IS-IS向けTLVパーサ設計レビュー改訂版の印刷用ドキュメント。Mercuryリポジトリのリンクと、McGrewらのTLSフィンガープリンティング論文の書誌を参照に追加。

IS-IS 向け TLV パーサ設計レビュー

自作Cライブラリのための手法整理（改訂版）

目的

IS-IS の TLV（Type 1バイト + Length 1バイト + Value 0〜255バイト、サブTLVも同形式で入れ子）を、綺麗・安全に扱う C ライブラリを自作するにあたり、既存実装と命名済みの手法をレビューする。

二つの参照実装の対照的な設計

FRRouting isisd — アンマーシャル型。 受信バイト列を C 構造体群へ完全変換する設計。入口は isis_unpack_tlvs(avail_len, stream, dest, error_log)。入力抽象は「カーソル付きバッファ（stream）」で、現在の読み取り位置（オフセット）を内部に保持し、stream_getc() 等の逐次読み出しのたびにカーソルが前進する。TLV型ごとに pack / unpack / copy / free / print の5点セットを揃える。これは単なる規約ではなく、定型シグネチャによる量産が肝で、実装上は unpack_tlv_* 群がすべて (context, tlv_type, tlv_len, stream, log, dest, indent) という統一シグネチャで並ぶ。この統一シグネチャがあるからこそ、後述のハンドラ表と素直に組み合わせられる。（なお、このカーソルは残量情報 STREAM_READABLE も保持しており、後述のバグはこの物理残量を見ずに進めてしまったことに起因する。）

> 補足: 「アンマーシャル型」= unmarshal type
>
> 綴りはプログラミングでは L 一つの marshal / unmarshal が標準（Go の Marshal/Unmarshal、Java JAXB 等）。動名詞は英 unmarshalling / 米 unmarshaling。人名・地名の Marshall（L二重）とは別。
>
> marshalling はメモリ上のデータを保存・転送用のバイト列へ変換すること（直列化）。unmarshalling はその逆で、バイト列からメモリ上の構造体を復元すること（逆直列化）。
>
> FRR をこう呼べるのは、ワイヤ上の TLV 列を丸ごと C 構造体へ復元する＝まさに unmarshal だから。対する Wireshark の in-situ 型は、この完全な構造体表現を作る工程を持たない。
>
> 語源: marshal はゲルマン語の「馬 + 召使い」に遡り、やがて「物事を整列・配置する高官」を指すように。軍の整列、鉄道の操車場（marshalling yard）と同じ「順序立てて並べる」の語感が、

データを転送用に整える計算機用語へ受け継がれた。

Wireshark dissector — セレクティブ / in-situ 型。 構造体化せず元バッファを指したまま必要箇所だけ読む。中心は全CLVをなめて型表とマッチさせディスパッチする isis_dissect_clvs()。値参照は境界情報を持つ tvbuff_t 上の proto_tree_add_item() 経由のみで、生ポインタを直接操作しない。

両者に共通するのが table-driven dispatch（ハンドラ表による分岐）。{type, name, handler} 配列を引く形で、種別が多く拡張も多い IS-IS に向く。

実バグから抽出した安全パースの核心

FRR の Router Capability TLV（IS-IS では TLV 242、RFC 4971 / 7981）のパーサに、実際の脆弱性があった。パーサはプロトコルレベルの tlv_len を信用する一方で、ストリームバッファの状態（STREAM_READABLE）を無視していた。FRR の stream API は防御的アサーションを備えており、枯渇したバッファに対して読み取りを試みると即座に abort を発火させる。結果として、入力パケットが切り詰められていると stream_getc が lib/stream.c のアサーションを発火させ、リモートDoSになる。核心は、ループ条件が残りTLV長をチェックしていても、ストリームがサブTLVヘッダの2バイト（TypeとLength）を物理的に供給できることまでは保証していない点にある。つまり while (tlv_len > 2) だけではサブTLVヘッダ2バイトの物理的存在を保証できない。

導かれる原則:

1. 長さフィールドを信用しない — 主張された論理長と、物理バッファ残量の両方を必ず検証する。
2. ヘッダを読む前にヘッダ分の残量を確認する — type/length の2バイトを読む前に、2バイトあることを確かめる。
3. 不正入力で abort せず、警告して破棄 — ライブラリは失敗を戻り値で返し、呼び出し側に委ねる。修正側でも望ましい挙動は、TLV長をストリームの読み取り可能バイト数に対して検証し、切り詰めや不正があれば警告ログを出してPDUを破棄することとされている。

長さ検証は「3層」で考える

上記原則を入れ子TLVに一般化すると、信用すべき長さは2層ではなく3層ある:

- (a) 物理バッファ残量
- (b) 親TLVが主張する長さ
- (c) 子サブTLVが主張する長さ

このバグの本質は「(b)を信じて(a)を確認しなかった」ことにある。自作ライブラリでは、子の読み取り範囲を min(親の残り, 物理残り) でクランプし、各階層で残量を縮約していく不変条件として実装するとよい。これは後述の再帰コンビネータの設計にそのまま落ちる。

名前の付いた手法

Selective / in-situ parsing — 関心ある要素だけ変換し、参照はバッファ内を指して確保を避ける。出典は Cisco Mercury の設計ノート doc/safe-parsing.md。同ノートは「selective = 関心ある要素だけ変換し不要部は読み飛ばす」「in-situ = パケットバッファ内への参照を作りメモリ確保を避ける」と定義し、境界チェックを構造化プログラミングで強制する方針を掲げる。

