

Linux カーネル CVE-2026-98432 脆弱性対処: カーネルパッチ不要の暫定対策説明書 (by Claude Methos 5.1)

IPv4 TCP ハンドシェイクにおける ireq_opt / inet_opt の use-after-free (競合状態) の原理と、パケットフィルタ (iptables / nftables) による暫定遮断がなぜ効くのかの解説 — Linux を運用管理する日本国内組織のシステム管理者向け

2026/10/31 公開: 登 大遊

- ① P. 1 ~ P. 41: 「ミュトス 5.1」の作成した分析結果スライド
- ② P. 42 ~ P. 82: ① の内容に対する「ChatGPT 6 Astra Pro」の意見書

本スライドは AI により作成されました

AI モデル	Claude Mythos 5.1 (Anthropic)
思考レベル	拡張思考 (Extended Thinking) — 最大 (max)
セッション開始日時	2026-10-30 22:03 JST
応答生成日時	2026-10-30 23:03 JST
基準カーネル	Linux v7.0 (Ubuntu 26.04 LTS が採用する系統) — 行番号は本資料共通
情報基準日	2026 年 10 月 30 日 (公開情報・カーネルソースの解析に基づく)

この資料でできること

- 今すぐ実行できる 2 つのフィルタコマンドと、その永続化・確認方法を得る
- 脆弱性の原理 (構造体・RCU・マルチコア競合・RCE への道筋) を図とソースで理解する
- コンテナ・ルータ・パブリッククラウド (IaaS/PaaS/SaaS) での適用範囲と限界を把握する
- 注意: 暫定対策は根本対策 (修正済みカーネルへの更新) までの橋渡しです。

対象読者: Linux システム (サーバ、コンテナホスト、仮想化基盤、ルータ) を運用管理する組織のシステム管理者 / セキュリティ担当者

「ミュトス」に作成させた分析結果スライドを個人的に無償で配布するものであり、内容の正確性等の保証はありませんが、P. 42 以降で ChatGPT 6 Astra Pro による意見書が添付されています。

本資料の構成と読み方

第 1 部 すぐ使う

スライド 3~7, 27~32

- エグゼクティブサマリ
- 最初に実行する 2 つのコマンド
- 対策の全体像 (a / b / c / 根本)
- 脆弱性プロファイルと成立条件
- 適用手順・永続化・再起動・副作用
- ルータ / コンテナでの適用範囲

第 2 部 原理を知る

スライド 8~26

- IPv4 ヘッダと IHL、呪文の解説
- Netfilter フックとパケットの旅
- カーネル内部構造体とポインタの移管
- マルチコアで競合が起こる瞬間
- RCU の約束と 2 行の修正
- ソースコード挿絵 (6 か所)
- 情報漏洩・RCE の原理、MPTCP の役割

第 3 部 クラウドと責任分界

スライド 33~41

- 層構造と『誰が直せるか』
- IaaS / PaaS / SaaS それぞれの結論
- AWS / Azure / GCP の技術推定
- 自前基盤の安全上の利点
- チェックリスト / 参考文献

凡例と表記

即実行

設定・コマンドなど、そのまま使える内容

原理

脆弱性やフィルタが効く仕組みの解説

ソース

Linux v7.0 のソースコード抜粋 (ファイル名・行番号付き、英語コメントは日本語化)

クラウド

パブリッククラウド利用時の限界と責任分界

重要な前置き

- 本資料の暫定対策 (対策 a/b/c) は 2026-10-30 時点の公開情報とソース解析に基づく評価であり、全カーネル版で実証済みという意味ではありません。
- Red Hat は『推奨基準を満たす緩和策は無い』と表明しています。Go セキュリティチームはパケットフィルタ案を『未確認だが十分かもしれない』としています。
- 根本対策は修正済みカーネルへの更新です。暫定対策はそれまでの橋渡しとして位置づけてください。

エグゼクティブサマリ: 何が起き、誰が危なく、今すぐ何をするか

即実行

9.8

CVSS (Linux カーネル CNA 評価)

4.4 →

影響範囲: 2015 年以降の全カーネル

2 行

修正差分 (kfree → kfree_rcu)

48+5 分

公開 PoC: KASLR 突破 + root 実行

0 件

実被害の報告なし / KEV 未登録 (10/28-29 時点)

何が起きたか

- Linux の IPv4 TCP 受付処理で、SYN に付いた IPv4 オプションを保持する小さなオブジェクト (ip_options_rcu) が RCU 猶予期間を待たずに解放され、別 CPU の読者が解放済みメモリを読む (use-after-free)。
- 子ソケットが要求ソケットから『ポインタを渡されるだけ』で複製されないことと、破棄時の kfree() が原因。IPv6 は複製するため非該当。
- 報告者は Anthropic の Xinyang Ge 氏。Anthropic は AI がゼロクリックの遠隔 root 実行を構築したと公表 (2026-10-27)。悪用コードは非公開。
- 修正は 2026-10-05 に適用、10-15 に CVE 公開。安定版 5.10 / 5.15 / 6.1 / 6.6 / 6.12 / 6.18 / 7.2 に反映済み。

誰が危ないか

- 未修正カーネルで IPv4 TCP を待ち受ける全ての Linux: サーバ、コンテナホスト (コンテナ内の待受も同じカーネル)、VM、ルータ、NAS、組込機器。
- 攻撃者に必要なのは『IPv4 オプション付きの SYN を待受ポートへ届けられること』と複数 CPU。認証や操作は不要。
- 公開 PoC は MPTCP 待受を前提にする。Go 1.24 以降の net.Listen は既定で MPTCP 待受を作るため、多くの Go 製サーバが該当。
- MPTCP 待受が無くても根本の競合は残る。『MPTCP を使っていないから無関係』とは言えない (Go セキュリティチームも同旨)。

今すぐやること (優先順)

- ① パケットフィルタで『IPv4 オプション付き TCP』を PREROUTING で DROP する (対策 a)。次のスライドのコマンド。
- ② MPTCP を無効化する (対策 b: sysctl / 対策 c: Go の GODEBUG 等)。公開 PoC の前提を外す。
- ③ ルールを永続化し、残存状態を消すために再起動する。
- ④ 修正済みカーネルへ計画的に更新する (根本対策)。
- ⑤ クラウド事業者・ホスティング事業者の対応状況を確認する (第 3 部)。
- 副作用: IP オプションを使う TCP 通信は遮断される (通常的环境ではほぼ存在しない)。

最初に実行する 2 つのコマンド (どちらか一方): IPv4 オプション付き TCP を 門前払いする

即実行

iptables 版 (nftables 環境でも iptables-nft 互換レイヤが入っていれば利用可)

```
sudo iptables -t raw -I PREROUTING 1 -p tcp -m u32 --u32 '0>>24&0x0F=0x06:0x0F' -j DROP
```

nftables 版 (推奨: counter で遮断数を観測できる。ファイルに保存して sudo nft -f ファイル名 で読み込む)

```
table inet cve_2026_98432 {
  chain pre { type filter hook prerouting priority -310; policy accept;
    ip protocol tcp ip hdrlength gt 5 counter drop; }
}
```

何をするルールか

- 受信した IPv4 パケットのうち、プロトコルが TCP (6) で、ヘッダ長 IHL が 5 より大きい (= IP オプションが付いている) ものを、受信直後のフック (PREROUTING) で黙って捨てます。
- 通常の TCP (IHL=5、20 バイトヘッダ) には一切触れません。IPv6 は対象外 (非該当)。
- 原理: カーネルは『オプションが有る時だけ』脆弱なオブジェクトを作るため、オプション付き SYN が届かなければ競合の対象物が生まれません (第 2 部)。

適用後に必ず行うこと

- 永続化: nftables.conf への include / iptables-persistent 等。起動時に『信頼できない通信が TCP 処理に届く前』に適用される構成にする。
- 再起動: 適用前に生成された残存オブジェクトを確実に消すため、永続化後にホストを再起動することを推奨 (詳細はスライド 28)。
- 確認: sudo nft list table inet cve_2026_98432 / sudo iptables -t raw -L PREROUTING -v -n で一致数を監視。

注意事項

- iptables 版と nftables 版は同じ判定。両方入れても害は無いが、管理の一貫性のためどちらかに統一。
- firewalld / ufw / クラウドの管理エージェントが自動でルールを書き換える環境では、再適用の仕組みに組み込む。
- コンテナ (macvlan/ipvlan/SR-IOV) や VM は、ホストの PREROUTING を通らない経路がある (スライド 32)。
- これは暫定対策。修正済みカーネルへの更新を計画する。

対策の全体像: 対策 a / b / c と根本対策の関係

即実行

対策	内容	効く範囲 (原理)	前提・制約	推奨
対策 a パケットフィルタ	PREROUTING で『IPv4 オプション付き TCP』を DROP (前スライドのルール)。ホスト単体にも、境界ルータにも適用できる。	脆弱なオブジェクト (ireq_opt) の生成そのものを防ぐ。公開 PoC (MPTCP 経由) も、未公開の他経路も、『受信 IPv4 オプションに依存する』攻撃経路をすべて遮断する。	対象 TCP 処理に至る全経路でルールが有効なこと (コンテナ・トンネル・迂回経路に注意)。NetLabel/CIPSO で内部生成する構成では不十分。適用前の残存状態は再起動等で解消。	必須 (最優先)
対策 b MPTCP 無効化 (sysctl)	sysctl net.mptcp.enabled=0 (ネットワーク名前空間ごと)。加えて net.ipv4.tcp_migrate_req=0 を確認 (既定 0)。	公開 PoC の前提 (MPTCP 待受 + 即時破棄経路) を外す。移行経路 (5.14+) のポインタ共有も閉じる。	既存の MPTCP 待受ソケットは終了・再作成が必要。古いカーネルには存在しない sysctl がある。根本の競合は残る。	a に併用
対策 c アプリ側 MPTCP 無効化	Go 1.24+: GODEBUG=multipathtcp=0、go.mod の godebug 行、//go:debug 指示。他言語も個別設定。	sysctl が触れない環境 (コンテナ、権限分離) でも、待受ソケットを MPTCP にしない。	SetMultipathTCP() 等で明示的に有効化しているコードには効かない。『脆い』緩和策 (Go チーム)。	a に併用
根本対策 カーネル更新	修正済み安定版 (5.10.272 / 5.15.223 / 6.1.190 / 6.6.159 / 6.12.113 / 6.18.56 / 7.2.10 / 7.3) 以降、またはディストリの修正版へ更新して再起動。	kfree → kfree_rcu により、読者の RCU 区間中に解放されなくなる。	Ubuntu は作業中 (High)、RHEL は影響あり・修正未公開 (10/30 時点)。ディストリの修正版を待つ間は a (+b+c)。	計画的に実施

推奨の組合せ

a (必須) → + b + c (多層防御) → 根本対策。a だけで『受信 IPv4 オプションに依存する攻撃経路』は遮断できる。b + c だけでは根本の競合が残り、クラッシュやメモリ破損のリスクを排除できない。

『効く』の意味 (評価の前提)

本評価は 2026-10-30 時点の公開情報とソース解析に基づく。『対象の TCP 処理に至るすべての攻撃経路でルールが正しく適用され、順序が維持され、適用前の残存状態が解消されている』ことが前提 (※本資料の『前提条件』)。

脆弱性プロファイル (2026-10-30 時点)

即実行

項目	内容
識別子 / 公開日	CVE-2026-98432 — 2026-10-15 (Linux カーネル CVE チーム)。修正自体は 2026-10-05 に適用。
種別 / 箇所	競合状態に起因する use-after-free。net/ipv4/tcp_ipv4.c (ireq_opt) と net/ipv4/af_inet.c (inet_opt)。IPv4 TCP の受付処理 (3 ウェイハンドシェイク) のみ。IPv6 は非該当。
影響範囲	Linux 4.4 以降 (コミット 079096f103fa『要求ソケットを ehash に入れる』で露出)。加えて 5.14 以降の要求ソケット移行 (c905dee62232) にも同種の競合。
修正コミット	upstream e3f33b0a1d89d738e2b913dab519226eaf76e15b『ipv4: free inet_opt and ireq_opt after an RCU grace period』(Eric Dumazet, 7.3-rc7)。stable 例: 11a03ce3c4a6。
修正済み版	5.10.272 / 5.15.223 / 6.1.190 / 6.6.159 / 6.12.113 / 6.18.56 / 7.2.10 / 7.3
深刻度 (評価者により差)	カーネル CNA: 9.8 (10/24 付与)。Red Hat: CVE ページ 5.9 (Moderate) → アドバイザリ 8.1 (Important、10/29)。Oracle / SUSE: 8.1。Amazon: 7.0。NVD: 10/28 時点で未評価。
報告者 / 悪用	Xinyang Ge (Anthropic)。Anthropic は内部モデルが構築した 2 段階の遠隔 root 実行 (KASLR 突破 48 分 + コマンド実行 5 分) を公表。コード・対象版・構成は非公開。KEV 未登録、実被害の報告なし (10/28-29 時点)。
MPTCP との関係	CVE 記録と修正は MPTCP に言及しない。公開 PoC が MPTCP 待受を前提にするのみ。Go 1.24 以降の net パッケージは既定で待受に MPTCP を有効化。

ディストリビューション	状況 (10/30 時点)
Ubuntu 22.04 / 24.04 / 26.04 / 26.10	作業中 (Priority High)、修正版未公開
Ubuntu 16.04 / 18.04 / 20.04	脆弱、修正版未公開
RHEL 8 / 9 / 10, OpenShift 4	影響あり、修正版未公開 (RHEL 7 は非該当)
Debian 13 trixie / 12 bookworm	修正済み (6.12.113-1 / 6.1.190-1 / 6.12.113-1~deb12u1)
Oracle Linux 7-10 (UEK)	修正済み (ELSA-2026-500397~500399)
Amazon Linux 2023	kernel6.18-6.18.56 / kernel6.12-6.12.113 で修正済み。AL2 等は保留
SUSE / openSUSE	Micro 6.2, Leap 16.0 修正済み。SLES 15 SP4-SP7, 16.0 はテスト中/作業中

※ ディストリの状況は変化が速い。適用判断の際は各ベンダの CVE ページで最新状況を確認すること。Red Hat は『推奨基準を満たす緩和策は無い』と表明しており、本資料の対策 a/b/c はコミュニティ/ソース解析に基づく暫定措置である。

攻撃が成立する 6 つの条件と、どの対策がどの条件を断ち切るか

即実行

1 IPv4 の TCP

IPv6 は非該当 (オプションを複製する実装)

断ち切る手段: —

2 SYN に IPv4 オプション

IHL > 5. これが無いと ireq_opt は作られない

断ち切る手段: 対策 a が遮断

3 待受ソケットへ到達

公開ポートに SYN を届けられること (踏み台含む)

断ち切る手段: FW / 露出削減

4 マルチコア

2 つの softirq が同時に走る必要 (ほぼ全サーバ)

断ち切る手段: —

5 child を即時破棄する経路

公開 PoC は MPTCP (MP_JOIN) の破棄経路を使用

断ち切る手段: 対策 b / c が遮断

6 未修正カーネル (4.4 以降)

根本原因。kfree → kfree_rcu の修正が未適用

断ち切る手段: カーネル更新

すべてが同時に満たされた時にだけ発動する。②を断てば (オプション付き SYN を門前払い) 残りの条件が揃っていても発動しない — これが対策 a の原理。

⑤を断つ (MPTCP 無効化) のは『公開された PoC の前提』を外すだけで、根本の UAF は残る。①④⑥は管理者には変えにくい前提条件。

例外: NetLabel/CIPSO を使う構成では、受信パケットにオプションが無くてもカーネル内部で ireq_opt を生成する経路がある (cipso_v4_req_setattr)。

読み方

攻撃は 6 条件がすべて同時に成立した時だけ発動する。管理者が確実に断ち切れるのは ②③⑤⑥。②を断つ対策 a は、公開 PoC (MPTCP 経由) と未公開の他経路の双方に効く唯一の暫定策。

注意

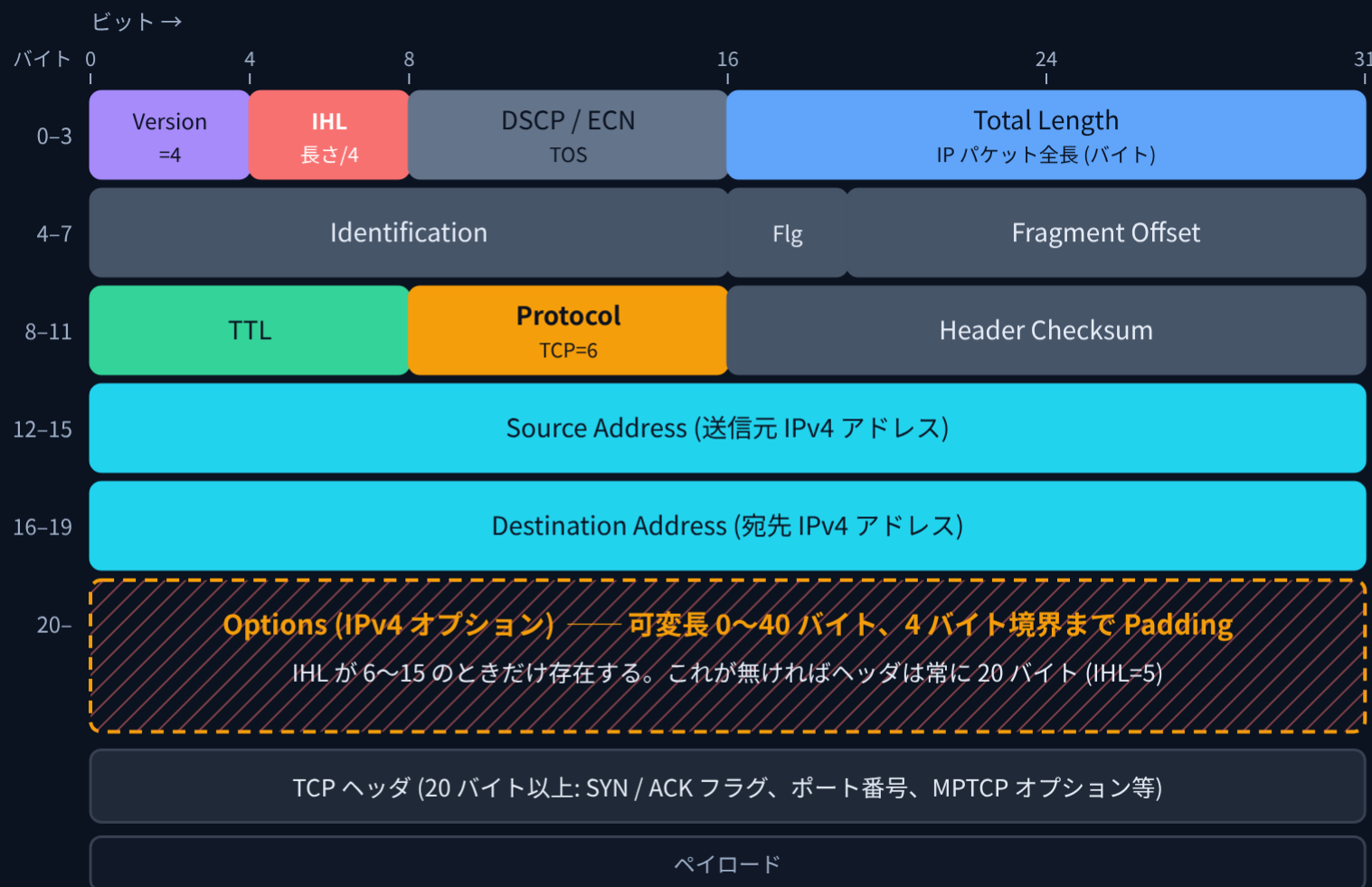
⑤を断つだけ (MPTCP 無効化) では根本の競合が残る。NetLabel/CIPSO 構成ではカーネル内部で ireq_opt が生成されるため、②を断っても十分でない → 該当ホストは最優先でカーネル更新。

第 2 部 原理を知る — 呪文の解説から、カーネル内部の競合、RCE の道筋まで

パケットヘッダ → Netfilter フック → 構造体とポインタ → マルチコア競合 → RCU の約束 → ソースコード → 悪用の原理 → MPTCP の役割

IPv4 ヘッダの構造と IHL: フィルタが見ている 4 ビット

原理



IHL = ヘッダ長 (4 バイト単位、4 ビット値)

$IHL \times 4 = \text{IPv4 ヘッダのバイト長}$

IHL = 5 → 20 バイト = オプション無し (通常)

IHL = 6~15 → 24~60 バイト = オプション有り

IHL < 5 → 不正ヘッダ (ip_rcv_core()) が破棄)

先頭バイトの読み方 (Version 4 ビット + IHL 4 ビット)

0x45 = 0100 0101 → Version 4, IHL 5 (20 B)

0x47 = 0100 0111 → Version 4, IHL 7 (28 B)

0x4F = 0100 1111 → Version 4, IHL 15 (60 B)

代表的な IPv4 オプション (種別番号)

RR (7) 経路記録 / TS (68) タイムスタンプ

LSRR (131) / SSRR (137) 送信元経路指定 (危険)

Router Alert (148) 中継ルータへの注意喚起

CIPSO (134) MLS セキュリティラベル (NetLabel)

NOP (1) / EOOL (0) 詰め物 / 終端

本脆弱性の前提: 受信した TCP SYN にこの Options 領域が存在すること (IHL > 5)。

存在しなければ ireq_opt は NULL のままで、解放競合の対象物そのものが生成されない。

呪文の解説 (1): iptables -m u32 '0>>24&0x0F=0x06:0x0F' は何を見ているのか

原理

iptables 版のルール (1 行) をトークン毎に分解

```
sudo iptables -t raw -I PREROUTING 1 -p tcp -m u32 --u32 '0>>24&0x0F=0x06:0x0F' -j DROP
```

外側の呪文: どこで・何に・どうするか

- t raw
raw テーブル: conntrack より前に評価される (優先度 -300)
- I PREROUTING 1
受信直後のフック (PREROUTING) の先頭 (1 番目) に挿入
- p tcp
IPv4 ヘッダの Protocol フィールド = 6 (TCP) のみ対象
- m u32
xt_u32: パケット中の任意 32 ビット語を取り出し演算・比較
- j DROP
一致したら黙って破棄 (応答も返さない)

内側の呪文 '0>>24&0x0F=0x06:0x0F' = 「IHL が 6~15 か?」

IPv4 ヘッダ先頭の 32 ビット語 (オフセット 0)

	Version	IHL	DSCP/ECN	Total Length	例: 0x47 00 00 3C
1	0				オフセット 0 から 32 ビットを読む (ビッグエンディアン) 0x4700003C
2	>>24				右に 24 ビットシフト → 先頭 1 バイトだけが残る 0x00000047
3	&0x0F				下位 4 ビットだけをマスク → IHL フィールドの値 0x7 (=7)
4	=0x06:0x0F				値が 6~15 の範囲内なら一致 (IHL>5 = オプション有り) 7 ∈ [6, 15] → 一致 → DROP

具体例: 通常の SYN は通過し、オプション付き SYN だけが落ちる

- 0x45 00 00 3C 1C 46 40 00 40 06 ...
先頭 0x45 → IHL=5 → 範囲外 → 一致せず → 次のルールへ (通常の TCP は影響なし)
 - 0x47 00 00 44 ... 40 06 ... [07 07 04 ..]
先頭 0x47 → IHL=7 → 範囲内 → 一致 → DROP (28 B ヘッダ = 20 B + 8 B の RR オプション)
- 注: IPv6 パケットは -p tcp (IPv4 用 iptables) の対象外。IPv6 は本脆弱性の影響を受けない (カーネルはオプションを複製するため)。

呪文の解説 (2): nftables 'ip protocol tcp ip hdrlength gt 5 counter drop' の各トークン

原理

nftables 版のルール (table 定義 1 ブロック) をトークン毎に分解

```
table inet cve_2026_98432 {
  chain pre {
    type filter hook prerouting priority -310; policy accept;
    ip protocol tcp ip hdrlength gt 5 counter drop;
  }
}
```



table inet NAME	inet ファミリ = IPv4/IPv6 両用テーブル。名前は任意 (識別用)
chain pre	チェーン名は任意。基本チェーン (フックに直結するチェーン)
type filter hook prerouting	受信直後 (PREROUTING) のフックで評価する filter 型
priority -310	raw(-300) より前、defrag(-400) より後に実行される
policy accept	一致しないパケットは通常どおり通過
ip protocol tcp	IPv4 ヘッダ Protocol = 6。『ip』 式は IPv4 専用 (IPv6 は対象外)
ip hdrlength gt 5	IPv4 ヘッダの IHL > 5 (= オプション有り)。u32 版と同じ判定
counter	一致パケット数・バイト数を計上 (監視・副作用確認に有用)
drop	黙って破棄 (応答を返さない)

適用と確認 (抜粋)

```
sudo nft -f /etc/nftables.d/cve-2026-98432.nft # 即時適用
sudo nft list table inet cve_2026_98432 # 確認
    ip protocol tcp ip hdrlength gt 5 counter packets 12 bytes 768 drop
    ↑ counter の値が増えれば遮断が動作している
sudo systemctl enable --now nftables # 永続化 (nftables.conf に include)
```

呪文の裏側: カーネルは PREROUTING で何を実行しているのか (ip_rcv / xt_u32)

ソース

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

net/ipv4/ip_input.c — ip_rcv()

```
564 int ip_rcv(struct sk_buff *skb, struct net_device *dev, struct packet_type *pt,
565           struct net_device *orig_dev)
566 {
567     struct net *net = dev_net(dev);
568     skb = ip_rcv_core(skb, net);          /* 版・IHL<5・チェックサム等の基本検査 */
569     if (skb == NULL)
570         return NET_RX_DROP;
571     return NF_HOOK(NFPROTO_IPV4, NF_INET_PRE_ROUTING, /* ★ 本ルールが評価される場所 */
572                   net, NULL, skb, dev, NULL, ip_rcv_finish); /* 通過した時だけ続きへ */
573 }
574
575 }
```

PREROUTING: ローカル宛も転送も、受信した IPv4 パケットは全てここを通る。DROP されたパケットは ip_options_compile にも TCP にも到達しない。

