OpenSSL

Security Assessment

April 18, 2024

Prepared for:
Anton Arapov
Matt Caswell
OpenSSL
Organized by the Open Source Technology Improvement Fund, Inc.

Prepared by: **Max Ammann, Fredrik Dahlgren, Spencer Michaels, and Jim Miller**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering directors were associated with this project:

David Pokora, Engineering Director, Application Security
david.pokora@trailofbits.com

Jim Miller, Engineering Director, Cryptography
james.miller@trailofbits.com

The following engineers were associated with this project:

Max Ammann, Consultant
maximilian.ammann@trailofbits.com

Fredrik Dahlgren, Consultant
fredrik.dahlgren@trailofbits.com

Spencer Michaels, Consultant
spencer.michaels@trailofbits.com

Jim Miller, Consultant
jim.miller@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
August 21, 2023	Pre-project kickoff call
September 7, 2023	Status update meeting #1
September 12, 2023	Status update meeting #2
September 19, 2023	Status update meeting #3
September 27, 2023	Delivery of report draft
September 27, 2023	Report readout meeting
April 18, 2024	Delivery of comprehensive report with fix review appendix

Executive Summary

Engagement Overview

OSTIF engaged Trail of Bits to review the security of the OpenSSL cryptographic library. The focus of the engagement was the new provider architecture and eight new cryptographic primitives, all of which were introduced in version 3 of OpenSSL.

A team of four consultants conducted the review from August 28 to September 22, 2023, for a total of nine engineer-weeks of effort. Our testing efforts focused on the implementation of the new provider architecture, including the implementations of library contexts, encoders and decoders, and the provider-based implementation of the high-level EVP API. We also reviewed a number of new cryptographic primitives included in version 3 of the library. With full access to the source code, documentation, Coverity reports, Coveralls test coverage data, and fuzzing coverage data from OSS-Fuzz, we performed static and dynamic testing of the OpenSSL codebase, using automated and manual processes.

Observations and Impact

Overall, we found the OpenSSL library to be defensively implemented and well tested. The project has an extensive test suite with known test vectors for implemented cryptographic primitives. Code coverage is tracked and improved when needed, and the project also regularly runs static analysis through Coverity and continuous fuzzing through OSS-Fuzz. However, during the engagement, we identified a number of development practices that could have security implications for future releases of the library.

We found that the C integer types are used inconsistently throughout the codebase. Signed types are often used to represent unsigned quantities, signed and unsigned integers are often mixed in arithmetic expressions, and larger types are passed to APIs that expect smaller types, which leads to implicit truncations. Although we did not identify any security-relevant issues due to integer truncation or implicit integer promotions or conversions during this engagement, we believe that this practice introduces unnecessary risks that should be avoided.

We also noted that internal APIs often lack source-level documentation. This makes it hard to understand the exact security properties the API is expected to satisfy. Adding source-level documentation and documenting the security properties expected and upheld by internal provider APIs would go a long way in making the codebase easier to review and maintain. Such documentation would also be a useful resource for any developers looking to implement third-party providers for OpenSSL.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that OpenSSL take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.

- **Standardize the use of C integer types.** Using signed types to represent unsigned values, mixing signed and unsigned types in arithmetic expressions, and passing larger types to APIs that expect smaller types all represent latent security risks to downstream consumers of the library. These anti-patterns should generally be avoided, as they are known to be the cause of truncation issues and overflows and could lead to memory-safety issues or undefined behavior. We recommend that the OpenSSL team develop a secure coding standard for integer types. This standard could initially apply to only new or refactored code to allow the team to safely transition the entire codebase over time. For our related recommendations, refer to appendix G.

- **Add source-level documentation for internal APIs.** Currently, internal APIs are mostly undocumented. This leaves maintainers and code reviewers guessing as to which security properties are expected to hold when the APIs are called. Better source-level documentation would make the codebase easier to review and maintain. It would also make it generally easier to ensure that source-level documentation is up to date, as it would already be part of the codebase.

- **Introduce a deprecation schedule for weak algorithms.** We found that the default provider contains a number of algorithms based on both the two-key and three-key variants of Triple-DES (TOB-OSSL-3). This algorithm is considered broken by the cryptographic community, and there are known and practical attacks on Triple-DES-based ciphers that enable plaintext recovery. We understand that the team has to balance development velocity against backward compatibility and cannot immediately remove algorithms that are found to be insecure. However, we think it would be a good idea to introduce a deprecation schedule that outlines how and when weak and legacy algorithms are moved from the default provider to the legacy provider. This, along with a regular release schedule, would make it easier for downstream consumers to plan how and when to move away from legacy algorithms.

- **Develop new fuzzers to increase fuzzing coverage.** The OpenSSL project already runs a number of fuzzers on its codebase to detect issues related to memory corruption and undefined behavior. Implemented fuzzers focus mainly on APIs that receive untrusted user input. This effort could be extended to include APIs that may

be prone to misuse. An example of this is the provider fuzzer developed as part of this engagement, described in appendix D.

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	4
Low	6
Informational	13
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Configuration	2
Cryptography	2
Data Exposure	1
Data Validation	7
Denial of Service	5
Error Reporting	1
Undefined Behavior	5

Project Goals

The engagement was scoped to provide a security assessment of the OpenSSL cryptographic library. Specifically, we sought to answer the following non-exhaustive list of questions:

- Do library contexts and included providers manage memory correctly? Are allocated pointers checked for null and freed correctly before going out of scope?
- Are reference counts for shared resources incremented and decremented correctly?
- Are resource locks managed correctly by the library context implementation?
- Are the default library context and the default provider resolved correctly?
- Is the provider scaffolding for each supported algorithm implemented correctly?
- Does the EVP API use the new provider architecture correctly?
- Are provider implementations written defensively to prevent misuse?
- Are legacy engine fallbacks in the EVP API implemented correctly?
- Are OpenSSL configuration files parsed correctly?
- Is the configuration file format resistant to misuse?
- Are cryptographic primitives implemented correctly, according to their specifications?
- Are cryptographic primitives implemented using constant-time code?
- Is sensitive data like key material zeroized when it goes out of scope?

Project Targets

The engagement involved a review and testing of the following target.

OpenSSL

Repository	https://github.com/openssl/openssl
Version	3.1.2 (commit `17a2c5111864d8e016c5f2d29c40a3746b559e9d`)
Type	C
Platforms	Linux, macOS, Windows

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Providers:** We manually reviewed the provider architecture and utility code, focusing on memory management, reference counting, and resource locks. We manually reviewed the default, FIPS, legacy, base, and null providers included with OpenSSL. Here, we focused on correctness, and we also ensured that the default and FIPS providers do not support any legacy algorithms. Finally, we reviewed the high-level provider implementations under the `providers/implementations` directory. Here, we focused on overall correctness, memory management, and misuse resistance of the APIs. We also gave a best-effort review of the implemented cryptographic primitives. However, since the focus of the review was the new provider architecture, we did not perform an in-depth cryptographic review of each primitive as part of this engagement.

  The following provider implementations were reviewed as part of the engagement:

  - The RSA and SM2 asymmetric ciphers
  - The AES-CBC-HMAC-SHA256, AES-CCM, AES-GCM, and ChaCha20-Poly1305 ciphers
  - The Blake2, MD2, MD4, MD5, RipeMD, SHA2, SHA3, SM3, and Whirlpool hash functions
  - RSA KEM
  - The HKDF, KBKDF, PBKDF1, PBKDF2, Scrypt, SSH KDF, SSKDF, and X9.42 KDFs
  - The DH, DSA, EC, ECX, and RSA key management functions
  - The Blake2, GMAC, KMAC, HMAC, Poly1305, and Siphash MACs
  - The ECDSA, EDDSA, RSA, and SM2 signature providers
  - All pseudo-random number generators (PRNGs), and random number generator seeding for the ARM64, x86, and Unix platforms
  - All encoders and decoders (refer to the bullet point on coverage of encoders and decoders below)

- **LibCTX:** We performed a manual review of the library context implementation, focusing on overall correctness, memory management, and potential concurrency-related issues.
- **Encoders and decoders:** We manually reviewed the implementation of encoders and decoders. This review included the low-level API that runs data or objects through a chain of coders, and the high-level API for public and private keys (`OSSL_ENCODER_CTX_new_for_pkey`).
- **EVP:** We manually reviewed the parts of the high-level EVP API that interacts with the new provider architecture and legacy engine APIs. As the EVP API is large and it is unrealistic to manually audit the whole API surface, we focused on the interaction points between the EVP code and the new provider code to check that it was correctly implemented. For example, we audited the instantiation of EVP objects that use the provider API and EVP APIs that use the new LibCTX code to fetch provider implementations.
- **Cryptographic primitives:** The following new cryptographic primitives are included in version 3.0 of the OpenSSL library and were reviewed as part of the engagement:
    - SIV and CTS cipher modes
    - Blake2
    - Scrypt
    - SSH KDF
    - SSKDF
    - KBKDF
    - Siphash

  We reviewed each primitive against the relevant specification or RFCs, focusing on correctness and on identifying potential issues related to input parameter validation, timing side channels, and the zeroization of sensitive data.
- **HTTP client:** We manually reviewed the new HTTP client implementation, focusing on functionality where we typically see issues, like URL parsing, HTTP header parsing, and HTTP redirects.
- **Fuzz testing:** We developed several fuzzers for internal OpenSSL APIs, provider implementations, and the configuration file parser. For more detail on the fuzzers developed during the engagement, refer to appendix D.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- **Providers:** The main focus of the review was the new provider architecture. For this reason, we performed only a best-effort review of the cryptographic code under the `providers/implementations` directory. Additionally, we did not manage to review all of the primitives implemented. In particular, the following provider implementations were not reviewed during this engagement:
    - AES variants other than the ones listed above, as well as the Aria, Blowfish, Camellia, CAST5, DES, IDEA, RC2, RC4, RC5, Seed, SM4, and Triple-DES ciphers
    - The ECX and KDF key exchanges
    - The TLS1 PRF and the KRB5 and PKCS12 KDFs
    - The ECX, KDF legacy, and MAC legacy key management functions
    - CMAC
    - PRNG seeding using RDTSC, and OpenVMS-, VXWorks-, and Windows-specific seeding
- **EVP:** We did not have time to review the entire EVP API implementation as part of this review. Instead, we focused on how the EVP API interacts with the new provider and legacy engine architectures.
- **Random number generation:** We did not have time to perform an end-to-end review of the random number generator seeding and generation during this engagement.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
Clang	An open-source LLVM front end for C and C++	Appendix C.1
CodeQL	A code analysis engine developed by GitHub to automate security checks	Appendix C.2
Cppcheck	An open-source static analysis tool focusing on detecting undefined behavior and dangerous coding constructs in C and C++ codebases	Appendix C.3
Semgrep	An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time	Appendix C.4
LibFuzzer	An open-source library for in-process, coverage-guided fuzz testing	Appendix D

Areas of Focus

Our automated testing and verification work focused on the following:

- Code quality issues and potentially fragile code patterns
- Overflow and truncation issues due to implicit integer conversions
- General undefined behavior

Test Results

The results of this focused testing are detailed below.

OpenSSL: We built the OpenSSL library using Clang with warnings for integer truncation and implicit sign conversions enabled. We also ran the static analysis tools CodeQL, Semgrep, and Cppcheck on the codebase and triaged the results. Here, we focused on issues related to the new provider architecture. Finally, we fuzzed the configuration, property list parsers, and provider implementations using LibFuzzer.

Property	Tool	Result
The project adheres to best practices by avoiding implicit conversions that truncate the input.	Clang	Appendix C.1
The project avoids common issues and fragile coding constructs often found in C codebases.	CodeQL Semgrep	**Passed**
The codebase does not contain compiler-specific or undefined behavior.	Cppcheck	TOB-OSSL-13
Input parsers are robust against malformed or malicious inputs.	LibFuzzer	TOB-OSSL-4 TOB-OSSL-5 TOB-OSSL-6

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The library often mixes signed and unsigned integer types in arithmetic expressions. Also, parts of the codebase use signed integer types to represent unsigned quantities like buffer sizes. Due to these practices, the codebase contains numerous cases of implicit integer promotions and conversions, which could cause hard-to-diagnose signed-overflow or truncation issues. That being said, we did not identify any security issues due to implicit truncations as part of this engagement.	**Moderate**
Auditing	The library does not implement auditing or logging.	**Not Applicable**
Authentication / Access Controls	The library does not implement access controls.	**Not Applicable**
Complexity Management	The new provider architecture is well engineered and provides an easy way to load additional cryptographic modules. The design also provides a clear and logical separation between library contexts, providers, and cryptographic primitives. The new implementation of encoders and decoders is well designed but is currently targeted at a narrow use case, which means that the public APIs are less ergonomic.	**Satisfactory**
Configuration	The library can be configured using a configuration file, which allows the end user to load and activate different providers. We found the sections of the configuration file format related to providers to be easy to misuse (TOB-OSSL-2) and the corresponding parser to be vulnerable to malicious inputs (TOB-OSSL-4, TOB-OSSL-5, TOB-OSSL-6). However, since the configuration file is	**Moderate**

	never attacker controlled, this is typically not a serious issue.	
Cryptography and Key Management	The reviewed cryptographic primitive implementations all match the relevant specifications and RFCs, and each implementation comes with known test vectors, which also gives some confidence that the implementation is correct. We did not identify any side-channel leakages in any of the implementations. Sensitive data is generally scrubbed from memory as it goes out of scope. However, we found two issues in which key material in memory is not zeroized correctly by the corresponding implementation (TOB-OSSL-21 and TOB-OSSL-22).	**Satisfactory**
Data Handling	We found that parameters for cryptographic algorithms are validated to ensure that the code follows the relevant specifications, and that the implementation generally protects against potential memory-safety issues like out-of-bounds reads and writes. However, we did identify one issue that could lead to an out-of-bounds read in PBKDF1 (TOB-OSSL-23). The library performs a minimal amount of pointer validation for user-provided inputs. This is typically enough to be safe against adversarial inputs, but failing to check for null pointers often makes the high-level APIs less resistant to misuse.	**Moderate**
Documentation	The high-level APIs and library design are well documented through man pages and internal documentation. However, source-level documentation is very scant, and it is often difficult to know which security invariants functions expect to hold or uphold. This makes the codebase difficult to review for security and correctness.	**Moderate**
Low-Level Manipulation	The low-level, platform-specific cryptographic implementations were not reviewed as part of this engagement.	**Not Considered**
Maintenance	The project's maintenance practices were not reviewed as part of this engagement.	**Not Considered**
Memory Safety and Error	Functions typically signal errors by returning `0` (or null for functions returning pointers) to the caller. We found that	**Satisfactory**

