an initialization vector [18, 19].

We note the following disadvantages of CBC.

- CBC is vulnerable to blockwise-adaptive chosen-plaintext attack, while CTR is not.
- An error in one block of CBC renders unreadable the remainder of the message, while an error in one block of CTR only renders that block unreadable.
- CBC cannot be parallelized, as can CTR. In CBC each block depends on the encryption of the block before it. In CTR, each block is encrypted independently.
- CBC is subject to the padding attack of Vaudenay [24], but this is avoidable if one pads according to the algorithm given in that paper. In CTR, one can pad arbitrarily and have the padding length as an extra plaintext block at the end.
- CBC offers no protection versus Chosen Ciphertext Attack (CCA), as would HPCBC, XCBC, or OCB [2, 15]. However, Counter Mode also offers no CCA protection.
- An initialization vector need only be calculated once in CTR mode, not per message as in CBC. The counter i can be statefully maintained, and so will not repeat until 2^{64} or 2^{128} blocks.
- While this does not necessarily apply to SSL, the $F_{sk}(i + i_0)$ can be pre-computed, leaving only XOR operations with the plaintext to compute the ciphertext. If data needs to be transmitted only occasionally, but urgently when ready, this can be an advantage.

Since SSL uses Message Authentication Code (MAC) algorithms for the datagrams, the maliability attacks and other CCA attacks that plague CTR mode are of no interest. Therefore it is clear that CTR and not CBC would be a better choice for future versions of TLS and SSL, as CTR has advantages and no disadvantages over CBC [19].

Other Problems with SSL's Encryption: We note that the mode of encryption should probably be changed to address other concerns as well. For example, it is well-known that applying a message authentication code to the ciphertext itself *after* encryption is preferable to applying it to the message *before* encryption [5, 23]. Currently, SSL does the latter rather than the former.

B The Application of Splitting Blocks to Guessing Passwords

While the attack presented in this paper really can only provide for up to about 30–50 guesses at best, under ideal conditions, other chosen plaintext attacks might allow for several more. Therefore it is interesting to point out the effect of splitting upon passwords. If the printable ASCII character set of 95 choices is used, and the passwords are 8 bytes long, and nearly randomly chosen (very generous assumptions), then there are $95^8 = 6.6 \times 10^{15} = 2^{52.6}$ possible passwords, and $2^{51.6} \approx 3.4 \times 10^{15}$ guesses would be required in expectation.

However, the probability of the password not being broken in two is 12.5%. Even with AES and 16-byte blocks, $\frac{7}{16} = 43.8\%$ will be broken into pieces. If divided down the center, (regardless of 64-bit or 128-bit blocks) the number of guesses expected would be

$$\frac{2^{\frac{52.6}{2}} + 2^{\frac{52.6}{2}}}{2} = 2^{26.3} = 8.26 \times 10^7$$

Since 3.4×10^{15} guesses are expected in the unsplit case, and 8.3×10^7 in the split case, the attack becomes 4.1×10^7 times faster when the password is split. While our present attack scenario is not suited for anywhere near this number of guesses, there may be similar scenarios which can tolerate a few thousand guesses, and if a few thousand users are targeted, then at least one password recovery would be expected.