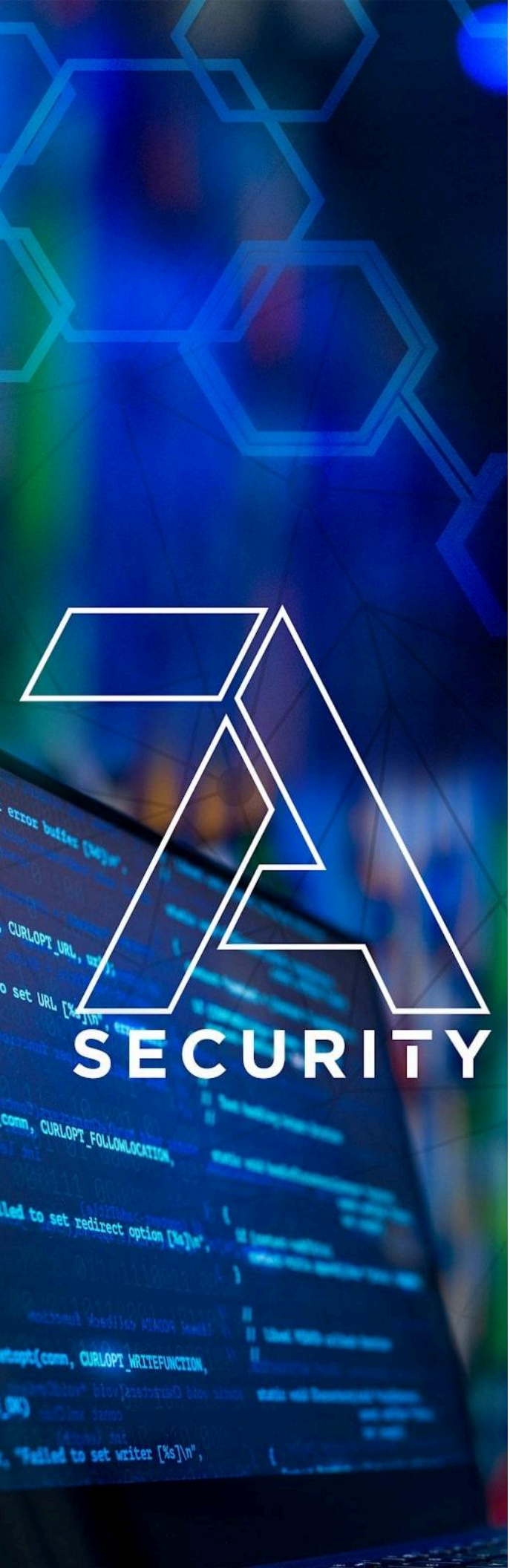


PLATINUM

ISO/IEC 27001:2022
ISMS Certified
by Consilium Labs (IAS)

AICPA SOC 2
Formerly SAS 70 Reports

# Pentest Report

Client:
*DEfO-2*

*in collaboration with the*

*Open Source Technology Improvement Fund, Inc.*

## DEfO Test Targets:

ECH Patchset
OpenSSL Core Integration
Code Review
Configuration & Regression
Threat Model

**7ASecurity Test Team:**

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Dariusz Jastrzębski
- Dheeraj Joshi, BTech.
- Miroslav Štampar, PhD.
- Szymon Grzybowski, MSc.



7ASecurity
*Protect Your Site & Apps*
*From Attackers*
sales@7asecurity.com
7asecurity.com

# INDEX

# Introduction

*"The encrypted ClientHello (ECH) mechanism (draft-spec) is a way to plug a few privacy-holes that remain in the Transport Layer Security (TLS) protocol that's used as the security layer for the web.[...]The DEfO project has developed an implementation of ECH for OpenSSL, and proof-of-concept implementations of various clients and servers that use OpenSSL as a demonstration and for interoperability testing."*

From https://defo.ie/

This document outlines the results of a penetration test and *whitebox* security review conducted against the DEfO project. The security audit was solicited by the DEfO maintainers, facilitated by the *Open Source Technology Improvement Fund, Inc (OSTIF)*, funded by the *Sovereign Tech Fund*, and executed by 7ASecurity in November and December 2025. The audit team dedicated 35 working days to complete this assignment. Please note that this is the second penetration test for this project. Consequently, the identification of security weaknesses was expected to be more difficult during this engagement, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration the goal was to review the solution as thoroughly as possible, to ensure DEfO users can be provided with the best possible security. The methodology implemented was *whitebox*: 7ASecurity was provided with access to documentation and source code. A team of 6 senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of necessary arrangements were in place by November 2025, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email, as well as a shared Element channel. The DEfO team was helpful and responsive throughout the audit, which ensured that 7ASecurity was provided with the necessary access and information at all times, thus avoiding unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

The audit was split across the following work packages:

- WP1: Whitebox Tests against DEfO ECH Patchset & OpenSSL Core Integration
- WP2: Automated and Manual Code Review with Active Tests
- WP3: Configuration & Regression Review against OpenSSL Hardening
- WP4: Lightweight Threat Model for OpenSSL ECH Clients & Servers

The findings of the security audit (WP1-3) can be summarized as follows:

| *Identified Vulnerabilities* | *Hardening Recommendations* | *Total Issues* |
|---|---|---|
| *5* | *6* | *11* |

Please note the results of WP4 are described in the following report sections:

- WP4: Lightweight Threat Model

Moving forward, the scope section elaborates on the items under review, while the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security posture of the DEfO implementation.

# About OSTIF

The *Open Source Technology Improvement Fund (OSTIF)* is dedicated to resourcing and managing security engagements for open source software projects through partnerships with corporate, government, and non-profit donors. We bridge the gap between resources and security outcomes, while supporting and championing the open source community whose efforts underpin our digital landscape.

Over the past ten years, OSTIF has been responsible for the discovery of over 800 vulnerabilities, (121 of those being Critical/High), over 13,000 hours of security work, and millions of dollars raised for open source security. Maximizing output and security outcomes while minimizing labor and cost for projects and funders has resulted in partnerships with multi-billion dollar companies, top open source foundations, government organizations, and respected individuals in the space. Most importantly, we've helped over 120 projects and counting improve their security posture.

Our directive is to support and enrich the open source community through providing public-facing security audits, educational resources, meetups, tooling, and advice. OSTIF's experience positions us to be able to share knowledge of auditing with maintainers, developers, foundations, and the community to further secure our infrastructure in a sustainable manner.

We are a small team working out of Chicago, Illinois. Our website is ostif.org. You can follow us on social media to keep up to date on audits, conferences, meetups, and opportunities with OSTIF, or feel free to reach out directly at contactus@ostif.org or our Github.

Derek Zimmer, Executive Director
Amir Montazery, Managing Director
Helen Woeste, Communications and Community Manager
Tom Welter, Project Manager

# Scope

The following list outlines the items in scope for this project:

- **WP1: DEfO ECH Patchset & OpenSSL Core Integration**
  - Target: https://github.com/openssl/openssl/tree/feature/ech
  - Version Audited: https://github.com/openssl/openssl/tree/daf[…]e
  - Project website: https://defo.ie/
  - Documentation: https://datatracker.ietf.org/doc/draft-ietf-tls-esni/25/
- **WP2: Automated and Manual Code Review with Active Tests**
  - As above
- **WP3: Configuration & Regression Review against OpenSSL Hardening**
  - As above
- **WP4: Lightweight Threat Model for OpenSSL ECH Clients & Servers**
  - As above

# Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *DEF-02-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

## DEF-02-001 WP1/2: DoS via OOB Read in Base64 Decoding *(Medium)*

**Retest Notes:** Related upstream changes are tracked in OpenSSL PR 29593[1].

A heap out-of-bounds read vulnerability was identified in the parsing of Base64-encoded *ECHConfigLists*. The function *ech_read_priv_echconfiglist* passes a raw buffer obtained from the internal helper function *ech_bio2buf* to *BIO_new_mem_buf* with a length argument of -1. This length argument instructs OpenSSL to treat the buffer as a NUL-terminated C string and to compute its length with *strlen*. However, *ech_bio2buf* does not NUL-terminate the allocated buffer. The buffer is populated using *BIO_read_ex*, which performs a raw read and does not append a terminator. Although *ech_bio2buf* initially uses *OPENSSL_zalloc*, it switches to *OPENSSL_realloc* when the input exceeds the initial chunk size, which leaves the bytes immediately following the data uninitialized.