Handling	return values are checked consistently throughout the codebase and that returned pointers are checked to ensure they are not null. We did identify a number of possible segmentation faults due to null pointer dereferences (TOB-OSSL-6, TOB-OSSL-8, TOB-OSSL-15, TOB-OSSL-16, TOB-OSSL-19), one instance of a possible double free (TOB-OSSL-12), and one instance of a possible use after free (TOB-OSSL-14) during the engagement. These all resulted from invoking the high-level APIs in unexpected ways.	
Testing and Verification	The library comes with an extensive test suite covering both low-level cryptographic primitives and high-level APIs. Cryptographic primitives are tested against known test vectors, tests cover both the happy path and different failure cases, and coverage is tracked continuously through Coveralls. In addition to this, OpenSSL relies on the test suite from the Python cryptography project for integration testing. OpenSSL uses Coverity for static analysis and regularly triages found issues. The project also runs continuous fuzzing campaigns as part of OSS-Fuzz. However, fuzz tests running on OSS-Fuzz cover only around 28% of the codebase, and parts of the new provider architecture are not covered by fuzz testing.	**Satisfactory**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Risk of signed integer overflows when parsing property queries	Undefined Behavior	**Informational**
2	The provider configuration format is prone to misuse	Configuration	**Low**
3	The default provider supports insecure algorithms	Configuration	**Informational**
4	Provider configuration section can cause a stack overflow	Denial of Service	**Informational**
5	Risk of heap buffer overflow during parsing of OIDs	Undefined Behavior	**Informational**
6	Risk of segmentation fault when loading property list in “stable” configuration section	Denial of Service	**Informational**
7	The ossl_prov_memdup function does not update dst_len if the call fails	Error Reporting	**Informational**
8	API misuse may lead to unexpected segmentation fault	Undefined Behavior	**Informational**
9	Insufficient validation in dh_gen_common_set_params	Data Validation	**Low**
10	HTTP client redirects to local host instead of remote one	Data Validation	**Informational**
11	OCSP requests might hang if the server responds with infinite headers	Denial of Service	**Medium**

12	Calling EVP_KDF_CTX_reset causes a double free when the context is freed	Undefined Behavior	**Low**
13	The aesni_cbc_hmac_sha256_cipher function depends on compiler-specific behavior	Data Validation	**Low**
14	Use after free when setting invalid properties on the Scrypt algorithm or if SHA-256 is missing	Undefined Behavior	**Low**
15	Setting OSSL_MAC_PARAM_DIGEST_NOINIT for HMAC causes segmentation fault	Denial of Service	**Informational**
16	Functions of EVP_CIPHER_CTX are missing null checks	Denial of Service	**Informational**
17	Assertion could be hit when fetching algorithms by name	Data Validation	**Informational**
18	Reinitialization of EVP_MAC for GMAC fails if parameters are not provided	Data Validation	**Low**
19	Creation of X.509 extensions can lead to undefined behavior	Data Validation	**Informational**
20	Missing null checks in OSSL_PARAM getters	Data Validation	**Informational**
21	The ossl_blake2b_final function fails to zeroize sensitive data	Cryptography	**Medium**
22	The kdf_pbkdf1_do_derive function fails to zeroize sensitive data	Cryptography	**Medium**
23	Out-of-bounds read in kdf_pbkdf1_do_derive	Data Exposure	**Medium**

Detailed Findings

1. Risk of signed integer overflows when parsing property queries

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-OSSL-1
Target: `crypto/property/property_parse.c`	

Description

The `parse_number`, `parse_hex`, and `parse_oct` functions are used to parse strings to an `OSSL_PROPERTY_LIST`; their arithmetic operations could result in signed integer overflows, which is undefined behavior.

```
static int parse_number(const char *t[], OSSL_PROPERTY_DEFINITION *res)
{
    const char *s = *t;
    int64_t v = 0;

    if (!ossl_isdigit(*s))
        return 0;
    do {
        v = v * 10 + (*s++ - '0');
    } while (ossl_isdigit(*s));

    // ...
}
```

Figure 1.1: Passing a string representing a large number to `parse_number` causes undefined behavior. (`crypto/property/property_parse.c`)

The following figures show example inputs to these functions that cause undefined behavior due to overflow.

```
char* input =
"f.a=0x00ffffffffffffffffffffffffffffffffffffffffffffffff0fffffffffffffffffffffffffff
fffffffffffff0fffffff"
// crypto/property/property_parse.c:124:11: runtime error: left shift of
6148914691236517205 by 4 places cannot be represented in type 'int64_t' (aka 'long
long')
```

Figure 1.2: An overflow that results from parsing a large hexadecimal number

```
char* input = "a.a=401846744073709551615"
```

```
// crypto/property/property_parse.c:103:15: runtime error: signed integer overflow:
4018467440737095516 * 10 cannot be represented in type 'long long'
```

Figure 1.3: An overflow that results from parsing a large decimal number

```
char* input = "a.a=0000000000200000000000000000000000000000"
// crypto/property/property_parse.c:149:16: runtime error: left shift of
2305843009213693952 by 3 places cannot be represented in type 'int64_t' (aka 'long
long')
```

Figure 1.4: An overflow that results from parsing a large octal number

The following code can be used to reproduce the bug. In order to log the same messages shown in the above examples, UndefinedBehaviorSanitizer (UBSan) must be enabled (`enable-ubsan` in OpenSSL).

```
OSSL_PROPERTY_LIST *list  = ossl_parse_property(NULL, input);
if (list) {
    ossl_property_free(list);
}
```

Figure 1.5: Code that reproduces the signed long integer overflows

This finding was discovered by the provider fuzzer described in appendix D.

Recommendations

Short term, add checks to prevent overflows to the arithmetic operations in `parse_number`, `parse_hex`, and `parse_oct`.

Long term, review the project's current fuzzing coverage to ensure that all input parsers have sufficient coverage.

2. The provider configuration format is prone to misuse

Severity: **Low**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-OSSL-2
Target: `crypto/provider_conf.c`	

Description

Users can load and activate providers using the OpenSSL library configuration file. The file format appears to be inspired by the Windows INI configuration file format. The documentation in the provider README file contains the following example, describing how to load and activate the default and legacy providers.

```
openssl_conf = openssl_init

[openssl_init]
providers = provider_sect

[provider_sect]
default = default_sect
legacy = legacy_sect

[default_sect]
activate = 1

[legacy_sect]
activate = 1
```

Figure 2.1: An example provider configuration section from the provider README file (`README-PROVIDERS.md`)

From the example and the overall file format, end users could easily infer that they could use the syntax `activate = 0` to ensure that a particular provider is *not* used. This would also be consistent with the INI file format, in which values such as `1`, `yes`, `true`, and `on` would typically be interpreted as true, and in which `0`, `no`, `false`, and `off` would be interpreted as false. However, by looking at the provider section parser function `provider_conf_load`, we see that the value assigned to the `activate` key is ignored by the parser.

```
for (i = 0; i < sk_CONF_VALUE_num(ecmds); i++) {
    CONF_VALUE *ecmd = sk_CONF_VALUE_value(ecmds, i);
    const char *confname = skip_dot(ecmd->name);
    const char *confvalue = ecmd->value;
```

```
    OSSL_TRACE2(CONF, "Provider command: %s = %s\n",
                confname, confvalue);

    /* First handle some special pseudo confs */

    /* Override provider name to use */
    if (strcmp(confname, "identity") == 0)
        name = confvalue;
    else if (strcmp(confname, "soft_load") == 0)
        soft = 1;
    /* Load a dynamic PROVIDER */
    else if (strcmp(confname, "module") == 0)
        path = confvalue;
    else if (strcmp(confname, "activate") == 0)
        activate = 1;
}

if (activate) {
    ok = provider_conf_activate(libctx, name, value, path, soft, cnf);
} else {
    // ...
}
```

Figure 2.2: The value assigned to `activate` is ignored by the `provider_conf_load` function. (`crypto/provider_conf.c`)

We note that this surprising behavior is described in the man page for the OpenSSL configuration file format, which says the following about the `activate` key:

> *If present, the module is activated. The value assigned to this name is not significant.*

However, users who are not aware of this behavior may end up activating insecure providers by mistake.

Exploit Scenario

An OpenSSL end user wants to ensure that an application is using only FIPS-compliant algorithms. To ensure that the legacy provider is not active, she includes the following section in her OpenSSL configuration file and thus enables the legacy provider by mistake instead of disabling it as intended.

```
[provider_sect]
# ...
legacy = legacy_sect

[legacy_sect]
activate = 0
```

Figure 2.3: An end user could enable an insecure provider by mistake by setting the value for the corresponding `activate` key to `0`.

Recommendations

Short term, have the `provider_conf_load` function return `0`, signaling a fatal error, if a user attempts to set the `activate` key to a value different from 1.

Long term, extend the parser to take the value assigned to the `activate` key into account, document the values accepted by the parser along with their interpretations, and have the parser activate the corresponding provider only on truthy values.

3. The default provider supports insecure algorithms

Severity: **Informational**	Difficulty: **Not Applicable**
Type: Configuration	Finding ID: TOB-OSSL-3
Target: `providers/defltprov.c`	

Description

The default provider includes multiple versions of Triple-DES (based on both the two-key and three-key variants of the algorithm). The DES block size is only 64 bits, and the cipher is vulnerable to (practical) birthday attacks against long-lived sessions.

```
#ifndef OPENSSL_NO_DES
    ALG(PROV_NAMES_DES_EDE3_ECB, ossl_tdes_ede3_ecb_functions),
    ALG(PROV_NAMES_DES_EDE3_CBC, ossl_tdes_ede3_cbc_functions),
    ALG(PROV_NAMES_DES_EDE3_OFB, ossl_tdes_ede3_ofb_functions),
    ALG(PROV_NAMES_DES_EDE3_CFB, ossl_tdes_ede3_cfb_functions),
    ALG(PROV_NAMES_DES_EDE3_CFB8, ossl_tdes_ede3_cfb8_functions),
    ALG(PROV_NAMES_DES_EDE3_CFB1, ossl_tdes_ede3_cfb1_functions),
    ALG(PROV_NAMES_DES3_WRAP, ossl_tdes_wrap_cbc_functions),
    ALG(PROV_NAMES_DES_EDE_ECB, ossl_tdes_ede2_ecb_functions),
    ALG(PROV_NAMES_DES_EDE_CBC, ossl_tdes_ede2_cbc_functions),
    ALG(PROV_NAMES_DES_EDE_OFB, ossl_tdes_ede2_ofb_functions),
    ALG(PROV_NAMES_DES_EDE_CFB, ossl_tdes_ede2_cfb_functions),
#endif /* OPENSSL_NO_DES */
```

Figure 3.1: The default provider supports a number of Triple-DES based algorithms. (`providers/defltprov.c`)

NIST SP 800-131A revision 2 disallows the use of the two-key variant of Triple-DES for encryption and has deprecated use of the three-key variant.

Algorithm	Status
Two-key TDEA Encryption	Disallowed
Two-key TDEA Decryption	Legacy use
Three-key TDEA Encryption	Deprecated through 2023 Disallowed after 2023
Three-key TDEA Decryption	Legacy use

Figure 3.2: Table 1 in NIST SP 800-131A revision 2 details the current status of two-key and three-key Triple-DES (TDEA).

Exploit Scenario

An application that supports cipher negotiation relies on OpenSSL for cryptographic operations. Because the default provider is loaded, the application supports legacy algorithms like the two-key variant of Triple-DES, making it vulnerable to birthday attacks like Sweet32.

Recommendations

Short term, publish a deprecation schedule for Triple-DES-based algorithms.

Long term, move all Triple-DES-based algorithms to the legacy provider in the next major release of OpenSSL.

4. Provider configuration section can cause a stack overflow

Severity: **Informational**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-OSSL-4
Target: `crypto/provider_conf.c`	

Description

Parsing a configuration containing a self-referencing string value causes the `provider_conf_params` function to call itself recursively and overflow the stack. For example, loading the following configuration file, which references the `provider_sect` section within the same section, causes OpenSSL to crash with a stack overflow.

```
openssl_conf = openssl_init

[openssl_init]
providers = provider_sect

[provider_sect]
 = provider_sect
```

Figure 4.1: A configuration file that causes a stack overflow

The following code snippet shows the vulnerable code. If the `value` references the section in which the corresponding key-value pair is defined, the function will call itself recursively. The recursion depth is limited by the `name` buffer size of 512 bytes. However, if `name` is empty, then up to 512 recursive calls are possible. This is because each recursive call will append only a single period character `[.]` to the `name` buffer if `name` is empty. Experiments show that the stack size limit is hit quickly.

```
static int provider_conf_params(OSSL_PROVIDER *prov,
                                OSSL_PROVIDER_INFO *provinfo,
                                const char *name, const char *value,
                                const CONF *cnf)
{
    STACK_OF(CONF_VALUE) *sect;
    int ok = 1;

    sect = NCONF_get_section(cnf, value);
    if (sect != NULL) {
        int i;
        char buffer[512];
        size_t buffer_len = 0;
```

```
        OSSL_TRACE1(CONF, "Provider params: start section %s\n", value);

        if (name != NULL) {
            OPENSSL_strlcpy(buffer, name, sizeof(buffer));
            OPENSSL_strlcat(buffer, ".", sizeof(buffer));
            buffer_len = strlen(buffer);
        }

        for (i = 0; i < sk_CONF_VALUE_num(sect); i++) {
            CONF_VALUE *sectconf = sk_CONF_VALUE_value(sect, i);

            if (buffer_len + strlen(sectconf->name) >= sizeof(buffer))
                return 0;
            buffer[buffer_len] = '\0';
            OPENSSL_strlcat(buffer, sectconf->name, sizeof(buffer));
            if (!provider_conf_params(prov, provinfo, buffer, sectconf->value,
                                      cnf))
                return 0;
        }

        OSSL_TRACE1(CONF, "Provider params: finish section %s\n", value);
    } else {
        // ...
    }

    return ok;
}
```

Figure 4.2: The `provider_conf_params` function can cause a stack overflow. (`crypto/provider_conf.c#67–111`)

Recommendations

Short term, have the `provider_conf_params` function count the number of recursive calls that will result depending on the configuration file; impose a hard limit (e.g., 10) on the number of recursive calls allowed. Alternatively, rewrite this function to store the allocations on the heap instead of the stack and iteratively go over the configuration.

Long term, use `clang-tidy` to detect recursive calls and verify that a recursion base case prevents the stack from overflowing. Also, improve the project's fuzzing coverage by fuzzing not only the configuration parsing code but also the configuration module initialization code.