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

net/netfilter/xt_u32.c — u32_match_it() (iptables -m u32 のカーネル側、簡略化)

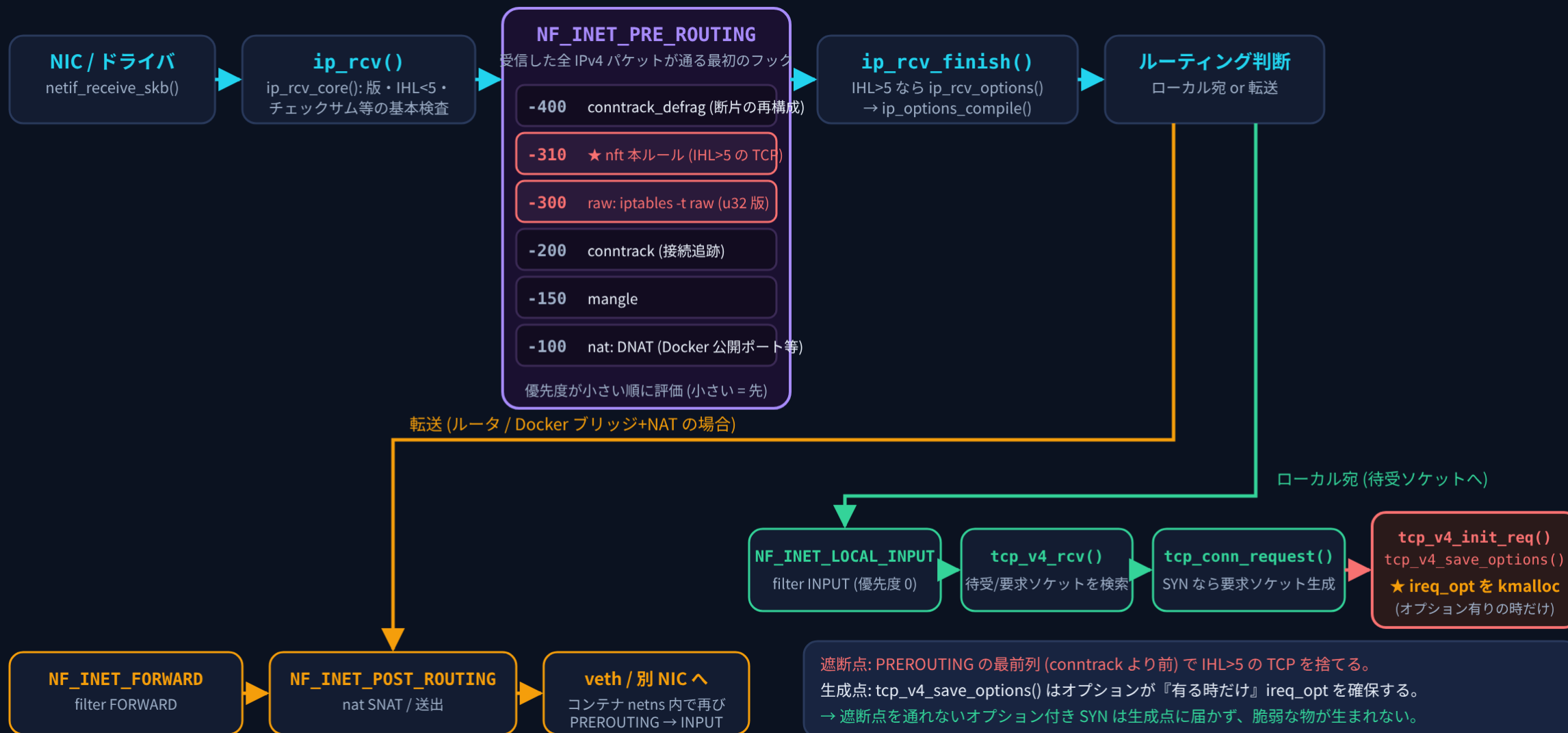
```
37 pos = ct->location[0].number;          /* '0' = オフセット 0 バイト */
39 if (skb->len < 4 || pos > skb->len - 4) return false;
44 val = ntohs(n);                          /* 先頭 32 ビット語をホスト順に */
48 for (i = 1; i < nnums; ++i) {            /* 内側ループ: &, <<, >>, @ */
50     switch (ct->location[i].nextop) {
51         case XT_U32_AND : val &= number; break; /* '&0x0F' → 下位 4 ビット = IHL */
52         case XT_U32_RIGHTSH : val >>= number; break; /* '>>24' → 先頭 1 バイト */
53     }
54 }
55 for (i = 0; i < nvals; ++i)              /* ", " と ":" の各値 (範囲) と比較 */
56     if (ct->value[i].min <= val && val <= ct->value[i].max) break; /* '=0x06:0x0F' */
57 if (i >= ct->nvalues) return false;      /* 範囲外 (IHL=5) → 次のルールへ */
```

呪文の正体: ユーザ空間で '0>>24&0x0F=0x06:0x0F' が [位置 0, 演算子 列 (>>24, &0x0F), 範囲 6..15] に翻訳され、この汎用ループが評価する。

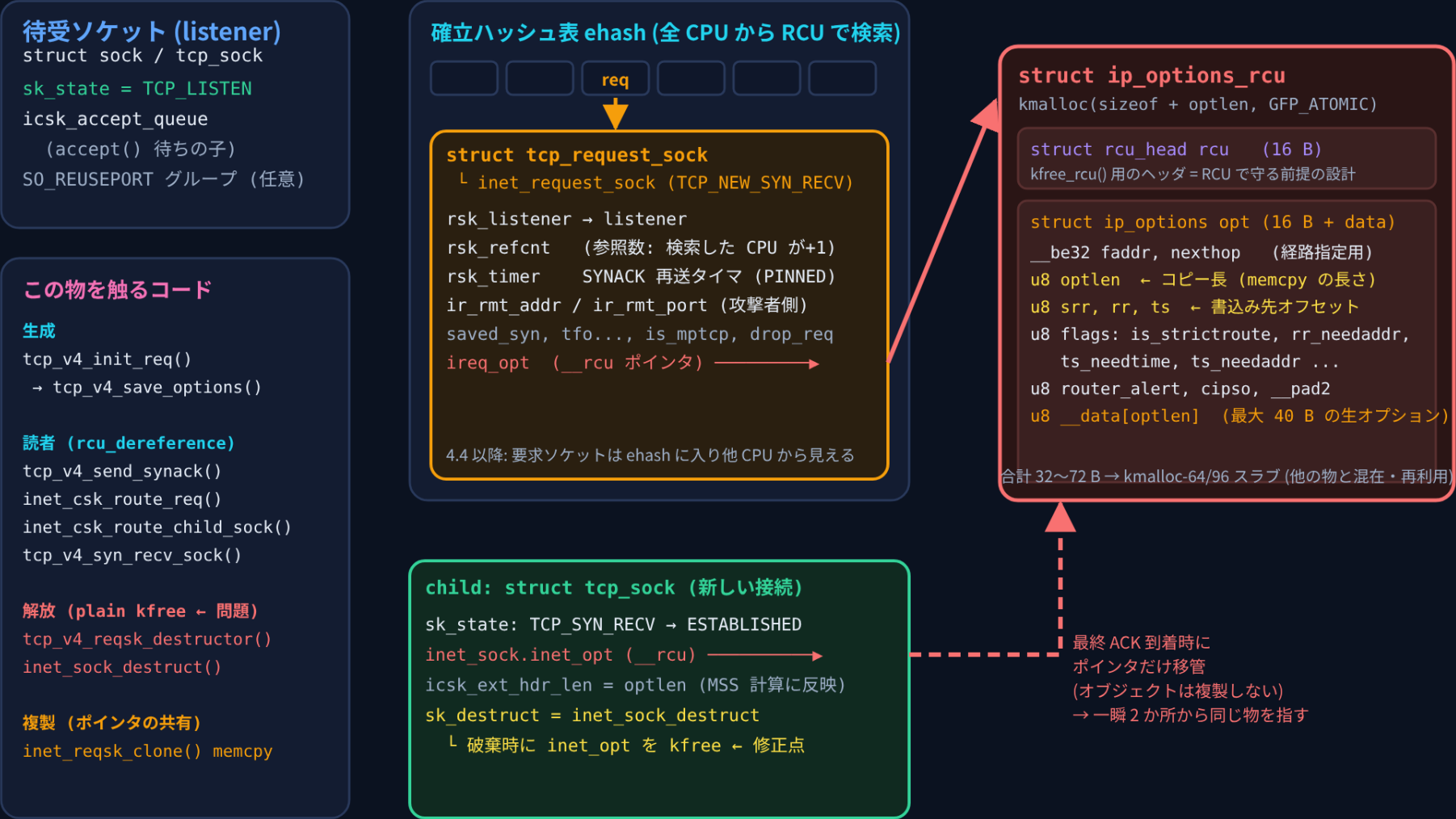
nftables 版の 'ip hlength gt 5' は nft_payload 式がヘッダ先頭バイトの下位 4 ビットを取り出して比較する等価な判定。

パケットの旅: 受信から要求ソケット生成まで — 遮断点と生成点

原理

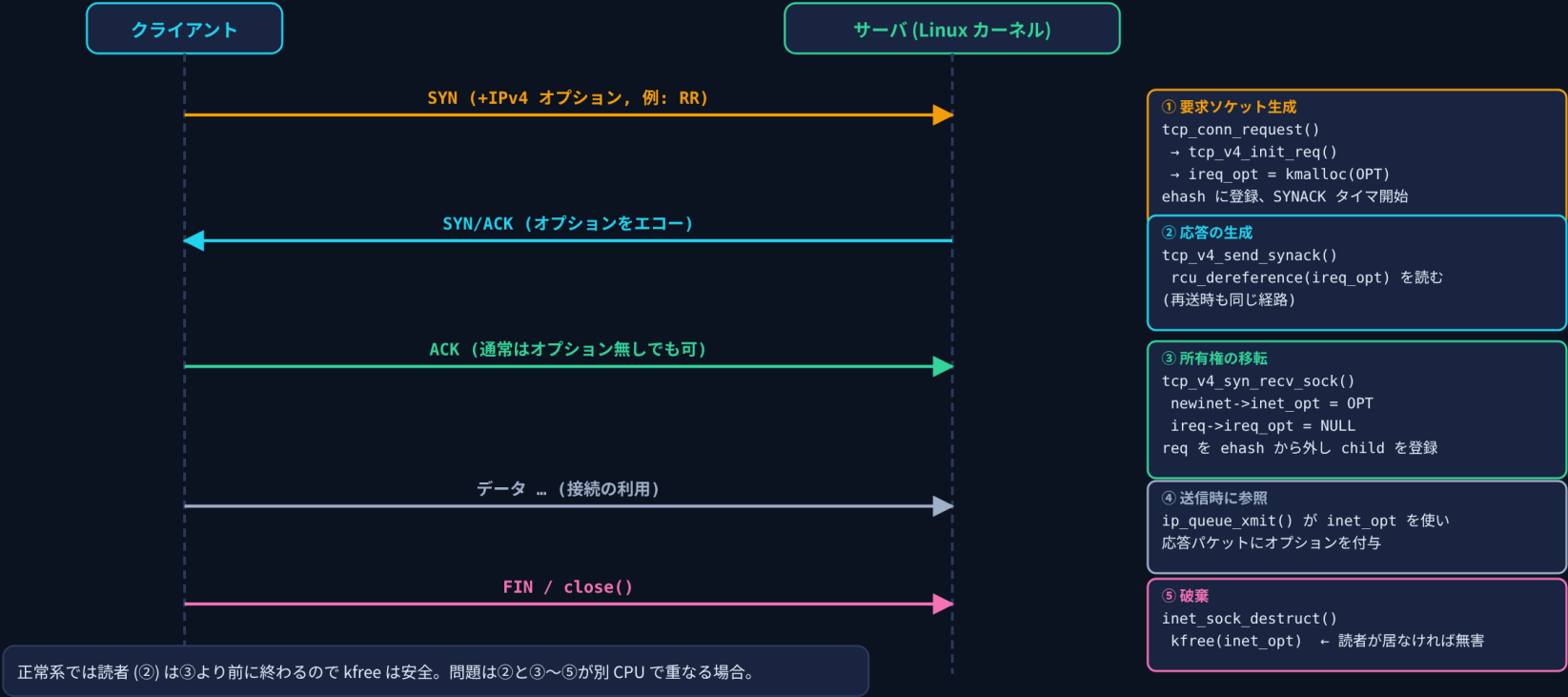


カーネル内部の構造体とリンク: 1 つの ip_options_rcu を誰が指すか



正常系のライフサイクル: SYN から close() まで、OPT はどう受け渡されるか

原理



要点

- OPT は『SYN にオプションがあった時だけ』①で生まれ、SYNACK の送信 (②) と再送のたびに読まれ、ACK 到着 (③) で child に所有権が移る。
- ③で req 側のポインタは NULL になるが、これは plain store。『同時に読んでいる他 CPU』を待つ仕組みは RCU の猶予期間だけ。
- ⑤の kfree() が猶予期間を待たないため、②の読者が③〜⑤と同時に走ると use-after-free になる — 次のスライド。
- 再送 SYN は攻撃者が任意に送れる (②を任意のタイミングで起こせる)。

マルチコアで壊れる瞬間: CPU 0 が読んでいる最中に CPU 1 が解放する

原理



RCU の約束と、それを破った 2 か所: 修正はなぜ 2 行で済むのか

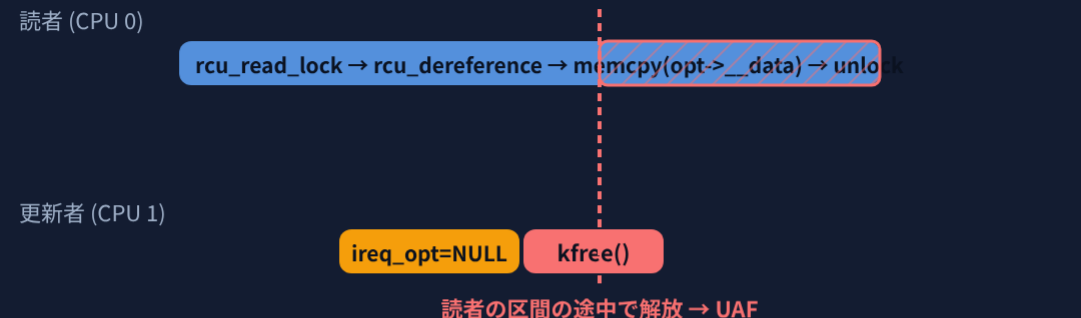
原理

RCU の約束 (正しい使い方): 読者は待たせない、解放は猶予期間の後



- 読者はロックを取らず rcu_read_lock() の区間でポインタを辿る (高速、CPU 間で待たない)。
- 更新者はポインタ付替え後、kfree_rcu()/call_rcu() で『全読者が区間を抜けるまで』解放を遅らせる。
- struct ip_options_rcu が rcu_head を持つのは、まさにこの約束を守るため。

CVE-2026-98432: 約束を破った 2 か所 (plain kfree)



- child を壊す inet_sock_destruct() と、req を壊す tcp_v4_reqsk_destructor() だけが kfree() だった。
- 『所有者が居ない = 読者も居ない』という仮定は、ehash 経由の読者やタイマには成立しない。
- 他の経路 (setsockopt IP_OPTIONS, CIPSO) は以前から kfree_rcu() を使っていた。

修正コミット e3f33b0a1d89 (Eric Dumazet, 2026-10) の差分 - 2 ファイル、各 1 行

```
net/ipv4/af_inet.c @@ inet_sock_destruct()
- kfree(rcu_dereference_protected(inet->inet_opt, 1));
+ kfree_rcu(rcu_dereference_protected(inet->inet_opt, 1), rcu);

net/ipv4/tcp_ipv4.c @@ tcp_v4_reqsk_destructor()
- kfree(rcu_dereference_protected(inet_rsk(req)->ireq_opt, 1));
+ kfree_rcu(rcu_dereference_protected(inet_rsk(req)->ireq_opt, 1), rcu);
```

kfree_rcu は『猶予期間の後に kfree』。読者は一切変更不要で、既存の rcu_head を使うだけ。

Fixes: 079096f103fa (4.4) / c905dee62232 (5.14)

Reported-by: Xinyang Ge (Anthropic)

ソース (1) 誕生: tcp_v4_save_options() はオプションが無ければ何も作らない

ソース

include/net/tcp.h — tcp_v4_save_options()

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

```
2640 static inline struct ip_options_rcu *tcp_v4_save_options(struct net *net, struct sk_buff *skb)
2642 {
2643     const struct ip_options *opt = &TCP_SKB_CB(skb)->header.h4.opt; /* IHL>5 の時のみ非空 */
2644     struct ip_options_rcu *dopt = NULL;
2646     if (opt->optlen) { /* 受信 SYN にオプションが有る時だけ */
2647         int opt_size = sizeof(*dopt) + opt->optlen;
2649         dopt = kmalloc(opt_size, GFP_ATOMIC); /* ★ 脆弱な対象物 (OPT) の誕生 */
2650         if (dopt && __ip_options_echo(net, &dopt->opt, skb, opt)) { /* 応答用に整形 */
2651             kfree(dopt); dopt = NULL;
2653         }
2654     }
2655     return dopt; /* オプション無しなら NULL のまま */
2656 }
```

フィルタが効く一次情報: optlen は受信 IPv4 ヘッダの IHL から決まる。IHL=5 なら 0 → kmalloc されず NULL。

誕生: rcu_head + ip_options + 生オプション分を kmalloc-64/96 から確保。これが競合で解放される OPT。

net/ipv4/tcp_ipv4.c — tcp_v4_init_req()

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

```
1629 static void tcp_v4_init_req(struct request_sock *req,
1630                             const struct sock *sk_listener, struct sk_buff *skb)
1632 {
1633     struct inet_request_sock *ireq = inet_rsk(req);
1634     struct net *net = sock_net(sk_listener);
1636     sk_rcv_saddr_set(req_to_sk(req), ip_hdr(skb)->daddr);
1637     sk_daddr_set(req_to_sk(req), ip_hdr(skb)->saddr);
1638     RCU_INIT_POINTER(ireq->ireq_opt, tcp_v4_save_options(net, skb)); /* ★ 所有者 1: req */
1639 }
```

所有者 1: RCU 保護ポインタとして要求ソケットに格納。この req は ehash に入り、他 CPU から検索できる (4.4 以降)。

呼出し順: SYN 受信 → tcp_conn_request() (tcp_input.c:7640) → af_ops->route_req = tcp_v4_route_req() → tcp_v4_init_req()

ソース (2) 移管: tcp_v4_syn_recv_sock() はコピーせずポインタを渡す

ソース

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

net/ipv4/tcp_ipv4.c — tcp_v4_syn_recv_sock() 抜粋

```
1732     newtp      = tcp_sk(newsk);
1733     newinet    = inet_sk(newsk);
1734     ireq       = inet_rsk(req);
1735     inet_opt   = rcu_dereference(ireq->ireq_opt);      /* OPT を読む */
1736     RCU_INIT_POINTER(newinet->inet_opt, inet_opt);    /* ★ ポインタを渡すだけ (複製しない) */
1741     if (inet_opt)
1742         inet_csk(newsk)->icsk_ext_hdr_len = inet_opt->opt.optlen;
...
/* ... 経路・MSS・MD5/AO の初期化 (省略) ... */
1789     *own_req = inet_ehash_nolisten(newsk, req_to_sk(req_unhash),
1790                                   &found_dup_sk);      /* ehash 上で req → child に置換 */
1791     if (likely(*own_req)) {
1792         tcp_move_syn(newtp, req);
1793         ireq->ireq_opt = NULL;      /* ★ 所有権は child へ (plain store) */
1794     } else {
1795         newinet->inet_opt = NULL;    /* 負けた側は何も所有しない */
1796     }
1806     return newsk;
```

1 つの OPT を 2 つの所有者が指す瞬間。
IPv6 版 (tcp_v6_syn_recv_sock) は
ipv6_dup_options() で複製するため非該当。

置換の後も、置換前に req を見つけていた
CPU (再送 SYN 処理中) は req 経由で OPT を
読み続ける。NULL 代入は他 CPU の読者を待たない。

修正コミットの説明: 『tcp_v4_syn_recv_sock() は ireq->ireq_opt をコピーせず child に渡す。並行する再送 SYN は RCU だけでそれを読み、解放後に使い得る』

IPv6 の tcp_v6_syn_recv_sock() は ipv6_dup_options() で複製してから child に持たせるため所有者が 1 つしか無く、本問題の影響を受けない (修正コミットも明記)。

ソース (3) 読者: 再送 SYN → tcp_check_req() → tcp_rtx_synack() → tcp_v4_send_synack()

ソース

net/ipv4/tcp_minisocks.c — tcp_check_req() 抜粋

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

```
727     /* 純粋な再送 SYN を再送信する */
728     if (TCP_SKB_CB(skb)->seq == tcp_rsk(req)->rcv_isn &&
729         flg == TCP_FLAG_SYN && !paws_reject) {
731         /* RFC 1122: SYN-RECV で SYN を再受信したら SYN+ACK を再送信する */
754         if (!tcp_oow_rate_limited(sock_net(sk), skb, LINUX_MIB_TCPACKSKIPPEDSYNRECV,
756             &tcp_rsk(req)->last_oow_ack_time)) {
765             if (!tcp_rtx_synack(sk, req)) { /* ★ 読者: SYNACK を作り直す */
...             /* ... 再送タイマの更新 (省略) ... */
777         }
778     }
779     return NULL;
780 }
```

攻撃者が任意に起こせる読者: 同じ seq の SYN をもう一度送るだけで、SYN-RECV 状態の req に対し SYNACK 再送 (= OPT の読み出し) が走る。

tcp_rtx_synack() (tcp_output.c:4636) → af_ops->send_synack = tcp_v4_send_synack() (tcp_ipv4.c:1182) へ。

net/ipv4/tcp_ipv4.c — tcp_v4_send_synack() 抜粋

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

```
1215     rcu_read_lock();
1216     err = ip_build_and_send_pkt(skb, sk, ireq->ir_loc_addr, ireq->ir_rmt_addr,
1218         rcu_dereference(ireq->ireq_opt), tos); /* ★ OPT の生ポインタ */
1220     rcu_read_unlock();
1221     err = net_xmit_eval(err);
```

読者は正しく RCU 区間に入っている。ここで得た生ポインタは区間が終わるまで有効であるべき — kfree_rcu なら守られる約束。

同じ読み出しは SYNACK 再送タイマ (inet_connection_sock.c:1027 reqsk_timer_handler → tcp_rtx_synack) と経路検索 (inet_csk_route_req:770) にもある。

ソース (4) 根源: inet_sock_destruct() と tcp_v4_reqsk_destructor() の kfree() — ここがパッチされた

ソース

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

net/ipv4/af_inet.c — inet_sock_destruct()

```
141 void inet_sock_destruct(struct sock *sk)
142 {
143     struct inet_sock *inet = inet_sk(sk);
144     __skb_queue_purge(&sk->sk_receive_queue);
145     __skb_queue_purge(&sk->sk_error_queue);
146     sk_mem_reclaim_final(sk);
147     ... /* ... 状態の健全性検査 (WARN_ON_ONCE 等、省略) ... */
165     kfree(rcu_dereference_protected(inet->inet_opt, 1)); /* ★ 根源 1: 即時解放 */
166     dst_release(rcu_dereference_protected(sk->sk_dst_cache, 1));
167     dst_release(rcu_dereference_protected(sk->sk_rx_dst, 1));
168 }
```

根源 1 (child 側): child の最後の参照が落ちると sk_destruct → ここへ。MPTCP の dispose_child 経路では ACK 処理と同じ softirq 内で到達する。

修正後:
kfree_rcu(rcu_dereference_protected(inet->inet_opt, 1), rcu); — 既存の rcu_head を使って解放を猶予期間後に遅延。

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

net/ipv4/tcp_ipv4.c — tcp_v4_reqsk_destructor()

```
1228 /* IPv4 要求ソケットのデストラクタ */
1229 static void tcp_v4_reqsk_destructor(struct request_sock *req)
1230 {
1231     ...
1232     kfree(rcu_dereference_protected(inet_rsk(req)->ireq_opt, 1)); /* ★ 根源 2: 即時解放 */
1233 }
```

根源 2 (req 側): 要求ソケットの最後の参照 (reqsk_put) で呼ばれる。移行で OPT を共有した nreq が落ちる時も同じ kfree。修正後は kfree_rcu(..., rcu)。

upstream e3f33b0a1d89 (Eric Dumazet) / stable 11a03ce3c4a6 — Fixes: 079096f103fa (4.4), c905dee62232 (5.14) / Reported-by: Xinyang Ge (Anthropic)

暫定対策はこの 2 行を変えられない。だから『解放される物を作らせない』(対策 a) か『解放を起こすレバーを外す』(対策 b/c) のどちらかになる。

ソース (5) 消費者: ip_build_and_send_pkt() / ip_options_build() が解放済みメモリをどう使うか

ソース

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

net/ipv4/ip_output.c — ip_build_and_send_pkt() 抜粋

```
160     /* IP ヘッダを組み立てる */
161     skb_push(skb, sizeof(struct iphdr) + (opt ? opt->opt.optlen : 0)); /* ★ 偽 optlen で溢れ */
168     iph->daddr = (opt && opt->opt.srr ? opt->opt.faddr : daddr); /* ★ 偽 srr → 宛先改変 */
    ...
    /* ... version / ihl / tos / ttl / id 等の設定 (省略) ... */
186     if (opt && opt->opt.optlen) {
187         iph->ihl += opt->opt.optlen>>2;
188         ip_options_build(skb, &opt->opt, daddr, rt); /* ★ 解放済み OPT を使用 */
189     }
```