The same buffer is then subjected to a format check using *strspn* before *BIO_new_mem_buf* is called, which demonstrates that the implementation treats this binary buffer as a C string. Because *strspn* relies on NUL-termination, this check alone may trigger a heap out-of-bounds read before the BIO is created. If the input data consists of only Base64 characters and its length exceeds the 512-byte chunk size, the buffer may be followed by uninitialized heap memory. The *strlen* call triggered by *BIO_new_mem_buf* may traverse past the valid data and continue reading uninitialized memory until a null byte is encountered or unmapped memory is reached. This results in a heap out-of-bounds read that may cause a segmentation fault and a DoS condition.

**Affected File:**
https://github.com/openssl/openssl/[...]/crypto/bio/bss_mem.c

**Affected Code:**

```
BIO *BIO_new_mem_buf(const void *buf, int len)
{
    BIO *ret;
    BUF_MEM *b;
    BIO_BUF_MEM *bb;
    size_t sz;
```

[1] https://github.com/openssl/openssl/pull/29593

```
    if (buf == NULL) {
        ERR_raise(ERR_LIB_BIO, ERR_R_PASSED_NULL_PARAMETER);
        return NULL;
    }
    sz = (len < 0) ? strlen(buf) : (size_t)len;
    if ((ret = BIO_new(BIO_s_mem())) == NULL)
        return NULL;
    bb = (BIO_BUF_MEM *)ret->ptr;
    b = bb->buf;
    /* Cast away const and trust in the MEM_RDONLY flag. */
    b->data = (void *)buf;
    b->length = sz;
    b->max = sz;
    *bb->readp = *bb->buf;
    ret->flags |= BIO_FLAGS_MEM_RDONLY;
    /* Since this is static data retrying won't help */
    ret->num = 0;
    return ret;
}
```

**Affected File:**
https://github.com/openssl/openssl/[...]/ssl/ech/ech_store.c

**Affected Code:**

```
static int ech_bio2buf(BIO *in, unsigned char **buf, size_t *len)
{
    unsigned char *lptr = NULL, *lbuf = NULL, *tmp = NULL;
    size_t sofar = 0, readbytes = 0;
    int done = 0, brv, iter = 0;

    if (buf == NULL || len == NULL)
        return 0;
    sofar = OSSL_ECH_BUFCHUNK;
    lbuf = OPENSSL_zalloc(sofar);
    if (lbuf == NULL)
        return 0;
    lptr = lbuf;
    while (!BIO_eof(in) && !done && iter++ < OSSL_ECH_MAXITER) {
        brv = BIO_read_ex(in, lptr, OSSL_ECH_BUFCHUNK, &readbytes);
        if (brv != 1)
            goto err;
        if (readbytes < OSSL_ECH_BUFCHUNK) {
            done = 1;
            break;
        }
        sofar += OSSL_ECH_BUFCHUNK;
        tmp = OPENSSL_realloc(lbuf, sofar);
        if (tmp == NULL)
            goto err;
```

```
            lbuf = tmp;
            lptr = lbuf + sofar - OSSL_ECH_BUFCHUNK;
        }
        if (BIO_eof(in) && done == 1) {
            *len = sofar + readbytes - OSSL_ECH_BUFCHUNK;
            *buf = lbuf;
            return 1;
        }
err:
    OPENSSL_free(lbuf);
    return 0;
}
[...]
static int ech_read_priv_echconfiglist(OSSL_ECHSTORE *es, BIO *in,
                                       EVP_PKEY *priv, int for_retry)
    [...]
    if (ech_bio2buf(in, &encodedval, &encodedlen) != 1) {
        ERR_raise(ERR_LIB_SSL, ERR_R_INTERNAL_ERROR);
        return 0;
    }
    [...]
    if (detfmt == OSSL_ECH_FMT_B64TXT) {
        btmp = BIO_new_mem_buf(encodedval, -1);
        if (btmp == NULL) {
            ERR_raise(ERR_LIB_SSL, ERR_R_INTERNAL_ERROR);
            goto err;
        }
    [...]
}
```

It is recommended to replace the -1 length argument with the explicit *encodedlen* variable when calling *BIO_new_mem_buf*. This ensures that the function uses the known buffer size rather than relying on an unsafe string length calculation on non-string binary data.

### DEF-02-003 WP3: Silent FIPS Compliance Violation *(High)*

**Retest Notes:** The issue was addressed in OpenSSL PR 29439[2], and 7ASecurity confirmed that the fix is valid.

A compliance vulnerability was identified in which the ECH implementation silently bypasses FIPS restrictions. While the design documentation explicitly states that FIPS compliance for ECH is not yet supported, the implementation fails to enforce this limitation securely. Instead of rejecting non-compliant algorithms when operating in FIPS mode, the code decouples ECH cryptographic operations from the parent security context. This allows otherwise FIPS-restricted applications to silently utilize non-validated cryptography from the *Default* provider, creating a "silent compliance violation" in which an application appears FIPS-compliant while operating outside FIPS 140-3 requirements[3]. In FIPS-regulated environments, data protected using non-validated cryptography is treated as not FIPS-approved for compliance purposes[4]. Consequently, this bypass results in a complete loss of confidentiality assurance and renders the module non-compliant with FIPS-approved operation for the duration of the session.

The technical root cause is that the ECH cryptographic initialization logic calls OSSL_HPKE_CTX_new with NULL for both the library context and the property query string. By passing NULL, the implementation causes OpenSSL to use the global default context and properties, explicitly ignoring any specific constraints (such as fips=yes) enforced on the *SSL_CONNECTION* or *SSL_CTX*.

This behavior was verified dynamically against the target build. A client configured with strict FIPS enforcement (*-propquery "fips=yes"*) was observed successfully generating an ECH *ClientHello* using the x25519 KEM. In a secure implementation, this operation would have failed immediately, as x25519 is not FIPS-approved for FIPS 140-3 approved-mode operation. The successful generation demonstrates that the intended FIPS constraint was not enforced for this code path.

**Affected File:**
https://github.com/openssl/openssl/[...]/ssl/ech/ech_internal.c

**Affected Code:**

```
static unsigned char *hpke_decrypt_encch(SSL_CONNECTION *s,
                                         OSSL_ECHSTORE_ENTRY *ee,
                                         OSSL_ECH_ENCCH *the_ech,
                                         size_t aad_len, unsigned char *aad,
```

[2] https://github.com/openssl/openssl/pull/29439
[3] https://edu.chainguard.dev/chainguard/fips/non-approved-algorithms/
[4] https://csrc.nist.gov/projects/cryptographic-module-validation-program

```
                                        int forhrr, size_t *innerlen)
{
    [...]
    hctx = OSSL_HPKE_CTX_new(hpke_mode, hpke_suite, OSSL_HPKE_ROLE_RECEIVER,
                             NULL, NULL);
    [...]
}
```

**Affected File:**
https://github.com/openssl/openssl/[...]/ssl/statem/extensions_clnt.c

**Affected Code:**

```
EXT_RETURN tls_construct_ctos_ech(SSL_CONNECTION *s, WPACKET *pkt,
                                  unsigned int context, X509 *x,
                                  size_t chainidx)
{
    [...]
    s->ext.ech.hpke_ctx = OSSL_HPKE_CTX_new(hpke_mode, hpke_suite,
                                            OSSL_HPKE_ROLE_SENDER,
                                            NULL, NULL);
    [...]
}
```

**Affected File:**
https://github.com/openssl/openssl/[...]/doc/designs/ech-api.md

**Affected Contents:**

```
- There are all the usual algorithm variations, and those will likely result in
  the same x25519 versus p256 combinatorics. How that plays out has yet to be
  seen as FIPS compliance for ECH is not (yet) a thing. For OpenSSL, it seems
  wise to be agnostic and support all relevant combinations.
```

**Affected File:**
https://github.com/openssl/openssl/[...]/doc/man1/openssl-ech.pod.in

**Affected Contents:**

```
For example the default is: x25519, hkdf-sha256, aes128gcm
```

It is recommended to modify the *OSSL_HPKE_CTX_new* calls in both *ssl/ech/ech_internal.c* and *ssl/statem/extensions_clnt.c* to propagate the *libctx* and *propq* from the parent *SSL_CONNECTION* instead of passing *NULL*. This ensures that if FIPS is enforced, non-compliant algorithms will correctly fail to load.

### DEF-02-005 WP1/2: ECH Downgrade Protection Bypass *(High)*

**Retest Notes:** Related upstream changes are tracked in OpenSSL PR 29593[5].

A cryptographic logic flaw exists in the *tls_construct_server_hello* function that neutralizes the TLS 1.3 anti-downgrade protection mechanism. The TLS 1.3 specification[6] mandates that a TLS 1.3-capable server negotiating TLS 1.2 must overwrite the last eight bytes of *ServerHello.random* with a specific sentinel value (*44 4F 57 4E 47 52 44 01*) to signal support for a higher protocol version. The ECH implementation introduces a logic path that overwrites these same eight bytes whenever ECH was attempted and accepted (that is, when *backend == 1* or *ech.success == 1*), without verifying the negotiated protocol version. The ECH specification explicitly warns against this exact behavior[7]: "*The backend server MUST NOT perform this operation if it negotiated TLS 1.2 or below. Note that doing so would overwrite the downgrade signal for TLS 1.3*".