5. Risk of heap buffer overflow during parsing of OIDs

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-OSSL-5
Target: `crypto/asn1/asn_moid.c`	

Description

The ASN1 configuration module reads 1 byte out of bounds when interpreting OIDs starting with a comma. The following configuration file contains an OID section that causes the parser to read out of bounds.

```
openssl_conf = openssl_init
[openssl_init]
oid_secti = asdf
[asdf]
lt = ,comma
```

Figure 5.1: A configuration file that causes an out-of-bounds read

The out-of-bounds read happens in the `do_create` function. The function first looks for the pointer `p` to the first comma. Then, it decrements the pointer by 1 byte. If the input `value` starts with a comma, `p` will then point to an out-of-bounds memory region.

```
p = strrchr(value, ',');
if (p == NULL) {
    // ...
} else {
    // ...
    p--;
    while (ossl_isspace(*p)) {
        // ...
    }
    // ...
}
```

Figure 5.2: The `do_create` function may read 1 byte out of bounds. (`crypto/asn1/asn_moid.c#66–93`)

Recommendations

Short term, have the `do_create` function check that `p` will be in bounds before decrementing it.

Long term, improve the project's fuzzing coverage by fuzzing not only the configuration parsing code but also the configuration module initialization code, which contains further parsing code (e.g., for OIDs).

6. Risk of segmentation fault when loading property list in “stable” configuration section

Severity: **Informational**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-OSSL-6
Target: `crypto/asn1/asn_mstbl.c`	

Description

Parsing a configuration containing a malicious property string in a “stable” section can cause a segmentation fault. The following configuration file contains the property string `min`. Loading this configuration will cause a null pointer dereference because the value of the property named `min` is null.

```
openssl_conf = openssl_init

[openssl_init]
s = mstbl

[mstbl]
id-tc26 = min
```

Figure 6.1: An example configuration that causes a segmentation fault

The null pointer dereference happens in the `do_tcreate` function. When parsing the property list, the value is assumed to be non-null. Passing a null value to `strtoul` is undefined behavior. On macOS, this causes OpenSSL to crash with a segmentation fault.

```
lst = X509V3_parse_list(value);
if (!lst)
    goto err;
for (i = 0; i < sk_CONF_VALUE_num(lst); i++) {
    cnf = sk_CONF_VALUE_value(lst, i);
    if (strcmp(cnf->name, "min") == 0) {
        tbl_min = strtoul(cnf->value, &eptr, 0);
        if (*eptr)
            goto err;
    } else if (strcmp(cnf->name, "max") == 0) {
        tbl_max = strtoul(cnf->value, &eptr, 0);
        if (*eptr)
            goto err;
    } else if (strcmp(cnf->name, "mask") == 0) {
        if (!ASN1_str2mask(cnf->value, &tbl_mask) || !tbl_mask)
            goto err;
    } else if (strcmp(cnf->name, "flags") == 0) {
```

```
        if (strcmp(cnf->value, "nomask") == 0)
            tbl_flags = STABLE_NO_MASK;
        else if (strcmp(cnf->value, "none") == 0)
            tbl_flags = STABLE_FLAGS_CLEAR;
        else
            goto err;
    } else
        goto err;
}
```

Figure 6.2: The implementation of `do_tcreate` fails to check whether `cnf->value` is null. (`crypto/asn1/asn_mstbl.c#70–95`)

Recommendations

Short term, add a null check before the use of `cnf->value`.

Long term, improve the project's fuzzing coverage by fuzzing not only the configuration parsing code but also the configuration module initialization code.

7. The ossl_prov_memdup function does not update dst_len if the call fails

Severity: **Informational**	Difficulty: **High**
Type: Error Reporting	Finding ID: TOB-OSSL-7
Target: `providers/common/provider_util.c`	

Description

The `ossl_prov_memdup` function is used throughout the provider implementations to securely duplicate a contiguous block of memory. If the copy operation succeeds, the function updates `dst_len` to the value of `src_len`. However, if the allocation fails, the function sets `dst` to `NULL` but fails to set `dst_len` to `0`.

```
/* Duplicate a lump of memory safely */
int ossl_prov_memdup(const void *src, size_t src_len,
                     unsigned char **dest, size_t *dest_len)
{
    if (src != NULL) {
        if ((*dest = OPENSSL_memdup(src, src_len)) == NULL) {
            ERR_raise(ERR_LIB_PROV, ERR_R_MALLOC_FAILURE);
            return 0;
        }
        *dest_len = src_len;
    } else {
        *dest = NULL;
        *dest_len = 0;
    }
    return 1;
}
```

Figure 7.1: If the `src` argument is `NULL`, then `dst_len` is set to `0`, but if the allocation fails, `dst_len` is not updated. (`providers/common/provider_util.c`)

Recommendations

Short term, have `ossl_prov_memdup` set `dst_len` to `0` if the call to `OPENSSL_memdup` fails.

Long term, ensure that return values are always initialized before returning control to the calling function.

8. API misuse may lead to unexpected segmentation fault

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-OSSL-8
Target: Multiple files	

Description
Several API usage patterns might lead to unexpected segmentation faults.

1. If the encoder API is used without calling the `OSSL_ENCODER_CTX_set_cleanup` function, a null pointer dereference will occur in the `encoder_process` function.

```
OSSL_ENCODER_CTX *ctx = NULL;

if ((ctx = OSSL_ENCODER_CTX_new()) == NULL) {
    ERR_raise(ERR_LIB_OSSL_ENCODER, ERR_R_MALLOC_FAILURE);
    return 0;
}

OSSL_ENCODER_CTX_set_construct(ctx, test_construct);

OSSL_ENCODER *encoder = OSSL_ENCODER_fetch(NULL, "RSA",
"output=pem,structure=SubjectPublicKeyInfo");

OSSL_ENCODER_CTX_add_encoder(ctx, encoder);

// Not including this call leads to a SEGV.
// OSSL_ENCODER_CTX_set_cleanup(ctx, cleanup);

OSSL_ENCODER_to_bio(ctx, mem);
```

Figure 8.1: An example of an invalid use of OSSL_ENCODER_CTX

2. The following dispatch array definition passes the initialization checks but causes null pointer dereferences when used later on. This provider is missing a `NEWCTX` and `FREE` function. However, during the initialization checks (figure 8.3), only the number of `OSSL_FUNC_KDF_NEWCTX` entries is checked, regardless of whether they are null.

```
const OSSL_DISPATCH ossl_kdf_hkdf_functions[] = {
    { OSSL_FUNC_KDF_NEWCTX, NULL },
    { OSSL_FUNC_KDF_NEWCTX, NULL },
    { OSSL_FUNC_KDF_DUPCTX, (void(*)(void))kdf_hkdf_dup },
    { OSSL_FUNC_KDF_RESET, (void(*)(void))kdf_hkdf_reset },
    { OSSL_FUNC_KDF_DERIVE, (void(*)(void))kdf_hkdf_derive },
```

```
    { OSSL_FUNC_KDF_SETTABLE_CTX_PARAMS,
      (void(*)(void))kdf_hkdf_settable_ctx_params },
    { OSSL_FUNC_KDF_SET_CTX_PARAMS, (void(*)(void))kdf_hkdf_set_ctx_params },
    { OSSL_FUNC_KDF_GETTABLE_CTX_PARAMS,
      (void(*)(void))kdf_hkdf_gettable_ctx_params },
    { OSSL_FUNC_KDF_GET_CTX_PARAMS, (void(*)(void))kdf_hkdf_get_ctx_params },
    { 0, NULL }
};
```

Figure 8.2: An invalid provider definition

```
for (; fns->function_id != 0; fns++) {
    switch (fns->function_id) {
    case OSSL_FUNC_KDF_NEWCTX:
        if (kdf->newctx != NULL)
            break;
        kdf->newctx = OSSL_FUNC_kdf_newctx(fns);
        fnctxcnt++;
        break;
    // ...
}
// ...
if (fnkdfcnt != 1 || fnctxcnt != 2) {
    /*
     * In order to be a consistent set of functions we must have at least
     * a derive function, and a complete set of context management
     * functions.
     */
    evp_kdf_free(kdf);
    ERR_raise(ERR_LIB_EVP, EVP_R_INVALID_PROVIDER_FUNCTIONS);
    return NULL;
}
```

Figure 8.3: Initialization checks (openssl/crypto/evp/kdf_meth.c#78–85)

```
EVP_KDF *kdf = EVP_KDF_fetch(NULL, "HKDF", NULL);
EVP_KDF_CTX *kctx = EVP_KDF_CTX_new(kdf);
EVP_KDF_CTX_free(kctx);
```

Figure 8.4: Example code that causes a null pointer dereference when used with the above dispatch array

This finding is related to this GitHub issue, which discusses the lack of the EVP_CIPHER_CTX_copy function.

Recommendations

Short term, add null checks to the relevant implementation. For the first issue, the code should check whether a cleanup function is defined. For the second issue, add null checks for the functions in the dispatch array.

Long term, develop more precise guidelines on the parameters and functions for which users are responsible for adding null checks.

9. Insufficient validation in dh_gen_common_set_params

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-OSSL-9
Target: `providers/implementations/keymgmt/dh_kmgmt.c`	

Description

The `dh_gen_common_set_params` function is used to set or update the parameters held by a Diffie-Hellman (DH) key management context. (It is invoked if a user calls `evp_keymgmt_set_params` on a DH `EVP_KEYMGMT` object.) One of the settable parameters that the function accepts is the generation type, which determines how DH parameters (like primes and sub-group generators) are generated.

```
p = OSSL_PARAM_locate_const(params, OSSL_PKEY_PARAM_FFC_TYPE);
if (p != NULL) {
    if (p->data_type != OSSL_PARAM_UTF8_STRING
        || ((gctx->gen_type =
             dh_gen_type_name2id_w_default(p->data, gctx->dh_type)) == -1)) {
        ERR_raise(ERR_LIB_PROV, ERR_R_PASSED_INVALID_ARGUMENT);
        return 0;
    }
}
```

Figure 9.1: The `gen_type` field on `gctx` could be updated with an invalid value (-1). (`providers/implementations/keymgmt/dh_kmgmt.c`)

If the parameter value is invalid, the function will return `0`, signaling an error, but will still update the generation type `gctx->gen_type` to `-1`, which does not represent a valid parameter generation type.

```
/* DH parameter generation types used by EVP_PKEY_CTX_set_dh_paramgen_type() */
# define DH_PARAMGEN_TYPE_GENERATOR     0   /* Use a safe prime generator */
# define DH_PARAMGEN_TYPE_FIPS_186_2    1   /* Use FIPS186-2 standard */
# define DH_PARAMGEN_TYPE_FIPS_186_4    2   /* Use FIPS186-4 standard */
# define DH_PARAMGEN_TYPE_GROUP         3   /* Use a named safe prime group */
```

Figure 9.2: Valid parameter generation types (`include/openssl/dh.h`)

Since the value of the parameter generation type is typically not checked exhaustively, this could lead to type confusion issues or segmentation faults.

```
if (gctx->gen_type == DH_PARAMGEN_TYPE_GROUP
        && gctx->ffc_params == NULL) {
```

```
    // ...
} else {
    // ...

    if ((gctx->selection & OSSL_KEYMGMT_SELECT_DOMAIN_PARAMETERS) != 0) {
        if (gctx->gen_type == DH_PARAMGEN_TYPE_GENERATOR)
            // ...
        else
            ret = ossl_dh_generate_ffc_parameters(dh, gctx->gen_type,
                                                  gctx->pbits, gctx->qbits,
                                                  gencb);
        // ...
    }
}
```

Figure 9.3: An invalid generation type would not be detected in dh_gen during DH parameter generation. (providers/implementations/keymgmt/dh_kmgmt.c)

Exploit Scenario

An application that relies on OpenSSL for key management attempts to set the parameter generation type for a DH key exchange. This fails, but the parameter generation type is still updated. When the context is used to generate DH parameters, the wrong parameter type is generated by the library. When the application attempts to use the context to complete the key exchange, the library crashes with a segmentation fault.

Recommendations

Short term, have the `dh_gen_common_set_params` function check the return value before updating the generation type on the context.

Long term, unit test error paths to ensure parameters are not updated if a call fails.

10. HTTP client redirects to local host instead of remote one

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-OSSL-10
Target: `crypto/http/http_client.c`	

Description

The HTTP client redirects to a local host even if the redirection response contains a URL with a remote host. A server responding with the following HTTP response redirects a client to the same server instead of a different one.

```
HTTP/1.1 302 Everything Is Just Fine
Server: netcat
Location: //openssl.org
```

Figure 10.1: HTTP server response

This is due to an invalid assumption about URLs. The HTTP client assumes that URLs starting with a slash [/] generally refer to a host-relative resource location. However, URLs can start with a double slash to indicate that a resource is located on a different host but is accessible over the same protocol (i.e., HTTP/HTTPS). The bug exists in the following code, where the redirection URL is compared with a slash.

```
if (resp == NULL && redirection_url != NULL) {
    if (redirection_ok(++n_redirs, current_url, redirection_url)
            && may_still_retry(max_time, &timeout)) {
        (void)BIO_reset(bio);
        OPENSSL_free(current_url);
        current_url = redirection_url;
        if (*redirection_url == '/') { /* redirection to same server */
            // ...
            goto new_rpath;
        }
        // ...
        (void)OSSL_HTTP_close(rctx, 1);
        // ...
        continue;
    }
    // ...
}
```

Figure 10.2: The invalid assumption about URLs when interpreting redirection URLs (openssl/crypto/http/http_client.c#1183–1210)

A similar comparison is done in the `redirection_ok` function. Even though the check in that function does not conform to the URL specification, it does not constitute a bug.

The issue can be reproduced by launching a server using the command `while true; do cat $HTTP_RESPONSE | nc -l 8000; done`, where `$HTTP_RESPONSE` points to a file containing the contents of figure 10.1.

Note that this issue could allow an attacker to circumvent a web application firewall (WAF) protecting an application. Consider using a WAF that disallows requests to local hosts because they could be used to launch SSRF attacks against server A. Now if server A makes a request to server B using the OpenSSL client, which is redirected to `//openssl.org`, the OpenSSL client will attempt to load a local resource on server A. However, the WAF would not recognize this as a request to a local host and would allow it. This opens up server A to SSRF attacks from malicious third-party servers.

Recommendations

Short term, modify the code to check whether the URL starts with a double slash and to not redirect to the `new_rpath goto` if so.

Long term, replace the URL parser with a tested implementation. Also, avoid implementing checks on plain URL strings. Instead, provide the functionality in `http_lib.c`, which can be tested.