DoS の芽: 再利用後の optlen が大きいと
skb_push がヘッドルームを超え
skb_under_panic() → カーネルパニック。

ここから下が『解放済みメモリの消費者』。

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

net/ipv4/ip_options.c — ip_options_build()

```
44 void ip_options_build(struct sk_buff *skb, struct ip_options *opt,
45     __be32 daddr, struct rtable *rt)
46 {
47     unsigned char *iph = skb_network_header(skb);
49     memcpy(&(IPCB(skb)->opt), opt, sizeof(struct ip_options));
50     memcpy(iph + sizeof(struct iphdr), opt->__data, opt->optlen); /* ★ 越境読み → 漏洩 */
51     opt = &(IPCB(skb)->opt);
53     if (opt->srr)
54         memcpy(iph + opt->srr + iph[opt->srr + 1] - 4, &daddr, 4); /* ★ 任意位置へ 4B 書込 */
56     if (opt->rr_needaddr)
57         ip_rt_get_source(iph + opt->rr + iph[opt->rr + 2] - 5, skb, rt); /* ★ 同上 */
    ...
    /* ... ts_needaddr / ts_needtime も同様に 4 バイトを書く (省略) ... */
66 }
```

情報漏洩の機構: 攻撃者制御の optlen 分だけ
隣接ヒープを SYNACK のオプション欄へコピ
ー → KASLR 突破に必要なカーネルポインタ
等が流出。

書込みの機構: srr と長さバイト iph[srr+1] は
共に攻撃者制御 → skb データ領域の外 (隣の
slab 物) に daddr (任意 4 バイト) を書ける。

注: これらの行自体は正しいコード。『opt が生きている』前提が破られたときだけ凶器になる。修正後は opt の寿命が読者区間を覆うので無害に戻る。

悪用の原理: 同じ 1 つの use-after-free から、情報漏洩 (KASLR 突破) と root 実行へ

原理

段階 1: 情報漏洩で KASLR を突破 (公開 PoC: 約 48 分)

kmalloc-64 スラブの 1 ページ (同サイズの物が隣合う)

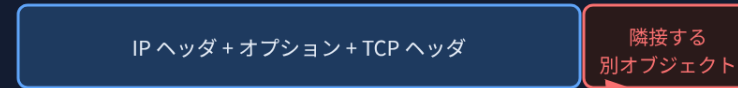


次の kmalloc-64 が同じスロットを再利用
(攻撃者が選んだ値で optlen が上書きされる)

- ① SYN(+IP オプション) → req と OPT を作らせる
- ② 『再送 SYN』と『最終 ACK(破棄経路)』を同時投入 → 競合
- ③ OPT が kfree → 別の確保要求で同スロットが再利用される
- ④ CPU 0 の memcpy(..., opt->__data, opt->optlen) が
偽の大きな optlen で隣のメモリまで読む (越境読み)
- ⑤ 読んだ内容は SYNACK の IP オプション欄に乗って攻撃者へ
- ⑥ その中にカーネルのポインタがあれば KASLR の基点が判明
負けても handshake が失敗するだけ (通常はパニックしない) → 延々と再試行できる

段階 2: 越境書込みで制御フローを乗っ取る (さらに約 5 分)

SYNACK 用 skb のデータ領域 (ip_options_build が書く場所)



iph + opt->srr + iph[opt->srr+1] - 4 に 4 バイト (daddr) を書く:
srr / rr / ts と長さバイトは攻撃者が置いた値 → 書込み先が外へ出る

- ① 同じ UAF で optlen / srr / rr / ts を攻撃者の値にする
- ② ip_options_build() の memcpy / 4 B 書込みが skb の外へ溢れる
- ③ 隣に置いた対象物のコールバック (関数ポインタ) を上書き
- ④ ポインタの先は、通常のプロトコル機能で事前に仕込んだ
ペイロード (段階 1 で得たアドレスを使う)
- ⑤ カーネルがそのコールバックを呼んだ瞬間に ROP/JOP 連鎖
- ⑥ 最終的に root 権限で任意コマンド実行 (ゼロクリック RCE)
Anthropic 公開情報の範囲で記載。悪用コード・対象カーネル版・構成は非公開。

結果として起こり得ること (同じ 1 つの UAF から)

DoS / クラッシュ

偽 optlen で skb_push が
ヘッドルームを超える →
skb_under_panic (カーネルパニック)
ihl の破損 → 不正パケット

機密情報の漏洩

ヒープ上の隣接データが
SYNACK に載って外部へ
(ポインタ、鍵、他接続の断片)
= KASLR 突破・認証情報の露出

権限昇格 / RCE

関数ポインタ上書きで
root 権限のコード実行
カーネル = ホスト全体の掌握
(コンテナ・VM 内全テナント)

公開 PoC の前提

IPv4 + IP オプション付き SYN
マルチコア
MPTCP 待受 (即時破棄経路)
到達可能な待受ポート

MPTCP が引き金になる理由: 攻撃者が『解放のタイミング』を握れる経路

原理



なぜ MPTCP なのか

脆弱性の本体は IPv4 TCP の `ireq_opt` 寿命管理で、MPTCP 固有ではない。しかし MPTCP には『ACK 受信直後に `child` を即時破棄する』経路が用意されている (`subflow_syn_rcv_sock` の `dispose_child`)。

攻撃者は HMAC を壊した `MP_JOIN` ACK を送るだけで、任意のタイミングで `kfree` を起こせる = 競合の片側を確実に制御できる。

MPTCP 待受が無いホストでも同じ UAF は存在する。別の破棄経路 (`listener close` 等) で成立する可能性は否定されていない (Go セキュリティチーム)。

Go 1.24 以降の `net.Listen` は既定で MPTCP 待受を作る → 多くの Go 製サーバが前提を満たす

ソース (6) レバー: subflow_syn_recv_sock() の dispose_child — MP_JOIN の ACK が不正なら即時破棄

ソース

net/mptcp/subflow.c — subflow_syn_recv_sock() 抜粋

Linux v7.0 — net/mptcp/subflow.c:862-966 (抜粋)

```
862     if (child && *own_req) {
...         /* ... ctx 取得、drop_req = false (省略) ... */
871     if (!ctx || fallback) {
872         if (fallback_is_fatal) {             /* MP_JOIN は fallback 不可 */
874             goto dispose_child;             /* ★ 即時破棄へ */
877         }
902     } else if (ctx->mp_join) {
909         if (!subflow_hmac_valid(subflow_req, &mp_opt)) { /* ★ 攻撃者が故意に壊せる */
910             SUBFLOW_REQ_INC_STATS(req, MPTCP_MIB_JOINACKMAC);
912             goto dispose_child;
913         }
...         /* ... (省略) ... */
955     return child;
957 dispose_child:
958     mptcp_subflow_drop_ctx(child);
959     tcp_rsk(req)->drop_req = true;
960     inet_csk_prepare_for_destroy_sock(child);
961     tcp_done(child);             /* ★ 即座に閉じる → 参照解放で kfree(OPT) */
963     req->rsk_ops->send_reset(sk, skb, reason);
966     return child;             /* 最後の参照は呼出し側が解放 */
```

仕様として正しい: MP_JOIN の ACK が検証を通らなければ、作ったばかりの child をその場で捨てる。

攻撃者の『レバー』: 故意に壊した HMAC を持つ ACK を送るだけで、望むタイミングで dispose_child を踏ませられる。

tcp_done(child) → 呼出し側の sock_put で refcnt 0 → inet_sock_destruct → kfree(OPT)。これが競合の片側。

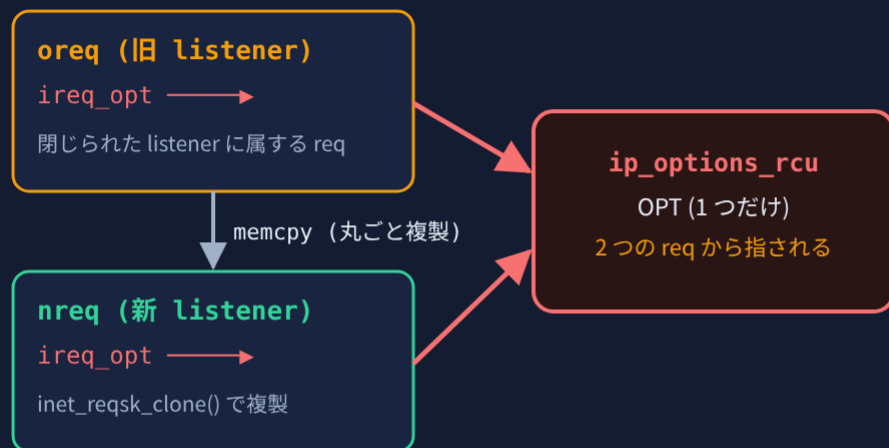
MPTCP 無効化 (net.mptcp.enabled=0 / Go の GODEBUG=multipathtcp=0) はこのレバーを外すだけ。根本の kfree は tcp_ipv4.c / af_inet.c 側に残る。

Go 1.24 以降の net.Listen は既定で MPTCP 待受を作る。GODEBUG=multipathtcp=0 / go.mod の godebug 行 //go:debug 指示で無効化できる (SetMultipathTCP 明示呼出しには効かない)。

もう 1 つの共有経路: 要求ソケット移行 (tcp_migrate_req) と inet_reqsk_clone() の memcpy

原理

要求ソケット移行 (tcp_migrate_req, Linux 5.14+) が作るもう 1 つの共有



reqsk_migrate_reset() が片方を NULL にして二重解放は防ぐが、NULL 化は plain store で、他 CPU の読者は古いポインタを持ち続ける。

その後 nreq 側が破棄されると tcp_v4_reqsk_destructor() の kfree() で OPT が消え、oreq を参照中の読者が UAF に陥る。

成立条件: SO_REUSEPORT のグループ内で listener を close() する運用 (graceful restart 等) + sysctl net.ipv4.tcp_migrate_req=1
既定値は 0 (無効)。有効化している環境では、暫定対策として 0 に戻すことで『この共有経路』を閉じられる。
修正コミットはこの経路も Fixes: c905dee62232 として明示している (『SYN+ACK 再送時の移行』を導入したコミット)。

確認と設定

```
# 現在値を確認 (netns ごと)
sysctl net.ipv4.tcp_migrate_req
net.ipv4.tcp_migrate_req = 0 # ← 既定

# 1 なら暫定的に 0 へ
sudo sysctl -w net.ipv4.tcp_migrate_req=0

# MPTCP ソケットの作成を禁止 (対策 b)
sudo sysctl -w net.mptcp.enabled=0

# 永続化 (/etc/sysctl.d/99-cve-2026-98432.conf)
net.mptcp.enabled = 0
net.ipv4.tcp_migrate_req = 0

# 既存の MPTCP 待受を確認 (iproute2)
ss -Mlnp # MPTCP 待受ソケット一覧
```

ソースの要点 (net/ipv4/inet_connection_sock.c)

918-958 行 inet_reqsk_clone(): req を obj_size 分 memcpy → ireq_opt のポインタも複製される。968-977 行 reqsk_migrate_reset(): 片方を NULL にする (CONFIG_IPV6 有効時は union 経由)。1027 行～ reqsk_timer_handler(): SYNACK 再送時に listener が閉じていれば移行する。

運用上の判断

net.ipv4.tcp_migrate_req は既定 0 (無効)。有効化している環境 (SO_REUSEPORT + graceful restart の運用) では、暫定的に 0 へ戻すことでこの共有経路を閉じられる。無効化しても MPTCP 経路とは独立なので、対策 a/b/c と併用する。

対策 a の適用手順: 即時適用 → 確認 → 永続化 → 再起動 → 監視

即実行

1 即時適用

nftables: `sudo nft -f /etc/nftables.d/cve-2026-98432.nft`
iptables: 前掲の 1 行 (raw テーブル、PREROUTING の先頭)。混在環境は `nft list ruleset` で全体を確認

2 動作確認

`sudo nft list table inet cve_2026_98432` (counter の増加) / `sudo iptables -t raw -L PREROUTING -v -n`
検証: 別ホストから IP オプション付き SYN を送る (例: `nping --ip-options R, hping3, scapy`)
受信側で `tcpdump -ni eth0 'ip[0] & 0x0f > 5'` を観察し、SYNACK が返らないことを確認

3 永続化

nftables: `/etc/nftables.conf` に `include` を追加し `systemctl enable nftables`
iptables: `iptables-persistent / netfilter-persistent` (`iptables-save > /etc/iptables/rules.v4`)
RHEL 系: `nftables.service`。firewalld 利用時は独自 table との共存を確認
ネットワーク有効化より前にルールが入る順序 (ユニット依存関係) を必ず確認

4 再起動

永続化を確認してからホストを再起動し、適用前に生まれた残存オブジェクトを消す (次スライド)
再起動できない場合は、影響を受け得る待受サービスの再起動で代替

5 監視・記録

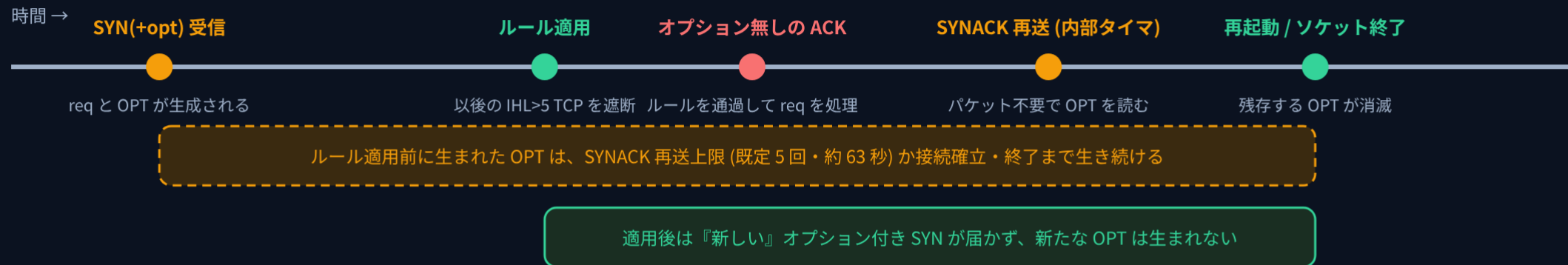
counter の推移を監視 (急増は攻撃試行の兆候、少量の一致は正当な通信の可能性 → 送信元を確認)
変更記録に『CVE-2026-98432 暫定対策、カーネル更新後に撤去可』と明記

ルール配置の原則

- 『信頼できない通信が対象の TCP 処理に到達する前』に有効であること。起動直後にネットワークが上がってからルールが入る構成だと、その隙間が無防備になる。
- PREROUTING の先頭 (iptables: `-I 1`、nft: `priority -310`) に置く理由: 他のルール (ACCEPT や NAT) に先回りされないため。
- 既存の FW がすべての受信を ACCEPT していても、このルールは独立して効く (filter より前の段階で捨てる)。
- ネットワーク名前空間ごとに独立。コンテナ内・VM 内は別途 (スライド 32)。
- 撤去: 修正済みカーネルで再起動した後、ルールを削除してよい (残しても害は少ない)。

なぜ再起動を推奨するのか: フィルタは『これから届くパケット』にしか効かない

即実行



- ・パケットフィルタは『これから通るパケット』にしか作用しない。既に ireq_opt を持つ要求ソケットや、オプション付きで確立した接続 (inet_opt) は残る。
- ・残存した OPT に対しても、再送 SYN (オプション付き) はルールで落ちるため読者経路の 1 つは断たれるが、カーネル内部のタイマ経路や別の破棄経路の組合せを完全には否定できない。確実に初期状態へ戻す保守的な方法は『ルールを永続化してから再起動』(起動時にルールが先に入る構成にしておく)。
- ・再起動できない場合: 影響を受け得る待受サービスの再起動 (要求ソケット・接続の作り直し) と、nft counter / ss による残存状況の観測で代替する。

確実な方法

- ・ルールを永続化 → 起動時にネットワーク有効化より前に入ることを確認 → ホスト再起動。これで『適用前に生まれた OPT』は存在しなくなる。
- ・ルータで集約保護している場合も、保護対象ホスト側の残存状態はルータ再起動では消えない点に注意。

再起動が難しい場合の代替

- ・対象の待受サービス (特に MPTCP 待受、公開ポートのサーバ) を再起動し、要求ソケットと接続を作り直す。SYNACK 再送上限 (既定 5 回、約 63 秒) を過ぎれば残存する要求ソケットは消える。
- ・確立済み接続の inet_opt は接続終了まで残るが、その解放は『読者の居ない』close 時であり、攻撃者が新たに読者経路を作ることはフィルタで阻止される。

対策 b / c: MPTCP と要求ソケット移行を無効化する (公開 PoC の前提を外す)

即実行

対策 b: `sysctl` (ネットワーク名前空間ごとに設定。コンテナ・`netns` を使う場合はそれぞれで実施)

```
sudo sysctl -w net.mptcp.enabled=0          # MPTCP ソケットの作成を禁止 (5.6 以降に存在)
sudo sysctl -w net.ipv4.tcp_migrate_req=0    # 要求ソケット移行を無効化 (5.14 以降。既定 0 なので確認が主)
printf 'net.mptcp.enabled = 0¥nnet.ipv4.tcp_migrate_req = 0¥n' | sudo tee /etc/sysctl.d/99-cve-2026-98432.conf # 永続化
ss -Mlnp                                     # 既存の MPTCP 待受ソケットを列挙 (iproute2)。該当サービスは再起動して待受を作り直す
```

対策 c: アプリケーション側で MPTCP 待受を作らせない (`sysctl` に触れない環境向け)

```
# Go 1.24 以降 (既定で net.Listen が MPTCP 待受) - いずれか 1 つ
GODEBUG=multipathtcp=0 ./server             # 既存バイナリの変数で無効化 (再ビルド不要)
godebug multipathtcp=0                       # go.mod (main モジュール) に記述して再ビルド
//go:debug multipathtcp=0                    # package main の直前に記述して再ビルド
# 注意: コードが ListenConfig.SetMultipathTCP(true) を明示していると上記は効かない
```

効果と限界

- 公開された RCE の前提 (MPTCP 待受 + 即時破棄のレバー) を外す。移行経路の共有も閉じる。
- 根本の `kfree` は残る。別の破棄経路での発動可能性は未確認 (否定もされていない)。

適用時の注意

- MPTCP を実際に使っている通信 (マルチパス経路の冗長化、一部のモバイル向け API) は単一パス TCP に戻る。
- `sysctl` は `netns` ごと。Docker/K8s では各 Pod/コンテナの `netns` で別途設定するか、イメージ側 (対策 c) で対応。
- 古いカーネルには `net.mptcp.enabled` が無い (= MPTCP 自体が無い) 場合がある。

確認コマンド

```
sysctl net.mptcp.enabled
net.ipv4.tcp_migrate_req
ss -Mlnp / nstat -az | grep -i mptcp
go version -m ./server | grep -i godebug
(Go バイナリの設定確認)
対策 a の counter と合わせて、攻撃試行の有無を観測する。
```

副作用の評価: IPv4 オプションを使う TCP 通信は実在するか (RFC 7126 の推奨と照らす)

即実行

IPv4 オプション	用途	RFC 7126 (BCP 186) の推奨	本ルールによる影響と判断
LSRR (131) / SSRR (137) 送信元経路指定	送信元が経路を指定。歴史的にスプーフィング・ACL 迂回に悪用	既定で破棄すべき	遮断して問題なし (多くの機器が既に破棄)
RR (7) 経路記録 / TS (68) タイムスタンプ	経路診断 (ping -R / -T)。TCP で使う実運用はほぼ無い	既定で破棄してよい	遮断しても実害はほぼ無い (ping は ICMP なので対象外)
Router Alert (148)	RSVP、IGMP、PGM 等が中継ルータへの処理要求に使用	必要な場合のみ許可	TCP での使用は稀。RSVP-TE 等は TCP ではないため対象外
CIPSO (134) / 汎用セキュリティ	MLS 環境 (SELinux + NetLabel、Trusted Solaris 等) のラベル付け	閉域網内でのみ	CIPSO を TCP で使う閉域網では本ルールが通信を止める。かつカーネル内部生成経路があるため対策 a だけでは不十分 → 当該ホストは優先的にカーネル更新
Quick-Start / 実験的 (25, 30, 94 等)	実験・研究用	破棄してよい	遮断して問題なし

判断の仕方

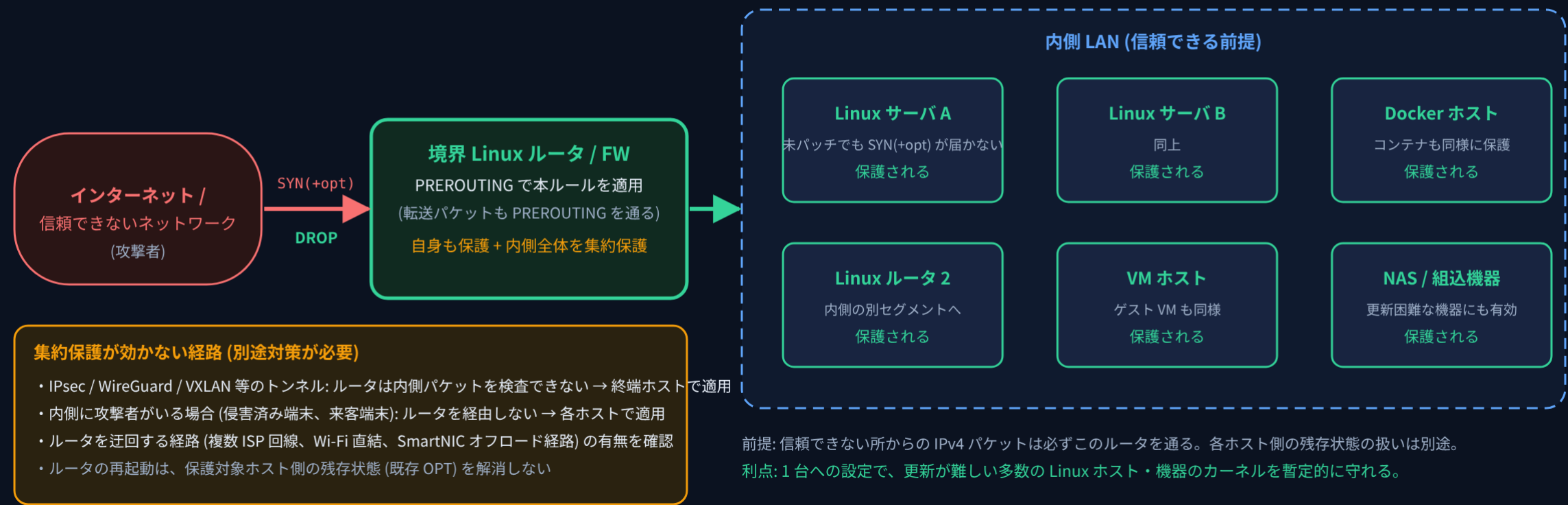
- ルール適用前に `tcpdump -ni eth0 -c 1000 'ip[0] & 0x0f > 5 and tcp'` で、IP オプション付き TCP が実際に流れているか観測する (多くの環境で 0 件)。
- 適用後は counter の一致数と送信元を確認。正当な通信が含まれる場合は、送信元アドレスを限定した ACCEPT をこのルールの前に置くか (露出が増えるので非推奨)、当該ホストを先にカーネル更新する。
- インターネット境界では RFC 7126 の推奨どおり、IP オプション付きパケットを全面的に破棄する運用が以前から一般的。

結論

- 通常のサーバ・コンテナホスト・ルータでは、IPv4 オプションを用いる TCP 通信は事実上存在しないため、本ルールの副作用は極めて限定的。
- 仮に一部の通信が影響を受けても、カーネル更新までの間 CVE-2026-98432 から保護される利益が、オプション付き TCP が一時的に疎通しない不利益を上回る場合が多い。
- 必要な通信の有無と重要度を確認して判断する (観測 → 適用 → 監視)。

適用場所 (1): 境界ルータで集約保護する — 1 台の設定で内側の全ホストを暫定保護

即実行

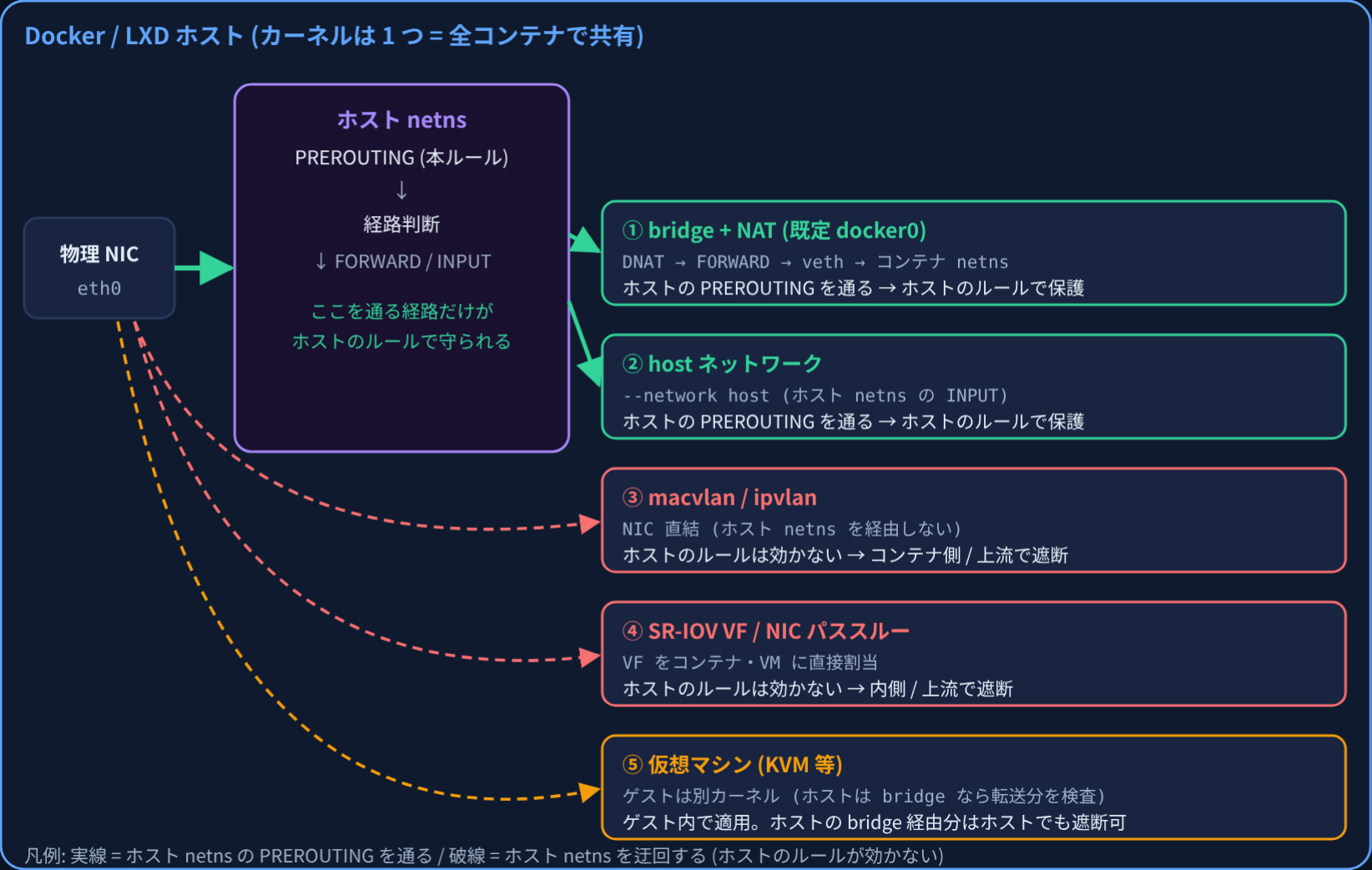


成立条件のまとめ

(1) 信頼できない所からの IPv4 パケットが必ずこのルータの PREROUTING を通る (2) ルータ自身の Linux カーネルでルールが正しく評価される (3) トンネル (IPsec / WireGuard / VXLAN 等) の内側パケットは検査できないため終端ホストで別途適用 (4) 内側に攻撃者が居る前提なら各ホストでも適用 (5) 保護対象ホスト側の残存状態 (適用前に生まれた OPT) はルータの再起動では消えない