The implementation incorrectly treats *"ECH attempted and accepted"* and the negotiated protocol version as coupled states. Because *ossl_ech_early_decrypt* executes before protocol version selection, a server can successfully decrypt an ECH *ClientHello* and subsequently negotiate TLS 1.2 due to client preference, local configuration, or active network manipulation. The overwrite is triggered whenever ECH was attempted and either backend mode is active or *ech.success* is set, regardless of the final protocol version. This corrupts *ServerHello.random* in TLS 1.2 handshakes by overwriting the downgrade sentinel required by RFC 8446, preventing the client from detecting the downgrade.

This flaw may allow an active attacker to mask downgrade attacks. An attacker can strip TLS 1.3 capabilities from the *ClientHello*, forcing negotiation of TLS 1.2. Under normal conditions, the server would insert the downgrade sentinel to warn the client. However, the ECH logic overwrites this sentinel with the ECH accept confirmation value, even though ECH acceptance must not be signaled for TLS 1.2 handshakes. This behavior directly contradicts the ECH specification, which mandates that the accept confirmation value is inserted only when TLS 1.3 or higher is negotiated. Inserting it into a TLS 1.2 *ServerHello* violates this requirement and inherently breaks the TLS 1.3 downgrade-signal mechanism.

**Affected File:**
https://github.com/openssl/openssl/[...]/ssl/statem/statem_srvr.c

[5] https://github.com/openssl/openssl/pull/29593
[6] https://datatracker.ietf.org/doc/html/rfc8446#section-4.1
[7] https://www.ietf.org/archive/id/draft-ietf-tls-esni-18.html#section-7.2

**Affected Code:**

```
CON_FUNC_RETURN tls_construct_server_hello(SSL_CONNECTION *s, WPACKET *pkt)
{
    [...]
    if (s->ext.ech.attempted_type == TLSEXT_TYPE_ech
        && (s->ext.ech.backend == 1
            || (s->ext.ech.es != NULL && s->ext.ech.success == 1))) {
        [...]
        if (ossl_ech_calc_confirm(s, hrr, acbuf, shlen) != 1) {
            SSLfatal(s, SSL_AD_INTERNAL_ERROR, ERR_R_INTERNAL_ERROR);
            return CON_FUNC_ERROR;
        }
        memcpy(s->s3.server_random + SSL3_RANDOM_SIZE - 8, acbuf, 8);
        [...]
    }
    [...]
}
```

It is recommended to update the server-side logic to ensure that the ECH accept confirmation overwrite is performed only when the final negotiated protocol version is TLS 1.3 or higher. This can be enforced by ensuring the overwrite occurs only once TLS 1.3 has been selected (once the negotiated version is known), rather than gating solely on ECH state. This change preserves the integrity of the TLS 1.3 downgrade protection mechanism and prevents accidental erasure of downgrade sentinel values that are required to protect clients against version stripping attacks.

### DEF-02-007 WP3: HPKE Algorithms Not Subject to Security Policy *(Medium)*

**Note:** Related upstream discussion and changes are tracked in OpenSSL PR 29593[8].

A security policy gap exists in the ECH cipher suite selection logic. OpenSSL provides an SSL/TLS security framework (security levels and an optional security callback) to enforce cryptographic strength requirements and to ensure that negotiated algorithms meet the configured Security Level (for example, Level 2 requires at least 112 bits of security). The standard TLS handshake logic checks cipher suites and key exchange groups against this policy. However, the ECH implementation does not integrate HPKE suite selection into this framework.

In the *ossl_ech_pick_matching_cfg* (client) and *ech_decode_one_entry* (server) functions, candidate ECH cipher suites are validated through the *OSSL_HPKE_suite_check* helper. The implementation of *OSSL_HPKE_suite_check* accepts only suite identifiers as arguments and has no access to the *SSL_CTX* or *SSL* object. As a result, it is inherently unable to verify compliance with the configured

[8] https://github.com/openssl/openssl/pull/29593

security level or cipher restrictions. It performs an availability check to confirm that the suite is supported locally, rather than a policy-based check.

This allows ECH to operate outside the application-defined security policy. In deployments that rely on high security levels or restricted cipher lists to enforce specific cryptographic standards (for example, an AES-256-only policy), ECH may still negotiate algorithms that fall below the required threshold if they are offered in the ECH configuration. This creates a discrepancy between the operator-configured cryptographic policy and the algorithms actually used for ECH. HPKE protects only the ECH *Inner ClientHello* (SNI, ALPN, and related metadata), not the TLS record layer, but some organizations may still require these elements to be protected with algorithms that meet the required security level. This weakens the confidentiality guarantees for handshake metadata by allowing encryption with algorithms that do not meet the mandatory security threshold of the organization.

**Affected File:**
https://github.com/openssl/openssl/[...]/ssl/ech/ech_internal.c

**Affected Code:**
```
int ossl_ech_pick_matching_cfg(SSL_CONNECTION *s, OSSL_ECHSTORE_ENTRY **ee,
                               OSSL_HPKE_SUITE *suite)
{
   [...]
    // 7asec NOTE: Checks only for implementation support, ignoring Security Policy
    if (OSSL_HPKE_suite_check(lee->suites[csuite]) == 1) {
        if (tee == NULL) {
            tee = lee;
            [...]
        }
    }
    [...]
}
```

**Affected File:**
https://github.com/openssl/openssl/[...]/crypto/hpke/hpke.c

**Affected Code:**
```
int OSSL_HPKE_suite_check(OSSL_HPKE_SUITE suite)
{
    return hpke_suite_check(suite, NULL, NULL, NULL);
}
```

It is recommended to integrate ECH HPKE suite selection with the OpenSSL security policy framework. This can be achieved by extending the *ssl_security* callback to understand and validate HPKE algorithms, or by implementing an equivalent check

within the ECH logic that uses the *SSL_CONNECTION* context to reject HPKE suites that do not meet the configured security level or cipher restrictions.

### DEF-02-009 WP1/2: Certificate Verification DoS in ECH Retry *(Medium)*

**Note:** Related upstream changes are tracked in OpenSSL PR 29593[9].

A vulnerability exists in the logic that handles rejected ECH attempts. When a server rejects an ECH offer (for example, due to key rotation) and provides *retry_configs*, the client is required to authenticate the server public name before accepting the new configuration. The implementation updates the SNI metadata (*s->ext.hostname*) to the public name but does not propagate this update to the *X509_VERIFY_PARAM* structure (*s->param*). As a result, the verification engine continues to validate the server certificate against the original inner hostname. In deployments where the “public name” used for retry differs from the inner origin name (including shared-mode and split-mode scenarios), this leads to a verification mismatch (X509_V_ERR_HOSTNAME_MISMATCH).

This flaw creates a denial-of-service (DoS) vector. The ECH protocol is designed to recover from stale keys through *retry_configs*. This bug breaks that recovery mechanism and turns the protocol-defined retry into immediate connection termination. Section 8.1.1 of the ECH specification[10] explicitly identifies this retry mechanism as a means to repair state inconsistencies, such as standard key rotation. Therefore, the failure constitutes a protocol-level denial of service for legitimate clients with stale DNS state, rather than a client configuration issue.

An attacker who can influence the client view of the ECH configuration (for example, through DNS cache poisoning or replay of an expired DNS record) can supply the client with a stale ECH key. Under normal conditions, the protocol would self-heal. Due to this vulnerability, however, verification fails permanently and the attacker can deny service to the client for as long as the stale DNS record is maintained.

**Affected File:**
https://github.com/openssl/openssl/[...]/ssl/statem/statem_clnt.c

**Affected Code:**

```
MSG_PROCESS_RETURN tls_process_server_hello(SSL_CONNECTION *s, PACKET *pkt)
{
    [...]
    } else if (!hrr) {
        /*
```

[9] https://github.com/openssl/openssl/pull/29593
[10] https://www.ietf.org/archive/id/draft-ietf-tls-esni-18.html#name-misconfiguration-and-deploy

```
             * If we got retry_configs then we should be validating
             * the outer CH, so we better set the hostname for the
             * connection accordingly.
             */
            s->ext.ech.former_inner = s->ext.hostname;
            s->ext.hostname = NULL;
            if (s->ext.ech.outer_hostname != NULL) {
                s->ext.hostname = OPENSSL_strdup(s->ext.ech.outer_hostname);
                if (s->ext.hostname == NULL) {
                    SSLfatal(s, SSL_AD_INTERNAL_ERROR, ERR_R_INTERNAL_ERROR);
                    goto err;
                }
            }
        }
        [...]
}
```

The error-handling path must call *SSL_set1_host(ssl, s->ext.ech.outer_hostname)* immediately after updating *s->ext.hostname*. This keeps the *X509_VERIFY_PARAM* structure synchronized with the new public identity, allowing the client to authenticate the server successfully and process *retry_configs* to restore connectivity.

# Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

### DEF-02-002 WP1/2: Asymmetric DoS via Unbounded Trial Decryption *(Medium)*

**Retest Notes:** Related upstream changes are tracked in OpenSSL PR 29593[11].

An issue was identified in the server-side Encrypted *ClientHello* trial decryption logic that permits an asymmetric DoS attack when the optional trial decryption feature is enabled. The implementation allows a client to trigger a large number of expensive cryptographic operations by sending a single malicious packet containing an unknown configuration identifier. When a *ClientHello* with an unrecognized configuration ID is received and the *SSL_OP_ECH_TRIALDECRYPT* option is enabled, the server attempts to decrypt the message using every key in the key store. This linear iteration through all available keys creates a computational imbalance, with complexity proportional to the number of keys, so that a low-bandwidth flood of malicious *ClientHello* messages can quickly cause CPU exhaustion on the server.

The root cause is the linear iteration through all configured trial-decryption keys when SSL_OP_ECH_TRIALDECRYPT is enabled, combined with the lack of deployment-level DoS mitigations (for example, rate limiting) for this optional feature.

The *ossl_ech_early_decrypt* function contains an unbounded loop that iterates through every loaded ECH configuration key when no direct configuration match is found. For each iteration, the server calls *hpke_decrypt_encch* and performs a full HPKE decapsulation, which requires expensive elliptic-curve scalar multiplications. By sending a *ClientHello* with a randomized or unrecognized *config_id*, an attacker ensures that every decryption attempt fails, forcing the server to attempt decryption against every key in the store, which matches the “wasteful decryption” scenario warned about in the TLS ECH specification[12], before the packet is rejected.

The impact of this flaw is a potential loss of service availability due to resource exhaustion on servers that enable the trial decryption feature. The ECH protocol

[11] https://github.com/openssl/openssl/pull/29593
[12] https://www.ietf.org/archive/id/draft-ietf-tls-esni-18.html#section-10.4

specification permits trial decryption to preserve privacy and reduce partitioning risk, but the TLS ECH specification explicitly recommends rate limiting and related DoS mitigations, which are missing in this implementation. If a server administrator enables this option without understanding these risks, an attacker can exhaust server processing capacity without requiring high bandwidth.

**Affected File:**
https://github.com/openssl/openssl/[...]/ssl/ech/ech_internal.c

**Affected Code:**

```
int ossl_ech_early_decrypt(SSL_CONNECTION *s, PACKET *outerpkt, PACKET *newpkt)
{
    [...]
    num = (es == NULL || es->entries == NULL ? 0
           : sk_OSSL_ECHSTORE_ENTRY_num(es->entries));
    [...]
    if (foundcfg == 1) {
        clear = hpke_decrypt_encch(s, ee, extval, aad_len, aad,
                                   forhrr, &clearlen);
        if (clear == NULL)
            s->ext.ech.grease = OSSL_ECH_IS_GREASE;
    }
    /* if still needed, trial decryptions */
    if (clear == NULL && (s->options & SSL_OP_ECH_TRIALDECRYPT)) {
        foundcfg = 0; /* reset as we're trying again */
        for (cfgind = 0; cfgind != num; cfgind++) {
            ee = sk_OSSL_ECHSTORE_ENTRY_value(es->entries, cfgind);
            clear = hpke_decrypt_encch(s, ee, extval,
                                       aad_len, aad, forhrr, &clearlen);
            if (clear != NULL) {
                foundcfg = 1;
                s->ext.ech.grease = OSSL_ECH_NOT_GREASE;
                break;
            }
        }
    }
    [...]
}
```

It is recommended to document the DoS characteristics of *SSL_OP_ECH_TRIALDECRYPT* and to treat it as an operationally sensitive option when enabled. Operational controls should be used to bound per-connection work, for example by keeping the trial-decryption key store small (pruning stale keys and prioritizing the most likely configuration first) to keep per-packet work bounded and predictable. It is also recommended to maintain the current default behavior where *SSL_OP_ECH_TRIALDECRYPT* is disabled, to preserve secure-by-default behavior for deployments that do not require this specific privacy feature. It is recommended, for

applications using the OpenSSL library, to implement rate limiting for connections that trigger trial decryption, as described in the ECH draft.

### DEF-02-004 WP2: ECH Decryption Failures Misclassified as GREASE *(Low)*

**Note:** Discussed in OpenSSL PR 29593[13].

A hardening gap was identified in the *hpke_decrypt_encch* function, where the ECH implementation does not distinguish between a client intentionally sending a GREASE value and an ECH decryption failure after a matching configuration identifier is found. Server-side HPKE errors during ECH decryption are suppressed from the OpenSSL error stack through the use of *ERR_set_mark* and *ERR_pop_to_mark*. All failure modes, including incorrect keys, incorrect suites, provider failures, and internal HPKE errors, are collapsed into a generic decryption failure and a *NULL* return value, which the caller interprets as *"ECH did not work, fall back to Outer ClientHello"*.

The principal challenge is that ECH decryption failures are indistinguishable from GREASE traffic in server logs and metrics unless the integrating server records and exports this state (for example, via *SSL_ech_get1_status()*). As a result, operational telemetry cannot distinguish between benign GREASE, configuration mismatches, and other decryption failures. This conflation compromises both security monitoring and deployment health metrics. ECH failure rates cannot accurately reflect genuine configuration or interoperability issues. Due to the one-octet *config_id*, collisions between GREASE values and in-use configuration identifiers are possible.

**Affected File:**
https://github.com/openssl/openssl/blob/[...]/ssl/ech/ech_internal.c

**Affected Code:**

```
static unsigned char *hpke_decrypt_encch(SSL_CONNECTION *s,
                                         OSSL_ECHSTORE_ENTRY *ee,
                                         OSSL_ECH_ENCCH *the_ech,
                                         size_t aad_len, unsigned char *aad,
                                         int forhrr, size_t *innerlen)
{
    ...
    ERR_set_mark();
    hctx = OSSL_HPKE_CTX_new(hpke_mode, hpke_suite,
                             OSSL_HPKE_ROLE_RECEIVER, NULL, NULL);
    ...
    rv = OSSL_HPKE_decap(hctx, ...);
    if (rv != 1)
        goto clearerrs;
```

[13] https://github.com/openssl/openssl/pull/29593

```
    rv = OSSL_HPKE_open(hctx, clear, &clearlen, aad, aad_len,
                        cipher, cipherlen);

 clearerrs:
    /* close off our error handling */
    ERR_pop_to_mark();
    ...
    OSSL_HPKE_CTX_free(hctx);
    if (rv != 1) {
        OSSL_TRACE(TLS, "HPKE decryption failed somehow\n");
        OPENSSL_free(clear);
        return NULL;
    }
    ...
}
```

**Affected File:**
https://github.com/openssl/openssl/blob/[...]/ssl/ech/ech_internal.c

**Affected Code:**

```
int ossl_ech_early_decrypt(SSL_CONNECTION *s, PACKET *outerpkt, PACKET *newpkt)
{
  [...]

  if (foundcfg == 1) {
      clear = hpke_decrypt_encch(s, ee, extval, aad_len, aad, forhrr, &clearlen);
      if (clear == NULL)
          s->ext.ech.grease = OSSL_ECH_IS_GREASE;
  }
```

The severity of this finding has been reduced because the client-side implementation enforces ECH success through cryptographic acceptance signals and aborts the connection if ECH was attempted but not successfully confirmed. This significantly reduces the risk of a silent ECH downgrade.