11. OCSP requests might hang if the server responds with infinite headers

Severity: **Medium**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-OSSL-11
Target: `crypto/http/http_client.c`	

Description

An OCSP request sent by the OpenSSL library might cause a hang in the HTTP client. This is because the HTTP client accepts an unbounded number of HTTP headers. The behavior can be reproduced by creating an HTTP server that sends headers in a loop. The following figure (which is an excerpt from figure F.1 in appendix F) shows how to create a malicious server that never stops sending HTTP headers.

```
char validreq[] = "HTTP/1.1 200 OK\x0D\x0A"
        "Content-Type: application/ocsp-response\x0D\x0A";

void send_payload(int fd) {
    send(fd, validreq, sizeof(validreq) - 1, MSG_MORE);
    while (1) {
        send(fd, "a:b\x0d\x0a", 5, MSG_MORE);
    }
}

// driver code from figure F.1
```

Figure 11.1: This is a malicious HTTP server that sends an infinite stream of HTTP headers. The driver code from appendix F is required to execute this.

When the following OpenSSL OCSP command is invoked against a malicious OCSP server, the program will hang indefinitely:

> `openssl ocsp -issuer cert1.pem -cert cert.pem -url http://localhost:8080`.

This is due to the following code in `http_client.c`, which loops indefinitely:

```
/* Attempt to read a line in */
next_line:
       // ...
       n = BIO_get_mem_data(rctx->mem, &p);
       // ...
       n = BIO_gets(rctx->mem, buf, rctx->buf_size);
       // ...
```

```
key = buf;
value = strchr(key, ':');
if (value != NULL) {
    // ...
}
if (value != NULL && line_end != NULL) {
    if (rctx->state == OHS_REDIRECT
            && OPENSSL_strcasecmp(key, "Location") == 0) {
        // ...
    }
    if (OPENSSL_strcasecmp(key, "Content-Type") == 0) {
        // ...
    }
    // ...
}
/* Look for blank line indicating end of headers */
for (p = rctx->buf; *p != '\0'; p++) {
    if (*p != '\r' && *p != '\n')
        break;
}
if (*p != '\0') /* not end of headers */
    goto next_line;
```

Figure 11.2: The code responsible for parsing headers, which can loop indefinitely (openssl/crypto/http/http_client.c#639–756)

This finding is inspired by CVE-2023-38039.

Exploit Scenario

A server application is checking the validity of certificates using OpenSSL. A malicious OCSP server causes the server application to hang by sending an infinite stream of headers.

Recommendations

Short term, limit the number of headers received to a reasonable number (e.g., 30). This is already done for the length of HTTP lines using `OSSL_HTTP_DEFAULT_MAX_LINE_LEN`.

Long term, consider switching to a more battle-hardened HTTP client library. The third-party library could be an optional dependency and the current implementation could be used as fallback. The PicoHTTPParser (MIT/Perl licensed) by the h2o project could be a candidate for such a library.

12. Calling EVP_KDF_CTX_reset causes a double free when the context is freed

Severity: **Low**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-OSSL-12
Target: `providers/implementations/kdfs/scrypt.c`	

Description

A KDF context allows the current state to be reset using the `EVP_KDF_CTX_reset` function. After a reset, the Scrypt implementation will cause a double free either when it is reset again or when it is eventually freed. Since pointer fields in the context are not explicitly set to null after the corresponding data is freed, the next reset will cause the `OPENSSL_free` function to be called on already freed data. This behavior is implemented in the `kdf_scrypt_reset` function, which frees data but does not set the pointers to null, like the `kdf_hkdf_reset` function does, for example.

```
static void kdf_scrypt_reset(void *vctx)
{
    KDF_SCRYPT *ctx = (KDF_SCRYPT *)vctx;

    OPENSSL_free(ctx->salt);
    OPENSSL_clear_free(ctx->pass, ctx->pass_len);
    kdf_scrypt_init(ctx);
}
```

Figure 12.1: The function that frees the current `salt` and `pass` field but does not set them to null (`openssl/providers/implementations/kdfs/scrypt.c#92–99`)

The code is reachable through the following test case.

```
EVP_KDF *kdf;
EVP_KDF_CTX *kctx = NULL;
OSSL_PARAM params[6], *p = params;

if ((kdf = EVP_KDF_fetch(NULL, "scrypt", NULL)) == NULL) {
    goto end;
}

kctx = EVP_KDF_CTX_new(kdf);
EVP_KDF_free(kdf);
if (kctx == NULL) {
    goto end;
}
```

```
*p++ = OSSL_PARAM_construct_utf8_string("digest", "sha256", (size_t)7);
*p++ = OSSL_PARAM_construct_octet_string("salt", "salt", (size_t)4);
*p++ = OSSL_PARAM_construct_octet_string("key", "secret", (size_t)6);
*p++ = OSSL_PARAM_construct_octet_string("info", "label", (size_t)5);
*p = OSSL_PARAM_construct_end();

if (EVP_KDF_CTX_set_params(kctx, params) <= 0) {
    goto end;
}

EVP_KDF_CTX_reset(kctx);
// calling reset here again also causes a double free: EVP_KDF_CTX_reset(kctx);

end:
EVP_KDF_CTX_free(kctx);
return 1;
```

Figure 12.2: A test case that resets and clears the KDF context

Exploit Scenario

A user of OpenSSL implements a function that conditionally resets the Scrypt KDF before freeing it. During testing, the double free is not triggered because the branch that executes `EVP_KDF_CTX_reset` is not tested. In the production system, this branch is reachable through a specific input. An attacker could use this behavior to either crash the system or cause undefined behavior.

Recommendations

Short term, have the code set the `salt`, `pass`, and `pass_len` fields to `0`. Alternatively, have the code clear out the memory of the whole context if this is desired (i.e., `memset(ctx, 0, sizeof(*ctx))`).

Long term, add tests that call all operation functions for each provider implementation. Also, deploy the fuzzer for the providers, which is provided in appendix D.

13. The aesni_cbc_hmac_sha256_cipher function depends on compiler-specific behavior

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-OSSL-13
Target: `crypto/evp/e_aes_cbc_hmac_sha256.c`	

Description

The implementation of the `aesni_cbc_hmac_sha256_cipher` function uses signed integer right-shifts when verifying the HMAC. The type of shift used is implementation-dependent according to the C99 standard, which means that the behavior of the function may vary between compilers.

```
for (res = 0, i = 0, j = 0; j < maxpad + SHA256_DIGEST_LENGTH;
     j++) {
    c = p[j];
    cmask =
        ((int)(j - off - SHA256_DIGEST_LENGTH)) >>
        (sizeof(int) * 8 - 1);
    res |= (c ^ pad) & ~cmask; /* ... and padding */
    cmask &= ((int)(off - 1 - j)) >> (sizeof(int) * 8 - 1);
    res |= (c ^ pmac->c[i]) & cmask;
    i += 1 & cmask;
}
```

Figure 13.1: HMAC verification in `aesni_cbc_hmac_sha256_cipher` depends on compiler-specific behavior. (`crypto/evp/e_aes_cbc_hmac_sha256.c`)

Signed integer right-shifts may be implemented as either arithmetic or logical right-shifts according to section 6.5.7 of the C99 standard:

> *If E1 has a signed type and a negative value, the resulting value is implementation-defined.*

This means that if the shifted value E1 is negative, `E1 >> (sizeof(int) * 8 - 1)` may be either 1 or -1, depending on the compiler. It follows that the behavior of the code above may also be compiler-dependent.

The same issue is also present in the implementations of the following functions:

- `aesni_cbc_hmac_sha1_cipher` (in `e_aes_cbc_hmac_sha1.c`)
- `aesni_cbc_hmac_sha1_cipher` (in `cipher_aes_cbc_hmac_sha1_hw.c`)

- `aesni_cbc_hmac_sha256_cipher` (in `cipher_aes_cbc_hmac_sha256_hw.c`)

Exploit Scenario
A developer builds OpenSSL with a C99-compliant compiler that uses a logical right-shift for signed integer right-shifts. This causes the library to fail to validate TLS-record HMACs.

Recommendations
Short term, rewrite the HMAC verification in all of the implementations of `aesni_cbc_hmac_sha256_cipher` and `aesni_cbc_hmac_sha1_cipher` to not use signed integer right-shifts.

Long term, regularly run static analysis tools that detect undefined and implementation-specific behavior like Cppcheck.

14. Use after free when setting invalid properties on the Scrypt algorithm or if SHA-256 is missing

Severity: **Low**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-OSSL-14
Target: `providers/implementations/kdfs/scrypt.c`	

Description

The Scrypt KDF implementation frees the EVP_KDF_CTX data if it fails to fetch the SHA-256 algorithm. This can happen either if an invalid `properties` string is set through the OSSL_PARAM array (OSSL_KDF_PARAM_PROPERTIES) or if SHA-256 is not available through the currently loaded providers. After the unexpected free, a use-after-free bug can occur.

Figure 14.1 shows a test case that will trigger the use-after-free bug.

```
int r = 1;
EVP_KDF *kdf;
EVP_KDF_CTX *kctx = NULL;
unsigned char derived[32];
OSSL_PARAM params[9], *p = params;
uint64_t s_N = 2;
uint64_t s_r = 8;
uint64_t s_P = 1;

if ((kdf = EVP_KDF_fetch(NULL, "scrypt", NULL)) == NULL) {
    goto end;
}

kctx = EVP_KDF_CTX_new(kdf);
EVP_KDF_free(kdf);
if (kctx == NULL) {
    goto end;
}
/* Build up the parameters for the derivation */
*p++ = OSSL_PARAM_construct_octet_string("secret", "secret", (size_t)6);
*p++ = OSSL_PARAM_construct_octet_string("pass", "pass", (size_t)4);
*p++ = OSSL_PARAM_construct_octet_string("salt", "salt", (size_t)4);
*p++ = OSSL_PARAM_construct_uint64("n", &s_N);
*p++ = OSSL_PARAM_construct_uint64("r", &s_r);
*p++ = OSSL_PARAM_construct_uint64("p", &s_P);
// The following line causes a use-after-free later on.
*p++ = OSSL_PARAM_construct_utf8_string("properties", "invalid", (size_t)1);
*p = OSSL_PARAM_construct_end();
if (EVP_KDF_CTX_set_params(kctx, params) <= 0) {
    r = 0;
```

```
    goto end;
}
if (EVP_KDF_CTX_set_params(kctx, params) <= 0) {
    r = 0;
    goto end;
}
if (EVP_KDF_derive(kctx, derived, sizeof(derived), NULL) <= 0) {
    r = 0;
    goto end;
}
end:
EVP_KDF_CTX_free(kctx);
```

Figure 14.1: A test case that causes a use after free

When the `EVP_KDF_CTX_set_params` function is called, the Scrypt implementation will try to set a digest based on the provided `properties` string, which is stored in `ctx->propq` (figure 14.2). If the `properties` string is invalid or the SHA-256 algorithm cannot be fetched because it is not available, then `EVP_KDF_CTX_set_params` returns false, and the context is freed. Freeing the context at the end of the test case in figure 14.1 will then trigger a use after free in the `EVP_KDF_CTX_free` function.

```
static int set_digest(KDF_SCRYPT *ctx)
{
    EVP_MD_free(ctx->sha256);
    ctx->sha256 = EVP_MD_fetch(ctx->libctx, "sha256", ctx->propq);
    if (ctx->sha256 == NULL) {
        OPENSSL_free(ctx);
        ERR_raise(ERR_LIB_PROV, PROV_R_UNABLE_TO_LOAD_SHA256);
        return 0;
    }
    return 1;
}
```

Figure 14.2: The function that frees the whole context in the error case (`openssl/providers/implementations/kdfs/scrypt.c#164-174`)

This bug is a use after free because before the actual context is freed, the members are freed in `EVP_KDF_CTX_free` (refer to `scrypt:85`).

This finding was discovered through the fuzzer described in appendix D. Interestingly, the unit tests did not exercise the branch that led to the use after free, as shown in the Coveralls report.

Exploit Scenario

A user of OpenSSL implements a function that conditionally sets properties on the Scrypt algorithm. During testing, the use after free is not triggered because the branch that adds `OSSL_KDF_PARAM_PROPERTIES` is not tested. In the production system, this branch is

reachable through a specific input. An attacker uses this behavior to either crash the system or cause undefined behavior.

Recommendations

Short term, remove the call to the `OPENSSL_free` function from the `set_digest` function.

Long term, deploy and run the provider fuzzer described in appendix D.

15. Setting OSSL_MAC_PARAM_DIGEST_NOINIT for HMAC causes segmentation fault

Severity: **Informational**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-OSSL-15
Target: `crypto/evp/digest.c`	

Description

Using the parameter `OSSL_MAC_PARAM_DIGEST_NOINIT` along with an HMAC causes a segmentation fault during HMAC initialization. The parameter is translated to the `EVP_MD` flag `EVP_MD_CTX_FLAG_NO_INIT`. This digest parameter skips certain initialization steps. Users are supposed to set a custom update function by calling the function `EVP_MD_CTX_set_update_fn`. However, the new provider API does not provide an API to set an update for the internal digest. The following figure presents a test case that crashes during the execution of the `EVP_MAC_init` function.

```
int r = 1;
const char *key = "mac_key";
EVP_MAC_CTX *ctx = NULL;
OSSL_PARAM params[6], *p = params;
EVP_MAC *evp_mac = NULL;
// ...

if ((evp_mac = EVP_MAC_fetch(NULL, "hmac", NULL)) == NULL) {
    goto end;
}

int noinit = 1;
*p++ = OSSL_PARAM_construct_int(OSSL_MAC_PARAM_DIGEST_NOINIT, &noinit);
*p++ = OSSL_PARAM_construct_utf8_string("digest", "SHA3-224", 9);
*p = OSSL_PARAM_construct_end();

if ((ctx = EVP_MAC_CTX_new(evp_mac)) == NULL
    || !EVP_MAC_init(ctx, (const unsigned char *) key, strlen(key), params)) {
    r = 0;
    goto end;
}
// ...
```

Figure 15.1: A test case that causes a segmentation fault

The segmentation fault happens when the digest calls the noninitialized `update` function (figure 15.2).

```
int EVP_DigestUpdate(EVP_MD_CTX *ctx, const void *data, size_t count)
{
    // ...
    if (ctx->pctx != NULL
            && EVP_PKEY_CTX_IS_SIGNATURE_OP(ctx->pctx)
            && ctx->pctx->op.sig.algctx != NULL) {
    // ...
    if (ctx->digest == NULL
            || ctx->digest->prov == NULL
            || (ctx->flags & EVP_MD_CTX_FLAG_NO_INIT) != 0)
        goto legacy;
    // ...
    /* Code below to be removed when legacy support is dropped. */
 legacy:
    return ctx->update(ctx, data, count);
}
```

Figure 15.2: The digest update function that calls the internal `update` function (openssl/crypto/evp/digest.c#388–426)

This finding was discovered using the fuzzer described in appendix D.