> 補足: Cisco Mercury と datum 設計
>
> Mercury は Cisco（McGrew らのグループ）公開の高速ネットワークメタデータ抽出ツール（github.com/cisco/mercury）。TLS/DTLS/SSH/HTTP/TCP 等のフィンガープリントを生成し、ゼロコピーのリングバッファで 40Gbps 級まで処理する。この「大量パケットを安全に高速に舐める」要件が datum 設計を生んだ。
>
> 基本型は struct datum { const unsigned char *data; const unsigned char *data_end; }（始点・終点のポインタ対）。「ポインタ＋長さ」ではなくポインタ対を選んだ理由は、境界チェック前のポインタ演算を避け二つのポインタ比較で済むこと、data を進めても長さの減算が要らないこと。
>
> 重要な思想: datum が readable でも参照先メモリが完全とは限らない（20バイトTCPヘッダの先頭10バイトだけ等）。状態は null / readable / empty の3つ。ゆえに参照データへのアクセスは必ず境界チェック付き関数経由に限定する。これは本書が FRR バグから得た「論理長と物理残量は別物」と同じ教訓の別実装からの裏付け。
>
> 一次資料（優先順）: ①doc/safe-parsing.md（datum 設計の思想・理由。main と master で記述差あり）／②src/libmerc/datum.h（datum 実体と read_uint 系アクセサ）／③src/libmerc/dtls.h, tls.h（実プロトコルでの分解例）／④doc/npf.md（切り詰め入力の扱い・前方一致）。設計の背景論文は次項の Anderson & McGrew (2020) だが、パーサ設計の一次資料はリポジトリ。

Bounded pointer / datum パターン — 生ポインタの代わりに struct datum { uint8_t *data; uint8_t *data_end; } のような「始点・終点ペア」を全APIの基本型にし、チェック付き関数経由でのみアクセスさせる。C++17 の string_view（basic_string_view）が類似機能だが、Mercury は文字列処理を超える機能が必要なため datum を独立実装している。FRR の stream も本質は同じ。

Parse, don't validate — 「検証して bool を返す」のではなく「解析して型付きの値か失敗を返す」（Alexis King の同名エッセイ, 2019）。後段が生バイトに触れず型で守られる。

Parser combinator — 小さなパーサを合成して再帰下降を組む。Cでは関数ポインタで近似でき、TLV→サブTLV→サブサブTLVの再帰構造に向く。

Non-malleability（非展性）/ 検証済みパーサ — 同じ値が一通りのバイト列にしかエンコードされない性質。Microsoft の PulseParse は非展性バイナリ形式向けの検証済みパーサ／シリアライザコンビネータのライブラリで、分離論理を用いて仕様と証明を記述する。これを使って実装された EverCBOR は、CBOR の決定的符号化が非展性であることを証明し、C と安全な Rust の両方の検証済みコードを生成する。これらのコードはメモリ安全性・算術安全性・関数的正当

性・非曖昧性・（決定的サブセットでの）非展性について形式的に証明されている（arXiv:2505.17335、ACM CCS 2025採録）。Apple の swift-binary-parsing も同方向の実装で、型・メモリ安全性を保ちつつ整数オーバーフロー等の値由来の未定義動作を排除し、信頼できない解析値の算術にはオプショナルを返す演算子や例外を投げるメソッドを用意している。長さ計算の引き算（tlv_len - HEADER_SIZE）のアンダーフロー対策に直結する考え方である。

自作ライブラリへの骨子

- 範囲付きスライスを全APIの基本型にする
- 読み取りは残量チェック付き関数経由のみ
- 長さは「物理残量 / 親TLV主張長 / 子サブTLV主張長」の3層で検証し、子の範囲は min でクランプ。失敗は戻り値で返す
- 分岐はハンドラ表（統一シグネチャのハンドラを {type, name, handler} で引く）
- サブTLV再帰はコンビネータ的に合成
- 長さ計算のオーバーフロー・アンダーフローを必ず確認

実装上の注意（datum パターンを使う場合）

長さは必ず data_end - data（ポインタ差）で導出し、外部から受け取った長さフィールドとは独立に保持する。tlv_len - HEADER_SIZE のアンダーフロー対策としては、引き算の前に tlv_len >= HEADER_SIZE を必ず確認するか、符号付き比較に倒すのが定石。

IS-IS固有の注意

拡張IP到達可能性TLV（135番、RFC 5305）で、サブTLV存在ビットが1なのに長さ0という不整合が実際にトラブルの原因になった例がある。「フラグは立っているが対応する可変部が空」のケースを、仕様に沿って明示的に扱うこと。

参照

- FRR isisd の Router Capability TLV クラッシュ報告（GitHub FRRouting/frr Issue #20329）
- FRR isisd/isis_tlvs.c（統一シグネチャの実装）
- Cisco Mercury リポジトリ — https://github.com/cisco/mercury
- Cisco Mercury doc/safe-parsing.md（in-situ / selective・datum 設計の一次資料）、src/libmerc/datum.h / dtls.h / tls.h（datum 実装と利用例）、doc/npf.md（切り詰め入力の扱い）
- Blake Anderson and David A. McGrew, "Accurate TLS Fingerprinting using Destination Context and Knowledge Bases," arXiv:2009.01939 (2020) — https://arxiv.org/abs/2009.01939（Mercury の背景理論）
- Alexis King, "Parse, don't validate" (2019)

- T. Ramananandro et al., *Secure Parsing and Serializing with Separation Logic Applied to CBOR, CDDL, and COSE*, arXiv:2505.17335, ACM CCS 2025（PulseParse / EverCBOR）
- apple/swift-binary-parsing（ParserSpan / 安全な算術）
- RFC 4971 / 7981（IS-IS Router Capability TLV 242）、RFC 5305（拡張IP到達可能性 TLV 135）