**Example File:**
https://github.com/openssl/openssl/blob/[...]/ssl/statem/statem_clnt.c

**Example Code:**

```
if (ossl_ech_find_confirm(s, hrr, s_signal) != 1
    || memcmp(s_signal, c_signal, sizeof(c_signal)) != 0) {
    s->ext.ech.success = 0;  // ECH failed
} else {
    s->ext.ech.success = 1;  // ECH succeeded
}

// Lines 3389-3395 - Client enforcement
```

```
if (s->ext.ech.es != NULL
    && s->ext.ech.attempted == 1
    && s->ext.ech.success != 1
    && s->ext.ech.grease != OSSL_ECH_IS_GREASE) {
    SSLfatal(s, SSL_AD_ECH_REQUIRED, SSL_R_ECH_REQUIRED);
    return 0;  // ABORT CONNECTION
}
```

**Example File:**
https://github.com/openssl/openssl/blob/[...]/ssl/statem/statem_clnt.c

**Example Code:**

```
if (s->ext.ech.es != NULL
   && s->ext.ech.attempted == 1
   && s->ext.ech.success != 1
   && s->ext.ech.grease != OSSL_ECH_IS_GREASE) {
   SSLfatal(s, SSL_AD_ECH_REQUIRED, SSL_R_ECH_REQUIRED);
   return 0;  // ABORT CONNECTION
}
```

It is recommended to modify the logic in ssl/ech/ech_internal.c to record a distinct, rate-limited failure state when a configuration identifier is matched (foundcfg == 1) but HPKE decryption fails, rather than defaulting to the GREASE state. If the server successfully matches a configuration identifier (*foundcfg == 1*) but decryption fails, a distinct decryption failure state should be recorded or logged rather than defaulting to the GREASE state. This improves operational visibility while preserving the required fallback to the *Outer ClientHello*.

### DEF-02-006 WP1/2: Inner ClientHello Protocol Version Downgrade *(Info)*

**Note:** This issue was later confirmed to be mitigated by design. Related upstream changes are tracked in OpenSSL PR 29593[14].

A protocol compliance and security flaw exists in the construction logic for the ECH *Inner ClientHello*. The ECH specification mandates[15] that the *Inner ClientHello* "*MUST NOT offer to negotiate TLS 1.2 or below*". This requirement ensures that ECH protected connections are bound to TLS 1.3 or higher and prevents downgrade attacks in which encrypted parameters are used in a weaker protocol context.

In unmodified builds, this constraint is enforced because the ECH extension is emitted only for TLS 1.3 (SSL_EXT_TLS1_3_ONLY). The *tls_construct_client_hello* function reuses the standard protocol version selection logic (*ssl_set_client_hello_version*) for

[14] https://github.com/openssl/openssl/pull/29593
[15] https://www.ietf.org/archive/id/draft-ietf-tls-esni-18.html#section-6.1

both the *Outer* and *Inner ClientHellos*. This logic relies solely on the *SSL_CTX* configuration and does not account for the ECH context (*s->ext.ech.ch_depth*). As a result, if the client *SSL_CTX* enables TLS 1.2, either as the maximum protocol version or as part of an allowed range, the constructed *Inner ClientHello* offers TLS 1.2 parameters. This occurs either through emission of a legacy TLS 1.2 ClientHello (when the maximum version is set to TLS 1.2) or by including TLS 1.2 in the *supported_versions* extension (when the maximum version is TLS 1.3). Both cases violate the ECH specification.

If the TLS 1.3-only restriction were regressed, constructing an Inner ClientHello that advertises TLS 1.2 would violate the ECH specification and could interact with downgrade-related logic (for example, DEF-02-005). When such a message is processed by a vulnerable server, the ECH payload is decrypted successfully, but TLS 1.2 may be negotiated, for example if an attacker strips the TLS 1.3 offer. This state combination triggers the *Downgrade Protection Bypass* vulnerability (DEF-02-005) on the server side and results in the anti-downgrade sentinel being overwritten. Because the *Inner ClientHello* protocol version is not clamped to TLS 1.3, the client implementation may enable exploitation of server-side state machine defects under configurations that allow TLS 1.2.

**Affected File:**
https://github.com/openssl/openssl/[...]/ssl/statem/statem_clnt.c

**Affected Code:**

```
__owur CON_FUNC_RETURN tls_construct_client_hello(SSL_CONNECTION *s,
                                                  WPACKET *pkt)
    [...]
    // 7Asec NOTE: does not force TLS 1.3 minimum for the encrypted payload
    int protverr = ssl_set_client_hello_version(s);
    if (protverr != 0) {
        SSLfatal(s, SSL_AD_INTERNAL_ERROR, protverr);
        return 0;
    }
    [...]
    /*
     * Set CH depth flag so that other code (e.g. extension handlers)
     * know where we're at: 1 is "inner CH", 0 is "outer CH"
     */
    s->ext.ech.ch_depth = 1;
    if ((inner_mem = BUF_MEM_new()) == NULL
        || !WPACKET_init(&inner, inner_mem)
        || !ssl_set_handshake_header(s, &inner, mt)
        || tls_construct_client_hello_aux(s, &inner) != 1
        || !WPACKET_close(&inner)
        || !WPACKET_get_length(&inner, &innerlen)) {
        SSLfatal(s, SSL_AD_INTERNAL_ERROR, ERR_R_INTERNAL_ERROR);
```

```
            goto err;
        }
        [...]
}
```

It is recommended to add regression coverage to ensure ECH remains TLS 1.3-only in practice (that is, that the Inner ClientHello cannot advertise TLS 1.2 or below), preventing future regressions of this design invariant. When the *Inner ClientHello* is constructed (*s->ext.ech.ch_depth == 1*), the implementation must explicitly force *TLS1_3_VERSION* as the minimum protocol version and ensure that the *supported_versions* extension does not advertise lower versions. If the client configuration does not support TLS 1.3, ECH construction must fail or be disabled rather than generating a non-compliant *Inner ClientHello*.

### DEF-02-008 WP1/2: ClientHello Parser Accepts Truncated Input *(Low)*

**Note:** Discussed in OpenSSL PR 29593[16].

A hardening gap exists in the ClientHello offset parsing logic implemented in *ossl_ech_helper_get_ch_offsets*. The function can return success (value 1) for both well-formed *ClientHello* messages and malformed or truncated inputs. As a result, truncated *ClientHello* messages are treated as valid messages without extensions, rather than being rejected as decoding errors. This behavior weakens the robustness of ECH processing by allowing malformed inputs to proceed through parsing logic intended only for valid *ClientHellos*.

The implementation explicitly documents that error handling is suppressed to preserve compatibility with legacy test cases. As noted in the source code, truncated *ClientHello* messages are intentionally accepted to avoid breaking non-ECH test code, even though the input is incomplete. This design choice prioritizes test compatibility over strict input validation.

**Affected File:**
https://github.com/openssl/openssl/blob/[...]/ssl/ech/ech_helper.c

**Affected Contents:**

```
/*
* unexpectedly, we return 1 here, as doing otherwise will
* break some non-ECH test code that truncates CH messages
* The same is true below when looking through extensions.
* That's ok though, we'll only set those offsets we've
* found. [...]
```

[16] https://github.com/openssl/openssl/pull/29593

**Affected File:**
https://github.com/openssl/openssl/blob/[...]/ssl/ech/ech_internal.c

**Affected Code:**

```
if (ossl_ech_helper_get_ch_offsets(ch, ch_len, sessid_off, exts_off,
                                   &exts_len, ech_off, echtype, &ech_len,
                                   sni_off, &sni_len, inner) != 1) {
        SSLfatal(s, SSL_AD_DECODE_ERROR, SSL_R_BAD_EXTENSION);
        return 0;
    }
# ifdef OSSL_ECH_SUPERVERBOSE
    [...]
# endif
    s->ext.ech.sessid_off = *sessid_off;
    s->ext.ech.exts_off = *exts_off;
    s->ext.ech.ech_off = *ech_off;
    s->ext.ech.echtype = *echtype;
    s->ext.ech.inner = *inner;
    s->ext.ech.sni_off = *sni_off;
    s->ext.ech.ch_offsets_done = 1;
    return 1;
```

This behavior reduces parser strictness and may obscure malformed input handling during ECH processing, even if the ClientHello is later rejected by standard parsing. While no direct exploit is demonstrated, accepting truncated inputs as valid weakens defensive parsing guarantees and complicates reasoning about ECH failure modes.

It is recommended to modify *ossl_ech_helper_get_ch_offsets* to return a distinct failure indication for malformed or truncated *ClientHello* messages. A return value of 1 should indicate successful parsing of a well-formed *ClientHello*, with or without relevant extensions, while a return value of 0 should indicate malformed or truncated input. The calling code should treat a return value of 0 as a decode error and trigger *SSL_AD_DECODE_ERROR*. If strict rejection is not currently feasible due to test compatibility constraints, this behavior should be documented and revisited alongside upstream test adjustments to avoid weakening production parsing behavior.

### DEF-02-010 WP1: Retry Config Concatenation size_t Overflow *(Low)*

**Note:** Related upstream changes are tracked in OpenSSL PR 29593[17].

A defensive coding gap exists in the *ossl_ech_get_retry_configs* function, which concatenates a sequence of encoded retry configuration blobs into a single buffer. The allocation size is calculated using *retslen + ee->encoded_len* without verifying whether the addition overflows *size_t* or whether *ee->encoded_len* is within reasonable bounds.