Recommendations

Short term, add a null check for `ctx->update`. That way, the use of `OSSL_MAC_PARAM_DIGEST_NOINIT` cannot cause segmentation faults.

Long term, expose an API for the new KDFs that allows functions to be called on the underlying digest. Alternatively, deprecate the `OSSL_MAC_PARAM_DIGEST_NOINIT` parameter type and remove it from the next OpenSSL version.

16. Functions of EVP_CIPHER_CTX are missing null checks

Severity: **Informational**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-OSSL-16
Target: `crypto/evp/evp_lib.c`	

Description

Several functions that operate on an `EVP_CIPHER*` run into a segmentation fault if no cipher is set. None of the following functions can be called on an uninitialized context created using `EVP_CIPHER_CTX_new`:

- `EVP_CIPHER_CTX_get_key_length`
- `EVP_CIPHER_CTX_get_nid`
- `EVP_CIPHER_CTX_get_block_size`
- `EVP_CIPHER_CTX_get_iv_length`
- `EVP_CIPHER_CTX_get1_cipher`
- `EVP_Cipher`
- `EVP_CIPHER_param_to_asn1`
- `EVP_CIPHER_asn1_to_param`
- `EVP_CIPHER_get_asn1_iv`
- `EVP_CIPHER_set_asn1_iv`

For example, the following code will crash:

```
EVP_CIPHER_CTX* cipher_ctx = EVP_CIPHER_CTX_new();
if (!cipher_ctx) {
    return 0;
}
EVP_CIPHER_CTX_get_key_length(cipher_ctx);
```

Figure 16.1: Example code that crashes in the second function call

This is because `EVP_CIPHER_CTX_get_key_length` does not check whether `cipher_ctx->cipher` is non-null before dereferencing it. We believe null checks in these

functions are worth the potential performance impact because this code is reachable through higher level APIs like EVP_RAND. The following example initializes an HMAC-DRBG that uses GMAC but does not set a cipher:

```
unsigned char buf[4096];
int r = 1;
EVP_RAND_CTX *ctx = NULL;
OSSL_PARAM params[6], *p = params;
EVP_RAND *evp_rand = NULL;

if ((evp_rand = EVP_RAND_fetch(NULL, "HMAC-DRBG", NULL)) == NULL) {
    goto end;
}

// Missing cipher: *p++ = OSSL_PARAM_construct_utf8_string(OSSL_MAC_PARAM_CIPHER,
"AES-256-GCM", sizeof("AES-256-GCM"));
*p++ = OSSL_PARAM_construct_utf8_string("mac", "GMAC", 9);
*p = OSSL_PARAM_construct_end();

if (!(ctx = EVP_RAND_CTX_new(evp_rand, NULL))) {
    r = 0;
    goto end;
}
if (EVP_RAND_CTX_set_params(ctx, params) <= 0) {
    r = 0;
    goto end;
}
if (!EVP_RAND_generate(ctx, buf, sizeof(buf), 0, 0, NULL, 0)) {
    r = 0;
    goto end;
}
// ...
```

Figure 16.2: Example code that crashes because the underlying cipher is not set

This finding was discovered by the fuzzer described in appendix D.

Recommendations

Short term, add null checks for `cipher_ctx->cipher` in each of the above functions.

Long term, deploy the provider fuzzer described in appendix D.

17. Assertion could be hit when fetching algorithms by name

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-OSSL-17
Target: `crypto/evp/evp_fetch.c`	

Description

If the name in an algorithm fetch operation (i.e., the `name` argument to a function like `EVP_MD_fetch` or `EVP_CIPHER_fetch`) contains a colon after the algorithm name, then an assertion is hit in the `evp_method_id` function. For example, the call `EVP_CIPHER_fetch(NULL, "AES256:something", 0)` aborts with the message `OpenSSL internal error: Assertion failed: name_id > 0 && name_id <= METHOD_ID_NAME_MAX`. This assertion is hit because of a logic bug.

```
static void * inner_evp_generic_fetch(/* ...  */) {
    // ...
    if (meth_id == 0
        || !ossl_method_store_cache_get(store, prov, meth_id, propq, &method)) {
        // ...
        methdata->names = name;
        // ...
        if ((method = ossl_method_construct(methdata->libctx, operation_id,
                                            &prov, 0 /* !force_cache */,
                                            &mcm, methdata)) != NULL) {
            /*
             * If construction did create a method for us, we know that
             * there is a correct name_id and meth_id, (...)
             */
            if (name_id == 0)
                name_id = ossl_namemap_name2num(namemap, name);
            meth_id = evp_method_id(name_id, operation_id);
            if (name_id != 0)
                ossl_method_store_cache_set(store, prov, meth_id, propq,
                                            method, up_ref_method, free_method);
        }
        // ...
    }
    // ...
    return method;
}
```

Figure 17.1: The invalid logic for `name_id` (openssl/crypto/evp/evp_fetch.c#239–349)

The `inner_evp_generic_fetch` function first constructs a method using the `ossl_method_construct` function. The name of the algorithm is passed through

`metadata->names`. The `ossl_method_construct` function honors the colon that is used to give algorithms alternative names. Then, the algorithm name is used to get a `name_id` using the `ossl_namemap_name2num` function. This function cannot handle the colon in the `name` and thus returns `0` for the `name_id`. This means that the comment in figure 17.1 is incorrect. A successful method construction does not mean that there is a `name_id` for the `name` in this case. The next call to `evp_method_id` raises an assertion error because the `name_id` passed to the function is `0`:

```
static uint32_t evp_method_id(int name_id, unsigned int operation_id)
{
    if (!ossl_assert(name_id > 0 && name_id <= METHOD_ID_NAME_MAX)
        || !ossl_assert(operation_id > 0
                        && operation_id <= METHOD_ID_OPERATION_MAX))
        return 0;
    // ...
}
```

Figure 17.2: The assertion in `evp_method_id` (`openssl/crypto/evp/evp_fetch.c#110–118`)

The OpenSSL library aborts only in debug mode, not release mode. Therefore, this is not a security issue. Still, a failed assertion indicates a bug.

This finding was discovered by the provider fuzzer described in appendix D.

Recommendations

Short term, have the code call `evp_method_id` only if `name_id` is not `0`. That way, the fetch operation will fail gracefully.

Long term, consider making `ossl_namemap_name2num` honor the colon, just like the method construction.

18. Reinitialization of EVP_MAC for GMAC fails if parameters are not provided

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-OSSL-18
Target: `providers/implementations/macs/gmac_prov.c`	

Description

Reinitialization of an `EVP_MAC` that uses GMAC does not completely reset its state. This means that calling the `EVP_MAC_init` function on a context that was previously finished using the `EVP_MAC_final` function does not completely reset the `EVP_MAC` from that context. A successive call to the `EVP_MAC_update` function will error out.

The unit test in figure 18.2 demonstrates this behavior. The test runs the chain of initializing, updating, and finalizing the `EVP_MAC` twice. The second update call fails in the `gcm_cipher_internal` function because the IV cannot be reused (figure 18.1).

```
static int gcm_cipher_internal(PROV_GCM_CTX *ctx, unsigned char *out,
                               size_t *padlen, const unsigned char *in,
                               size_t len)
{
    // ...
    if (!ctx->key_set || ctx->iv_state == IV_STATE_FINISHED)
        goto err;
    // ...
    if (in != NULL) {
        // ...
    } else {
        // ...
        ctx->iv_state = IV_STATE_FINISHED; /* Don't reuse the IV */
        goto finish;
    }
    // ...
}
```

Figure 18.1: The internal GCM function that requires a fresh IV (openssl/providers/implementations/ciphers/ciphercommon_gcm.c#388–444)

The IV should have been reset with the second call to `EVP_MAC_init` in the test. This happens when calling the `EVP_MAC_CTX_set_params` function before the call to `EVP_MAC_init`, or if the parameters are passed directly to `EVP_MAC_init`. This is because the cipher is (re)initialized only when the cipher, key, or IV parameters are set (refer to `gmac_prov.c:215–242`).

```
int r = 1;
EVP_MAC_CTX *ctx = NULL;
unsigned char buf[4096];
OSSL_PARAM params[6], *p = params;
size_t final_l;
EVP_MAC *evp_mac = NULL;
char *key = OPENSSL_zalloc(32);

if ((evp_mac = EVP_MAC_fetch(NULL, "gmac", NULL)) == NULL) {
    goto end;
}

*p++ = OSSL_PARAM_construct_octet_string(OSSL_MAC_PARAM_KEY, key, 32);
*p++ = OSSL_PARAM_construct_utf8_string(OSSL_MAC_PARAM_CIPHER, "AES-256-GCM",
sizeof("AES-256-GCM"));
*p = OSSL_PARAM_construct_end();

if ((ctx = EVP_MAC_CTX_new(evp_mac)) == NULL) {
    r = 0;
    goto end;
}

if (EVP_MAC_CTX_set_params(ctx, params) <= 0) {
    r = 0;
    goto end;
}

if (!EVP_MAC_init(ctx, (const unsigned char *) key, 32, params) ||
    !EVP_MAC_update(ctx, (unsigned char *) text, sizeof(text)) ||
    !EVP_MAC_final(ctx, buf, &final_l, sizeof(buf)) ||
    !EVP_MAC_init(ctx, (const unsigned char *) key, 32, NULL) ||
    // The following update call fails. Adding EVP_MAC_CTX_set_params(ctx, params)
would fix it.
    !EVP_MAC_update(ctx, (unsigned char *) text, sizeof(text)) ||
    !EVP_MAC_final(ctx, buf, &final_l, sizeof(buf))) {
    r = 0;
    goto end;
}

end:
EVP_MAC_CTX_free(ctx);
```

Figure 18.2: The unit test that fails for GMAC

The use of an EVP_MAC like in the above unit test is common. The following figure shows an existing use in the random number generator, which contains a similar call chain.

```
static int do_hmac(PROV_DRBG_HMAC *hmac, unsigned char inbyte,
                   const unsigned char *in1, size_t in1len,
                   const unsigned char *in2, size_t in2len,
                   const unsigned char *in3, size_t in3len)
{
```

```
    EVP_MAC_CTX *ctx = hmac->ctx;

    if (!EVP_MAC_init(ctx, hmac->K, hmac->blocklen, NULL)
            /* K = HMAC(K, V || inbyte || [in1] || [in2] || [in3]) */
            || !EVP_MAC_update(ctx, hmac->V, hmac->blocklen)
            || !EVP_MAC_update(ctx, &inbyte, 1)
            || !(in1 == NULL || in1len == 0 || EVP_MAC_update(ctx, in1, in1len))
            || !(in2 == NULL || in2len == 0 || EVP_MAC_update(ctx, in2, in2len))
            || !(in3 == NULL || in3len == 0 || EVP_MAC_update(ctx, in3, in3len))
            || !EVP_MAC_final(ctx, hmac->K, NULL, sizeof(hmac->K)))
        return 0;

   /* V = HMAC(K, V) */
    return EVP_MAC_init(ctx, hmac->K, hmac->blocklen, NULL)
           && EVP_MAC_update(ctx, hmac->V, hmac->blocklen)
           && EVP_MAC_final(ctx, hmac->V, NULL, sizeof(hmac->V));
}
```

Figure 18.3: The use of an EVP_MAC in the DRBG_HMAC (openssl/providers/implementations/rands/drbg_hmac.c#57–78)

According to the OpenSSL documentation, the IV is generated automatically for GCM:

> *For EVP_CIPH_GCM_MODE the IV will be generated internally if it is not specified.*

If we use an HMAC rather than GMAC in the unit test above, the code works without resetting the parameters. This is because HMAC does not depend on an IV.

Exploit Scenario

A user of OpenSSL implements a function that conditionally chooses GMAC or HMAC. During testing, the error is not hit because the branch that uses GMAC is not tested. In the production system, this branch is reachable through a specific input. An attacker uses this behavior to cause an unexpected and potentially unhandled error.

Recommendations

Short term, have the code reset the cipher for GMAC when `EVP_MAC_init` is called, if that is the intended functionality. If reusing an `EVP_MAC_CTX` context for GMAC should not be allowed, then have the `EVP_MAC_init` function return an error when called with a reused GMAC.

Long term, add the fuzzer for providers described in appendix D. In order to detect this issue automatically, additional API call flows must be added. In this case, the fuzzer must assert that executing an algorithm twice with the same input gives the same output.

19. Creation of X.509 extensions can lead to undefined behavior

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-OSSL-19
Target: `X509V3_EXT_METHOD` implementations	

Description

Several configurations for X.509 extension creation cause undefined behavior. Depending on the platform, these configurations could cause a segmentation fault. X.509 extensions must not be confused with TLS certificate extensions here.

Multiple `X509V3_EXT_METHOD` implementations falsely assume that key-value pairs in an X509V3 list (created through the `X509V3_parse_list` function) have non-null values. For example, the `issuerSignTool` extension expects the `signTool` value to be non-null (figure 19.1).

```
static ISSUER_SIGN_TOOL *v2i_issuer_sign_tool(X509V3_EXT_METHOD *method,
    X509V3_CTX *ctx, STACK_OF(CONF_VALUE) *nval)
{
    // ...
    for (i = 0; i < sk_CONF_VALUE_num(nval); ++i) {
        CONF_VALUE *cnf = sk_CONF_VALUE_value(nval, i);

        if (cnf == NULL) {
            continue;
        }
        if (strcmp(cnf->name, "signTool") == 0) {
            ist->signTool = ASN1_UTF8STRING_new();
            if (ist->signTool == NULL ||
                !ASN1_STRING_set(ist->signTool, cnf->value, strlen(cnf->value))) {
                ERR_raise(ERR_LIB_X509V3, ERR_R_ASN1_LIB);
                goto err;
            }
        } else if (strcmp(cnf->name, "cATool") == 0) {
            // ...
        } else if (strcmp(cnf->name, "signToolCert") == 0) {
            // ...
        } else if (strcmp(cnf->name, "cAToolCert") == 0) {
            // ...
        } else {
            // ...
        }
    }
    // ...
```

```
}
```

Figure 19.1: The code that does not check `cnf->value` for null (openssl/crypto/x509/v3_ist.c#35–85)

A segmentation fault can be triggered using the following OpenSSL command.

```
openssl x509 -req -in request.csr -signkey key.pem -out certificate.crt -days 3650
-extensions ext -extfile openss-ext.conf
```

Figure 19.2: An OpenSSL command that crashes

The configuration file `openss-ext.conf` must contain a specifically crafted extension configuration. The following table summarizes our findings, by showing the configurations along with a reference to the code where the crash occurs.