適用場所 (2): コンテナ (Docker / LXD / Kubernetes) と仮想マシン — ホストのルールが届く経路・届かない経路

即実行



コンテナ環境での要点

コンテナは独立 OS ではない。
待受ソケットはコンテナ内でも
処理するのはホストカーネル 1 つ。
コンテナの待受経由で発動すると
ホスト全体 (全コンテナ) が侵害
され得る。

ホスト側ルールは ①② を覆う。
DNAT (-100) より前 (-310/-300)
で落ちるので、公開ポートへの
SYN(+opt) はコンテナに届かない。

③④ はホスト netns を通らない。
コンテナ netns 内に同じルールを
入れる (nft は netns 単位)、
または上流の L3 機器で遮断する。

Kubernetes: CNI (Calico/Cilium
/eBPF 経路) により通過フックが
異なる。eBPF でパケットを早期に
リダイレクトする構成では
Netfilter を通らないことがある →
ノード毎に実経路を確認。

第 3 部 パブリッククラウドと責任分界 — 自分で直せる層、事業者を待つしかない層

『クラウドを使っていればセキュリティは万全』という前提は、カーネルレベルの脆弱性には当てはまらない。どの層の Linux カーネルを誰が、いつ直し、誰が確認できるのかを整理する。

層構造で見る CVE-2026-98432: 自分で対処できる層と、事業者の対応を待つしかない層

クラウド

パブリッククラウドの層構造と「誰が CVE-2026-98432 を直せるか」

アプリケーション / データ あなたのコード・SaaS のアプリ
コンテナ / サンドボックス (gVisor, Firecracker 等) 共有カーネル or 専用 microVM。設計で分離度が大きく異なる
ゲスト VM の Linux カーネル IaaS: あなたが管理 / PaaS・SaaS: 事業者が管理
ハイパバイザ (ホストカーネル) AWS: KVM 派生 (ネットワークスタック無し) / Azure: Hyper-V (Windows) / GCP: KVM (Linux)
SmartNIC / オフロードカード Nitro Card (組込み Linux の示唆あり) / Azure Boost / Google IPU: 事業者のみ
共有サービス面 (LB, API, コンソール, IAM, ストレージ FE) 事業者の Linux フリートで稼働と合理的に推定される部分が多い

IaaS	PaaS	SaaS
自分で対処可	自分で対処可	事業者のみ
自分で対処可	事業者のみ	事業者のみ
自分で対処可	事業者のみ	事業者のみ
事業者のみ	事業者のみ	事業者のみ
事業者のみ	事業者のみ	事業者のみ
事業者のみ	事業者のみ	事業者のみ

緑 = 自分の手で本資料の対策 (フィルタ/設定/カーネル更新) を即日実施し、結果を自分で検証できる層。
赤 = 事業者の対応完了を待つしかない層。ここに残る脆弱性は、その上で動く全てのテナントのデータ・サービスに影響し得る。
ハイパバイザ層の直撃には『待受 TCP ソケットに SYN(+opt) が届く』ことが必要。AWS Nitro はホスト側に TCP スタックが無いと公表しており、Azure ホストは Windows ベースだが、周辺の Linux 部分 (カード、LB、PaaS ノード、管理面) の状況は顧客から見えない点は各社共通。

IaaS (ベアメタル / EC2 等) の結論: ゲストは自分で即日対処できるが、その下は見えない

クラウド

自分でできること (ゲスト VM / ベアメタル)

- 対策 a / b / c をゲストカーネルに即日適用し、counter や ss で結果を自分で検証できる。
- 修正済みカーネル (ディストリの修正版、または Amazon Linux 2023 のように既に修正済みのもの) へ自分の日程で更新できる。
- ベアメタルではハイパバイザ層が存在せず、自分のカーネルがそのまま最下層になる (SmartNIC を除く)。

自分では見えない・触れないもの

- ハイパバイザ (ホスト) のカーネル、オフロードカードのファームウェア、ハイパバイザの管理ネットワーク、共有サービス面 (LB / API / コンソール / IAM / ストレージのフロントエンド)。
- これらに本脆弱性が残っていれば、ゲストでいくら対策しても、同じ物理ホスト上の全 VM と、そのサービスを使う全顧客に影響し得る。
- **顧客が原理的に対策できない領域が、今回のようなカーネル脆弱性ではかなり広く存在する。**

ゼロトラストの前提で考える

- ハイパバイザの管理ネットワークには『既に攻撃パケットを発し得る攻撃者が居る』と仮定する (ゼロトラスト)。
- その仮定では、管理網に露出した未修正の Linux (ハイパバイザ、カード、管理面) は乗っ取りの対象になり得る。
- したがって事業者には、該当コードの有無と修正完了を顧客に開示する透明性が求められ、顧客はそれを確認する合理的な理由を持つ。

事業者への確認事項 (文例) — 完了報告が得られるまでは『安全』と判定しない

- Q1. 私の VM が稼働しているハイパバイザ (ホスト) の Linux カーネルは、Linux 4.4 以降で導入されたコミット 079096f103fa のコード (要求ソケットを ehash に登録する実装) を含んでいますか。含まない場合、その根拠 (非 Linux、該当経路が存在しない等) を示してください。
- Q2. 含む場合、CVE-2026-98432 の修正 (upstream e3f33b0a1d89 相当) は、私の VM を収容する全ホスト、オフロードカード、および私が利用する管理面・共有サービス面に適用済みですか。完了日時と適用範囲を示してください。
- 回答が『該当しない』であっても、その根拠 (例: ホストにネットワークスタックが存在しない) を確認する。完了報告書 (日付・範囲・担当) を入手し、変更記録に添付する。

法令・契約の観点

- 日本の個人情報保護法が求める安全管理措置 (委託先の監督を含む) や各種ガイドラインの観点から、委託先であるクラウド事業者の脆弱性対応状況を確認することは、顧客側の合理的な注意義務の一部と位置づけられる。
- 具体的な法的義務の範囲は法令・契約・業界ガイドラインに依存する。契約上の脆弱性通知義務・報告期限の有無を確認する。
- 本資料は法的助言ではない。

PaaS / SaaS の結論: 管理者が対策できる層が無い — 事業者の表明まで『直撃と波及』のリスクが残る

クラウド

直撃のリスク

- PaaS / SaaS では、顧客のコードやデータを処理する Linux カーネルは事業者が管理する。パケットフィルタも sysctl もカーネル更新も、顧客は触れない。
- 事業者が修正を完了するまでの間、そのカーネルで待ち受ける公開サービス (API、Web フロント、LB) は、本脆弱性による直撃を受け得る。
- 顧客にできる『技術的対策』は原理的に存在しない。できるのは、置くデータと使い方を見直すことだけ。

波及のリスク (マルチテナント)

- カーネルが乗っ取られると、そのカーネル上で動く全テナントのプロセス・メモリ・ストレージ接続が攻撃者の手に落ちる。共有カーネル型のコンテナ基盤では被害範囲 = ノード全体。
- 分離の設計により範囲は異なる: microVM (Firecracker)、gVisor (ユーザ空間 TCP スタックで、ホストカーネルの該当経路を使わない)、VM 単位分離は被害を限定し得る。
- ただし、どの設計が自分のテナントに使われ、その基盤カーネルが修正済みかは、事業者の表明がなければ顧客は知り得ない。

確実性のある実効策 (事業者が『基盤の Linux カーネルについて CVE-2026-98432 の対策を完了した』と責任をもって表明するまで)

- ① 機密性・完全性・可用性を要するデータを、当該 PaaS / SaaS およびそこからアクセス可能な場所に新たに保管しない。既存データは暗号化・アクセス制限・退避を検討する。
- ② 当該サービスから自組織の他システムへ到達できる認証情報 (API キー、サービスアカウント、VPN) の範囲を最小化し、横展開の経路を絞る。
- ③ 事業者に対し、対策完了の表明 (日付・範囲) と、未完了の間の補償的統制 (監視強化、ネットワーク分離) の説明を求める。契約上の報告義務を確認する。
- ④ 代替手段 (自前基盤、修正済み IaaS 上での自前運用) への一時的な切替可否を評価する。
- これは過剰反応ではなく、『顧客側に対策手段が無い』場合に残された唯一の確実な方法である。表明が得られ次第、通常運用に戻す。

主要事業者の基盤に関する技術的推定（公開情報に基づく。事業者の正式表明が優先する）

クラウド

観点	AWS	Microsoft Azure	Google Cloud
ハイパバイザ (ホスト OS)	Nitro Hypervisor: AWS は『KVM をベースに最小化したカスタムハイパバイザ』と公表 → Linux カーネル派生。ただし『ネットワークスタックを持たず EC2 ネットワークへのアクセスが無い』『シェル無し』と明言 → SYN を受け付ける待受経路は無いと考えられる (AWS の説明が正確である限り)。	Azure Host OS『Cloud Host』: Windows (OneCore) ベースの専用 SKU + Hyper-V (Type-1) → Linux ではないため CVE-2026-98432 の直接の対象外。	KVM (Linux カーネル機能) + Google 独自のユーザ空間 VMM (QEMU 不使用、2017 年公表) → ホストは Linux カーネル。ホスト側の待受経路の有無は非公開。
オフロードカード / SmartNIC	Nitro Card: Annapurna Labs 設計 SoC + AWS 専用ファームウェア。AWS 求人情報は『Nitro システムの中核にある組込み Linux 環境』に言及 → Linux ベースと推定。VPC データ面はカード側で処理し、カード自身の TCP 待受に顧客パケットは届かない設計と推定されるが、管理網側の露出は非公開 → 該当コードの有無・修正状況の確認が必要。	Azure Boost: FPGA + 専用 SoC (OS は非公開)。OpenHCL パラバイザは各 VM 内 (VTL2) に『最小 Linux カーネル + Rust VMM』を含むが、VM 単位で動きテナント間共有ではない。	Google 製 NIC / IPU、Andromeda (ホスト上のユーザ空間データ面)。詳細は非公開。
共有サービス面 (LB / API / コンソール / IAM / ストレージ FE)	ALB/CLB は EC2 インスタンス上、NLB は Hyperplane。S3 / IAM / コンソール等のフロントエンドは Linux フリートで稼働と合理的に推定。未修正ならハイパバイザが無事でも管理面経由で波及し得る (技術的には『あり得る』)。	Azure Front Door / Load Balancer (Ananta 系 SDN) の実装 OS は非公開。多数の第一者サービスが Azure Linux (CBL-Mariner) 等の Linux 上で稼働していることは公表済み。	GFE (Google Front End) 等のフロントエンドは Linux (Borg) 上で TCP を終端。Maglev (L4 LB) はカーネルバイパス転送のため、転送経路自体は本 CVE の対象外だが、終端側は Linux。
サーバレス / PaaS 基盤	Lambda / Fargate: Firecracker (KVM ベース) microVM を Linux ホスト上で稼働。AL2023 のゲスト側は修正版あり。	App Service (Linux) / AKS / Container Apps: Linux VM。ホストは Hyper-V。	Cloud Run / App Engine: gVisor (ユーザ空間 TCP スタック) → アプリ側の TCP (は該当経路を使わない設計。GKE ノードは Linux VM)。
顧客が確認できること	AL2023 等ゲスト側の修正版の有無。基盤側は事業者の表明のみ。	Azure の Linux PaaS ノード / ゲストの修正状況。基盤側は事業者の表明のみ。	COS / GKE ノードイメージの修正状況。基盤側は事業者の表明のみ。

よくある問い: 『Azure は Windows ベースだから安全か』 『Nitro は Linux だから危険か』

クラウド

Azure のハイパバイザは本 CVE の影響を受けないか

- 結論: ホスト OS (Cloud Host = Windows) と Hyper-V 自体は Linux ではないため、CVE-2026-98432 の直接の対象ではない。
- 『ハイパバイザ層がこの CVE に該当するか』だけを切り出せば、Linux (KVM) 派生の AWS / GCP より該当性は低い。
- ただし『だから Azure の方が総合的に安全』とは言えない: (1) Windows には Windows の脆弱性がある (2) OpenHCL 内の最小 Linux、Azure Boost、Linux PaaS ノード、内部サービスなど Linux 層は存在する (3) それらの修正状況は顧客から見えない点は各社共通。

AWS Nitro は Linux 派生だから危険か

- 結論: Nitro Hypervisor は KVM ベース (Linux 派生) だが、AWS は『ネットワークスタック無し・EC2 ネットワークへのアクセス無し』と公表しており、本 CVE の発動に必要な『待受 TCP ソケットに SYN が届く』経路はハイパバイザ上に存在しないと考えられる。
- 一方、Nitro Card (組込み Linux の示唆あり)、Firecracker ホスト、共有サービス面は Linux であり、管理網側の露出や修正状況は顧客には見えない。
- 『Linux 派生 = 直撃可能』でも『Windows = 無関係』でもない。問うべきは『待受経路の有無』と『修正完了の確認可能性』。

正しい問いの立て方 — 事業者横断で同じ基準を当てる

- ① 自分が制御できない Linux カーネルはどこにあるか (ハイパバイザ / オフロードカード / 管理面 / LB / PaaS 基盤 / 内部サービス)。
- ② それぞれに『IPv4 TCP の待受ソケットへ、信頼できない所から SYN が届く経路』があるか (無ければ本 CVE の直撃は成立しない)。
- ③ 経路がある層について、該当コード (4.4 以降) の有無と修正完了を、事業者が日付と範囲を示して表明しているか。
- ④ 表明が無い層が自組織の機密データ・重要サービスに触れているなら、PaaS/SaaS の結論 (前々スライド) と同じ補償的統制を適用する。
- CVE 1 件の該当性で事業者の優劣を決めるのではなく、『確認できるか』を基準にする。確認できない層の広さそのものが、パブリッククラウド固有のリスクである。

教訓：『直ちに全層を直さなければならない』脆弱性と、自前基盤 (オンプレミス / プライベートクラウド) の利点

クラウド



示唆: 『直ちに全層を直さなければならない』種類の脆弱性では、全ての Linux カーネル (ゲストだけでなくハイパバイザ・コンテナ基盤・管理面) を自分の手で即日修正し検証できることが、自前オンプレミス / プライベートクラウドの明確な安全上の利点になる (運用負荷という代償はある)。

自前基盤の安全上の利点

- ゲストだけでなく、ハイパバイザ・コンテナ基盤・ルータ・管理面を含む全ての Linux カーネルを、自分の手で即日 (フィルタ) / 数日 (更新) で修正し、結果を自分で検証できる。

パブリッククラウドの利点も正しく評価する

- 専門チームによる運用、多層防御、迅速な基盤更新 (事業者によっては顧客より早い) は現実の利点。問題は『速いか』ではなく『顧客から確認できない層が広い』こと。

回復すべき心がけ

- 『クラウドだから万全』という過信を捨て、自組織のデータがどのカーネルの上を流れ、誰がいつ直せるのかを常に把握しておく。重要度の高いシステムほど、自分で直せる範囲を確保する。

対策チェックリスト: 今日 / 今週 / 恒久

即実行

今日 (暫定対策)

- 対策 a を全 Linux ホスト (サーバ、コンテナホスト、VM、ルータ) に適用し、counter / pkts で動作を確認
- 境界ルータにも対策 a を適用 (集約保護)、トンネルの終端ホストは個別に適用
- 対策 b (sysctl) と対策 c (Go の GODEBUG 等) を適用し、既存の MPTCP 待受を作り直す
- コンテナ: macvlan / ipvlan / SR-IOV / eBPF 経路の有無を確認し、必要なら netns 内に同じルール
- ルールを永続化し、起動順序 (ネットワーク有効化より前) を確認
- 計画再起動 (残存状態の解消)。できない場合は待受サービスの再起動
- tcpdump / counter で IP オプション付き TCP の有無と送信元を観測・記録

今週 (確認と調整)

- 各ディストリの CVE ページで修正版の公開状況を毎日確認 (Ubuntu / RHEL は 10/30 時点で未公開)
- Go 1.24+ 製サーバの棚卸し (go version -m) と、サードパーティ製品の MPTCP 利用有無の確認
- NetLabel / CIPSO を使う MLS 構成の有無を確認 (該当ホストは最優先でカーネル更新)
- クラウド事業者・ホスティング事業者へ Q1 / Q2 を送付し、完了報告書を入手
- PaaS / SaaS に置く機密データの棚卸しと、事業者表明までの補償的統制の決定
- 監視: counter 急増、SYNACK 再送の異常、カーネル警告 (skb_under_panic 等) をアラート化

恒久 (根本対策と体制)

- 修正済みカーネル (安定版または各ディストリの修正版) へ更新し再起動。暫定ルールは撤去可
- RFC 7126 に沿い、境界での IP オプション付きパケット破棄を標準設定として維持
- 更新困難な Linux 機器 (NAS、アプライアンス、組込) の台帳化と、ルータ集約保護の恒常化
- 自前基盤 (オンプレ / プライベートクラウド) で『全層を自分で直せる範囲』を見直す
- 事業者との契約に脆弱性通知・対応期限・完了報告の条項があるか確認
- 本件を題材に、カーネル脆弱性時の初動手順 (フィルタ → 無効化 → 更新 → 確認) を手順書化

参考文献（一次情報を優先。URL は 2026-10-30 時点）

[**CVE**] Linux Kernel CVE Team, CVE-2026-98432: ipv4: free inet_opt and ireq_opt after an RCU grace period, 2026-10-15. <https://lists.openwall.net/linux-cve-announce/2026/10/15/1>

[**FIX**] E. Dumazet, ipv4: free inet_opt and ireq_opt after an RCU grace period, commit e3f33b0a1d89d738e2b913dab519226eaf76e15b (7.3-rc7). <https://github.com/torvalds/linux/commit/e3f33b0a1d89d738e2b913dab519226eaf76e15b>

[**SRC**] Linux v7.0 ソース: net/ipv4/tcp_ipv4.c, net/ipv4/af_inet.c, net/ipv4/tcp_minisocks.c, net/ipv4/inet_connection_sock.c, net/ipv4/ip_output.c, net/ipv4/ip_options.c, net/ipv4/ip_input.c, net/mptcp/subflow.c, net/netfilter/xt_u32.c, include/net/tcp.h, include/net/inet_sock.h (github.com/torvalds/linux, tag v7.0)

[**ANTHROPIC**] Anthropic Frontier Red Team, Discovering and exploiting vulnerabilities in critical, real-world software with Claude, 第 5 節, 2026-10-27. <https://www.anthropic.com/research/discovering-vulnerabilities-with-claude>

[**GO-SEC**] golang/go Issue 82717, proposal: net: Disable MPTCP by default (2026-10-29, neild 他のコメント). <https://github.com/golang/go/issues/82717>

[**NEWS**] The Hacker News, Linux Kernel TCP Flaw With 9.8 CVSS Score Affects Ubuntu, Red Hat and Other Distributions (2026-10). Debian Security Tracker CVE-2026-98432. pwn.watch, linux-kernel-ipv4-tcp-uaf-cve-2026-98432.

[**UBUNTU**] Canonical Kernel Team, 26.04 LTS (Resolute Raccoon) shipping with the final 7.0 Linux kernel, 2026-04-23 (discourse.ubuntu.com)

[**NFT/IPT**] Netfilter Project: nft(8) マニュアル (netfilter.org); iptables-extensions(8) u32 節; xtables-nft(8); Netfilter hooks (wiki.nftables.org)

[**RFC7126**] F. Gont, R. Atkinson, C. Pignataro, Recommendations on Filtering of IPv4 Packets Containing IPv4 Options, BCP 186 / RFC 7126, 2014-02

[**KDOC**] Linux カーネル文書: Documentation/networking/ip-sysctl.rst (tcp_migrate_req), mptcp-sysctl.rst (enabled), networking/bridge.html; net/netlabel/netlabel_kapi.c (netlbl_req_setattr), net/ipv4/cipso_ipv4.c (cipso_v4_req_setattr)

[**AWS**] AWS Whitepaper, The Security Design of the AWS Nitro System (The Nitro System journey / The components of the Nitro System); Amazon 求人情報 Kernel/OS Engineer, AWS EC2 Nitro Operating System

[**AZURE**] Microsoft Tech Community, Azure Host OS – Cloud Host; OpenHCL: Evolving Azure's virtualization model (Windows OS Platform Blog)

[**GCP**] Google Cloud Blog, 7 ways we harden our KVM hypervisor at Google Cloud (2017-01-26); D. E. Eisenbud et al., Maglev: A Fast and Reliable Software Network Load Balancer, NSDI 2016

Claude Mythos 5.1 作成の『CVE-2026-98432_暫定対策説明書』に対する ChatGPT 6 Pro Astra によるチェック結果の意見書

【Claude Mythos 5.1 の作成したスライド (1/41)】

Linux カーネル CVE-2026-98432 脆弱性対処: カーネルパッチ不要の暫定対策説明書

IPv4 TCP ハンドシェイクにおける ireq_opt / inet_opt の use-after-free (競合状態) の原理と、パケットフィルタ (iptables / nftables) による暫定遮断がなぜ効くのかの解説 — Linux を運用管理する日本国内組織のシステム管理者向け

本スライドは AI により作成されました

AI モデル	Claude Mythos 5.1 (Anthropic)
思考レベル	拡張思考 (Extended Thinking) — 最大 (max)
セッション開始日時	2026-10-30 22:03 JST
応答生成日時	2026-10-30 23:03 JST
基準カーネル	Linux v7.0 (Ubuntu 26.04 LTS が採用する系統) — 行番号は本資料共通
情報基準日	2026 年 10 月 30 日 (公開情報・カーネルソースの解析に基づく)

この資料でできること

- 今すぐ実行できる 2 つのフィルタコマンドと、その永続化・確認方法を得る
- 脆弱性の原理 (構造体・RCU・マルチコア競合・RCE への道筋) を図とソースで理解する
- コンテナ・ルータ・パブリッククラウド (IaaS/PaaS/SaaS) での適用範囲と限界を把握する
- 注意: 暫定対策は根本対策 (修正済みカーネルへの更新) までの橋渡しです。

対象読者: Linux システム (サーバ、コンテナホスト、仮想化基盤、ルータ) を運用管理する組織のシステム管理者 / セキュリティ担当者

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (1/41)】

Mythos 5.1 が作った本スライド全体の評点: 90 点 / 100 点

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

Mythosの資料は、UAFの原因と受信オプションを遮断する発想をよく捉えています。ただ、そのまま実施手順として配布するには、BPFによる要求移行、残存状態、ブリッジの通過経路、RFCの解釈、クラウドの断定を直す必要があります。私は全41頁の文脈を踏まえ、後頁の留保で解決する点や、安全側の運用判断そのものは減点していません。

AI バージョン : ChatGPT (製品版番号は取得不可) モデル : GPT-6 Astra Pro

思考レベル : 内部設定値は取得不可

セッション開始日時 (JST) : 正確な値は取得不可／本作業の記録開始 2026-10-31 00:50:55

応答生成日時 (JST) : 2026-10-31 01:24:16 (本意見書の生成時刻)

【1】LinuxカーネルCVEチームによる脆弱性公表

<https://lists.openwall.net/linux-cve-announce/2026/10/15/1>

【2】IPv4オプションをRCU猶予期間後に解放する修正

<https://github.com/torvalds/linux/commit/e3f33b0a1d89d738e2b913dab519226eaf76e15b>

【3】Goセキュリティチーム : Goプログラム利用時のCVE対策

<https://www.openwall.com/lists/oss-security/2026/10/30/1>

【Claude Mythos 5.1 の作成したスライド (2/41)】

本資料の構成と読み方

第 1 部　すぐ使う

スライド 3～7, 27～32

- エグゼクティブサマリ
- 最初に行う 2 つのコマンド
- 対策の全体像 (a / b / c / 根本)
- 脆弱性プロファイルと成立条件
- 適用手順・永続化・再起動・副作用
- ルータ / コンテナでの適用範囲

第 2 部　原理を知る

スライド 8～26

- IPv4 ヘッダと IHL、呪文の解説
- Netfilter フックとパケットの旅
- カーネル内部構造体とポインタの移管
- マルチコアで競合が起こる瞬間
- RCU の約束と 2 行の修正
- ソースコード挿絵 (6 か所)
- 情報漏洩・RCE の原理、MPTCP の役割

第 3 部　クラウドと責任分界

スライド 33～41

- 層構造と『誰が直せるか』
- IaaS / PaaS / SaaS それぞれの結論
- AWS / Azure / GCP の技術推定
- 自前基盤の安全上の利点
- チェックリスト / 参考文献