On platforms where *size_t* is a narrower integer type, this pattern can introduce a memory safety risk. A malicious or malformed configuration containing multiple entries with large encoded_len values could cause the addition *retslen + ee->encoded_len* to wrap around. This overflow would result in an undersized allocation, potentially leading to a heap overwrite during the subsequent *memcpy* operation. While protocol-level constraints make this difficult to exploit in typical TLS deployments, the underlying pattern represents an unsafe allocation practice.

**Affected File:**
https://github.com/openssl/openssl/blob/ab8a5bc1ea2afa6afe648fe3f1681ad6c5dec1be/ssl/ech/ech_internal.c#L236-L270

**Affected Code:**

```
[...]
int ossl_ech_get_retry_configs(SSL_CONNECTION *s, unsigned char **rcfgs,
                               size_t *rcfgslen)
{
    [...]
    size_t retslen = 0;
    unsigned char *tmp = NULL, *rets = NULL;
    [...]
    for (i = 0; i != num; i++) {
        ee = sk_OSSL_ECHSTORE_ENTRY_value(es->entries, i);
        if (ee != NULL && ee->for_retry == OSSL_ECH_FOR_RETRY) {
            tmp = (unsigned char *)OPENSSL_realloc(rets,
                                                   retslen + ee->encoded_len);
            if (tmp == NULL)
                goto err;
            rets = tmp;
            memcpy(rets + retslen, ee->encoded, ee->encoded_len);
            retslen += ee->encoded_len;
        }
    }
    *rcfgs = rets;
    *rcfgslen = retslen;
    [...]
```

[17] https://github.com/openssl/openssl/pull/29593

It is recommended to check whether the addition *retslen + ee->encoded_len* would overflow *size_t* before performing the allocation. Reasonable upper bounds should also be enforced on the total retry configuration buffer size to prevent unsafe allocations.

### DEF-02-011 WP1: ECH Transcript Buffer size_t Overflow *(Info)*

**Note:** TLS handshake message size limits constrain transcript growth; a *size_t* overflow is not expected in practice unless *size_t* is too small for TLS to operate.

A defensive coding gap similar to DEF-02-010 exists in the *ossl_ech_intbuf_add* function, where the ECH transcript buffer (*transbuf*) is grown using *OPENSSL_realloc* without validating arithmetic on the resulting allocation size. The new buffer size is computed by adding the existing buffer length to the incoming length without checking for *size_t* overflow.

If blen could be influenced by untrusted input, or if the transcript length could grow repeatedly, the addition could overflow size_t and wrap around. Note: TLS transcript sizes are bounded by protocol limits (logical maximum 2^32-1), so an overflow is not expected in practice on platforms where TLS can operate correctly. This would result in an allocation that is smaller than intended. Subsequent use of *memcpy* would then write beyond the bounds of the allocated buffer, potentially leading to heap corruption or a segmentation fault. While TLS transcript sizes are bounded by protocol constraints, the pattern represents an unsafe memory allocation practice.

**Affected File:**
https://github.com/openssl/openssl/blob/ab8a5bc1ea2afa6afe648fe3f1681ad6c5dec1be/ssl/ech/ech_internal.c#L2150-L2155

**Affected Code:**

```
[...]
int ossl_ech_intbuf_add(SSL_CONNECTION *s, const unsigned char *buf,
                        size_t blen, int hash_existing)
{
    [...]
    if (s == NULL || buf == NULL || blen == 0)
        goto err;
    [...]
    } else {
        /* just add new octets */
        if ((t1 = OPENSSL_realloc(s->ext.ech.transbuf,
                                  s->ext.ech.transbuf_len + blen)) == NULL)
            goto err;
        s->ext.ech.transbuf = t1;
        memcpy(s->ext.ech.transbuf + s->ext.ech.transbuf_len, buf, blen);
        s->ext.ech.transbuf_len += blen;
    }
    [...]
}
[...]
```

It is recommended to validate size_t additions before performing memory allocation in *ossl_ech_intbuf_add* as a defensive hardening measure. The implementation should ensure that *s->ext.ech.transbuf_len + blen* cannot overflow and should enforce reasonable upper bounds on transcript buffer growth to prevent out-of-bounds writes.

# WP4: Lightweight Threat Model

## Introduction

This section provides a high-level threat model for ECH-enabled clients, servers, and deployments. Many recommendations target integrators and operators (for example, browser and web-server implementations) and may not correspond to direct changes in the OpenSSL library itself. For additional context, please see the DEfO *"ECH Weakness Analysis"* report (D9.1)[18]. OpenSSL ECH, the Encrypted ClientHello, developed by the DEfO project for OpenSSL, is a fundamental, privacy-enhancing extension for the Transport Layer Security (TLS) protocol, implemented to address remaining privacy leaks in the TLS handshake.

By encrypting handshake metadata, including the origin Server Name Indication, it reduces the ability of network observers, such as Internet Service Providers (ISPs), public Wi-Fi operators, or censorship devices, to determine the origin hostname from the TLS handshake. This improvement in handshake metadata confidentiality strengthens user privacy, especially in environments where pervasive monitoring or censorship is a concern.

Technically, ECH operates by splitting the traditional ClientHello message into an unencrypted outer component and a cryptographically protected inner component, which contains the true destination. This process relies on a new public key exchange mechanism requiring correct server-side provisioning. Since OpenSSL is a widely used foundational cryptographic library for countless web services, any flaw in its ECH handling, such as in key management, configuration parsing, or cryptographic boundaries, has the potential for widespread security and privacy failures.

Because ECH fundamentally alters the visibility and trust model during connection setup, it is an attractive target for various attackers. Exploitation could lead to metadata disclosure, denial-of-service, or a circumvention of ECH privacy guarantees. Moreover, ECH impacts enterprise security solutions that rely on cleartext handshake data for threat inspection. Consequently, a detailed threat model is essential to examine interactions between the ECH protocol, the OpenSSL library, and external dependencies and to identify and mitigate attack vectors.

The threat model analysis in this document identifies security threats and risks affecting various groups of users. This approach was selected because the project is mature and prior reviews have already produced in-depth protocol-level analysis and identified vulnerabilities. A more holistic approach was therefore considered more valuable.

[18] https://defo.ie/ech-weakness-analysis-D9.1.pdf

This document, together with accompanying attack scenarios, establishes a baseline that encourages all team members to adopt a threat-led mindset beyond the implementation, focusing on security from the outset to address risks before they evolve into exploitable vulnerabilities. A lightweight STRIDE-based approach[19] was applied using documentation, source code, existing threat models, research into underlying technologies, and client input to assess the target.

### Relevant assets and threat actors

The following key assets were identified as significant for security:

- Project source code, including the main OpenSSL ECH implementation
- OpenSSL ECH-enabled application forks (e.g., *Nginx*, *HAProxy*)
- GitHub organization and CI/CD pipelines
- ECH private key material
- *ClientHelloInner* and *ClientHelloOuter*
- True SNI inside encrypted *ClientHelloInner*
- Advertised ECHConfig via DNS records
- Client, client-facing server, and backend server assets

The following threat actors are considered relevant for the analysis:

- On-path attacker (Man-in-the-Middle, e.g. LAN attacker, nation-state threat actor, censor, ISP)
- Passive network observer (as above, but not performing any traffic changes)
- Malicious insider or supply chain attacker
- Advanced persistent threat (e.g. hacking group, nation-state threat actor)
- External attacker (any Internet-based attacker, automated scripts, or bots targeting servers supporting ECH)

### Attack surface

In threat modeling, the attack surface includes all potential entry points that an attacker may exploit to compromise a system, access or manipulate sensitive data, or disrupt application availability. Identifying the attack surface helps locate vulnerabilities and implement defenses to reduce risk. By analyzing threats and attack scenarios, organizations gain insight into potential techniques that could undermine system security.

[19] https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats#stride-model

### Threat 01: Implementation Risks

OpenSSL Encrypted *ClientHello* (ECH) introduces multiple structural changes to the TLS protocol. Complex parsing logic for nested structures, Inner versus Outer *ClientHello*, must be implemented by library developers in memory-unsafe languages. Simultaneously, TLS 1.3 security guarantees must not be weakened, and no critical low-level or logical errors must be introduced. Even subtle bugs can potentially expose the entire ecosystem to critical vulnerabilities.