Configuration	Reference
`[ext]` `issuerSignTool = signTool`	#1 #2 #3 #4
`[ext]` `sbgp-autonomousSysNum = AS`	#5 #6
`[ext]` `issuingDistributionPoint = fullname`	#7
`[ext]` `sbgp-ipAddrBlock = IPv4-SAFI`	#8

This finding was discovered while looking for bugs similar to finding TOB-OSSL-6.

Recommendations

Short term, add null checks for `cnf->value`, where `cnf` refers to a pointer returned by the `sk_CONF_VALUE_value` function.

Long term, write a rule for a static analyzer like Semgrep or CodeQL that scans the codebase for similar issues.

20. Missing null checks in OSSL_PARAM getters

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-OSSL-20
Target: `crypto/params.c`	

Description

The getter functions for `OSSL_PARAM` values do not check that the data field is not null. Therefore, the getter functions cause a segmentation fault if they are invoked for a parameter value with a null data field. Users might accidentally construct a parameter value that points to null (figure 20.1), so this condition should be checked for.

```
OSSL_PARAM params[9], *p = params;

OSSL_PARAM res;
res.key = "n";
res.data_type = OSSL_PARAM_UNSIGNED_INTEGER;
res.data = NULL;
res.data_size = sizeof(uint64_t);
res.return_size = OSSL_PARAM_UNMODIFIED;

*p++ = res;
*p = OSSL_PARAM_construct_end();
```

Figure 20.1: The construction of an invalid `OSSL_PARAM`

If the above parameter is used for Scrypt, then a segmentation fault is encountered when the `OSSL_PARAM_get_uint64` function is called.

```
if ((p = OSSL_PARAM_locate_const(params, OSSL_KDF_PARAM_SCRYPT_N))
    != NULL) {
    if (!OSSL_PARAM_get_uint64(p, &u64_value)
        || u64_value <= 1
        || !is_power_of_two(u64_value))
        return 0;
    ctx->N = u64_value;
}
```

Figure 20.2: Scrypt gets the `N` parameter from the parameter array. (`openssl/providers/implementations/kdfs/scrypt.c#239–246`)

The reason for the crash is that `OSSL_PARAM_get_uint64` dereferences the `data` field without checking whether it is null.

```
int OSSL_PARAM_get_uint64(const OSSL_PARAM *p, uint64_t *val)
{
    // ...
    if (p->data_type == OSSL_PARAM_UNSIGNED_INTEGER) {
#ifndef OPENSSL_SMALL_FOOTPRINT
        switch (p->data_size) {
        case sizeof(uint32_t):
            *val = *(const uint32_t *)p->data;
            return 1;
        case sizeof(uint64_t):
            *val = *(const uint64_t *)p->data;
            return 1;
        }
#endif
        return general_get_uint(p, val, sizeof(*val));
    } else if (p->data_type == OSSL_PARAM_INTEGER) {
    // ...
}
```

Figure 20.3: The dereference without a null check (openssl/crypto/params.c#823–894)

The OSSL_PARAM struct is part of the public API of OpenSSL, which should aim to catch this type of mistake made by users.

Recommendations

Short term, add a check to all OSSL_PARAM_get_* functions to check whether the data field is non-null.

Long term, deploy the provider fuzzer described in appendix D to find similar occurrences in the provider API.

21. The ossl_blake2b_final function fails to zeroize sensitive data

Severity: **Medium**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-OSSL-21
Target: `providers/implementations/digests/blake2b_prov.c`	

Description

The `ossl_blake2b_final` function finalizes a Blake2b hash context and returns the resulting digest. If the output size is not a multiple of 8, a temporary stack buffer (`outbuffer`) is used to store the digest value. This buffer is not cleared, which means that the value remains on the stack.

If the hash function is used as a KDF to derive key material, a copy of the resulting key would remain in memory.

```
int ossl_blake2b_final(unsigned char *md, BLAKE2B_CTX *c)
{
    uint8_t outbuffer[BLAKE2B_OUTBYTES] = {0};
    uint8_t *target = outbuffer;
    int iter = (c->outlen + 7) / 8;
    int i;

    /* Avoid writing to the temporary buffer if possible */
    if ((c->outlen % sizeof(c->h[0])) == 0)
        target = md;

    // Finalize the hash function and store the result in the
    // buffer pointed to by target.

    if (target != md)
        memcpy(md, target, c->outlen);

    OPENSSL_cleanse(c, sizeof(BLAKE2B_CTX));
    return 1;
}
```

Figure 21.1: The Blake2b context is scrubbed, but `outbuffer` is not zeroized before the function returns. (`providers/implementations/digests/blake2b_prov.c`)

The same issue is present in the `ossl_blake2s_final` function.

Exploit Scenario

A server uses Blake2b as a KDF to derive session keys. Because of another issue in the server implementation, malicious users can send a specially crafted message to the server

that causes it to leak stack memory from the application process as part of the response. This is used by an attacker to leak session keys belonging to other users, allowing the attacker to decrypt their sessions.

Recommendations

Short term, ensure that the stack buffer `outbuffer` is cleared if `target` is different from `md` before `ossl_blake2b_final` and `ossl_blake2s_final` return.

Long term, regularly review new cryptographic implementations to ensure that sensitive data is scrubbed from memory.

22. The kdf_pbkdf1_do_derive function fails to zeroize sensitive data

Severity: **Medium**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-OSSL-22
Target: `providers/implementations/kdfs/pbkdf1.c`	

Description

The `kdf_pbkdf1_do_derive` function implements the PBKDF1 KDF. When the key is derived, the function uses the stack buffer `md_tmp` to hold intermediate outputs from the hash function. At the end of the function, this buffer holds the derived key. The `md_tmp` buffer is never cleared before the function returns, which means that the derived key is left on the stack.

```
static int kdf_pbkdf1_do_derive(const unsigned char *pass, size_t passlen,
                                const unsigned char *salt, size_t saltlen,
                                uint64_t iter, const EVP_MD *md_type,
                                unsigned char *out, size_t n)
{
    uint64_t i;
    int mdsize, ret = 0;
    unsigned char md_tmp[EVP_MAX_MD_SIZE];
    EVP_MD_CTX *ctx = NULL;

   // Derive the PBKDF1 key and store the result in mp_tmp.

    memcpy(out, md_tmp, n);
    ret = 1;
err:
    EVP_MD_CTX_free(ctx);
    return ret;
}
```

Figure 22.1: The implementation of PBKDF1 leaves the derived key on the stack. (`providers/implementations/kdfs/pbkdf1.c`)

Exploit Scenario

A server uses PBKDF1 as the legacy fallback algorithm for hashing passwords. Because of another issue in the server implementation, malicious users can send a specially crafted message to the server that causes it to leak stack memory from the application process as part of the response. This is used by an attacker to leak password hashes belonging to other users, allowing the attacker to recover other users' passwords through an offline brute-force attack.

Recommendations

Short term, ensure that the buffer `md_tmp` is cleared before the `kdf_pbkdf1_do_derive` function returns.

Long term, regularly review new cryptographic implementations to ensure that sensitive data is scrubbed from memory.

23. Out-of-bounds read in kdf_pbkdf1_do_derive

Severity: **Medium**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-OSSL-23
Target: `providers/implementations/kdfs/pbkdf1.c`	

Description

PBKDF1 key derivation is implemented by the function `kdf_pbkdf1_derive`, which calls through to the `kdf_pbkdf1_do_derive` function to compute the actual key. Neither function validates the requested output length `keylen`. If `keylen` is greater than the digest output size, the `kdf_pbkdf1_do_derive` function will read out of bounds and leak uninitialized stack memory as part of the returned buffer.

```
static int kdf_pbkdf1_derive(void *vctx, unsigned char *key, size_t keylen,
                             const ossl_param params[])
{
    kdf_pbkdf1 *ctx = (kdf_pbkdf1 *)vctx;
    const evp_md *md;

    if (!ossl_prov_is_running() || !kdf_pbkdf1_set_ctx_params(ctx, params))
        return 0;

    if (ctx->pass == null) {
        err_raise(err_lib_prov, prov_r_missing_pass);
        return 0;
    }

    if (ctx->salt == null) {
        err_raise(err_lib_prov, prov_r_missing_salt);
        return 0;
    }

    md = ossl_prov_digest_md(&ctx->digest);
    return kdf_pbkdf1_do_derive(ctx->pass, ctx->pass_len, ctx->salt, ctx->salt_len,
                                ctx->iter, md, key, keylen);
}
```

Figure 23.1: The `kdf_pbkdf1_derive` function fails to validate the requested output length. (`providers/implementations/kdfs/pbkdf1.c`)

```
static int kdf_pbkdf1_do_derive(const unsigned char *pass, size_t passlen,
                                const unsigned char *salt, size_t saltlen,
                                uint64_t iter, const EVP_MD *md_type,
                                unsigned char *out, size_t n)
```

```
{
    uint64_t i;
    int mdsize, ret = 0;
    unsigned char md_tmp[EVP_MAX_MD_SIZE];
    EVP_MD_CTX *ctx = NULL;

    ctx = EVP_MD_CTX_new();
    if (ctx == NULL) {
        ERR_raise(ERR_LIB_PROV, ERR_R_MALLOC_FAILURE);
        goto err;
    }

    if (!EVP_DigestInit_ex(ctx, md_type, NULL)
        || !EVP_DigestUpdate(ctx, pass, passlen)
        || !EVP_DigestUpdate(ctx, salt, saltlen)
        || !EVP_DigestFinal_ex(ctx, md_tmp, NULL))
        goto err;
    mdsize = EVP_MD_size(md_type);
    if (mdsize < 0)
        goto err;
    for (i = 1; i < iter; i++) {
        if (!EVP_DigestInit_ex(ctx, md_type, NULL))
            goto err;
        if (!EVP_DigestUpdate(ctx, md_tmp, mdsize))
            goto err;
        if (!EVP_DigestFinal_ex(ctx, md_tmp, NULL))
            goto err;
    }

    memcpy(out, md_tmp, n);
    ret = 1;
err:
    EVP_MD_CTX_free(ctx);
    return ret;
}
```

Figure 23.2: If the requested key length n is greater than the digest size, the `kdf_pbkdf1_do_derive` function will copy uninitialized stack memory to the output buffer. (`providers/implementations/kdfs/pbkdf1.c`)

Exploit Scenario

A server uses PBKDF1 as the legacy fallback algorithm for hashing passwords. A configuration issue causes the server to request an output from PBKDF1 that is longer than the digest size. This causes the output digest to contain uninitialized stack memory, making the authentication process based on the resulting password hash nondeterministic.

Recommendations

Short term, add a check to ensure that the requested output length from PBKDF1 is not longer than the digest size.

Long term, ensure that MACs and KDFs are tested with invalid input parameters to ensure that they behave as expected on invalid inputs.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Automated Testing

This section describes the setup of the automated analysis tools used during this audit.

Though static analysis tools frequently report false positives, there are certain categories of issues that they detect with essentially perfect precision, such as undefined behavior, misspecified format strings, and use of unsafe APIs. We recommend that the OpenSSL team periodically run these static tools and review their findings.

C.1 Clang

We built the codebase using Clang with the following warnings enabled:

- `-Walloca`
- `-Wassign-enum`
- `-Wbad-function-cast`
- `-Wcast-qual`
- `-Wcomma`
- `-Wfloat-equal`
- `-Wformat-nonliteral`
- `-Wimplicit-fallthrough`
- `-Wimplicit-int-conversion`
- `-Wshift-sign-overflow`
- `-Wshorten-64-to-32`
- `-Wsign-conversion`
- `-Wswitch-enum`
- `-Wunreachable-code-break`
- `-Wunreachable-code-return`
- `-Wunreachable-code`

This produced a large number of warnings. In particular, warnings concerning implicit integer conversions that may lead to sign or truncation issues should be investigated further.

C.2. CodeQL

We used CodeQL to detect any known vulnerabilities present in the codebase. This analysis did not identify any issues in the library. To build the CodeQL database, we ran the command `codeql database create` in the repository root directory.

```
codeql database create --language=cpp --command="make -j4" openssl-3.1.2.codeql
```

Figure C.1: The command used to build a CodeQL database for the OpenSSL library

We ran the query suite `cpp-lgtm-full.qls` included with CodeQL, as well as a number of internal queries, on the library. To run the query suite `cpp-lgtm-full.qls`, we used the following command.

```
codeql database analyze
       --format=sarif-latest
       --output=cpp-lgtm-full.sarif
       -- openssl-3.1.2.codeql cpp-lgtm-full.qls
```

Figure C.2: The command used to run the query suite `cpp-lgtm-all.qls` on the library

C.3. Cppcheck

To install Cppcheck, we followed the instructions on the official website. We ran the tool with a few analyses disabled to remove false positives:

```
  cppcheck --suppress=unusedFunction       \
           --suppress=missingInclude       \
           --suppress=missingIncludeSystem \
           providers 2> cppcheck.txt
```

Figure C.3: The command used to run Cppcheck on the `providers` directory

C.4. Semgrep

We ran the static analyzer Semgrep with the rule sets shown in figure C.4 to identify low-complexity weaknesses in the source code repositories. These runs did not identify any issues or code quality findings.

```
semgrep --config "p/c"
semgrep --config "p/security-code-scan"
```

Figure C.4: These Semgrep rule sets did not identify any issues in the codebase.

D. Fuzzing

Fuzzing is an essential software testing method. It typically increases test coverage and covers code paths that are difficult to cover completely using conventional unit testing. For example, the unit tests of OpenSSL do not test this branch, so finding TOP-OSSL-14 was not detected. The following are all of the findings that we discovered using fuzzing: TOP-OSSL-1, TOP-OSSL-12, TOP-OSSL-14 , TOP-OSSL-15, TOP-OSSL-16, and TOP-OSSL-17.

OpenSSL runs several libFuzzer-based fuzzers through OSS-Fuzz. They are located in the `fuzz` directory. We used the official OpenSSL build configuration to compile and run the fuzzers. Each fuzzer implements the following functions:

- `int FuzzerInitialize(int *argc, char ***argv)`
  - This function runs once when the fuzzer is started.
- `void FuzzerCleanup()`
  - This function runs once when the fuzzer is stopped.
- `int FuzzerTestOneInput(const uint8_t *buf, size_t len)`
  - This is the main fuzzing function, which runs the fuzzer once on the given byte buffer.