凡例と表記

即実行	設定・コマンドなど、そのまま使える内容
原理	脆弱性やフィルタが効く仕組みの解説
ソース	Linux v7.0 のソースコード抜粋 (ファイル名・行番号付き、英語コメントは日本語化)
クラウド	パブリッククラウド利用時の限界と責任分界

重要な前置き

- 本資料の暫定対策 (対策 a/b/c) は 2026-10-30 時点の公開情報とソース解析に基づく評価であり、全カーネル版で実証済みという意味ではありません。
- Red Hat は『推奨基準を満たす緩和策は無い』と表明しています。Go セキュリティチームはパケットフィルタ案を『未確認だが十分かもしれない』としています。
- 根本対策は修正済みカーネルへの更新です。暫定対策はそれまでの橋渡しとして位置づけてください。

Linux カーネル CVE-2026-98432 脆弱性対処 ― カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

2

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (2/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

この前置きは筋が通っています。Red Hatの記述は同社の推奨基準を満たす緩和策がないという趣旨で、緩和が物理的に不可能という意味ではありません。Go開発者のフィルタ案が未確認であること、全版で実証済みではないことも明示されています。

本書は、技術的正確性50点、成立条件・適用範囲30点、根拠の整合性20点で採点します。全体評点は実施判断に直結する頁を2倍、その他の本文を1倍とする加重平均です。表紙と章扉は集計から除きます。別頁の留保で補われる点や、慎重な運用方針という程度の違いは減点しません。

判定：妥当　採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

【1】Red Hatの脆弱性評価と緩和策方針
<https://access.redhat.com/security/cve/cve-2026-98432>
【2】Go開発者によるMPTCP既定動作と本脆弱性の検討
<https://github.com/golang/go/issues/82717>

共通注意：暫定策は更新までの措置です。更新できる場合は、修正済みカーネルへの更新を優先してください。

2 / 41

要約と公表された実証結果

【Claude Mythos 5.1 の作成したスライド (3/41)】

エグゼクティブサマリ: 何が起き、誰が危なく、今すぐ何をするか

即実行

9.8

CVSS (Linux カーネル CNA 評価)

4.4 →

影響範囲: 2015 年以降の全カーネル

2 行

修正差分 (kfree → kfree_rcu)

48+5 分

公開 PoC: KASLR 突破 + root 実行

0 件

実被害の報告なし / KEV 未登録 (10/28-29 時点)

何が起きたか

- Linux の IPv4 TCP 受付処理で、SYN に付いた IPv4 オプションを保持する小さなオブジェクト (ip_options_rcu) が RCU 猶予期間を待たずに解放され、別 CPU の読者が解放済みメモリを読む (use-after-free)。
- 子ソケットが要求ソケットから『ポインタを渡されるだけ』で複製されないことと、破棄時の kfree() が原因。IPv6 は複製するため非該当。
- 報告者は Anthropic の Xinyang Ge 氏。Anthropic は AI がゼロクリックの遠隔 root 実行を構築したと公表 (2026-10-27)。悪用コードは非公開。
- 修正は 2026-10-05 に適用、10-15 に CVE 公開。安定版 5.10 / 5.15 / 6.1 / 6.6 / 6.12 / 6.18 / 7.2 に反映済み。

誰が危ないか

- 未修正カーネルで IPv4 TCP を待ち受ける全ての Linux: サーバ、コンテナホスト (コンテナ内の待受も同じカーネル)、VM、ルータ、NAS、組込機器。
- 攻撃者に必要なのは『IPv4 オプション付きの SYN を待受ポートへ届けられること』と複数 CPU。認証や操作は不要。
- 公開 PoC は MPTCP 待受を前提にする。Go 1.24 以降の net.Listen は既定で MPTCP 待受を作るため、多くの Go 製サーバが該当。
- MPTCP 待受が無くても根本の競合は残る。『MPTCP を使っていないから無関係』とは言えない (Go セキュリティチームも同旨)。

今すぐやること (優先順)

- ① パケットフィルタで『IPv4 オプション付き TCP』を PREROUTING で DROP する (対策 a)。次のスライドのコマンド。
- ② MPTCP を無効化する (対策 b: sysctl / 対策 c: Go の GODEBUG 等)。公開 PoC の前提を外す。
- ③ ルールを永続化し、残存状態を消すために再起動する。
- ④ 修正済みカーネルへ計画的に更新する (根本対策)。
- ⑤ クラウド事業者・ホスティング事業者の対応状況を確認する (第 3 部)。
- 副作用: IP オプションを使う TCP 通信は遮断される (通常的环境ではほぼ存在しない)。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

3

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (3/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

要約の骨格は、私の照合でも裏付けられました。CNAの9.8、4.4での導入、二箇所の修正、Anthropicの48分と追加5分の実証は公表資料に合います。同頁が悪用コードの非公開を明記しているため、「公開PoC」は公表された実証の意味で読めます。MPTCPを止めてもUAF自体が残るという注意も正確です。KEVは取得できた2026年10月29日版で未登録でした。「被害の報告なし」と明記した範囲も尊重します。

判定：公表情報と全体文脈に照らして妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

【1】 CVE公式記録：影響版・修正版・深刻度の根拠

<https://raw.githubusercontent.com/CVEProject/cvelistV5/main/cves/2026/98xxx/CVE-2026-98432.json>

【2】 Claudeによる重要ソフトウェアの脆弱性発見と悪用の実証

<https://www.anthropic.com/research/discovering-vulnerabilities-with-claude>

【3】 Goセキュリティチーム：Goプログラム利用時のCVE対策

<https://www.openwall.com/lists/oss-security/2026/10/30/1>

【4】 CISA：悪用確認済み脆弱性カタログ（2026年10月29日版）

https://github.com/cisagov/kev-data/blob/develop/known_exploited_vulnerabilities.json

共通注意：暫定策は更新までの措置です。更新できる場合は、修正済みカーネルへの更新を優先してください。

3 / 41

最初に実行する 2 つのコマンド (どちらか一方): IPv4 オプション付き TCP を 門前払いする

iptables 版 (nftables 環境でも iptables-nft 互換レイヤが入っていれば利用可)

```
sudo iptables -t raw -I PREROUTING 1 -p tcp -m u32 --u32 '0>>24&0x0F=0x06:0x0F' -j DROP
```

nftables 版 (推奨: counter で遮断数を観測できる。ファイルに保存して sudo nft -f ファイル名 で読み込む)

```
table inet cve_2026_98432 {
  chain pre { type filter hook prerouting priority -310; policy accept;
    ip protocol tcp ip hdrlength gt 5 counter drop; }
}
```

何をするルールか

- 受信した IPv4 パケットのうち、プロトコルが TCP (6) で、ヘッダ長 IHL が 5 より大きい (= IP オプションが付いている) ものを、受信直後のフック (PREROUTING) で黙って捨てます。
- 通常の TCP (IHL=5、20 バイトヘッダ) には一切触れません。IPv6 は対象外 (非該当)。
- 原理: カーネルは『オプションが有る時だけ』脆弱なオブジェクトを作るため、オプション付き SYN が届かなければ競合の対象物が生まれません (第 2 部)。

適用後に必ず行うこと

- 永続化: nftables.conf への include / iptables-persistent 等。起動時に『信頼できない通信が TCP 処理に届く前』に適用される構成にする。
- 再起動: 適用前に生成された残存オブジェクトを確実に消すため、永続化後にホストを再起動することを推奨 (詳細はスライド 28)。
- 確認: sudo nft list table inet cve_2026_98432 / sudo iptables -t raw -L PREROUTING -v -n で一致数を監視。

注意事項

- iptables 版と nftables 版は同じ判定。両方入れても害は無いが、管理の一貫性のためどちらかに統一。
- firewalld / ufw / クラウドの管理エージェントが自動でルールを書き換える環境では、再適用の仕組みに組み込む。
- コンテナ (macvlan/ipvlan/SR-IOV) や VM は、ホストの PREROUTING を通らない経路がある (スライド 32)。
- これは暫定対策。修正済みカーネルへの更新を計画する。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

4

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (4/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

ここは、呪文に見えるコマンドにも十分な根拠があります。u32式とnftables式はともに、IPv4のIHLが6～15であるTCPを選択します。TCPオプションの有無を調べる式ではなく、SYN以外にも一致します。PREROUTINGで受付処理より先に破棄する位置も適切です。内部生成オプション、残存状態、名前空間ごとの迂回他頁で説明済みですから、この頁だけを切り出して不足とは判定しません。

判定：提示条件の下で妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

【1】Linux v7.0：u32条件の判定実装

https://github.com/torvalds/linux/blob/v7.0/net/netfilter/xt_u32.c

【2】nftables公式マニュアル

<https://netfilter.org/projects/nftables/manpage.html>

【3】Linux v7.0：IPv4受信処理と経路決定

https://github.com/torvalds/linux/blob/v7.0/net/ipv4/ip_input.c

【4】Linux v7.0：受信IPv4オプションの保存処理

<https://github.com/torvalds/linux/blob/v7.0/include/net/tcp.h>

共通注意：暫定策は更新までの措置です。更新できる場合は、修正済みカーネルへの更新を優先してください。

4 / 41

要求ソケット移行の停止条件

【Claude Mythos 5.1 の作成したスライド (5/41)】

対策の全体像: 対策 a / b / c と根本対策の関係

即実行

対策	内容	効く範囲 (原理)	前提・制約	推奨
対策 a パケットフィルタ	PREROUTING で『IPv4 オプション付き TCP』を DROP (前スライドのルール)。ホスト単体にも、境界ルータにも適用できる。	脆弱なオブジェクト (ireq_opt) の生成そのものを防ぐ。公開 PoC (MPTCP 経由) も、未公開の他経路も、『受信 IPv4 オプションに依存する』攻撃経路をすべて遮断する。	対象 TCP 処理に至る全経路でルールが有効なこと (コンテナ・トンネル・迂回経路に注意)。NetLabel/CIPSO で内部生成する構成では不十分。適用前の残存状態は再起動等で解消。	必須 (最優先)
対策 b MPTCP 無効化 (sysctl)	sysctl net.mptcp.enabled=0 (ネットワーク名前空間ごと)。加えて net.ipv4.tcp_migrate_req=0 を確認 (既定 0)。	公開 PoC の前提 (MPTCP 待受 + 即時破棄経路) を外す。移行経路 (5.14+) のポインタ共有も閉じる。	既存の MPTCP 待受ソケットは終了・再作成が必要。古いカーネルには存在しない sysctl がある。根本の競合は残る。	a に併用
対策 c アプリ側 MPTCP 無効化	Go 1.24+: GODEBUG=multipathtcp=0、go.mod の godebug 行、//go:debug 指示。他言語も個別設定。	sysctl が触れない環境 (コンテナ、権限分離) でも、待受ソケットを MPTCP にしない。	SetMultipathTCP() 等で明示的に有効化しているコードには効かない。『脆い』緩和策 (Go チーム)。	a に併用
根本対策 カーネル更新	修正済み安定版 (5.10.272 / 5.15.223 / 6.1.190 / 6.6.159 / 6.12.113 / 6.18.56 / 7.2.10 / 7.3) 以降、またはディストリの修正版へ更新して再起動。	kfree → kfree_rcu により、読者の RCU 区間中に解放されなくなる。	Ubuntu は作業中 (High)、RHEL は影響あり・修正未公開 (10/30 時点)。ディストリの修正版を待つ間は a (+b+c)。	計画的に実施

推奨の組合せ

a (必須) → + b + c (多層防御) → 根本対策。a だけで『受信 IPv4 オプションに依存する攻撃経路』は遮断できる。b + c だけでは根本の競合が残り、クラッシュやメモリ破損のリスクを排除できない。

『効く』の意味 (評価の前提)

本評価は 2026-10-30 時点の公開情報とソース解析に基づく。『対象の TCP 処理に至るすべての攻撃経路でルールが正しく適用され、順序が維持され、適用前の残存状態が解消されている』ことが前提 (※本資料の『前提条件』)。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

5

sysctl=0だけでは不十分。BPFによる要求移行も確認。

【ChatGPT 6 Astra Pro による意見 (5/41)】

Mythos 5.1 が作った本ページの評点: 85 点 / 100 点

対策を生成経路・MPTCP・アプリ設定に分けた整理は良いですね。ただ、「tcp_migrate_req=0で移行経路も閉じる」まで言い切るのは、少し先を急いでいます。BPF_SK_REUSEPORT_SELECT_OR_MIGRATE型プログラムがあると、sysctlが0でも要求移行は成立します。移行を止めたと判断するには、そのBPFプログラムの有無と移行方針まで確認してください。対策aの対象を受信オプション由来に限り、NetLabel/CIPSOを例外にした説明は妥当です。

判定：要求移行の停止条件に訂正が必要 採点内訳：正確性 44/50、成立条件 22/30、根拠 19/20

【1】LinuxカーネルCVEチームによる脆弱性公表
<https://lists.openwall.net/linux-cve-announce/2026/10/15/1>

【2】Linux v7.0 : SO_REUSEPORTの要求移行処理
https://github.com/torvalds/linux/blob/v7.0/net/core/sock_reuseport.c

【3】Goセキュリティチーム：Goプログラム利用時のCVE対策
<https://www.openwall.com/lists/oss-security/2026/10/30/1>

【4】CVE公式記録：影響版・修正版・深刻度の根拠
<https://raw.githubusercontent.com/CVEProject/cvelistV5/main/cves/2026/98xxx/CVE-2026-98432.json>

【Claude Mythos 5.1 の作成したスライド (6/41)】

脆弱性プロファイル (2026-10-30 時点)

即実行

項目	内容
識別子 / 公開日	CVE-2026-98432 — 2026-10-15 (Linux カーネル CVE チーム)。修正自体は 2026-10-05 に適用。
種別 / 箇所	競合状態に起因する use-after-free。net/ipv4/tcp_ipv4.c (ireq_opt) と net/ipv4/af_inet.c (inet_opt)。IPv4 TCP の受付処理 (3 ウエイハンドシェイク) のみ。IPv6 は非該当。
影響範囲	Linux 4.4 以降 (コミット 079096f103fa『要求ソケットを ehash に入れる』で露出)。加えて 5.14 以降の要求ソケット移行 (c905dee62232) にも同種の競合。
修正コミット	upstream e3f33b0a1d89d738e2b913dab519226eaf76e15b『ipv4: free inet_opt and ireq_opt after an RCU grace period』(Eric Dumazet, 7.3-rc7)。stable 例: 11a03ce3c4a6。
修正済み版	5.10.272 / 5.15.223 / 6.1.190 / 6.6.159 / 6.12.113 / 6.18.56 / 7.2.10 / 7.3
深刻度 (評価者により差)	カーネル CNA: 9.8 (10/24 付与)。Red Hat: CVE ページ 5.9 (Moderate) → アドバイザリ 8.1 (Important、10/29)。Oracle / SUSE: 8.1。Amazon: 7.0。NVD: 10/28 時点で未評価。
報告者 / 悪用	Xinyang Ge (Anthropic)。Anthropic は内部モデルが構築した 2 段階の遠隔 root 実行 (KASLR 突破 48 分 + コマンド実行 5 分) を公表。コード・対象版・構成は非公開。KEV 未登録、実被害の報告なし (10/28-29 時点)。
MPTCP との関係	CVE 記録と修正は MPTCP に言及しない。公開 PoC が MPTCP 待受を前提にするのみ。Go 1.24 以降の net パッケージは既定で待受に MPTCP を有効化。

※ ディストリの状況は変化が速い。適用判断の際は各ベンダの CVE ページで最新状況を確認すること。Red Hat は『推奨基準を満たす緩和策は無い』と表明しており、本資料の対策 a/b/c はコミュニティ/ソース解析に基づく暫定措置である。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

6

この8.1の一次資料は
特定できず。確認保
留。

今回の公式評価は8.1。
時点を明記して更新。

【ChatGPT 6 Astra Pro による意見 (6/41)】

Mythos 5.1 が作った本ページの評点: 94 点 / 100 点

脆弱性の識別、報告者、修正コミット、上流の修正版は一次資料と一致します。ただし、今回参照した Amazonの公式評価は8.1で、表の7.0は更新が必要です。SUSEも製品別に公開が進んでおり、修正状況は取得時刻付きで読む必要があります。IPv6の主たる移管問題は対象外ですが、公式通知が挙げるCALIPSOの移行問題は別パッチ扱いです。Red Hatの「8.1のアドバイザリ」は一次資料を特定できず、確認済みとして追認はしません。

判定：中核は妥当。時点情報と未確認の出典を整理 採点内訳：正確性 47/50、成立条件 29/30、根拠 18/20

- 【1】LinuxカーネルCVEチームによる脆弱性公表
- <https://lists.openwall.net/linux-cve-announce/2026/10/15/1>
- 【2】Amazon Linuxの脆弱性評価と修正状況
- <https://explore.alas.aws.amazon.com/CVE-2026-98432.html>
- 【3】SUSEの脆弱性評価と製品別修正状況
- <https://www.suse.com/security/cve/CVE-2026-98432.html>
- 【4】Red Hatの脆弱性評価と緩和策方針
- <https://access.redhat.com/security/cve/cve-2026-98432>

攻撃条件図の適用範囲

【Claude Mythos 5.1 の作成したスライド (7/41)】

攻撃が成立する 6 つの条件と、どの対策がどの条件を断ち切るか

即実行

1 IPv4 の TCP

IPv6 は非該当 (オプションを複製する実装)

断ち切る手段: —

2 SYN に IPv4 オプション

IHL > 5. これが無いと ireq_opt は作られない

断ち切る手段: 対策 a が遮断

3 待受ソケットへ到達

公開ポートに SYN を届けられること (踏み台含む)

断ち切る手段: FW / 露出削減

4 マルチコア

2 つの softirq が同時に走る必要 (ほぼ全サーバ)

断ち切る手段: —

5 child を即時破棄する経路

公開 PoC は MPTCP (MP_JOIN) の破棄経路を使用

断ち切る手段: 対策 b / c が遮断

6 未修正カーネル (4.4 以降)

根本原因. kfree → kfree_rcu の修正が未適用

断ち切る手段: カーネル更新

すべてが同時に満たされた時にだけ発動する。②を断てば (オプション付き SYN を門前払い) 残りの条件が揃っていても発動しない → これが対策 a の原理。
⑤を断つ (MPTCP 無効化) のは『公開された PoC の前提』を外すだけで、根本の UAF は残る。①④⑥は管理者には変えにくい前提条件。
例外: NetLabel/CIPSO を使う構成では、受信パケットにオプションが無くてもカーネル内部で ireq_opt を生成する経路がある (cipso_v4_req_setattr)。

読み方

攻撃は 6 条件がすべて同時に成立した時だけ発動する。管理者が確実に断ち切れるのは②③⑤⑥。②を断つ対策 a は、公開 PoC (MPTCP 経由) と未公開の他経路の双方に効く唯一の暫定策。

注意

⑤を断つだけ (MPTCP 無効化) では根本の競合が残る。NetLabel/CIPSO 構成ではカーネル内部で ireq_opt が生成されるため、②を断っても十分でない → 該当ホストは最優先でカーネル更新。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

7

⑤は全攻撃共通の必要条件にはできません。

このchild破棄経路の条件。26頁の要求移行は別経路です。

【ChatGPT 6 Astra Pro による意見 (7/41)】

Mythos 5.1 が作った本ページの評点: 82 点 / 100 点

条件を六つに分ける図は分かりやすいのですが、「全部そろった時だけ」とするには守備範囲が広すぎます。この図はIPv4のchild破棄経路のモデルです。公式通知と26頁にある要求ソケットの複製・移行の競合では、childの即時破棄を必要条件にはできません。MPTCP無効化だけでは不十分という下段の留保は正しいので、見出しと図の断定をその留保にそろえれば、対策判断に使える説明になります。

判定：図の必要条件を当該経路に限定 採点内訳：正確性 44/50、成立条件 20/30、根拠 18/20

【1】LinuxカーネルCVEチームによる脆弱性公表
<https://lists.openwall.net/linux-cve-announce/2026/10/15/1>

【2】Goセキュリティチーム：Goプログラム利用時のCVE対策
<https://www.openwall.com/lists/oss-security/2026/10/30/1>

共通注意：暫定策は更新までの措置です。更新できる場合は、修正済みカーネルへの更新を優先してください。

7 / 41

【Claude Mythos 5.1 の作成したスライド (8/41)】

第 2 部 原理を知る — 呪文の解読から、カーネル内部の競合、RCE の道筋まで

パケットヘッダ → Netfilter フック → 構造体とポインタ → マルチコア競合 → RCU の約束 → ソースコード → 悪用の原理 → MPTCP の役割

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

8

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (8/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

原理の説明順序には賛成です。ヘッダの条件から遮断位置へ進み、その後にポインタの所有権と寿命を追う構成なら、なぜこのフィルタを選ぶのかを理解できます。これ以降の図は通常のIPv4受付経路を軸にした説明として読みます。コードから確認できる挙動、公表された悪用実証、説明のための模式図を区別することが、技術資料としての信頼につながります。

判定：妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

- 【1】LinuxカーネルCVEチームによる脆弱性公表
<https://lists.openwall.net/linux-cve-announce/2026/10/15/1>
- 【2】IPv4オプションをRCU猶予期間後に解放する修正
<https://github.com/torvalds/linux/commit/e3f33b0a1d89d738e2b913dab519226eaf76e15b>

IPv4 ヘッダと IHL

【Claude Mythos 5.1 の作成したスライド (9/41)】



原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (9/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

ヘッダ図は丁寧で、数値にも問題を認めません。IHLは先頭バイトの下位4ビットで、単位は4バイトです。5なら20バイト、6～15なら24～60バイトとなり、0x45・0x47・0x4Fの例も正しく読めています。MPTCP等のTCPオプションをIPv4オプションと別の場所に描いた点も適切です。末尾の生成条件は、別頁で明記したNetLabel/CIPSO例外を除く通常の受信経路の説明として成立します。

判定：妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

【1】RFC 791：インターネットプロトコル仕様

<https://www.rfc-editor.org/rfc/rfc791.txt>

【2】Linux v7.0：IPv4受信処理と経路決定

https://github.com/torvalds/linux/blob/v7.0/net/ipv4/ip_input.c

【3】Linux v7.0：受信IPv4オプションの保存処理

<https://github.com/torvalds/linux/blob/v7.0/include/net/tcp.h>

u32 のビット演算

【Claude Mythos 5.1 の作成したスライド (10/41)】

呪文の解説 (1): iptables -m u32 '0>>24&0x0F=0x06:0x0F' は何を見ているのか

原理

iptables 版のルール (1 行) をトークン毎に分解

```
sudo iptables -t raw -I PREROUTING 1 -p tcp -m u32 --u32 '0>>24&0x0F=0x06:0x0F' -j DROP
```

外側の呪文: どこで・何に・どうするか

-t raw
raw テーブル: conntrack より前に評価される (優先度 -300)

-I PREROUTING 1
受信直後のフック (PREROUTING) の先頭 (1 番目) に挿入

-p tcp
IPv4 ヘッダの Protocol フィールド = 6 (TCP) のみ対象

-m u32
xt_u32: パケット中の任意 32 ビット語を取り出し演算・比較

-j DROP
一致したら黙って破棄 (応答も返さない)

内側の呪文 '0>>24&0x0F=0x06:0x0F' = 「IHL が 6~15 か?」

IPv4 ヘッダ先頭の 32 ビット語 (オフセット 0)

Version	IHL	DSCP/ECN	Total Length	例: 0x47 00 00 3C
1	0			オフセット 0 から 32 ビットを読む (ビッグエンディアン) 0x4700003C
2	>>24			右に 24 ビットシフト → 先頭 1 バイトだけが残る 0x00000047
3	&0x0F			下位 4 ビットだけをマスク → IHL フィールドの値 0x7 (=7)
4	=0x06:0x0F			値が 6~15 の範囲内なら一致 (IHL>5 = オプション有り) 7 ∈ [6, 15] → 一致 → DROP

具体例: 通常の SYN は通過し、オプション付き SYN だけが落ちる

0x45 00 00 3C 1C 46 40 00 40 06 ... 先頭 0x45 → IHL=5 → 範囲外 → 一致せず → 次のルールへ (通常の TCP は影響なし)

0x47 00 00 44 ... 40 06 ... [07 07 04 ..] 先頭 0x47 → IHL=7 → 範囲内 → 一致 → DROP (28 B ヘッダ = 20 B + 8 B の RR オプション)

注: IPv6 パケットは -p tcp (IPv4 用 iptables) の対象外。IPv6 は本脆弱性の影響を受けない (カーネルはオプションを複製するため)。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

10

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (10/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

ビット演算の説明は、私の確認でもそのまま使えます。先頭の32ビット値をネットワーク順から読み、24ビット右へずらし、0x0FでマスクするとIHLだけが残ります。6～15との比較でオプション付きIPv4 TCPを選べます。0x47の例はRRの7バイトとパディングを含む8バイトの領域にも整合します。下段はSYNの例ですが、規則そのものにはSYNフラグの条件がなく、同じIHL条件のTCP全体を破棄します。

判定：妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

- 【1】Linux v7.0：u32条件の判定実装
https://github.com/torvalds/linux/blob/v7.0/net/netfilter/xt_u32.c
- 【2】RFC 791：インターネットプロトコル仕様
<https://www.rfc-editor.org/rfc/rfc791.txt>
- 【3】Netfilterフックと処理優先順位
https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks

nftables の式とフックの優先順位

【Claude Mythos 5.1 の作成したスライド (11/41)】

呪文の解読 (2): nftables 'ip protocol tcp ip hdrlength gt 5 counter drop' の各トークン

原理

nftables 版のルール (table 定義 1 ブロック) をトークン毎に分解

```
table inet cve_2026_98432 {
  chain pre {
    type filter hook prerouting priority -310; policy accept;
    ip protocol tcp ip hdrlength gt 5 counter drop;
  }
}
```

ヘッダのどこを見ているか

▼ ip hdrlength gt 5 はここ (IHL) を見る

Ver	IHL	DSCP/ECN	Total Length
Identification		F	Fragment Offset
TTL	Protocol	← ip protocol tcp はここ (=6)	
Source Address			
Destination Address			
Options (IHL>5 のときのみ存在)			

