This POODLE Bites: Exploiting The SSL 3.0 Fallback

Security Advisory

Bodo Möller, Thai Duong, Krzysztof Kotowicz

Google

September 2014

{bmoeller, thaidn, koto}@google.com

Introduction

SSL 3.0 [RFC6101] is an obsolete and insecure protocol. While for most practical purposes it has been replaced by its successors TLS 1.0 [RFC2246], TLS 1.1 [RFC4346], and TLS 1.2 [RFC5246], many TLS implementations remain backwards-compatible with SSL 3.0 to interoperate with legacy systems in the interest of a smooth user experience. The protocol handshake provides for authenticated version negotiation, so normally the latest protocol version common to the client and the server will be used.

However, even if a client and server both support a version of TLS, the security level offered by SSL 3.0 is still relevant since many clients implement a protocol downgrade dance to work around server-side interoperability bugs. In this Security Advisory, we discuss how attackers can exploit the downgrade dance and break the cryptographic security of SSL 3.0. Our POODLE attack (Padding Oracle On Downgraded Legacy Encryption) will allow them, for example, to steal “secure” HTTP cookies (or other bearer tokens such as HTTP Authorization header contents).

We then give recommendations for both clients and servers on how to counter the attack: if disabling SSL 3.0 entirely is not acceptable out of interoperability concerns, TLS implementations should make use of TLS_FALLBACK_SCSV.

CVE-2014-3566 has been allocated for this protocol vulnerability.

The POODLE Attack

To work with legacy servers, many TLS clients implement a downgrade dance: in a first handshake attempt, offer the highest protocol version supported by the client; if this handshake fails, retry (possibly repeatedly) with earlier protocol versions. Unlike proper protocol version negotiation (if the client offers TLS 1.2, the server may respond with, say, TLS 1.0), this downgrade can also be triggered by network glitches, or by active attackers. So if an attacker that controls the network between the client and the server interferes with any attempted handshake offering TLS 1.0 or later, such clients will readily confine

themselves to SSL 3.0.

Encryption in SSL 3.0 uses either the RC4 stream cipher, or a block cipher in CBC mode. RC4 is well known to have biases [RC4-biases], meaning that if the same secret (such as a password or HTTP cookie) is sent over many connections and thus encrypted with many RC4 streams, more and more information about it will leak. We show here how to put together an effective attack against CBC encryption as used by SSL 3.0, again assuming that the attacker can modify network transmissions between the client and the server. Unlike with the BEAST [BEAST] and Lucky 13 [Lucky-13] attacks, there is no reasonable workaround. This leaves us with no secure SSL 3.0 cipher suites at all: to achieve secure encryption, SSL 3.0 must be avoided entirely.

The most severe problem of CBC encryption in SSL 3.0 is that its block cipher padding is not deterministic, and not covered by the MAC (Message Authentication Code): thus, the integrity of padding cannot be fully verified when decrypting. Padding by 1 to L bytes (where L is the block size in bytes) is used to obtain an integral number of blocks before performing blockwise CBC (cipher-block chaining) encryption. The weakness is the easiest to exploit if there's an entire block of padding, which (before encryption) consists of L-1 arbitrary bytes followed by a single byte of value L-1. To process an incoming ciphertext record C_1 ... C_n also given an initialization vector C_0 (where each C_i is one block), the recipient first determines P_1 ... P_n as $P_i = D_K(C_i) \oplus C_{i-1}$ (where D_K denotes block-cipher decryption using per-connection key K), then checks and removes the padding at the end, and finally checks and removes a MAC. Now observe that if there's a full block of padding and an attacker replaces C_n by any earlier ciphertext block C_i from the same encrypted stream, the ciphertext will still be accepted if $D_K(C_i) \oplus C_{n-1}$ happens to have L-1 as its final byte, but will in all likelihood be rejected otherwise, giving rise to a padding oracle attack [tls-cbc].

In the web setting, this SSL 3.0 weakness can be exploited by a man-in-the middle attacker to decrypt "secure" HTTP cookies, using techniques from the BEAST attack [BEAST]. To launch the POODLE attack (Padding Oracle On Downgraded Legacy Encryption), run a JavaScript agent on evil.com (or on http://example.com) to get the victim's browser to send cookie-bearing HTTPS requests to https://example.com, and intercept and modify the SSL records sent by the browser in such a way that there's a non-negligible chance that example.com will accept the modified record. If the modified record is accepted, the attacker can decrypt one byte of the cookies.

Assume that each block C has 16 bytes, C[0] ... C[15]. (Eight-byte blocks can be handled similarly.) Also assume, for now, that the size of the cookies is known. (Later we will show how to start the attack if it isn't.) The MAC size in SSL 3.0 CBC cipher suites is typically 20 bytes, so below the CBC layer, an encrypted POST request will look as follows:

`POST` */path* `Cookie:` *name*=*value*...`\r\n\r\n`*body* ‖ *20-byte MAC* ‖ *padding*

The attacker controls both the request path and the request body, and thus can induce requests such that the following two conditions hold:

- The padding fills an entire block (encrypted into C_n).
- The cookies' first as-of-yet unknown byte appears as the final byte in an earlier block (encrypted into C_i).