In order to increase the test coverage, we developed three new fuzzers and added support for OpenSSL 3.1.2 to the tlspuffin fuzzer, which does not use the above harnessing.

D.1. Property List Fuzzer

This fuzzer does not require initialization or cleanup. It tests the `ossl_parse_property` function.

```
#include "internal/property.h"

int FuzzerTestOneInput(const uint8_t *buf, size_t len)
{
    char *b;
    b = OPENSSL_malloc(len + 1);
    if (b != NULL) {
        memcpy(b, buf, len);
        b[len] = '\0';
        OSSL_PROPERTY_LIST *red  = ossl_parse_property(NULL, (const char *)b);
        if (red) {
            ossl_property_free(red);
        }
        OPENSSL_free(b);
```

```
    }
    return 0;
}
```

Figure D.1: The fuzzer for `ossl_parse_property`

Several signed integer overflows were discovered with the use of UBSan.

D.2. Extended Configuration Fuzzer

This fuzzer builds on top of the existing configuration fuzzer in `fuzz/conf.c`. The main difference is that it calls the `CONF_modules_load` function after loading the configuration. This uncovered several bugs in the interpretation of configurations, whereas the original fuzzer tests only configuration parsing.

```
int FuzzerInitialize(int *argc, char ***argv) {
    OPENSSL_load_builtin_modules();
    return 1;
}

int FuzzerTestOneInput(const uint8_t *buf, size_t len)
{
    long errorline = -1;
    int r = 0;
    OSSL_LIB_CTX *libctx = NULL;
    BIO *mem_bio = NULL;
    CONF *conf = NULL;

    if ((libctx = OSSL_LIB_CTX_new()) == NULL)
        goto end;

    if ((mem_bio = BIO_new(BIO_s_mem()))  == NULL)
        goto end;

    BIO_write(mem_bio, buf, len);

    if ((conf = NCONF_new_ex(libctx, NULL)) == NULL)
        goto end;

    if (NCONF_load_bio(conf, mem_bio, &errorline) <= 0)
        goto end;

    if (CONF_modules_load(conf, NULL, 0) <= 0)
        goto end;

    r = 1;

    end:
    CONF_modules_finish();
    NCONF_free(conf);
    BIO_free(mem_bio);
    OSSL_LIB_CTX_free(libctx);
```

```
    return r;
}
```

Figure D.2: The fuzzer for configuration parsing and interpretation

D.3. Provider Implementations Fuzzer

During this audit, we were looking for a way to fuzz the new provider API, and we found that the execution of algorithm implementations with random `OSSL_PARAM` arrays was a good fit for fuzzing. The high-level implementation of the fuzzing harness is summarized as follows:

1. First, the fuzzer initializes an `OSSL_LIB_CTX` with default options. For each operation, it collects all algorithms and stores them in a global variable. This results in several stacks, such as `STACK_OF(EVP_MD)`, `STACK_OF(EVP_KDF)`, and `STACK_OF(EVP_CIPHER)`.

2. For each random input buffer generated by the fuzzer (starting with an empty corpus), the fuzzer does the following:

    a. It selects a random operation and algorithm based on the first two integers of the input buffer.

    b. It gets all settable parameters for the algorithm from `EVP_MD_settable_ctx_params`.

    c. For each parameter, the fuzzer reads a random value from the random input buffer and honors its type. For example, for an `OSSL_PARAM_INTEGER`, it reads an `int64_t` and creates an `OSSL_PARAM` from it.

    d. Depending on the operation, the fuzzer executes a test case. For example, for digests, it calls `EVP_DigestInit_ex2`, `EVP_DigestUpdate`, and `EVP_DigestFinal_ex`. It executes similar code for processes such as symmetric encryption and key derivation.

3. Finally, the fuzzer frees all data allocated during step 1.

The fuzzer reads data from the input buffer and interprets it. Here, it is essential to use magic values to separate several inputs, rather than a type-length-value (TLV) encoding. We want the fuzzer to still be able to progress after flipping bytes, and operations like making a string longer usually require two mutations when using a TLV encoding (one for increasing the string length and one for inserting new bytes). When using magic values to separate input strings from other inputs, a single mutation (inserting new bytes) is enough. This is also described in the libFuzzer documentation.

There are also several parameters types, like `OSSL_KDF_PARAM_SCRYPT_N` and `OSSL_KDF_PARAM_ITER`, that we hard code to 1, because fuzzing the value of these parameters would stop the fuzzer from progressing due to the high execution time.

To improve parameter name generation, we also used a dictionary generated from `core-names.h`. We extracted all strings using grep (`grep -o '".*"' include/openssl/core_names.h > dictionary.txt`) and provided the resulting text file as a dictionary to libFuzzer.

We did not implement provider fuzzing for all primitives supported by the OpenSSL library. We currently support digests, symmetric encryption, KDFs, MACs, and RNGs. Support for KEMs, key management functions, key exchanges, asymmetric encryption, and signing/signature verification is not yet implemented due to the time constraints imposed by the audit. In order to support these, several stub methods have to be implemented.

The source code for the fuzzer is delivered alongside this report.

D.4. Dolev-Yao TLS Fuzzing Using tlspuffin

Since 2022, Trail of Bits has been researching stateful fuzzing of cryptographic protocols. The project started in 2021 as a research project at Inria Nancy (LORIA) in France. This research culminated in a paper on the Dolev-Yao (DY) fuzzing approach, which will be published at 2024 IEEE S&P. The corresponding fuzzer is called tlspuffin.

The current TLS fuzzer in the OpenSSL project essentially fuzzes only the Client/Server Hello messages, as they are the only messages in TLS 1.3 that are not encrypted. It is unlikely that the fuzzer triggers interesting states beyond the first message. This is where the idea of DY fuzzing comes into play. In the 1980s, the formal methods community identified and mathematically defined the DY model. It allows us to reason about cryptographic protocols on a logical and structural level. To fuzz a protocol specifically on a structural level, a DY fuzzer injects, omits, and modifies encrypted TLS messages. The fuzzer is capable of decrypting TLS messages and modifying individual fields. Using this approach, the tlspuffin fuzzer has discovered several CVEs of medium severity in wolfSSL.

The tlspuffin fuzzer is also capable of detecting logical security flaws. This class of bug usually does not result in a crash or memory corruption that would be detectable by AddressSanitzer. The current version of tlspuffin is capable of detecting issues like authentication bypasses, where a server or client can impersonate another one.

As part of the engagement, we ran tlspuffin on OpenSSL 3.1.2 for 72 hours. No issues were detected during this time.

The tlspuffin fuzzer is continuously improved, and development is ongoing. For example, a new feature promises to add classical bit-level fuzzing capabilities to tlspuffin. As already mentioned, tlspuffin works on a more structural level and does not flip single bits in its

current version. However, it makes perfect sense to combine both approaches. This feature is expected to be released later this year.

D.5. Recommendations for Future Fuzzing

Based on the results of this audit, we recommend continuing to invest efforts into fuzzing. Here, we summarize some potential directions for future work.

- **Implement missing operations for the provider fuzzer.** Even though we fuzzed only a selection of operations, we discovered several bugs in the APIs in this way. It makes sense to expand this effort to cover all primitives supported by the architecture. Also, additional execution flows for each operation could help identify more bugs. For instance, TOB-OSSL-18 would be detectable by adding an assertion to the fuzzing harness to ensure that executing an operation twice gives identical results.

- **Implement differential fuzzing using the results from the provider fuzzer.** During our fuzzing efforts for this audit, we ignored the results from the cryptographic computations from the provider fuzzer. However, the results could be used to perform differential fuzzing between different architectures (ARM64, x86-64, MIPS, etc.). This can be achieved by storing the cryptographic output of the test cases in the corpus after a long fuzzing campaign. The corpus could be reexecuted on different architectures, and the results could be compared with the previously stored output. The same could be done between different versions of OpenSSL to prevent regressions, or between OpenSSL and other cryptographic libraries. This is essentially the aim of cryptofuzz.

- **Benchmark the project's fuzzers.** The fuzzers implemented during this engagement have not yet been benchmarked for coverage. Investigating their coverage and investing in improvements could uncover more bugs.

- **Support different fuzzing engines.** The OpenSSL project uses only libFuzzer to drive its fuzzers. The integration of fuzzers like AFL++ or the novel LibAFL could yield improvements in terms of test case executions per second.

- **Implement OpenSSL TLS fuzzing modes.** The BoringSSL library features a compilation mode that disables several checks. A mode like this could improve the coverage of existing fuzzers in the OpenSSL projects.

E. Code Quality Recommendations

The following section contains code quality recommendations that do not have any immediate security implications.

1. Remove the reference to the `HAVE_ATOMICS` macro in the `provider_new` function. In `provider_new`, the provider reference count lock is initialized if `HAVE_ATOMICS` is not defined.

```
   if ((prov = OPENSSL_zalloc(sizeof(*prov))) == NULL
#ifndef HAVE_ATOMICS
       || (prov->refcnt_lock = CRYPTO_THREAD_lock_new()) == NULL
#endif
       ) {
        OPENSSL_free(prov);
        ERR_raise(ERR_LIB_CRYPTO, ERR_R_MALLOC_FAILURE);
        return NULL;
    }
```

Figure E.1: The reference count lock is initialized if `HAVE_ATOMICS` is not defined. (`crypto/provider_core.c`)

However, the implementations of `CRYPTO_UP_REF` and `CRYPTO_DOWN_REF` both assume that atomics are available and do not check whether `HAVE_ATOMICS` is defined.

2. Fix the spelling of the `s390x_keccakc_final` function. The name of the S390X finalization function for Keccak is misspelled and should be corrected.

```
static int s390x_keccakc_final(unsigned char *md, void *vctx, int padding)
```

Figure E.2: The function name `s390x_keccakc_final` is misspelled. (`providers/implementations/digests/sha3_prov.c`)

3. Have the engine initialization conditionally compiled in the `ossl_prov_set_macctx` function. The following lines could be wrapped in `#if !defined(OPENSSL_NO_ENGINE) && !defined(FIPS_MODULE) ... #endif` because the engine variable is never used if either `OPENSSL_NO_ENGINE` or `FIPS_MODULE` are defined.

```
if (engine == NULL) {
    if ((p = OSSL_PARAM_locate_const(params, OSSL_ALG_PARAM_ENGINE))
            != NULL) {
        if (p->data_type != OSSL_PARAM_UTF8_STRING)
            return 0;
        engine = p->data;
    }
}
```

Figure E.3: This part of `ossl_prov_set_macctx` could be conditionally compiled. (`providers/common/provider_util.c`)

4. Move the null check in the `hmac_setkey` function. The `hmac_setkey` function dereferences the `key` argument as part of a call to `memcpy` and then later checks whether the `key` is null before the call to `HMAC_Init_ex`. If the `key` should be checked, this check should occur before the call to `memcpy`.

5. Update NIST standard references in the `kbkdf` implementation. The `kbkdf` implementation currently references NIST standard SP 800-108, which was withdrawn in August of 2022. That standard was replaced with NIST standard SP 800-108 revision 1, which is now the most up to date. The comments in the `kbkdf` implementation should be updated to reflect this. In addition to updating the link to the standard, the comments referring to specific sections of the standard need to be updated. In particular, references to sections 5.1, 5.2, and 5.3 need to be changed to sections 4.1, 4.2, and 4.3, respectively.

7. Prefer iteration to recursion when iterating through encoding/decoding steps. The implementation of encoding and decoding is currently based on recursion (refer to examples here and here). For long encoding and decoding chains, this implementation could overflow the stack, as demonstrated by the example below. We recommend using an iterative approach that allocates memory on the heap instead of the stack, as this would be more robust.

```
OSSL_ENCODER_CTX *ctx = NULL;

if ((ctx = OSSL_ENCODER_CTX_new()) == NULL) {
    ERR_raise(ERR_LIB_OSSL_ENCODER, ERR_R_MALLOC_FAILURE);
    return 0;
}

OSSL_ENCODER_CTX_set_construct(ctx, test_construct1);

OSSL_ENCODER *encoder = OSSL_ENCODER_fetch(NULL, "ASDF",
"output=asdf,structure=type-specific");
for (int i = 0; i < 2500; ++i) {
    OSSL_ENCODER_CTX_add_encoder(ctx, encoder);
}

OSSL_ENCODER_CTX_set_cleanup(ctx, cleanup);
```

```
OSSL_ENCODER_to_bio(ctx, mem);
```

Figure E.4: The unit test that crashes if an encoder that converts from ASDF to ASDF is present

Note that for this example, we added an encoder that converts from the format ASDF to itself to `providers/encoders.inc`.

8. Update the comment for the `dh_builtin_genparams` function. The code comment for `dh_builtin_genparams` states that the function assumes that the generator argument is not `0`, 1, or `-1`. However, the function actually checks the generator argument and returns an error if it is less than or equal to 1.

9. Avoid using XOR as a logical operator in conditions. The function `common_check_sm2` (in `providers/implementations/keymgmt/ec_kmgmt.c`) uses XOR to define an `if` statement condition. This should be avoided, as it makes the code harder to read.

10. Update the comment for the `common_import` function. The comment for `common_import` (in `providers/implementations/keymgmt/ec_kmgmt.c`) claims that the function can import private keys and, optionally, the corresponding public key; however, the `ossl_ec_key_fromdata` function, which implements the import code, will always import the public key if it is available. Thus, in practice, importing only the private key is impossible.

11. Remove the superfluous length check in the `gcm_tls_iv_set_fixed` function. The length check in `gcm_tls_iv_set_fixed` against `0` is redundant because the length is known to be greater than `EVP_GCM_TLS_FIXED_IV_LEN`, which is 4.

12. Remove the duplication of the functionality in the `scrypt_set_membuf` function. The `scrypt_set_membuf` function is duplicated (with different names) across a number of KDF implementations (KRB5 KDF, PBKDF1, PBKDF2, PKCS12 KDF, Scrypt, and SSH KDF). This function should be moved to a common location like `provider_util.c`, and each KDF should be updated to reference this common implementation.

F. Driver Code for a Malicious HTTP Server

This section contains code that implements a malicious HTTP server, which never stops sending HTTP headers.