同じ『IHL > 5』判定を、u32 のビット演算ではなく nft の組込みヘッダ式で書いたもの。
counter により『実際に何パケット落ちたか』が nft list ruleset で読める。

table inet NAME	inet ファミリー = IPv4/IPv6 両用テーブル。名前は任意 (識別用)
chain pre	チェーン名は任意。基本チェーン (フックに直結するチェーン)
type filter hook prerouting	受信直後 (PREROUTING) のフックで評価する filter 型
priority -310	raw(-300) より前、defrag(-400) より後に実行される
policy accept	一致しないパケットは通常どおり通過
ip protocol tcp	IPv4 ヘッダ Protocol = 6。『ip』式は IPv4 専用 (IPv6 は対象外)
ip hdrlength gt 5	IPv4 ヘッダの IHL > 5 (= オプション有り)。u32 版と同じ判定
counter	一致パケット数・バイト数を計上 (監視・副作用確認に有用)
drop	黙って破棄 (応答を返さない)

適用と確認 (抜粋)

```
sudo nft -f /etc/nftables.d/cve-2026-98432.nft # 即時適用
sudo nft list table inet cve_2026_98432 # 確認
ip protocol tcp ip hdrlength gt 5 counter packets 12 bytes 768 drop
↑ counter の値が増えれば遮断が動作している
sudo systemctl enable --now nftables # 永続化 (nftables.conf に include)
```

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

11

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (11/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

nftablesの読み解きも正確です。inetテーブルでもip式はIPv4を対象とし、hdrlengthはIHLの値を比較します。優先度-310は標準のdefrag-400より後、raw-300とconntrack-200より前です。counterの説明も正しいものです。policy acceptは、この独立チェーンで一致しなかったパケットを後続処理へ渡す意味であり、既存の拒否規則を解除しません。この違いが分かる図になっています。

判定：妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

- 【1】nftables公式マニュアル
- <https://netfilter.org/projects/nftables/manpage.html>
- 【2】Netfilterフックと処理優先順位
- https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks

呪文の裏側: カーネルは PREROUTING で何を実行しているのか (ip_rcv / xt_u32)

ソース

net/ipv4/ip_input.c — ip_rcv()

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

```
564 int ip_rcv(struct sk_buff *skb, struct net_device *dev, struct packet_type *pt,
565           struct net_device *orig_dev)
566 {
567     struct net *net = dev_net(dev);
568     skb = ip_rcv_core(skb, net);           /* 版・IHL<5・チェックサム等の基本検査 */
569     if (skb == NULL)
570         return NET_RX_DROP;
571     return NF_HOOK(NFPROTO_IPV4, NF_INET_PRE_ROUTING, /* ★ 本ルールが評価される場所 */
572                   net, NULL, skb, dev, NULL, ip_rcv_finish); /* 通過した時だけ続きへ */
573 }
574
575
```

PREROUTING: ローカル宛も転送も、受信した IPv4 パケットは全てここを通る。DROP されたパケットは ip_options_compile にも TCP にも到達しない。

net/netfilter/xt_u32.c — u32_match_it() (iptables -m u32 のカーネル側、簡略化)

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

```
37 pos = ct->location[0].number;           /* '0' = オフセット 0 バイト */
39 if (skb->len < 4 || pos > skb->len - 4) return false;
44 val = ntohs(n);                          /* 先頭 32 ビット語をホスト順に */
48 for (i = 1; i < nnums; ++i) {            /* 内側ループ: &, <<, >>, @ */
50     switch (ct->location[i].nextop) {
51         case XT_U32_AND : val &= number; break; /* '&0x0F' → 下位 4 ビット = IHL */
52         case XT_U32_RIGHTSH : val >>= number; break; /* '>>24' → 先頭 1 バイト */
53     }
54 }
55 for (i = 0; i < nvals; ++i)              /* ", " と ":" の各値 (範囲) と比較 */
56     if (ct->value[i].min <= val && val <= ct->value[i].max) break; /* '=0x06:0x0F' */
57 if (i >= ct->nvalues) return false;       /* 範囲外 (IHL=5) → 次のルールへ */
58
```

呪文の正体: ユーザ空間で '0>>24&0x0F=0x06:0x0F' が [位置 0, 演算子 列 (>>24, &0x0F), 範囲 6..15] に翻訳され、この汎用ループが評価する。

nftables 版の 'ip hlength gt 5' は nft_payload 式がヘッダ先頭バイトの下位 4 ビットを取り出して比較する等価な判定。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

12

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

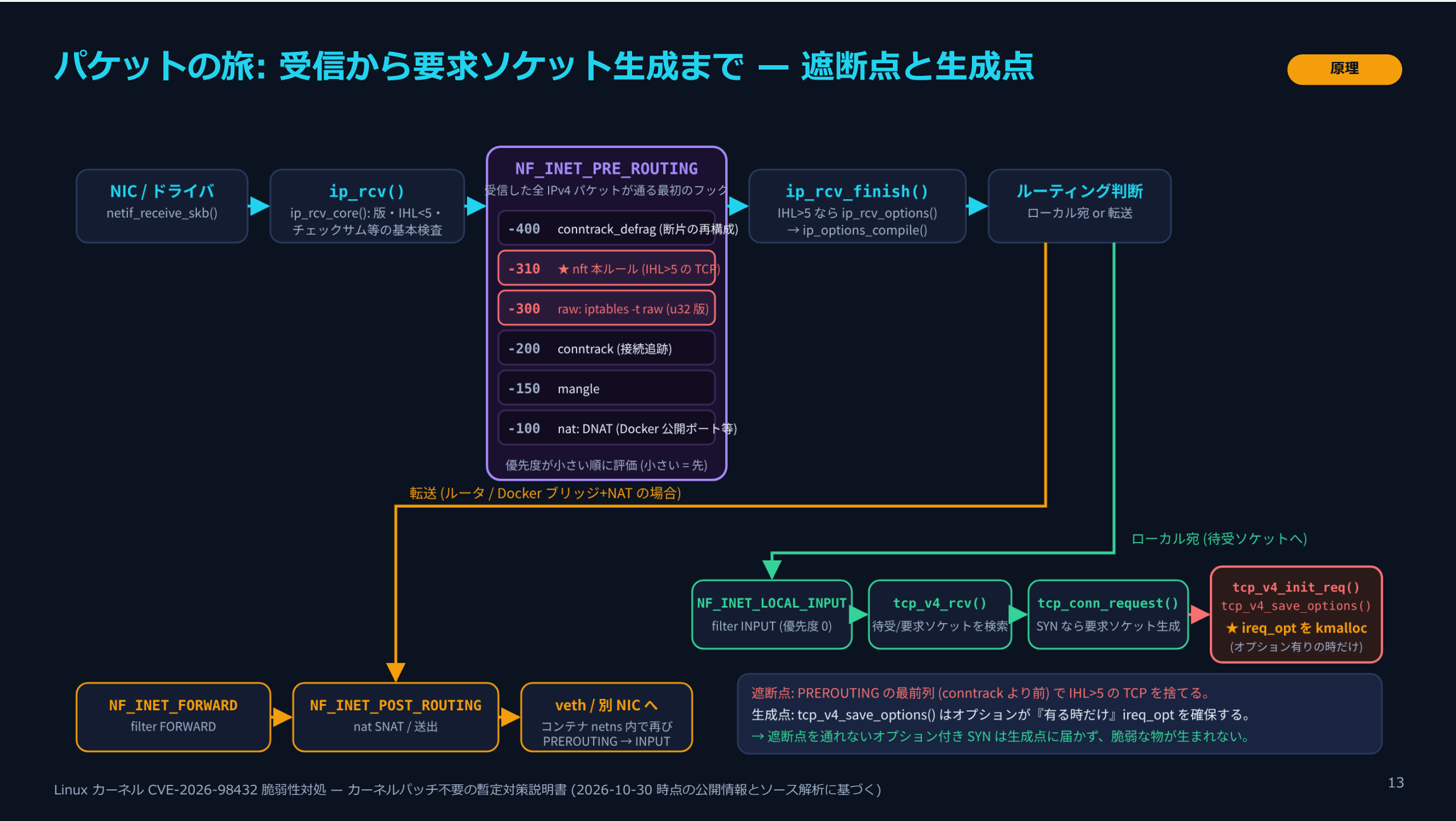
このコード図の要所は、Linux v7.0と一致しています。ip_rcv_coreの検査後にPREROUTINGを通り、通過時だけip_rcv_finishへ進む流れです。u32側のntohl、シフト、マスク、範囲比較も実装どおりでした。省略箇所は「簡略化」と示され、意味を変えていません。「全てここを通る」はこの通常のIPv4受信経路についての説明と読み、後頁のブリッジやオフロードの留保を重ねて欠落扱いにはしません。

判定：妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

- 【1】Linux v7.0：IPv4受信処理と経路決定
- https://github.com/torvalds/linux/blob/v7.0/net/ipv4/ip_input.c
- 【2】Linux v7.0：u32条件の判定実装
- https://github.com/torvalds/linux/blob/v7.0/net/netfilter/xt_u32.c
- 【3】nftables公式マニュアル
- <https://netfilter.org/projects/nftables/manpage.html>

受信経路とオプションの生成点

【Claude Mythos 5.1 の作成したスライド (13/41)】



原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (13/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

TCP受付より前に遮断する、という図の主眼は正しく伝わります。conntrackやDNATとの前後関係にも問題は
ありません。右上の「ルーティング判断」は、決まった経路に応じてローカル宛と転送へ処理を振り分け
る概念図として読めます。実装内部の細かな呼出し順との差は、この図の防御上の結論を変えません。ブリ
ッジやオフロードの例外は32頁で述べられているため、本頁では欠落として重ねて指摘しません。

判定：通常経路の概念図として妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

- 【1】Linux v7.0：IPv4受信処理と経路決定
https://github.com/torvalds/linux/blob/v7.0/net/ipv4/ip_input.c
- 【2】Netfilterフックと処理優先順位
https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks
- 【3】Linux v7.0：受信IPv4オプションの保存処理
<https://github.com/torvalds/linux/blob/v7.0/include/net/tcp.h>

カーネル内部の構造体とリンク: 1 つの ip_options_rcu を誰が指すか

原理

待受ソケット (listener)
struct sock / tcp_sock
sk_state = TCP_LISTEN
icsk_accept_queue
(accept() 待ちの子)
SO_REUSEPORT グループ (任意)

この物を触るコード

生成
tcp_v4_init_req()
→ tcp_v4_save_options()

読者 (rcu_dereference)
tcp_v4_send_synack()
inet_csk_route_req()
inet_csk_route_child_sock()
tcp_v4_syn_rcv_sock()

解放 (plain kfree ~ 問題)
tcp_v4_reqsk_destructor()
inet_sock_destruct()

複製 (ポインタの共有)
inet_reqsk_clone() memcpy

確立ハッシュ表 ehash (全 CPU から RCU で検索)

req

struct tcp_request_sock
↳ inet_request_sock (TCP_NEW_SYN_RECV)

rsk_listener → listener
rsk_refcnt (参照数: 検索した CPU が+1)
rsk_timer SYNACK 再送タイマ (PINNED)
ir_rmt_addr / ir_rmt_port (攻撃者側)
saved_syn, tfo..., is_mptcp, drop_req
ireq_opt (__rcu ポインタ) →

4.4以降: 要求ソケットは ehash に入り他 CPU から見える

child: struct tcp_sock (新しい接続)
sk_state: TCP_SYN_RECV → ESTABLISHED
inet_sock.inet_opt (__rcu) →
icsk_ext_hdr_len = optlen (MSS 計算に反映)
sk_destruct = inet_sock_destruct
↳ 破棄時に inet_opt を kfree ← 修正点

struct ip_options_rcu
kmalloc(sizeof + optlen, GFP_ATOMIC)

struct rcu_head rcu (16 B)
kfree_rcu() 用のヘッダ = RCU で守る前提の設計

struct ip_options opt (16 B + data)
__be32 faddr, nexthop (経路指定用)
u8 optlen ← コピー長 (memcpy の長さ)
u8 srr, rr, ts ← 書き込み先オフセット
u8 flags: is_strictroute, rr_needaddr, ts_needtime, ts_needaddr ...
u8 router_alert, cipso, __pad2
u8 __data[optlen] (最大 40 B の生オプション)

合計 32~72 B ⇒ kmalloc-64/96 スラブ (他の物と混在・再利用)

最終 ACK 到着時に
ポインタだけ移管
(オブジェクトは複製しない)
→ 一瞬 2 か所から同じ物を指す

バイト数・スラブ区分は典型的64ビット構成の例。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

14

所有者と読者を分けて描いた点は、私も評価します。req自体の参照数が残っていても、別オブジェクトOPTの寿命は守れません。ireq_optからinet_optへの移管と、同じ領域を別CPUが読む関係はソースに合っています。なお、rcu_headの16バイトやkmalloc-64/96という値は典型的な64ビット構成の例です。アーキテクチャや設定に依存する数字であることを小さく添えれば、図を広い環境へ持ち出しても誤解を招きません。

判定：中核は妥当。サイズは代表構成の値 採点内訳：正確性 49/50、成立条件 29/30、根拠 20/20

- 【1】Linux v7.0：IPv4ソケットとオプション構造体
- https://github.com/torvalds/linux/blob/v7.0/include/net/inet_sock.h
- 【2】Linux v7.0：IPv4 TCP要求生成・送信・所有権移管
- https://github.com/torvalds/linux/blob/v7.0/net/ipv4/tcp_ipv4.c
- 【3】IPv4オプションをRCU猶予期間後に解放する修正
- <https://github.com/torvalds/linux/commit/e3f33b0a1d89d738e2b913dab519226eaf76e15b>

通常の接続におけるポインタの移管

【Claude Mythos 5.1 の作成したスライド (15/41)】

正常系のライフサイクル: SYN から close() まで、OPT はどう受け渡されるか

原理

クライアント

サーバ (Linux カーネル)

SYN (+IPv4 オプション, 例: RR)

SYN/ACK (オプションをエコー)

ACK (通常はオプション無しでも可)

データ .. (接続の利用)

FIN / close()

① 要求ソケット生成
tcp_conn_request()
→ tcp_v4_init_req()
→ ireq_opt = kmalloc(OPT)
ehash に登録、SYNACK タイマ開始

② 応答の生成
tcp_v4_send_synack()
rcu_dereference(ireq_opt) を読む
(再送時同じ経路)

③ 所有権の移転
tcp_v4_syn_recv_sock()
newinet->inet_opt = OPT
ireq->ireq_opt = NULL
req を ehash から外し child を登録

④ 送信時に参照
ip_queue_xmit() が inet_opt を使い
応答バケットにオプションを付与

⑤ 破棄
inet_sock_destruct()
kfree(inet_opt) → 読者が居なければ無害

正常系では読者 (②) は③より前に終わるので kfree は安全。問題は②と③〜⑤が別 CPU で重なる場合。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

15

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (15/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

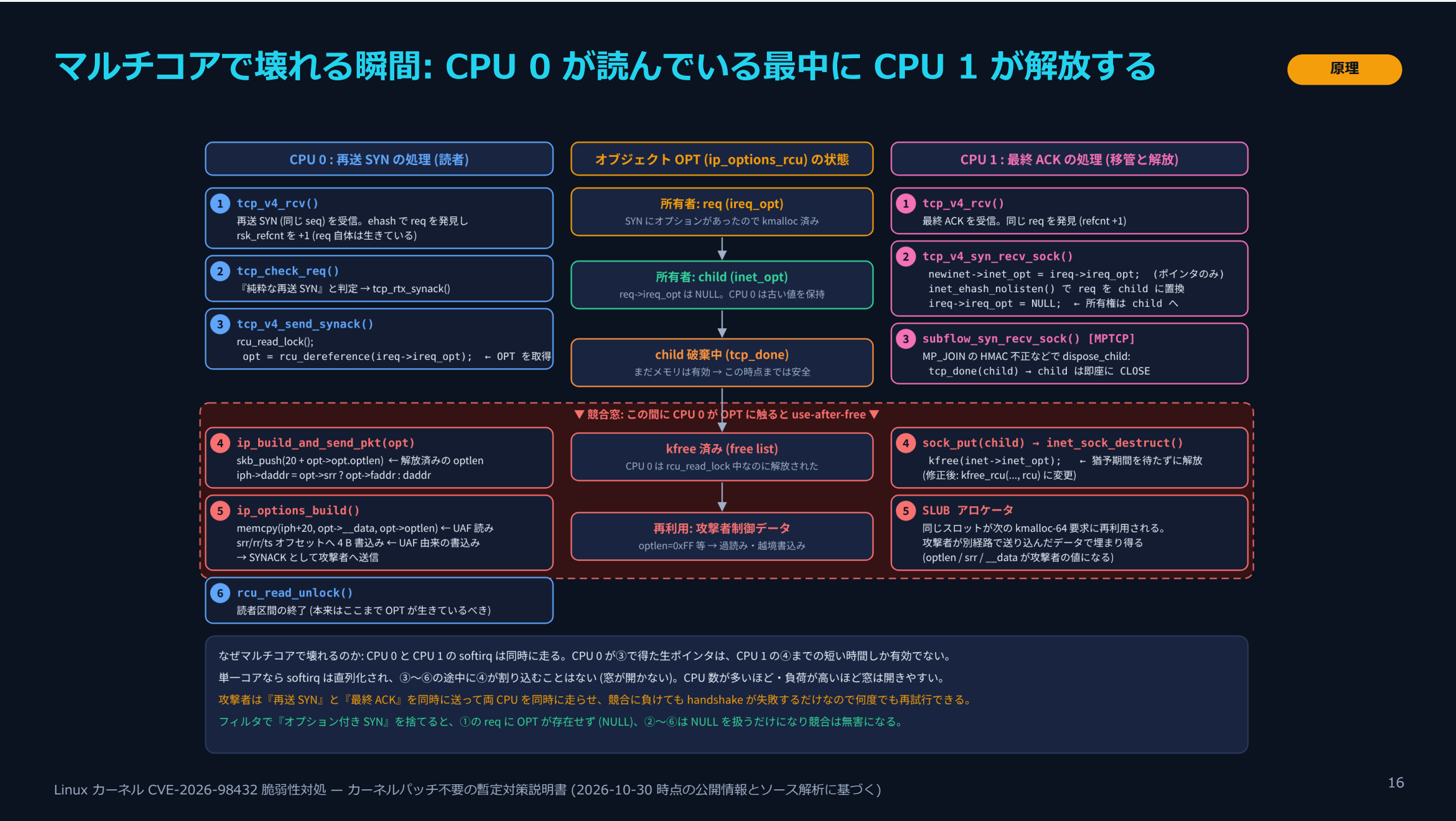
この流れは、通常の接続ライフサイクルの説明として妥当です。SYNのオプションをreqに保持し、childへ複製せず渡す点が核心です。req側をNULLにしても、すでに値を取り出した別CPUの処理は待ちません。所有者が変わることと、全読者が使い終わることを区別できている点は大切です。後続頁で再送SYNと早期破棄を具体化しているので、この正常系の図にその全条件を詰め直す必要はありません。

判定：通常経路の説明として妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

- 【1】Linux v7.0：受信IPv4オプションの保存処理
- <https://github.com/torvalds/linux/blob/v7.0/include/net/tcp.h>
- 【2】Linux v7.0：IPv4 TCP要求生成・送信・所有権移管
- https://github.com/torvalds/linux/blob/v7.0/net/ipv4/tcp_ipv4.c
- 【3】IPv4オプションをRCU猶予期間後に解放する修正
- <https://github.com/torvalds/linux/commit/e3f33b0a1d89d738e2b913dab519226eaf76e15b>

複数 CPU による競合の時系列

【Claude Mythos 5.1 の作成したスライド (16/41)】



RCU の約束と修正差分

【Claude Mythos 5.1 の作成したスライド (17/41)】

RCU の約束と、それを破った 2 か所: 修正はなぜ 2 行で済むのか

原理

RCU の約束 (正しい使い方): 読者は待たせない、解放は猶予期間の後

読者 A (CPU 0)

rcu_read_lock → rcu_dereference(p) → 使用 → rcu_read_unlock

読者 B (CPU 2)

rcu_read_lock → 使用 → unlock

更新者 (CPU 1)

p = NULL

kfree_rcu()

猶予期間

実解放

既存読者が全員 unlock した後

読者はロックを取らず rcu_read_lock() の区間でポインタを辿る (高速、CPU 間で待たない)。

更新者はポインタ付替え後、kfree_rcu()/call_rcu() で『全読者が区間を抜けるまで』解放を遅らせる。

struct ip_options_rcu が rcu_head を持つのは、まさにこの約束を守るため。

CVE-2026-98432: 約束を破った 2 か所 (plain kfree)

読者 (CPU 0)

rcu_read_lock → rcu_dereference → memcpy(opt->__data) → unlock

更新者 (CPU 1)

ireq_opt=NULL

kfree()

読者の区間の途中で解放 → UAF

child を壊す inet_sock_destruct() と、req を壊す tcp_v4_reqsk_destructor() だけが kfree() だった。

『所有者が居ない = 読者も居ない』という仮定は、ehash 経由の読者やタイマには成立しない。

他の経路 (setsockopt IP_OPTIONS, CIPSO) は以前から kfree_rcu() を使っていた。

修正コミット e3f33b0a1d89 (Eric Dumazet, 2026-10) の差分 — 2 ファイル、各 1 行

net/ipv4/af_inet.c @@ inet_sock_destruct()
- kfree(rcu_dereference_protected(inet->inet_opt, 1));
+ kfree_rcu(rcu_dereference_protected(inet->inet_opt, 1), rcu);

net/ipv4/tcp_ipv4.c @@ tcp_v4_reqsk_destructor()
- kfree(rcu_dereference_protected(inet_rsk(req)->ireq_opt, 1));
+ kfree_rcu(rcu_dereference_protected(inet_rsk(req)->ireq_opt, 1), rcu);

kfree_rcu は『猶予期間の後に kfree』。読者は一切変更不要で、既存の rcu_head を使うだけ。

Fixes: 079096f103fa (4.4) / c905dee62232 (5.14)

Reported-by: Xinyang Ge (Anthropic)

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

17

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (17/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

RCUの約束と二箇所の差分は、公式修正とおりです。取得済みの読者がRCU区間を抜けるまで、OPTの実解放を遅らせる必要があります。「タイマ」という記述も、26頁の要求移行を含む説明として読めば整合します。通常のSYNACK再送タイマは参照保持と同期停止で保護され、移行時の複製・共有には別の競合があります。この区別を踏まえると、二つのkfreeをkfree_rcuへ変える理由が明確になります。

判定：二つの解放経路を含む説明として妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

【1】IPv4オプションをRCU猶予期間後に解放する修正
<https://github.com/torvalds/linux/commit/e3f33b0a1d89d738e2b913dab519226eaf76e15b>

【2】LinuxカーネルCVEチームによる脆弱性公表
<https://lists.openwall.net/linux-cve-announce/2026/10/15/1>

共通注意：暫定策は更新までの措置です。更新できる場合は、修正済みカーネルへの更新を優先してください。

17 / 41

オプション生成条件のソース照合

【Claude Mythos 5.1 の作成したスライド (18/41)】

ソース (1) 誕生: tcp_v4_save_options() はオプションが無ければ何も作らない

ソース

include/net/tcp.h — tcp_v4_save_options()

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

```
2640 static inline struct ip_options_rcu *tcp_v4_save_options(struct net *net, struct sk_buff *skb)
2642 {
2643     const struct ip_options *opt = &TCP_SKB_CB(skb)->header.h4.opt; /* IHL>5 の時のみ非空 */
2644     struct ip_options_rcu *dopt = NULL;
2646     if (opt->optlen) { /* 受信 SYN にオプションが有る時だけ */
2647         int opt_size = sizeof(*dopt) + opt->optlen;
2649         dopt = kmalloc(opt_size, GFP_ATOMIC); /* ★ 脆弱な対象物 (OPT) の誕生 */
2650         if (dopt && __ip_options_echo(net, &dopt->opt, skb, opt)) { /* 応答用に整形 */
2651             kfree(dopt); dopt = NULL;
2653         }
2654     }
2655     return dopt; /* オプション無しなら NULL のまま */
2656 }
```

フィルタが効く一次情報: optlen は受信 IPv4 ヘッダの IHL から決まる。IHL=5 なら 0 → kmalloc されず NULL。

誕生: rcu_head + ip_options + 生オプション分を kmalloc-64/96 から確保。これが競合で解放される OPT。

net/ipv4/tcp_ipv4.c — tcp_v4_init_req()

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

```
1629 static void tcp_v4_init_req(struct request_sock *req,
1630                             const struct sock *sk_listener, struct sk_buff *skb)
1632 {
1633     struct inet_request_sock *ireq = inet_rsk(req);
1634     struct net *net = sock_net(sk_listener);
1636     sk_rcv_saddr_set(req_to_sk(req), ip_hdr(skb)->daddr);
1637     sk_daddr_set(req_to_sk(req), ip_hdr(skb)->saddr);
1638     RCU_INIT_POINTER(ireq->ireq_opt, tcp_v4_save_options(net, skb)); /* ★ 所有者 1: req */
1639 }
```

所有者 1: RCU 保護ポインタとして要求ソケットに格納。この req は ehash に入り、他 CPU から検索できる (4.4 以降)。

呼出し順: SYN 受信 → tcp_conn_request() (tcp_input.c:7640) → af_ops->route_req = tcp_v4_route_req() → tcp_v4_init_req()

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

18

「optlenが非ゼロ」と書けば意味が正確です。

【ChatGPT 6 Astra Pro による意見 (18/41)】

Mythos 5.1 が作った本ページの評点: 98 点 / 100 点

ここは原理を支える重要な抜粋で、コードと行番号はLinux v7.0に一致しています。optlenが0なら割当てを行わず、doptはNULLのまま返ります。細部ですが「IHL>5の時のみ非空」は、optポインタではなくopt->optlenが非ゼロになる意味と明記しましょう。opt自体はskb内の構造体を指しています。NetLabel/CIPSOによる別の生成経路は他頁に留保があるため、この関数の説明に対する反証とは扱いません。