The attacker then replaces C_n by C_i and forwards this modified SSL record to the server.

Usually, the server will reject this record, and the attacker will simply try again with a new request. Occasionally (on average, once in 256 requests), the server will accept the modified record, and the attacker will conclude that $D_K(C_i)[15] \oplus C_{n-1}[15] = 15$, and thus that $P_i[15] = 15 \oplus C_{n-1}[15] \oplus C_{i-1}[15]$. This reveals the cookies' first previously unknown byte. The attacker proceeds to the next byte by changing the sizes of request path and body simultaneously such that the request size stays the same but the position of the headers is shifted[1], continuing until it has decrypted as much of the cookies as desired. The expected overall effort is 256 SSL 3.0 requests per byte.

As the padding hides the exact size of the payload, the cookies' size is not immediately apparent, but inducing requests `GET /`, `GET /A`, `GET /AA`, ... allows the attacker to observe at which point the block boundary gets crossed: after at most 16 such requests, this will reveal the padding size, and thus the size of the cookies.

Recommendations

The attack described above requires an SSL 3.0 connection to be established, so disabling the SSL 3.0 protocol in the client or in the server (or both) will completely avoid it. If either side supports *only* SSL 3.0, then all hope is gone, and a serious update required to avoid insecure encryption. If SSL 3.0 is neither disabled nor the only possible protocol version, then the attack is possible if the client uses a downgrade dance for interoperability.

Disabling SSL 3.0 entirely right away may not be practical if it is needed occasionally to work with legacy systems. Also, similar protocol version downgrades are still a concern with newer protocol versions (although not nearly as severe as with SSL 3.0). The TLS_FALLBACK_SCSV mechanism from [draft-ietf-tls-downgrade-scsv-00] addresses the broader issue across protocol versions versions, and we consider it crucial especially for systems that maintain SSL 3.0 compatibility. The following recommendations summarize how TLS_FALLBACK_SCSV works.

TLS clients that use a downgrade dance to improve interoperability should include the value 0x56, 0x00 (TLS_FALLBACK_SCSV) in ClientHello.cipher_suites in any fallback handshakes. This value serves as a signal allowing updated servers to reject the connection in case of a downgrade attack. Clients should always fall back to the next lower version (if starting at TLS 1.2, try TLS 1.1 next, then TLS 1.0, then SSL 3.0) because skipping a protocol version forgoes its better security. (With TLS_FALLBACK_SCSV, skipping a version also could entirely prevent a successful handshake if it happens to be the version that should be used with the server in question.)

In **TLS servers,** whenever an incoming connection includes 0x56, 0x00 (TLS_FALLBACK_SCSV) in ClientHello.cipher_suites, compare ClientHello.client_version to the highest protocol version supported by the server. If the server supports a version higher than the one indicated by the client, reject the connection with a fatal alert (preferably, inappropriate_fallback(86) from [draft-ietf-tls-downgrade-scsv-00]).

---

[1] Experimenting with a proof of concept for the POODLE attack, we found that some browsers send request header and request body in separate SSL records. In this case, only the path size needs to be changed when proceeding to the next target byte.

This use of TLS_FALLBACK_SCSV will ensure that SSL 3.0 is used only when a legacy implementation is involved: attackers can no longer force a protocol downgrade. (Attacks remain possible if both parties allow SSL 3.0 but one of them is not updated to support TLS_FALLBACK_SCSV, provided that the client implements a downgrade dance down to SSL 3.0.)

References

[BEAST] T. Duong, J. Rizzo: "Here Come The ⊕ Ninjas", 2011.

[draft-ietf-tls-downgrade-scsv-00] B. Möller, A. Langley: "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks", Internet-Draft draft-ietf-tls-downgrade-scsv-00, 2014.

[Lucky-13] N.J. AlFardan, K.G. Paterson: "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols", IEEE Symposium on Security and Privacy, 2013.

[RC4-biases] N.J. AlFardan, D.J. Bernstein, K.G. Paterson, B. Poettering, J.C.N. Schuldt: "On the Security of RC4 in TLS and WPA", USENIX Security Symposium, 2013.

[RFC2246] T. Dierks, C. Allen: "The TLS Protocol Version 1.0", RFC 2246, 1998.

[RFC4346] T. Dierks, E. Rescorla: "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, 2006.

[RFC5246] T. Dierks, E. Rescorla: "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, 2008.

[RFC6101] A. Freier, P. Karlton, P. Kocher: "The Secure Sockets Layer (SSL) Protocol Version 3.0", RFC 6101, 1996 (published as Historic RFC in 2011).

[tls-cbc] B. Möller: "Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures", http://www.openssl.org/~bodo/tls-cbc.txt, 2004.