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define PORT 8080
#ifndef MSG_MORE
# define MSG_MORE 0
#endif

char validreq[] = "HTTP/1.1 200 OK\x0D\x0A"
        "Content-Type: application/ocsp-response\x0D\x0A";

void send_payload(int fd) {
    send(fd, validreq, sizeof(validreq) - 1, MSG_MORE);
    while (1) {
        send(fd, "a:b\x0d\x0a", 5, MSG_MORE);
    }
}

int main() {
    int sock, fd, len;
    struct sockaddr_in servaddr, cli;
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == -1) {
        exit(0);
    }
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);
    if ((bind(sock, (struct sockaddr *) &servaddr, sizeof(servaddr))) != 0) {
        exit(0);
    }
    if ((listen(sock, 5)) != 0) {
        exit(0);
    } else {
        printf("Server listening..\n");
    }
    len = sizeof(cli);
    fd = accept(sock, (struct sockaddr*) & cli, &len);
```

```
    if (fd < 0) {
        exit(0);
    }
    send_payload(fd);
    close(sock);
}
```

Figure F.1: The code that implements a malicious HTTP server

G. Integer Type Recommendations

Issues related to implicit integer truncation, sign conversion, and integer overflows are often the root cause of more serious vulnerabilities in low-level languages like C and C++. These types of issues are often hard to detect during manual code review, but they can easily be detected statically by the compiler or other static analysis tools like UBSan.

There are a number of best practices that developers can implement to decrease the risk of vulnerabilities due to implicit integer conversions.

G.1. Recommendations

Prefer fixed-width integer types. Using fixed-width integer types like `int8_t`, `uint8_t`, `int32_t`, `uint32_t`, `int64_t`, and `uint64_t` ensures that the variable size remains consistent across platforms and helps to make developers' expectations around size and sign clear from the choice of type. This is particularly important when implementing cryptographic primitives where the state is expected to have a fixed, platform-independent size.

Avoid using signed types to represent unsigned quantities. Avoiding this practice makes expectations around variable use clear to the reader and reduces the risk of undefined behavior due to signed shifts or signed overflows.

The following figure shows the use of a signed variable to determine an array length. This might be problematic if the user controls the `len` variable and can set it to negative values.

```
int len = 0;

// Here len is computed from user-controlled values.

if (len > MAX_COPY_LEN) {
    return -1;
}
memcpy(&dest, &src, len * sizeof(type));
```

Figure G.1: If the user can cause `len` to be negative, the check will fail and the buffer at `dest` could overflow when the data is copied.

Some care must be taken to prevent the introduction of new vulnerabilities when a signed variable is replaced by an unsigned one. For example, if a loop condition checks whether `i-- >= 0`, replacing the loop variable `i` with an unsigned variable would cause an infinite loop because the loop condition would always be true.

Avoid mixing signed and unsigned integer types in arithmetic expressions. Mixing different integers types introduces implicit integer promotions, which could lead to hard-to-diagnose issues and vulnerabilities. In particular, the overflow behavior of signed

and unsigned integers is different. Unsigned integers will wrap on overflow, while signed overflow is undefined behavior in C. If signed and unsigned types are used in the same expression, it is always better to make type conversions explicit.

Avoid implicit integer narrowing. Narrowing an integer value (e.g., when passing a 64-bit integer to a function that takes a 32-bit argument) may truncate the value and in the worst case can lead to memory-safety issues like out-of-bounds reads and writes. An example of when this can be an issue is given in figure G.2.

```
unsigned char * data = NULL;
unsigned long long data_size = 0;

// Here data_size is computed from user-controlled values.

data = malloc(data_size);
```

Figure G.2: If `data_size` is greater than 2^{32}, the input to `malloc` is silently truncated on 32-bit platforms. Writing to the allocated buffer may cause an out-of-bounds write.

Ensure that defined constants have the correct type. Defined integer literals default to either `int`, `long`, or `long long` in C. To ensure that they are typed correctly, defined literals should include the correct suffix and/or the corresponding type.

```
// MAX_SIZE will be interpreted as an int value in expressions.
#define MIN_SIZE    1024
// MAX_SIZE will be interpreted as an uint64_t value in expressions.
#define MAX_SIZE    ((uint64_t)2048ULL)
```

Figure G.3: Ensure that defined literals are typed correctly by including the type in the definition.

Avoid casting pointers to and from integers. Casting between integers and pointers should be avoided. Casting from integers to pointers is implementation-specific behavior in C. The result may not be properly aligned or may not point to data of the correct type. Casting from pointers to integers is also implementation-specific behavior. If the integer type is not large enough to represent the value, it is undefined behavior.

Enable compiler flags that detect implicit conversions. Both GCC and Clang support compiler flags that detect implicit integer conversions and integer truncation. Enabling these flags allows developers to find problematic integer conversions and truncations quickly. For Clang, the following compiler flags are helpful in detecting issues related to implicit integer conversions.

- `-Wimplicit-int-conversion`: Signals an implicit integer conversion
- `-Wshift-sign-overflow`: Signals that a signed shift sets the sign bit of the result
- `-Wshorten-64-to-32`: Signals an implicit conversion that loses integer precision

- `-Wsign-conversion`: Signals an implicit conversion that changes signedness
- `-Wsign-compare`: Signals a comparison of integers of different signs

For a list of all Clang compiler flags and their interpretations, refer to the Clang reference page for diagnostic flags.

Use UBSan during testing to detect undefined behavior due to integer arithmetic. Enabling UBSan during unit testing ensures that undefined behavior due to issues like signed integer overflows or out-of-bounds shifts is detected early in the development process.

G.2. References

- The SEI CERT C Coding Standard
- Core C++ guidelines on signed/unsigned usage
- Vulnerabilities in C : When integers go bad!

H. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From April 8 to April 9, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the OpenSSL team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

The OpenSSL team provided us with a list of pull requests (PRs) that we matched to the findings. The PRs were easy to follow, allowing us to focus on reviewing the fixes.

While reviewing the PRs, we also observed that the codebase's testing could be improved. We recommend including the fuzzer we provided through PR #22964.

In summary, of the 23 issues described in this report, OpenSSL has resolved 16 issues, has partially resolved 2 issues, has accepted the risk of 3 issues, and has yet not resolved the remaining 2 issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Risk of signed integer overflows when parsing property queries	Resolved
2	The provider configuration format is prone to misuse	Resolved
3	The default provider supports insecure algorithms	Partially Resolved
4	Provider configuration section can cause a stack overflow	Resolved
5	Risk of heap buffer overflow during parsing of OIDs	Resolved
6	Risk of segmentation fault when loading property list in "stable" configuration section	Resolved
7	The ossl_prov_memdup function does not update dst_len if the call fails	Risk Accepted
8	API misuse may lead to unexpected segmentation fault	Partially Resolved

9	Insufficient validation in dh_gen_common_set_params	Resolved
10	HTTP client redirects to local host instead of remote one	Risk Accepted
11	OCSP requests might hang if the server responds with infinite headers	Resolved
12	Calling EVP_KDF_CTX_reset causes a double free when the context is freed	Unresolved
13	The aesni_cbc_hmac_sha256_cipher function depends on compiler-specific behavior	Risk Accepted
14	Use after free when setting invalid properties on the Scrypt algorithm or if SHA-256 is missing	Unresolved
15	Setting OSSL_MAC_PARAM_DIGEST_NOINIT for HMAC causes segmentation fault	Resolved
16	Functions of EVP_CIPHER_CTX are missing null checks	Resolved
17	Assertion could be hit when fetching algorithms by name	Resolved
18	Reinitialization of EVP_MAC for GMAC fails if parameters are not provided	Resolved
19	Creation of X.509 extensions can lead to undefined behavior	Resolved
20	Missing null checks in OSSL_PARAM getters	Resolved
21	The ossl_blake2b_final function fails to zeroize sensitive data	Resolved
22	The kdf_pbkdf1_do_derive function fails to zeroize sensitive data	Resolved
23	Out-of-bounds read in kdf_pbkdf1_do_derive	Resolved

Detailed Fix Review Results

TOB-OSSL-1: Risk of signed integer overflows when parsing property queries
Resolved in PR #22874. The affected functions now check whether the parsed number exceeds a 64-bit signed integer, preventing overflows. Additionally, this PR fixes a bug causing the hexadecimal string `0xa` to be interpreted as `0` instead of `10`. Tests were added.

TOB-OSSL-2: The provider configuration format is prone to misuse
Resolved in PR #22906. Enabling a provider now requires a configuration value for the `activate` key to be explicitly set. The same is true for disabling a provider. An error results if a value is not provided. The following values are valid:

- `1/0`
- `yes/no`
- `YES/NO`
- `true/false`
- `TRUE/FALSE`
- `on/off`
- `ON/OFF`

The same changes were applied to the `soft_load` option. Tests were added.

TOB-OSSL-3: The default provider supports insecure algorithms
Partially resolved. The OpenSSL team is currently working on designing a policy for phasing out insecure algorithms. The team plans to move affected algorithms to the legacy provider in OpenSSL 4.0, which is not yet scheduled for release.

TOB-OSSL-4: Provider configuration section can cause a stack overflow
Resolved in PR #22898. The affected area of the code now prevents stack overflows caused by recursion; it requires that visited configuration sections in recursive call sequences are unique and returns an error if they are not. This means that referencing a section twice is valid, but recursively referencing a section twice is not. Tests were added.

TOB-OSSL-5: Risk of heap buffer overflow during parsing of OIDs
Resolved in PR #22957. The `do_create` function now checks whether the OID string starts with a comma. If it does, the function skips the character. This prevents an edge case in which an out-of-bounds read happens. Additionally, the `genstr` and `genconf` options were fixed, which previously hung unexpectedly. Tests were added.

TOB-OSSL-6: Risk of segmentation fault when loading property list in "stable" configuration section
Resolved in PR #22988. A null check for values in property lists was added to prevent segmentation faults. Tests were added.

TOB-OSSL-7: The ossl_prov_memdup function does not update dst_len if the call fails
Risk accepted. The OpenSSL team elected not to change the behavior of the `ossl_prov_memdup` function because the return value of `0` and the fact that `*dest` is set to `NULL` are enough to inform the caller that `*dest_len` remains uninitialized.

TOB-OSSL-8: API misuse may lead to unexpected segmentation fault
Partially resolved in PR #23069. Null checks for the cleanup function were added, which fixes the first part of the finding. However, checks for null dispatch array entries have not yet been added. According to a discussion in the PR, there are two concerns with adding these checks:

1. Dispatch arrays might contain unexpected null values due to bugs in how functions are counted (see the finding description for more information).
2. There are functions that should be mandatory when initializing a provider. However, the ones that should be mandatory differ between provider types.

We recommend at least addressing the first concern by adding checks for null function pointers (e.g., in `/crypto/evp/kdf_meth.c`).

TOB-OSSL-9: Insufficient validation in dh_gen_common_set_params
Resolved in PR #22991. The generation type used in Diffie-Hellman (DH) key management is no longer written to the context if it is invalid. Additionally, whenever the generation type is used, its value is now checked for validity. No tests were added.

TOB-OSSL-10: HTTP client redirects to local host instead of remote one
Risk accepted. The OpenSSL team does not consider this finding a security issue and has not fixed it. We still recommend using an external HTTP implementation, as explained in the finding.

TOB-OSSL-11: OCSP requests might hang if the server responds with infinite headers
Resolved in PR #23781. The maximum count of HTTP headers accepted is now set by default to 256. It can be configured using the introduced `OSSL_HTTP_REQ_CTX_set_max_response_hdr_lines` function. Tests were added.

TOB-OSSL-12: Calling EVP_KDF_CTX_reset causes a double free when the context is freed
Unresolved. The OpenSSL team has not yet fixed this finding, but plans to.

TOB-OSSL-13: The aesni_cbc_hmac_sha256_cipher function depends on compiler-specific behavior
Risk accepted. The OpenSSL team accepts the risk of potential future implementation differences for signed integer right-shifts, which can be implemented either as arithmetic or logical right-shifts. According to the team, no supported platforms perform logical shifts.

To catch this issue in case support for new architectures that perform logical shifts is added, we recommend adding a unit test that asserts the presence of arithmetic shifts:

```
assert(-1 >> 1 == -1, "Arithmetic shift is unsupported");
```

Figure H.1: Proposed addition to OpenSSL

The alternative solution to use a preprocessor macro might be inadequate as the arithmetic during runtime and while preprocessing might differ.

TOB-OSSL-14: Use after free when setting invalid properties on the Scrypt algorithm or if SHA-256 is missing
Unresolved. The issue has not yet been addressed. We revalidated the use-after-free bug by reexecuting the test case in figure 14.1 on the master branch (commit `4514e02`).

TOB-OSSL-15: Setting OSSL_MAC_PARAM_DIGEST_NOINIT for HMAC causes segmentation fault
Resolved in PR #23054. The `OSSL_MAC_PARAM_DIGEST_NOINIT` flag was deprecated, and functionality related to it was removed.

TOB-OSSL-16: Functions of EVP_CIPHER_CTX are missing null checks
Resolved in PR #22995. Checks for null ciphers were added to the affected functions. Tests were added for the `EVP_CIPHER_CTX_get_block_size` and `EVP_CIPHER_CTX_get_iv_length` functions, which now return `0` instead of dereferencing the pointer if it is null. The return values of the two functions are checked throughout the codebase. The documentation was updated to reflect the new behavior.

TOB-OSSL-17: Assertion could be hit when fetching algorithms by name
Resolved in PR #23110. If a colon-separated alternative name is used when fetching algorithms like `EVP_CIPHER_fetch(NULL, "AES256:something", 0)`, the code returns an error instead of hitting an assertion. A unit test was added to check for this condition.

TOB-OSSL-18: Reinitialization of EVP_MAC for GMAC fails if parameters are not provided
Resolved in PR #23235. The MAC documentation was updated to indicate that the behavior of the API can differ depending on the used algorithm.

TOB-OSSL-19: Creation of X.509 extensions can lead to undefined behavior
Resolved in PR #23183. Checks for null values in X.509 creation configurations, as well as unit tests, were added.

TOB-OSSL-20: Missing null checks in OSSL_PARAM getters
Resolved in PR #23083. Checks for null values for numeric types in `OSSL_PARAM` were added. Tests were added.

TOB-OSSL-21: The ossl_blake2b_final function fails to zeroize sensitive data
Resolved in PR #23173. The temporary stack buffers are now cleared after they are no longer used.

TOB-OSSL-22: The kdf_pbkdf1_do_derive function fails to zeroize sensitive data
Resolved in PR #23194. The temporary stack buffer is now cleared before the `kdf_pbkdf1_do_derive` function finishes. Tests were not added, as there is no simple way to check for remnant data on the stack.

TOB-OSSL-23: Out-of-bounds read in kdf_pbkdf1_do_derive
Resolved in PR #23174. A check for whether the key length is longer than the digest output size was added to the `kdf_pbkdf1_derive` function, preventing out-of-bounds reads. A unit test was added.

I. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Risk Accepted	The issue persists and is not planned to be resolved.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.