**Attack Scenarios**

- **Parsing Logic Errors:** Malformed *ECHConfig* structures or crafted *ClientHello* messages can trigger buffer overflows or logic errors. Specifically, *ECHConfig* parsing might be a valuable target for attackers attempting to execute remote code on clients visiting a malicious site.
- **Timing Side-Channels:** Observers may measure differences in processing time between accepted (decrypted) and rejected (plaintext) ECH handshakes to deduce information about the encrypted *ClientHelloInner* or the validity of the key, potentially allowing the collection of valuable metadata. The threat is explicitly described in the draft specification[20] as a requirement for developers to address.
- **Key Management Store API Potential Issues**: By implementing an API to manage configurations and keys (*OSSL_ECHSTORE*), developers must ensure that all data originating from potentially untrusted locations cannot be used to corrupt memory or trigger parsing flaws.
- **Inner vs Outer inconsistencies**: Parameter inconsistencies between *Outer* and *Inner ClientHello* messages can lead to unforeseen issues during *ClientHello* reconstruction, potentially resulting in weakened TLS properties or downgrade attacks.
- **Resource Consumption (DoS):** Issues allowing the bypass of implemented limits on, for example, decryption size or decompression can lead to resource exhaustion on the server, resulting in a denial-of-service condition.

**Recommendations**

- **Linear-Time Scanning:** Implementations must ensure linear-time processing of outer extensions to prevent packet amplification attacks, as specified in Appendix A (Linear-time Outer Extension Processing) of the TLS ECH draft, which must be rigorously tested using edge cases and fuzz testing.
- **Fuzz Testing:** Full, rigorous fuzz testing specifically targeting *ECHConfig* parsing, *ClientHelloInner* reconstruction logic, and *OSSL_ECHOSTORE* is recommended to discover subtle but potentially dangerous bugs.

[20] https://datatracker.ietf.org/doc/draft-ietf-tls-esni/25/...10.6.%20%20Inner

- **Constant-Time Operations:** Constant-time implementations must be rigorously verified to limit side-channel attacks. Robust test cases and security analysis focusing on timing side-channel attacks should be performed to ensure the privacy properties of the ECH protocol.

### Threat 02: Network Adversaries and Downgrade Attacks

Active network adversaries can aim to strip away privacy protections targeting end users browsing the internet, the primary beneficiaries of the OpenSSL ECH protocol. Because ECH relies on a fail-open design and the opportunistic nature of implementations by, for example, browser vendors, users should be aware that attackers currently have multiple techniques that can potentially strip privacy enhancements offered by OpenSSL ECH. Therefore, depending on the relevant threat model and adversary type, users should consciously decide whether the protocol is sufficient to protect their privacy.

**Attack Scenarios**

- **Bitflipping and Malleability:** An active attacker may bitflip the encrypted payload, potentially forcing a decryption or transcript-integrity failure and causing the connection to fall back to the *Outer ClientHello* or to fail.
- **Protocol Downgrade:** Attackers may drop DNS responses containing ECH keys or modify ECHConfig content, forcing the client into a standard TLS handshake with visible metadata. Such techniques are, for instance, described by Cloudflare in the context of enterprise networks[21].
- **Privacy Confusion (Tor vs. ECH):** Users may mistakenly perceive ECH as a strong anonymity tool comparable to Tor. Using ECH to access forbidden resources in highly adversarial environments can lead to severe consequences due to weaker protection mechanisms than those provided by Tor. Users must clearly understand the limitations of the protocol and its current adoption level[22].

**Recommendations**

- It is advised that documentation clearly states that ECH protects the target domain in typical network environments, but may not be effective in highly censored regions where broad network control is exercised by the threat actor.
- Given the current adoption level, it is suggested that clients, such as browsers, provide an option to enforce ECH and present a warning when a privacy-enhanced connection cannot be established. Furthermore, mechanisms similar to HSTS for ECH could be investigated to actively detect attempts at ECH tampering or stripping.

[21] https://developers.cloudflare.com/ssl/edge-certificates/ech/#enterprise-network-applicability
[22] https://blog.trnck.dev/ciphertrails-checkin/

### Threat 03: Infrastructure and External Dependency Attacks

Infrastructure operators manage the external systems upon which ECH relies, primarily DNS for key distribution, but also services offering client-facing servers in split-mode deployments where the OpenSSL ECH protocol plays a crucial role. Because DNS is often unauthenticated, it represents a significant external attack vector. Due to the complex nature of the connection reconstruction process responsible for forwarding connections to the backend server, edge cases can be encountered, introducing unforeseen interoperability vulnerabilities.

**Attack Scenarios**

- **DNS Cache Poisoning:** Attackers may inject fake configuration data via DNS (*HTTPS RR/SVCB*) containing a public key they control. If successful, the *Inner ClientHello* may be encrypted to an attacker-controlled key, leading to *ClientHelloInner* disclosure
- **Broad-scale DNS Stripping or Tampering:** Highly adversarial environments can actively strip ECH or modify DNS responses at country-level scale to disable privacy enhancements, even at the ISP level, and block DoH or DoT resolvers to negate ECH benefits.
- **Replay Attacks:** Attackers may attempt to replay old DNS records with expired Time-To-Live (TTL) values, forcing the use of obsolete keys that may have been compromised due to incorrectly configured key rotation.
- **Insecure OpenSSL ECH Key Handling**: Disclosed keys, incorrectly rotated keys, or keys generated using deprecated, policy-disallowed, or insufficient-strength algorithms used by the ECH protocol allow an attacker to decrypt the *ClientHelloInner*, breaking the intended privacy properties of the protocol.

**Recommendations**

- **DNS Security:** Validation using DNSSEC for relevant resource records to ensure the authenticity of ECH configuration data is recommended.
- **Key Rotation:** Automated, short-lived key rotation, as recommended in the draft specification, and strictly enforced TTL limits involving retry configurations must be correctly configured.
- **Malicious DNS Configuration Monitoring**: Research focused on DNS *ECHConfig* monitoring to detect configuration tampering should be conducted. Over time, users should be able to determine whether they are operating in networks that actively attempt to strip privacy enhancements. Browser vendors could implement warning messages similar to insecure connection indicators, but related to ECH and privacy.

- **Security of Key Material**: Secure handling of OpenSSL ECH key material, including protected storage mechanisms, is required to prevent data leakage that would undermine the purpose of the protocol.

### Threat 04: Operational Risks for Website Owners

Website owners face increased operational complexity. The computational cost of trial decryption and handling nested handshakes introduces new avenues for denial-of-service (DoS) attacks and misconfiguration. Introducing or enabling OpenSSL ECH for services may not only lead to privacy enhancements, but also to operational issues that are difficult to debug and reproduce.

**Attack Scenarios**

- **Resource Exhaustion (DoS):** Attackers may send large volumes of valid or invalid ECH messages to exhaust server CPU resources through excessive decryption operations.
- **Ignored Configuration Identifiers:** If a server is configured to ignore specific configuration identifiers to obscure anonymity set size, it must perform trial decryption on every incoming request, significantly exacerbating DoS risks[23].
- **Ossification:** Misconfigured servers or aggressive middleboxes may cause valid traffic to be dropped, leading to availability issues that force administrators to disable ECH entirely, thereby downgrading security for all users.
- **Split-mode Misconfigurations:** Discrepancies between servers can lead to difficult-to-reproduce and resolve bugs affecting availability. Administrators may be tempted to resolve such issues, for example, by disabling TLS 1.3 and, consequently, ECH, due to an inability to identify the root cause[24][25].

**Recommendations**

- Implementation of strict rate limiting for decryption operations, particularly for messages that fail decryption, as suggested in the OpenSSL ECH draft.
- Monitoring of error rates associated with ECH decryption failures and alerting to detect active attacks or misconfigurations.
- Development of tools that allow easy identification of parameter mismatches, particularly in split-mode deployments, to ensure that configuration issues can be efficiently tracked and resolved.

[23] https://datatracker.ietf.org/doc/draft-ietf-tls-esni/25/
[24] https://community.cloudflare.com/t/getting-ech-fallback-certificate-error/565167
[25] https://blog.trnck.dev/cloudflare-with-ech-again/

### Threat 05: Enterprise Network Security Blind Spots

Network defenders and security vendors rely on deep packet inspection (DPI) to enforce policy. ECH blinds these tools by encrypting the Server Name Indication (SNI), creating a conflict between privacy and organizational security. The same enterprises rely on such tools to enforce network policies and detect threats within an organization. In these environments, it may be more beneficial to disable OpenSSL ECH rather than allow it within the network, despite the undeniable privacy enhancements it provides.