判定：コードと行番号は正確。注釈を精密化 採点内訳：正確性 49/50、成立条件 29/30、根拠 20/20

- 【1】Linux v7.0：受信IPv4オプションの保存処理
<https://github.com/torvalds/linux/blob/v7.0/include/net/tcp.h>
- 【2】Linux v7.0：IPv4 TCP要求生成・送信・所有権移管
https://github.com/torvalds/linux/blob/v7.0/net/ipv4/tcp_ipv4.c
- 【3】Linux v7.0：IPv4受信処理と経路決定
https://github.com/torvalds/linux/blob/v7.0/net/ipv4/ip_input.c

複製せずに渡す IPv4 のポインタ

【Claude Mythos 5.1 の作成したスライド (19/41)】

ソース

net/ipv4/tcp_ipv4.c — tcp_v4_syn_recv_sock() 抜粋

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

1732 newtp = tcp_sk(newsk);
1733 newinet = inet_sk(newsk);
1734 ireq = inet_rsk(req);
1735 inet_opt = rcu_dereference(ireq->ireq_opt); /* OPT を読む */
1736 RCU_INIT_POINTER(newinet->inet_opt, inet_opt); /* ★ ポインタを渡すだけ (複製しない) */
1741 if (inet_opt)
1742 inet_csk(newsk)->icsk_ext_hdr_len = inet_opt->opt.optlen;
... /* ... 経路・MSS・MD5/AO の初期化 (省略) ... */
1789 *own_req = inet_ehash_nolisten(newsk, req_to_sk(req_unhash),
1790 &found_dup_sk); /* ehash 上で req → child に置換 */
1791 if (likely(*own_req)) {
1792 tcp_move_syn(newtp, req);
1793 ireq->ireq_opt = NULL; /* ★ 所有権は child へ (plain store) */
1794 } else {
1795 newinet->inet_opt = NULL; /* 負けた側は何も所有しない */
1796 }
1806 return newsk;

1つのOPTを2つの所有者が指す瞬間。
IPv6版(tcp_v6_syn_recv_sock)は
ipv6_dup_options()で複製するため非該当。

置換の後も、置換前にreqを見つけていた
CPU(再送SYN処理中)はreq経由でOPTを
読み続ける。NULL代入は他CPUの読者を待
たない。

修正コミットの説明: 『tcp_v4_syn_recv_sock()はireq->ireq_optをコピーせずchildに渡す。並行する再送SYNはRCUだけでそれを読み、解放後に使い得る』

IPv6のtcp_v6_syn_recv_sock()はipv6_dup_options()で複製してからchildに持たせるため所有者が1つしか無く、本問題の影響を受けない(修正コミットも明記)。

Linuxカーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

19

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (19/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

ここは Mythos さんの読みが正確です。IPv4 は要求ソケットのオプションを複製せず、子ソケットへ同じポインタを移します。ireq_opt を NULL にしても、別 CPU がすでに取得した参照は消えません。勝者・敗者で所有権を整理する分岐もソースと一致します。IPv6 との比較は、本頁の子ソケット生成経路について正しい説明です。別の CALIPSO・要求移行問題まで安全と広げずに読めば、訂正は要りません。

判定：妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

- 【1】Linux v7.0 : IPv4 TCP の子ソケット作成と SYNACK 送信
https://github.com/torvalds/linux/blob/v7.0/net/ipv4/tcp_ipv4.c
- 【2】Linux v7.0 : IPv6 TCP のオプション複製
https://github.com/torvalds/linux/blob/v7.0/net/ipv6/tcp_ipv6.c
- 【3】Linux カーネル CVE チームによる脆弱性公表
<https://lists.openwall.net/linux-cve-announce/2026/10/15/1>

再送 SYN から保存済み OPT を読む経路

【Claude Mythos 5.1 の作成したスライド (20/41)】

ソース (3) 読者: 再送 SYN → tcp_check_req() → tcp_rtx_synack() → tcp_v4_send_synack()

ソース

net/ipv4/tcp_minisocks.c — tcp_check_req() 抜粋

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

```
727      /* 純粋な再送 SYN を再送信したら SYN+ACK を再送信する */
728      if (TCP_SKB_CB(skb)->seq == tcp_rsk(req)->rcv_isn &&
729          flg == TCP_FLAG_SYN && !paws_reject) {
731          /* RFC 1122: SYN-RECV で SYN を再受信したら SYN+ACK を再送信する */
732          if (!tcp_oow_rate_limited(sock_net(sk), skb, LINUX_MIB_TCPACKSKIPPEDSYNRECV,
733                                  &tcp_rsk(req)->last_oow_ack_time)) {
765              if (!tcp_rtx_synack(sk, req)) { /* ★ 読者: SYNACK を作り直す */
...
777              }
778          }
779          return NULL;
780      }
```

攻撃者が任意に起こせる読者: 同じ seq の SYN をもう一度送るだけで、SYN-RECV 状態の req に対し SYNACK 再送 (= OPT の読み出し) が走る。

tcp_rtx_synack() (tcp_output.c:4636) → af_ops->send_synack = tcp_v4_send_synack() (tcp_ipv4.c:1182) へ。

net/ipv4/tcp_ipv4.c — tcp_v4_send_synack() 抜粋

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

```
1215      rcu_read_lock();
1216      err = ip_build_and_send_pkt(skb, sk, ireq->ir_loc_addr, ireq->ir_rmt_addr,
1217                                 rcu_dereference(ireq->ireq_opt), tos); /* ★ OPT の生ポインタ */
1220      rcu_read_unlock();
1221      err = net_xmit_eval(err);
```

読者は正しく RCU 区間に入っている。ここで得た生ポインタは区間が終わるまで有効であるべき — kfree_rcu なら守られる約束。

同じ読み出しは SYNACK 再送タイマ (inet_connection_sock.c:1027 reqsk_timer_handler → tcp_rtx_synack) と経路検索 (inet_csk_route_req:770) にもある。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

20

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (20/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

呼出し順と RCU 区間の説明には、私も賛成です。再送 SYN を tcp_check_req() が受けると、保存済み ireq_opt を使って SYNACK を作ります。受信した再送 SYN 自体に IP オプションがあることは、この分岐の条件ではありません。末尾の「タイマーにも読み出しがある」は事実ですが、読み出すことだけで危険な競合が成立するわけではありません。通常タイマーの安全性と、28頁の残存状態の説明を区別して読むのが適切です。

判定：妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

- 【1】Linux v7.0：再送 SYN と要求ソケットの処理
- https://github.com/torvalds/linux/blob/v7.0/net/ipv4/tcp_minisocks.c
- 【2】Linux v7.0：IPv4 TCP の子ソケット作成と SYNACK 送信
- https://github.com/torvalds/linux/blob/v7.0/net/ipv4/tcp_ipv4.c
- 【3】Linux カーネル CVE チームによる脆弱性公表
- <https://lists.openwall.net/linux-cve-announce/2026/10/15/1>

二つの解放箇所と kfree_rcu

【Claude Mythos 5.1 の作成したスライド (21/41)】

ソース

net/ipv4/af_inet.c — inet_sock_destruct()

```
141 void inet_sock_destruct(struct sock *sk)
142 {
143     struct inet_sock *inet = inet_sk(sk);
144     __skb_queue_purge(&sk->sk_receive_queue);
145     __skb_queue_purge(&sk->sk_error_queue);
146     sk_mem_reclaim_final(sk);
147     /* ... 状態の健全性検査 (WARN_ON_ONCE 等、省略) ... */
165 kfree(rcu_dereference_protected(inet->inet_opt, 1)); /* ★ 根源 1: 即時解放 */
166 dst_release(rcu_dereference_protected(sk->sk_dst_cache, 1));
167 dst_release(rcu_dereference_protected(sk->sk_rx_dst, 1));
168 }
```

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

根源 1 (child 側): child の最後の参照が落ちると sk_destruct → ここへ。MPTCP の dispose_child 経路では ACK 処理と同じ softirq 内で到達する。

修正後: kfree_rcu(rcu_dereference_protected(inet->inet_opt, 1), rcu); — 既存の rcu_head を使って解放を猶予期間後に遅延。

net/ipv4/tcp_ipv4.c — tcp_v4_reqsk_destructor()

```
1228 /* IPv4 要求ソケットのデストラクタ */
1230 static void tcp_v4_reqsk_destructor(struct request_sock *req)
1231 {
1232     kfree(rcu_dereference_protected(inet_rsk(req)->ireq_opt, 1)); /* ★ 根源 2: 即時解放 */
1233 }
```

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

根源 2 (req 側): 要求ソケットの最後の参照 (reqsk_put) で呼ばれる。移行で OPT を共有した nreq が落ちる時も同じ kfree。修正後は kfree_rcu(..., rcu)。

upstream e3f33b0a1d89 (Eric Dumazet) / stable 11a03ce3c4a6 — Fixes: 079096f103fa (4.4), c905dee62232 (5.14) / Reported-by: Xinyang Ge (Anthropic)

暫定対策はこの 2 行を変えられない。だから『解放される物を作らせない』(対策 a) か『解放を起こすレバーを外す』(対策 b/c) のどちらかになる。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

21

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (21/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

二つの解放箇所を並べた説明は、きちんと核心を押さえています。inet_sock_destruct() と tcp_v4_reqsk_destructor() の plain kfree を、既存の rcu_head を使う kfree_rcu に置き換えた差分は公式修正と一致します。子ソケットと、移行時に複製された要求ソケットを別の所有経路として示した点も適切です。ソケットの参照数がゼロでも、別オブジェクトを介する RCU 読者の終了までは保証しない、という理解で正しいです。

判定：妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

【1】IPv4 オプションを RCU 猶予期間後に解放する修正
<https://github.com/torvalds/linux/commit/e3f33b0a1d89d738e2b913dab519226eaf76e15b>

【2】Linux v7.0 : IPv4 ソケットの破棄
https://github.com/torvalds/linux/blob/v7.0/net/ipv4/af_inet.c

【3】Linux v7.0 : IPv4 TCP の子ソケット作成と SYNACK 送信
https://github.com/torvalds/linux/blob/v7.0/net/ipv4/tcp_ipv4.c

共通注意：暫定策は更新までの措置です。更新できる場合は、修正済みカーネルへの更新を優先してください。

21 / 41

範囲外書込みと「任意」の区別

【Claude Mythos 5.1 の作成したスライド (22/41)】

ソース (5) 消費者: ip_build_and_send_pkt() / ip_options_build() が解放済みメモリをどう使うか

ソース

net/ipv4/ip_output.c — ip_build_and_send_pkt() 抜粋

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

```
160  /* IP ヘッダを組み立てる */
161  skb_push(skb, sizeof(struct iphdr) + (opt ? opt->opt.optlen : 0)); /* ★ 偽 optlen で溢れ */
168  iph->daddr  = (opt && opt->opt.srr ? opt->opt.faddr : daddr); /* ★ 偽 srr → 宛先改変 */
...  /* ... version / ihl / tos / ttl / id 等の設定 (省略) ... */
186  if (opt && opt->opt.optlen) {
187      iph->ihl += opt->opt.optlen>>2;
188      ip_options_build(skb, &opt->opt, daddr, rt); /* ★ 解放済み OPT を使用 */
189  }
```

DoS の芽: 再利用後の optlen が大きいと skb_push がヘッドルームを超え skb_under_panic() → カーネルパニック。

ここから下が『解放済みメモリの消費者』。

net/ipv4/ip_options.c — ip_options_build()

Linux v7.0 (Ubuntu 26.04 LTS が採用する系統)

```
44  void ip_options_build(struct sk_buff *skb, struct ip_options *opt,
45      __be32 daddr, struct rtable *rt)
46  {
47      unsigned char *iph = skb_network_header(skb);
49      memcpy(&(IPCB(skb)->opt), opt, sizeof(struct ip_options));
50      memcpy(iph + sizeof(struct iphdr), opt->__data, opt->optlen); /* ★ 越境読み → 漏洩 */
51      opt = &(IPCB(skb)->opt);
53      if (opt->srr)
54          memcpy(iph + opt->srr + iph[opt->srr + 1] - 4, &daddr, 4); /* ★ 任意位置へ 4B 書込 */
56      if (opt->rr_needaddr)
57          ip_rt_get_source(iph + opt->rr + iph[opt->rr + 2] - 5, skb, rt); /* ★ 同上 */
...  /* ... ts_needaddr / ts_needtime も同様に 4 バイトを書く (省略) ... */
66  }
```

情報漏洩の機構: 攻撃者制御の optlen 分だけ隣接ヒープを SYNACK のオプション欄へコピー → KASLR 突破に必要なカーネルポインタ等が流出。

書込みの機構: srr と長さバイト iph[srr+1] は共に攻撃者制御 → skb データ領域の外 (隣の slab 物) に daddr (任意 4 バイト) を書ける。

注: これらの行自体は正しいコード。『opt が生きている』前提が破られたときだけ凶器になる。修正後は opt の寿命が読者区間を覆うので無害に戻る。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

22

「任意」は過大。位置は相対範囲、値は相手アドレスに制約。

【ChatGPT 6 Astra Pro による意見 (22/41)】

Mythos 5.1 が作った本ページの評点: 88 点 / 100 点

UAF が情報漏洩や範囲外書込みにつながる説明は妥当です。ただ、「任意位置」「daddr = 任意4バイト」とまで言い切るには、少し筆が進みすぎています。示された SRR の書込先は、IP ヘッダを基準に 8 ビットの値から求める有限の相対位置です。値も SYNACK の接続相手アドレスであり、RR/TS は経路由来のアドレスや時刻を書きます。RCE の危険は否定しませんが、この抜粋だけから無制限の任意アドレス・任意値の書込み能力までは導けません。

判定：書込み能力の表現を要訂正 採点内訳：正確性 42/50、成立条件 26/30、根拠 20/20

【1】Linux v7.0 : IPv4 オプションの構築

https://github.com/torvalds/linux/blob/v7.0/net/ipv4/ip_options.c

【2】Linux v7.0 : IPv4 オプション構造体の定義

https://github.com/torvalds/linux/blob/v7.0/include/net/inet_sock.h

【3】Linux v7.0 : IPv4 TCP の子ソケット作成と SYNACK 送信

https://github.com/torvalds/linux/blob/v7.0/net/ipv4/tcp_ipv4.c

共通注意：暫定策は更新までの措置です。更新できる場合は、修正済みカーネルへの更新を優先してください。

22 / 41

公表された実証と図の具体化

【Claude Mythos 5.1 の作成したスライド (23/41)】



MPTCP が早期破棄を誘発する経路

【Claude Mythos 5.1 の作成したスライド (24/41)】



原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (24/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

MPTCP の役割を「解放側を誘発する経路」と整理した点は、よくできています。MP_JOIN の認証不成立から子ソケットを破棄する実装と、IPv4 オプションの寿命管理の欠陥を分けて説明しています。図注で公開された悪用コードの再現ではないと断っているため、非公開の実証手順を知っているとは扱いません。再送 SYN にオプションを付けた矢印は一例であり、そのオプションが読者の成立条件ではない点は、20・28頁と併せて理解できます。

判定：概念図として妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

【1】RFC 8684：複数アドレスによる Multipath TCP の仕様

<https://www.rfc-editor.org/rfc/rfc8684.html>

【2】Linux v7.0：MPTCP 子ソケットの作成と破棄

<https://github.com/torvalds/linux/blob/v7.0/net/mptcp/subflow.c>

【3】Claude による重要ソフトウェアの脆弱性発見と悪用の実証

<https://www.anthropic.com/research/discovering-vulnerabilities-with-claude>

MP_JOIN の検証失敗と子ソケットの破棄

【Claude Mythos 5.1 の作成したスライド (25/41)】

ソース

net/mptcp/subflow.c — subflow_syn_recv_sock() 抜粋

Linux v7.0 — net/mptcp/subflow.c:862-966 (抜粋)

```
862     if (child && *own_req) {
...         /* ... ctx 取得、drop_req = false (省略) ... */
871     if (!ctx || fallback) {
872         if (fallback_is_fatal) {             /* MP_JOIN は fallback 不可 */
874             goto dispose_child;           /* ★ 即時破棄へ */
877         }
902     } else if (ctx->mp_join) {
909         if (!subflow_hmac_valid(subflow_req, &mp_opt)) { /* ★ 攻撃者が故意に壊せる */
910             SUBFLOW_REQ_INC_STATS(req, MPTCP_MIB_JOINACKMAC);
912             goto dispose_child;
913         }
...         /* ... (省略) ... */
955     return child;
957 dispose_child:
958     mptcp_subflow_drop_ctx(child);
959     tcp_rsk(req)->drop_req = true;
960     inet_csk_prepare_for_destroy_sock(child);
961     tcp_done(child);           /* ★ 即座に閉じる → 参照解放で kfree(OPT) */
963     req->rsk_ops->send_reset(sk, skb, reason);
966     return child;           /* 最後の参照は呼出し側が解放 */
```

仕様として正しい: MP_JOIN の ACK が検証を
通らなければ、作ったばかりの child を
その場で捨てる。

攻撃者の『レバー』: 故意に壊した HMAC を
持つ ACK を送るだけで、望むタイミングで
dispose_child を踏ませられる。

tcp_done(child) → 呼出し側の sock_put で
refcnt 0 → inet_sock_destruct →
kfree(OPT)。これが競合の片側。

MPTCP 無効化 (net.mptcp.enabled=0 / Go の GODEBUG=multipathtcp=0) はこのレバーを外すだけ。根本の kfree は tcp_ipv4.c / af_inet.c 側に残る。

Go 1.24 以降の net.Listen は既定で MPTCP 待受を作る。GODEBUG=multipathtcp=0 / go.mod の godebug 行 //go:debug 指示で無効化できる (SetMultipathTCP 明示呼出しには効かない)。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

25

【ChatGPT 6 Astra Pro による意見 (25/41)】

Mythos 5.1 が作った本ページの評点: 98 点 / 100 点

この抜粋は、Mythos さんが根拠を丁寧に追えている箇所です。HMAC 検証不成立から dispose_child に入り、tcp_done() と呼出し側の参照解放で子ソケットを破棄する説明は、Linux v7.0 と一致します。Go の無効化設定と明示的 SetMultipathTCP が優先する点も正確です。なお実際の MPTCP 既定値は Go のビルド版だけでなく、go.mod／go.work の互換性設定にも左右されます。棚卸しではバージョン名だけで該当性を確定せず、実効設定と待受を確認してください。

判定：妥当・Go の実設定は確認 採点内訳：正確性 49/50、成立条件 29/30、根拠 20/20

- 【1】Linux v7.0：MPTCP 子ソケットの作成と破棄
<https://github.com/torvalds/linux/blob/v7.0/net/mptcp/subflow.c>
- 【2】Go セキュリティチーム：Go プログラム利用時の CVE-2026-98432 対策
<https://seclists.org/oss-sec/2026/q4/437>
- 【3】Go の互換性設定：GODEBUG の既定値と設定方法
<https://go.dev/doc/godebug>

要求ソケット移行と BPF の条件

【Claude Mythos 5.1 の作成したスライド (26/41)】

原理

要求ソケット移行 (tcp_migrate_req, Linux 5.14+) が作るもう 1 つの共有

oreq (旧 listener)
ireq_opt →
閉じられた listener に属する req

memcpy (丸ごと複製)

nreq (新 listener)
ireq_opt →
inet_reqsk_clone() で複製

ip_options_rcu
OPT (1 つだけ)
2 つの req から指される

reqsk_migrate_reset() が片方を NULL にして二重解放は防ぐが、NULL 化は plain store で、他 CPU の読者は古いポインタを持ち続ける。

その後 nreq 側が破棄されると tcp_v4_reqsk_destructor() の kfree() で OPT が消え、oreq を参照中の読者が UAF に陥る。

成立条件: SO_REUSEPORT のグループ内で listener を close() する運用 (graceful restart 等) + sysctl net.ipv4.tcp_migrate_req=1
既定値は 0 (無効)。有効化している環境では、暫定対策として 0 に戻すことで『この共有経路』を閉じられる。
修正コミットはこの経路も Fixes: c905dee62232 として明示している (『SYN+ACK 再送時の移行』を導入したコミット)。

確認と設定

現在値を確認 (netns ごと)
sysctl net.ipv4.tcp_migrate_req
net.ipv4.tcp_migrate_req = 0 # ← 既定

1 なら暫定的に 0 へ
sudo sysctl -w net.ipv4.tcp_migrate_req=0

MPTCP ソケットの作成を禁止 (対策 b)
sudo sysctl -w net.mptcp.enabled=0

永続化 (/etc/sysctl.d/99-cve-2026-98432.conf)
net.mptcp.enabled = 0
net.ipv4.tcp_migrate_req = 0

既存の MPTCP 待受を確認 (iproute2)
ss -Mlnp # MPTCP 待受ソケット一覧

ソースの要点 (net/ipv4/inet_connection_sock.c)

918-958 行 inet_reqsk_clone(): req を obj_size 分 memcpy → ireq_opt のポインタも複製される。968-977 行 reqsk_migrate_reset(): 片方を NULL にする (CONFIG_IPV6 有効時は union 経由)。1027 行〜 reqsk_timer_handler(): SYNACK 再送時に listener が閉じていれば移行する。

運用上の判断

net.ipv4.tcp_migrate_req は既定 0 (無効)。有効化している環境 (SO_REUSEPORT + graceful restart の運用) では、暫定的に 0 へ戻すことでこの共有経路を閉じられる。無効化しても MPTCP 経路とは独立なので、対策 a/b/c と併用する。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

26

0 だけでは閉じません。
移行用BPFの有無・
方針も確認。

【ChatGPT 6 Astra Pro による意見 (26/41)】

Mythos 5.1 が作った本ページの評点: 76 点 / 100 点

共有ポインタを複製してしまう図は正確ですが、運用上の結論はこのまま採用できません。
tcp_migrate_req=0 が止めるのは通常の移行方針です。BPF_SK_REUSEPORT_SELECT_OR_MIGRATE 型のプログラムが移行を選ぶ場合は、この値が 0 でも経路が残ります。これは推測ではなく、CVE 公表と sock_reuseport.c に明記されています。「0 へ戻せば閉じる」と言う前に、SO_REUSEPORT に付いた BPF と移行方針まで確認する一手が必要です。

判定：移行停止条件に重要な漏れ 採点内訳：正確性 42/50、成立条件 14/30、根拠 20/20

【1】Linux カーネル CVE チームによる脆弱性公表
<https://lists.openwall.net/linux-cve-announce/2026/10/15/1>

【2】Linux v7.0 : SO_REUSEPORT のソケット移行処理
https://github.com/torvalds/linux/blob/v7.0/net/core/sock_reuseport.c

【3】Linux カーネル文書：IP の sysctl 設定
<https://docs.kernel.org/networking/ip-sysctl.html>

適用手順と ACCEPT の意味

【Claude Mythos 5.1 の作成したスライド (27/41)】

対策 a の適用手順: 即時適用 → 確認 → 永続化 → 再起動 → 監視

即実行

1 即時適用

nftables: sudo nft -f /etc/nftables.d/cve-2026-98432.nft
iptables: 前掲の 1 行 (raw テーブル、PREROUTING の先頭)。混在環境は nft list ruleset で全体を確認

2 動作確認

sudo nft list table inet cve_2026_98432 (counter の増加) / sudo iptables -t raw -L PREROUTING -v -n
検証: 別ホストから IP オプション付き SYN を送る (例: nping --ip-options R, hping3, scapy)
受信側で tcpdump -ni eth0 'ip[0] & 0x0f > 5' を観察し、SYNACK が返らないことを確認

3 永続化

nftables: /etc/nftables.conf に include を追加し systemctl enable nftables
iptables: iptables-persistent / netfilter-persistent (iptables-save > /etc/iptables/rules.v4)
RHEL 系: nftables.service。firewalld 利用時は独自 table との共存を確認
ネットワーク有効化より前にルールが入る順序 (ユニット依存関係) を必ず確認

4 再起動

永続化を確認してからホストを再起動し、適用前に生まれた残存オブジェクトを消す (次スライド)
再起動できない場合は、影響を受け得る待受サービスの再起動で代替

5 監視・記録

counter の推移を監視 (急増は攻撃試行の兆候、少量の一致は正当な通信の可能性 → 送信元を確認)
変更記録に『CVE-2026-98432 暫定対策、カーネル更新後に撤去可』と明記

ルール配置の原則

- 『信頼できない通信が対象の TCP 処理に到達する前』に有効であること。起動直後にネットワークが上がってからルールが入る構成だと、その隙間が無防備になる。
- PREROUTING の先頭 (iptables: -I 1、nft: priority -310) に置く理由: 他のルール (ACCEPT や NAT) に先回りされないため。
- 既存の FW がすべての受信を ACCEPT していても、このルールは独立して効く (filter より前の段階で捨てる)。
- ネットワーク名前空間ごとに独立。コンテナ内・VM 内は別途 (スライド 32)。
- 撤去: 修正済みカーネルで再起動した後、ルールを削除してよい (残しても害は少ない)。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

27

別ベースチェーンの ACCEPT は最終許可ではない。

【ChatGPT 6 Astra Pro による意見 (27/41)】

Mythos 5.1 が作った本ページの評点: 90 点 / 100 点

適用・確認・永続化・監視の手順は実用的です。ただ、優先度 −310 の理由を「他の ACCEPT に先回りされないため」とする説明は、nftables の判定規則と合いません。先のベースチェーンで ACCEPT されても、後のベースチェーンは評価され、DROP は効きます。同一 iptables チェーンの手頭挿入とは分けて説明してください。検証では遮断カウンタとともに通常の TCP 接続も確認すると、単なるサービス停止と区別できます。再起動の代替条件は28頁の訂正を反映します。

判定：優先度の理由と再起動条件を要補足 採点内訳：正確性 46/50、成立条件 24/30、根拠 20/20

【1】nftables のチェーン設定と判定の扱い

https://wiki.nftables.org/wiki-nftables/index.php/Configuring_chains

【2】Netfilter のフックと処理優先順位

https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks

【3】Linux v7.0：再送 SYN と要求ソケットの処理

https://github.com/torvalds/linux/blob/v7.0/net/ipv4/tcp_minisocks.c

共通注意：暫定策は更新までの措置です。更新できる場合は、修正済みカーネルへの更新を優先してください。

27 / 41

残存する OPT と、再送 SYN の扱い

【Claude Mythos 5.1 の作成したスライド (28/41)】

なぜ再起動を推奨するのか: フィルタは『これから届くパケット』にしか効かない

即実行

時間 →

SYN(+opt) 受信

ルール適用

オプション無しの ACK

SYNACK 再送 (内部タイマ)