**Attack Scenarios**

- **Malware C2 Concealment:** Malware may use ECH to conceal communications with command-and-control (C2) servers, bypassing traditional SNI-based blocklists and inspection tools.
- **Policy Bypass:** Employees may bypass corporate filtering rules to access blocked domains using ECH, rendering organizational security policies ineffective.
- **Traffic Differentiation:** Intermediaries may struggle to differentiate between GREASE (dummy) traffic and real ECH traffic. If unknown extensions are blocked to maintain visibility, valid privacy-enhanced traffic may be inadvertently blocked, contributing to network ossification.
- **Business Fallback:** If security tools become ineffective, enterprises may respond by blocking ECH entirely or canceling contracts with vendors unable to provide visibility, leading to financial impact.
- **Security Vendor Inefficiency**: Privacy enhancements offered by OpenSSL ECH can reduce detection rates for various security products. Without proper research evaluating the impact on this sector, end users and organizations may continue paying for expensive tools, thereby granting a false sense of security.

**Recommendations**

- Documentation and use of canary domains or specific DNS signals to indicate enterprise policy regarding ECH support, allowing managed networks to disable it gracefully if required, as documented by Cloudflare[26].
- Implementation of more robust detection methodologies beyond simple domain matching, for example by using behavioral analysis and heuristic anomaly detection.
- Research evaluating security tool efficiency following widespread OpenSSL ECH adoption, educating users and providing transparency to prevent a false sense of security.

[26] https://developers.cloudflare.com/ssl/edge-certificates/ech/#enterprise-network-applicability

### Threat 06: Supply Chain and Insider Threats

The integrity of the software supply chain is critical for all projects. Malicious code or weak defaults introduced during development or deployment can compromise the entire ecosystem. Widely used libraries, especially those modifying critical protocols such as TLS, are high-value targets for attackers, including nation-state threat actors and advanced persistent threats. Even subtle changes introduced by a malicious, well-positioned attacker can potentially pass undetected through the development process and weaken the security guarantees of an already well-established protocol.

**Attack Scenarios**

- **Malicious Injection:** Malicious logic could be inserted into the OpenSSL or DEfO code repositories. Without signed commits and rigorous review, authorship cannot be reliably proven, allowing backdoors to persist and be upstreamed[27][28]. Such code could be introduced into the OpenSSL ECH codebase itself as well as into development tools used during early stages, targeting early adopters of the protocol.
- **Artifact Injection:** Attacks targeting still-available but outdated experimental packages built by DEfO and hosted on the main domain could be performed with the goal of infecting artifacts, allowing attackers to pivot to early adopters of the OpenSSL ECH[29] protocol.

**Recommendations**

- Differential fuzzing and reasonable regression testing practices are advised. Security-related test cases should be explicitly designed to ensure the non-reintroduction of previously resolved vulnerabilities, such as side-channel attacks, that have historically affected SSL and TLS implementations. Known and mitigated attacks and techniques described in the draft specification should also be verified using dedicated test cases.
- Outdated and potentially vulnerable OpenSSL ECH-enabled packages should be removed from the DEfO.ie website to prevent their use by administrators and developers.

[27] https://tukaani.org/xz-backdoor/
[28] https://www.akamai.com/blog/security-research/critical-linux-backdoor-xz-utils-discovered-what-to-know
[29] https://defo.ie/debian/

# Conclusion

Despite the number of findings encountered in this exercise, the DEfO solution defended itself well against a broad range of attack vectors. The project will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

The following positive impressions were observed during this assignment:

- The source code is exceptionally well organized and supported by comprehensive documentation, demonstrating a strong commitment to high code quality.
- Project maintainers demonstrated a high level of responsibility and responsiveness, which significantly streamlined the audit process.
- Extensive input validation is implemented through length checks, type validation, and structural verification of ECH and *ClientHello*-related fields, reducing the risk of buffer overflows, malformed input handling, and state machine desynchronization.
- Cryptographic operations, including HPKE encryption, decryption, and transcript hash processing, consistently rely on mature OpenSSL primitives, avoiding ad-hoc cryptography and reducing the risk of cryptographic misuse and timing attacks.
- Memory handling practices are generally sound, with appropriate use of *OPENSSL_zalloc* to zero sensitive data and to ensure correct cleanup on error paths, reducing the risk of memory leaks, use-after-free issues, and exposure of sensitive material.
- HPKE primitives are integrated into the existing TLS state machine, and both shared and split deployment topologies are supported in accordance with the documented design.
- Parsing logic relies on the *OpenSSL PACKET API*, contributing to safer and more maintainable protocol handling.
- The transition between *Outer* and *Inner ClientHello* messages was observed to function correctly in standard operational scenarios.
- The threat models and attack analyses described in the relevant IETF draft are comprehensive and appear aligned with the implementation. Prior formal verification efforts appear to have contributed meaningfully to mitigations.
- Overall, the project demonstrates a high level of maturity, making use of modular, well-tested components and reflecting a strong understanding of relevant security threats and risks.

The security of the DEfO project will improve with a focus on the following areas:

- **Protocol Compliance and Cryptographic Enforcement:** Strict enforcement of TLS and ECH protocol requirements is necessary to preserve core security guarantees. The server logic must ensure that ECH accept confirmation overwrites are performed only when the final negotiated protocol version is TLS 1.3. The *Inner ClientHello* must not offer to negotiate TLS 1.2 or lower (DEF-02-005, DEF-02-006). Cryptographic context initialization must propagate the library context and property query string to prevent silent bypass of FIPS restrictions (DEF-02-003). HPKE algorithm selection should be subject to the configured security policy (DEF-02-007).
- **Resource Management and Denial-of-Service Protections:** Resource usage must be strictly bounded to prevent denial-of-service conditions. Trial decryption operations require explicit CPU and complexity limits (DEF-02-002), and parsing logic must eliminate out-of-bounds read conditions by using explicit buffer lengths in Base64 decoding (DEF-02-001). Allocation size calculations should be hardened by validating *size_t* arithmetic and enforcing reasonable upper bounds for retry configuration concatenation and transcript buffer growth (DEF-02-010, DEF-02-011).
- **Input Validation, Parsing, and Failure Semantics:** Input handling should be hardened to enforce fail-closed behavior and clear failure classification. The ECH extension parser must return distinct error codes for malformed input versus valid inputs without relevant extensions, remove test-specific workarounds from production, and reject truncated messages outside test environments (DEF-02-008). Failure classification should also distinguish between GREASE values and configuration-matched ECH decryption failures (DEF-02-004).
- **State Management and Retry Consistency:** Connection state handling during retries must be made consistent to prevent unnecessary connection failures. The client retry logic should keep certificate hostname verification aligned with the public name during retries (DEF-02-009).
- **Testing Strategy and Automation:** Security testing coverage should be expanded and systematically enforced as a hardening measure. Fuzz testing, particularly for parsers and edge cases, should be extended beyond existing efforts such as *echdnsfuzz* and integrated into CI/CD pipelines alongside automated and AI-assisted testing to detect regressions and newly introduced vulnerabilities early.
- **Documentation and Threat Model Alignment:** Project documentation should clearly align expectations with the intended threat model as an informational improvement. It should explicitly state that ECH addresses origin hostname exposure in the TLS handshake against passive observers such as ISPs or local network observers, and that it is not a substitute for anonymity systems in highly adversarial environments. Limitations imposed by opportunistic, fail-open

deployment models and current browser and enterprise behavior should be clearly documented.

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will strengthen the security posture of the application and reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, ideally including another whitebox source code audit, is highly recommended to ensure adequate security coverage of the platform.

Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing DEfO resources.

It is suggested to test the solution regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the application highly resilient against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank Kerry Hartnett, Stephen and the rest of the DEfO team, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the *Open Source Technology Improvement Fund (OSTIF)* for facilitating and managing this project, and the *Sovereign Tech Fund* for funding it.

# License and Legal Notice

This report is licensed under the *Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)*[30] license.
You are free to:

- **Share** – copy and redistribute the material in any medium or format
- **Adapt** – remix, transform, and build upon the material for any purpose, even commercially

Under the following terms:

- **Attribution** – You must give appropriate credit to 7ASecurity, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests 7ASecurity endorses you or your use.

- **ShareAlike** – If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

Exceptions and Restrictions:

- **Trademarks and Logos**: The 7ASecurity name, logo, and visual identity elements (such as custom fonts or design marks) are not licensed under CC BY-SA 4.0 and may not be used without explicit written permission.

- **Third-party Content:** Any third-party content (e.g., open source project logos, screenshots, excerpts) included in this report remains under its respective copyright and licensing terms.

- **No Endorsement:** Use of this report does not imply endorsement by 7ASecurity of any derivative works, use cases, or conclusions drawn from the material.

**Disclaimer:** This report is provided for informational purposes only and reflects the state of the target project at the time of testing. No warranties are provided. Use at your own risk.

[30] https://creativecommons.org/licenses/by-sa/4.0/