再起動 / ソケット終了

req と OPT が生成される

以後の IHL>5 TCP を遮断

ルールを通過して req を処理

パケット不要で OPT を読む

残存する OPT が消滅

ルール適用前に生まれた OPT は、SYNACK 再送上限 (既定 5 回・約 63 秒) か接続確立・終了まで生き続ける

適用後は『新しい』オプション付き SYN が届かず、新たな OPT は生まれない

・パケットフィルタは『これから通るパケット』にしか作用しない。既に ireq_opt を持つ要求ソケットや、オプション付きで確立した接続 (inet_opt) は残る。

・残存した OPT に対して、再送 SYN (オプション付き) はルールで落ちるため読者経路の 1 つは断たれるが、カーネル内部のタイマ経路や別の破棄経路の組合せを完全には否定できない。確実に初期状態へ戻す保守的な方法は『ルールを永続化してから再起動』(起動時にルールが先に入る構成にしておく)。

・再起動できない場合: 影響を受け得る待受サービスの再起動 (要求ソケット・接続の作り直し) と、nft counter / ss による残存状況の観測で代替する。

確実な方法

・ルールを永続化 → 起動時にネットワーク有効化より前に入ることを確認 → ホスト再起動。これで『適用前に生まれた OPT』は存在しなくなる。

・ルータで集約保護している場合も、保護対象ホスト側の残存状態はルータ再起動では消えない点に注意。

再起動が難しい場合の代替

・対象の待受サービス (特に MPTCP 待受、公開ポートのサーバ) を再起動し、要求ソケットと接続を作り直す。SYNACK 再送上限 (既定 5 回、約 63 秒) を過ぎれば残存する要求ソケットは消える。

・確立済み接続の inet_opt は接続終了まで残るが、その解放は『読者の居ない』close 時であり、攻撃者が新たに読者経路を作ることはフィルタで阻止される。

通常タイマー自体は本件競合から除外されている。

再送 SYN でタイマー更新。一律の消滅保証には使えない。

オプションなしの再送 SYN でも既存 OPT を読む。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

28

【ChatGPT 6 Astra Pro による意見 (28/41)】

Mythos 5.1 が作った本ページの評点: 63 点 / 100 点

残存状態を消す判断には賛成ですが、その理由付けは丁寧に直しておきましょう。通常のSYNACK再送タイマー自体は、参照保持と同期停止により公式公表が問題から除外しています。実際の注意点は、IPオプションなしの再送SYNでも既存reqのOPTを読み、フィルタを通過できることです。再送は期限を更新するため、既定の約63秒だけで残存状態の消滅を一律に保証できません。代替手順ではソケット継承を避け、関連する待受・要求・接続が完全に終了したことを確認してください。

判定：残存状態の安全性に重要な誤り 採点内訳：正確性 30/50、成立条件 13/30、根拠 20/20

【1】Linux カーネル CVE チームによる脆弱性公表

<https://lists.openwall.net/linux-cve-announce/2026/10/15/1>

【2】Linux v7.0：再送 SYN と要求ソケットの処理

https://github.com/torvalds/linux/blob/v7.0/net/ipv4/tcp_minisocks.c

【3】Linux カーネル文書：IP の sysctl 設定

<https://docs.kernel.org/networking/ip-sysctl.html>

共通注意：暫定策は更新までの措置です。更新できる場合は、修正済みカーネルへの更新を優先してください。

28 / 41

無効化設定の適用範囲と副作用

【Claude Mythos 5.1 の作成したスライド (29/41)】

対策 b / c: MPTCP と要求ソケット移行を無効化する (公開 PoC の前提を外す)

即実行

対策 b: sysctl (ネットワーク名前空間ごとに設定。コンテナ・netns を使う場合はそれぞれで実施)

```
sudo sysctl -w net.mptcp.enabled=0          # MPTCP ソケットの作成を禁止 (5.6 以降に存在)
sudo sysctl -w net.ipv4.tcp_migrate_req=0    # 要求ソケット移行を無効化 (5.14 以降。既定 0 なので確認が主)
printf 'net.mptcp.enabled = 0\nnet.ipv4.tcp_migrate_req = 0\n' | sudo tee /etc/sysctl.d/99-cve-2026-98432.conf # 永続化
ss -Mlnp          # 既存の MPTCP 待受ソケットを列挙 (iproute2)。該当サービスは再起動して待受を作り直す
```

対策 c: アプリケーション側で MPTCP 待受を作らせない (sysctl に触れない環境向け)

```
# Go 1.24 以降 (既定で net.Listen が MPTCP 待受) - いずれか 1 つ
GODEBUG=multipathhtcp=0 ./server          # 既存バイナリの変数で無効化 (再ビルド不要)
godebug multipathhtcp=0                   # go.mod (main モジュール) に記述して再ビルド
//go:debug multipathhtcp=0                # package main の直前に記述して再ビルド
# 注意: コードが ListenConfig.SetMultipathTCP(true) を明示していると上記は効かない
```

効果と限界

- 公開された RCE の前提 (MPTCP 待受 + 即時破棄のレバー) を外す。移行経路の共有も閉じる。
- 根本の kfree は残る。別の破棄経路での発動可能性は未確認 (否定もされていない)。

適用時の注意

- MPTCP を実際に使っている通信 (マルチパス経路の冗長化、一部のモバイル向け API) は単一パス TCP に戻る。
- sysctl は netns ごと。Docker/K8s では各 Pod/コンテナの netns で別途設定するか、イメージ側 (対策 c) で対応。
- 古いカーネルには net.mptcp.enabled が無い (= MPTCP 自体が無い) 場合がある。

確認コマンド

```
sysctl net.mptcp.enabled
net.ipv4.tcp_migrate_req

ss -Mlnp / nstat -az | grep -i mptcp
go version -m ./server | grep -i godebug
(Go バイナリの設定確認)
対策 a の counter と合わせて、攻撃試行の有無を観測する。
```

Linux カーネル CVE-2026-98432 脆弱性対処 - カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

29

移行用 BPF は別途確認。sysctl=0 だけでは不十分。

アプリのフォールバック次第。socket 作成失敗もある。

【ChatGPT 6 Astra Pro による意見 (29/41)】

Mythos 5.1 が作った本ページの評点: 77 点 / 100 点

MPTCP の設定例と既存待受を作り直す注意は適切です。ただ、26頁と同じく tcp_migrate_req=0 だけで移行経路を閉じたとは判定できません。移行用 BPF を併せて確認してください。また「通信は単一パス TCP に戻る」は、アプリがフォールバックする場合の動作です。MPTCP 必須のアプリでは新規 socket 作成が ENOPROTOOPT で失敗し得ます。Go の設定も、実行時環境変数とコードの明示 API まで含めて確認し、バイナリ情報だけで無効化済みと決めないのが確実です。

判定：移行経路とフォールバックを要訂正 採点内訳：正確性 40/50、成立条件 17/30、根拠 20/20

【1】Linux v7.0 : SO_REUSEPORT のソケット移行処理

https://github.com/torvalds/linux/blob/v7.0/net/core/sock_reuseport.c

【2】Linux v7.0 : MPTCP 無効時のソケット作成

<https://github.com/torvalds/linux/blob/v7.0/net/mptcp/protocol.c>

【3】Go セキュリティチーム : Go プログラム利用時の CVE-2026-98432 対策

<https://seclists.org/oss-sec/2026/q4/437>

【4】Go の互換性設定 : GODEBUG の既定値と設定方法

<https://go.dev/doc/godebug>

RFC 7126 の勧告と一律遮断の区別

【Claude Mythos 5.1 の作成したスライド (30/41)】

副作用の評価: IPv4 オプションを使う TCP 通信は実在するか (RFC 7126 の推奨と照らす)

即実行

IPv4 オプション	用途	RFC 7126 (BCP 186) の推奨	本ルールによる影響と判断
LSRR (131) / SSRR (137) 送信元経路指定	送信元が経路を指定。歴史的にスプーフィング・ACL 迂回に悪用	既定で破棄すべき	遮断して問題なし (多くの機器が既に破棄)
RR (7) 経路記録 / TS (68) タイムスタンプ	経路診断 (ping -R / -T)。TCP で使う実運用はほぼ無い	既定で破棄してよい	遮断しても実害はほぼ無い (ping は ICMP なので対象外)
Router Alert (148)	RSVP、IGMP、PGM 等が中継ルータへの処理要求に使用	必要な場合のみ許可	TCP での使用は稀。RSVP-TE 等は TCP ではないため対象外
CIPSO (134) / 汎用セキュリティ	MLS 環境 (SELinux + NetLabel、Trusted Solaris 等) のラベル付け	閉域網内でのみ	CIPSO を TCP で使う閉域網では本ルールが通信を止める。かつカーネル内部生成経路があるため対策 a だけでは不十分 → 当該ホストは優先的にカーネル更新
Quick-Start / 実験的 (25, 30, 94 等)	実験・研究用	破棄してよい	遮断して問題なし

判断の仕方

- ルール適用前に `tcpdump -ni eth0 -c 1000 'ip[0] & 0x0f > 5 and tcp'` で、IP オプション付き TCP が実際に流れているか観測する (多くの環境で 0 件)。
- 適用後は counter の一致数と送信元を確認。正当な通信が含まれる場合は、送信元アドレスを限定した ACCEPT をこのルールの前に置くか (露出が増えるので非推奨)、当該ホストを先にカーネル更新する。
- インターネット境界では RFC 7126 の推奨どおり、IP オプション付きパケットを全面的に破棄する運用が以前から一般的。

結論

- 通常のサーバ・コンテナホスト・ルータでは、IPv4 オプションを用いる TCP 通信は事実上存在しないため、本ルールの副作用は極めて限定的。
- 仮に一部の通信が影響を受けても、カーネル更新までの間 CVE-2026-98432 から保護される利益が、オプション付き TCP が一時的に疎通しない不利益を上回る場合が多い。
- 必要な通信の有無と重要度を確認して判断する (観測 → 適用 → 監視)。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

30

CIPSOの勧告は、既定で削除・破棄しないこと。

Quick-Startの既定勧告は「無視」です。

RFCは種類別の方針。一律の全面破棄を推奨していません。

【ChatGPT 6 Astra Pro による意見 (30/41)】

Mythos 5.1 が作った本ページの評点: 70 点 / 100 点

副作用を観測して判断する姿勢は評価できます。ただ、RFC のお墨付きとして説明する部分は書き直しが必要です。RFC 7126 はオプション別に方針を示し、EOL・NOP だけを理由に破棄しないこと、CIPSO を既定で削除・破棄しないこと、Quick-Start は既定で無視することを勧告します。「推奨どおり全面破棄」や「CIPSO は閉域網内でのみ」は正確ではありません。本件の TCP 向け暫定遮断は個別の安全判断として述べ、一般的な IPv4 オプションの恒久方針と分けてください。

判定：RFC の一般勧告を誤って要約 採点内訳：正確性 38/50、成立条件 24/30、根拠 8/20

【1】RFC 7126 : IPv4 オプションを含むパケットのフィルタリング勧告
<https://www.rfc-editor.org/rfc/rfc7126>

境界ルータによる集約保護の前提

【Claude Mythos 5.1 の作成したスライド (31/41)】

インターネット / 信頼できないネットワーク (攻撃者)

SYN(+opt)
DROP

境界 Linux ルータ / FW

PREROUTING で本ルールを適用 (転送/パケットも PREROUTING を通る)

自身も保護 + 内側全体を集約保護

集約保護が効かない経路 (別途対策が必要)

- IPsec / WireGuard / VXLAN 等のトンネル: ルータは内側パケットを検査できない → 終端ホストで適用
- 内側に攻撃者がいる場合 (侵害済み端末、来客端末): ルータを経由しない → 各ホストで適用
- ルータを迂回する経路 (複数 ISP 回線、Wi-Fi 直結、SmartNIC オフロード経路) の有無を確認
- ルータの再起動は、保護対象ホスト側の残存状態 (既存 OPT) を解消しない

内側 LAN (信頼できる前提)

Linux サーバ A

未パッチでも SYN(+opt) が届かない
保護される

Linux サーバ B

同上
保護される

Docker ホスト

コンテナも同様に保護
保護される

Linux ルータ 2

内側の別セグメントへ
保護される

VM ホスト

ゲスト VM も同様
保護される

NAS / 組込機器

更新困難な機器にも有効
保護される

前提: 信頼できない所からの IPv4 パケットは必ずこのルータを通る。各ホスト側の残存状態の扱いは別途。

利点: 1 台への設定で、更新が難しい多数の Linux ホスト・機器のカーネルを暫定的に守れる。

成立条件のまとめ

(1) 信頼できない所からの IPv4 パケットが必ずこのルータの PREROUTING を通る (2) ルータ自身の Linux カーネルでルールが正しく評価される (3) トンネル (IPsec / WireGuard / VXLAN 等) の内側パケットは検査できないため終端ホストで別途適用 (4) 内側に攻撃者が居る前提なら各ホストでも適用 (5) 保護対象ホスト側の残存状態 (適用前に生まれた OPT) はルータの再起動では消えない

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

31

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (31/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

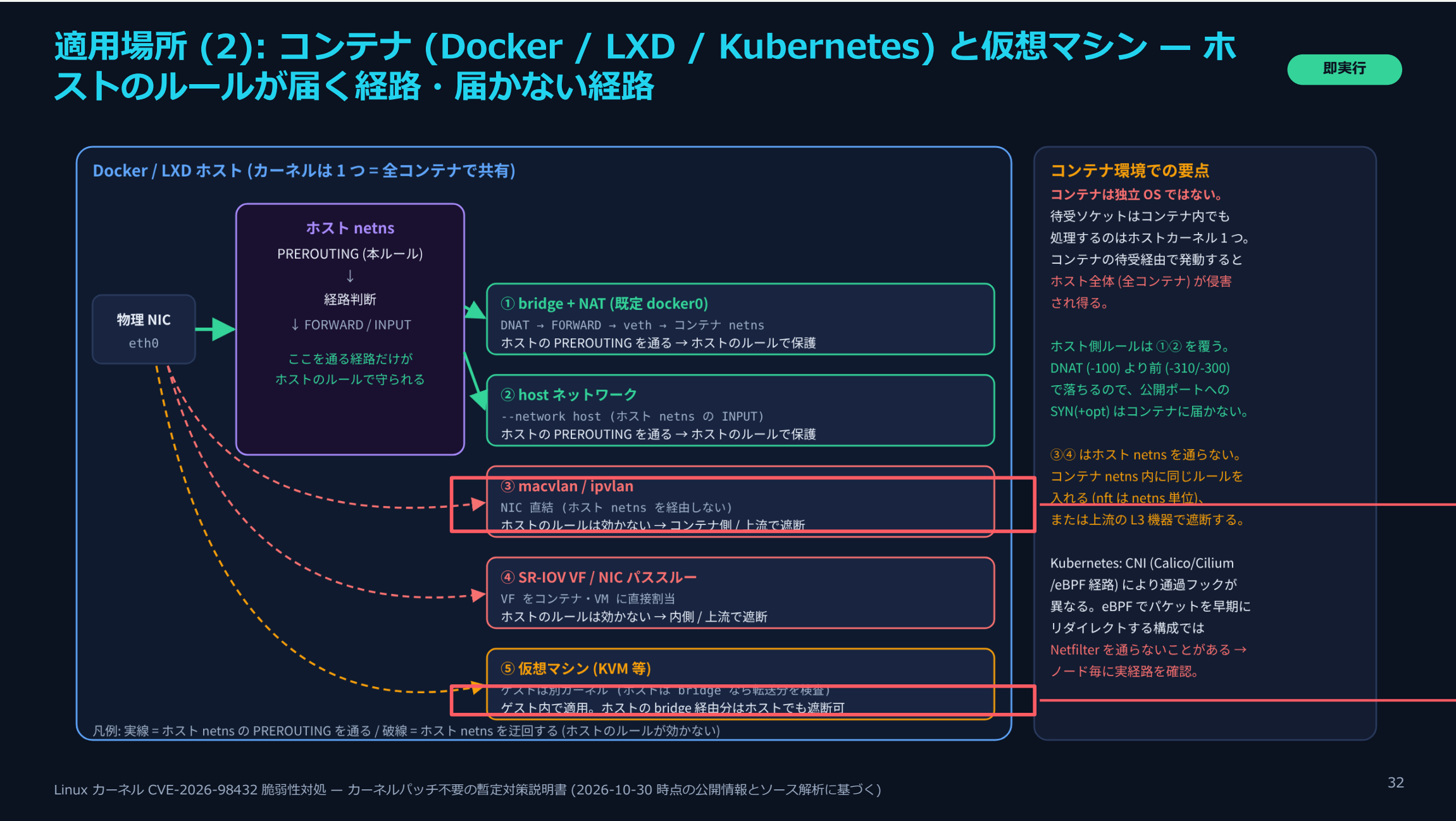
この集約保護の考え方には賛同します。ルーティング転送される IPv4 パケットも PREROUTING を通るため、図の前提を満たせば受信オプション由来の生成経路を入口で止められます。内部攻撃者、迂回経路、トンネル終端、保護先に残る状態まで挙げており、資料全体の NetLabel/CIPSO の留保とも整合します。トンネルの説明はこのルータを通過する通信を想定したものと読めます。ルータ自身が終端なら、復号・デカプセル化後の内側を検査する構成も評価対象になります。

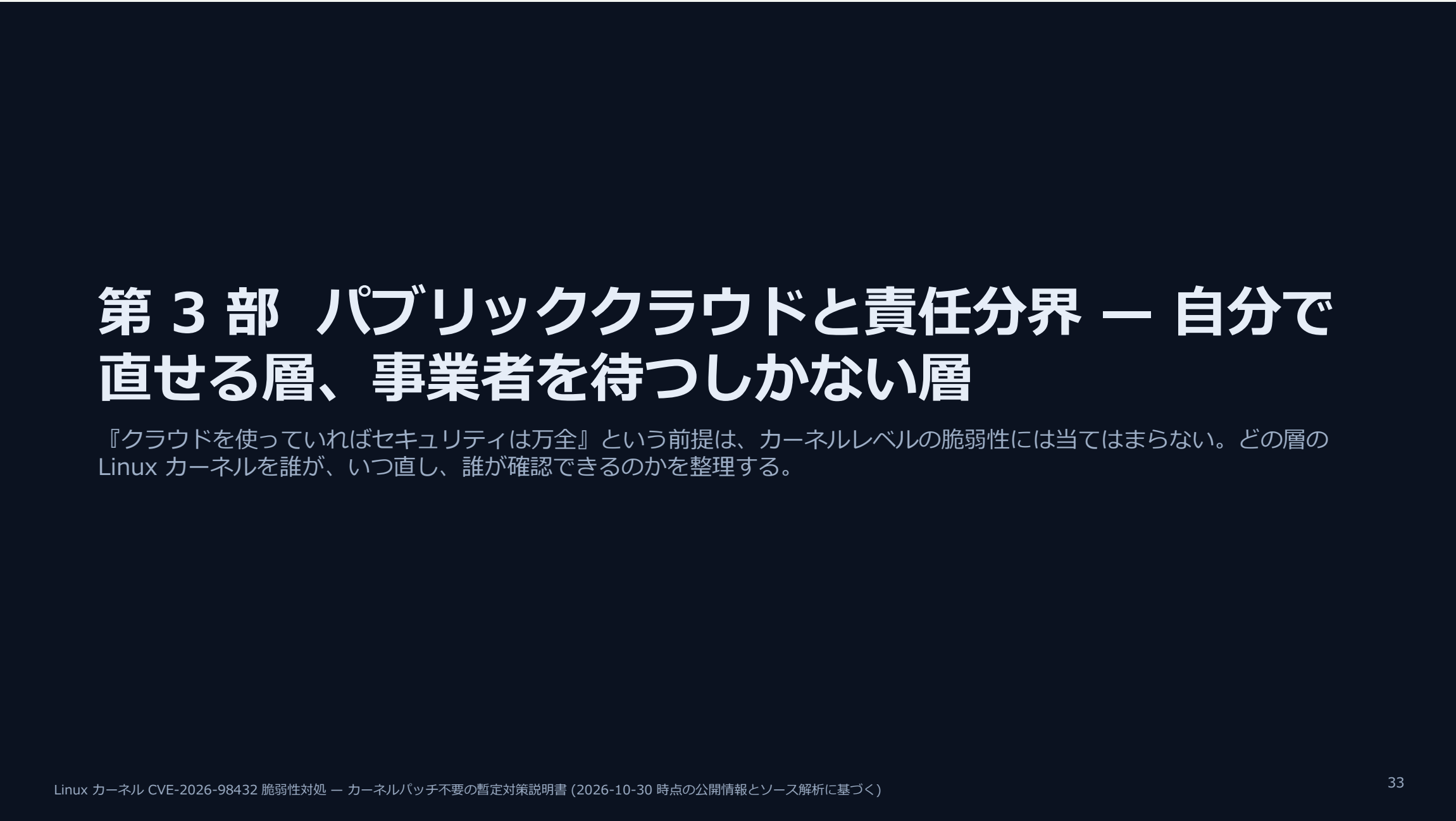
判定：明示された前提の下で妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

【1】Netfilter のフックと処理優先順位
https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks
【2】nftables のチェーン設定と判定の扱い
https://wiki.nftables.org/wiki-nftables/index.php/Configuring_chains

ブリッジ・ipvlan とフィルタの通過経路

【Claude Mythos 5.1 の作成したスライド (32/41)】





原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (33/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

この章の入口は堅実です。クラウド利用を安全の保証とみなさず、実際の実行層と変更権限を確かめる方針は、責任共有モデルと整合します。IaaSではゲストOS、PaaS/SaaSでは主に提供基盤を事業者が管理するという違いが、以後の判断の出発点です。事業者の対応確認を重視する姿勢は、安全側の運用方針として妥当であり、それ自体を減点しません。

判定：妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

【1】Microsoft Azure：クラウドにおける責任共有
<https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility>

変更権限を示す概念図

【Claude Mythos 5.1 の作成したスライド (34/41)】

層構造で見る CVE-2026-98432: 自分で対処できる層と、事業者の対応を待つしかない層

クラウド

パブリッククラウドの層構造と「誰が CVE-2026-98432 を直せるか」

	IaaS	PaaS	SaaS
アプリケーション / データ あなたのコード・SaaS のアプリ	自分で対処可	自分で対処可	事業者のみ
コンテナ / サンドボックス (gVisor, Firecracker 等) 共有カーネル or 専用 microVM。設計で分離度が大きく異なる	自分で対処可	事業者のみ	事業者のみ
ゲスト VM の Linux カーネル IaaS: あなたが管理 / PaaS・SaaS: 事業者が管理	自分で対処可	事業者のみ	事業者のみ
ハイパバイザ (ホストカーネル) AWS: KVM 派生 (ネットワークスタック無し) / Azure: Hyper-V (Windows) / GCP: KVM (Linux)	事業者のみ	事業者のみ	事業者のみ
SmartNIC / オフロードカード Nitro Card (組込み Linux の示唆あり) / Azure Boost / Google IPU: 事業者のみ	事業者のみ	事業者のみ	事業者のみ
共有サービス面 (LB, API, コンソール, IAM, ストレージ FE) 事業者の Linux フリートで稼働と合理的に推定される部分が多い	事業者のみ	事業者のみ	事業者のみ

緑 = 自分の手で本資料の対策 (フィルタ/設定/カーネル更新) を即日実施し、結果を自分で検証できる層。

赤 = 事業者の対応完了を待つしかない層。ここに残る脆弱性は、その上で動く全てのテナントのデータ・サービスに影響し得る。

ハイパバイザ層の直撃には『待受 TCP ソケットに SYN(+opt) が届く』ことが必要。AWS Nitro はホスト側に TCP スタックが無いと公表しており、Azure ホストは Windows ベースだが、周辺の Linux 部分 (カード、LB、PaaS ノード、管理面) の状況は顧客から見えない点は各社共通。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

34

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (34/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

層構造の整理は、今回の対策を誰が実施できるか考えるうえで役立ちます。とりわけPaaSのアプリケーション欄を『自分で対処可』とした点は、29頁のアプリ側MPTCP無効化と整合しています。下位基盤を利用者が直接修正できないことと、アプリや公開範囲を変更できることは両立します。この図の区別を36頁・39頁にも通していただければ、説明がすっきり揃います。

判定：概念図として妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

- 【1】Microsoft Azure：クラウドにおける責任共有
- <https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility>
- 【2】Azure App Service：アプリのプライベートエンドポイント
- <https://learn.microsoft.com/en-us/azure/app-service/overview-private-endpoint>

IaaS のゲスト対策と提供基盤の確認

【Claude Mythos 5.1 の作成したスライド (35/41)】

IaaS (ベアメタル / EC2 等) の結論: ゲストは自分で即日対処できるが、その下は見えない

自分でできること (ゲスト VM / ベアメタル)

- 対策 a / b / c をゲストカーネルに即日適用し、counter や ss で結果を自分で検証できる。
- 修正済みカーネル (ディストリの修正版、または Amazon Linux 2023 のように既に修正済みのもの) へ自分の日程で更新できる。
- ベアメタルではハイパバイザ層が存在せず、自分のカーネルがそのまま最下層になる (SmartNIC を除く)。

自分では見えない・触れないもの

- ハイパバイザ (ホスト) のカーネル、オフロードカードのファームウェア、ハイパバイザの管理ネットワーク、共有サービス面 (LB / API / コンソール / IAM / ストレージのフロントエンド)。
- これらに本脆弱性が残っていれば、ゲストでいくら対策しても、同じ物理ホスト上の全 VM と、そのサービスを使う全顧客に影響し得る。
- 顧客が原理的に対策できない領域が、今回のようなカーネル脆弱性ではかなり広く存在する。

ゼロトラストの前提で考える

- ハイパバイザの管理ネットワークには『既に攻撃バケットを発し得る攻撃者が居る』と仮定する (ゼロトラスト)。
- その仮定では、管理網に露出した未修正の Linux (ハイパバイザ、カード、管理面) は乗っ取りの対象になり得る。
- したがって事業者には、該当コードの有無と修正完了を顧客に開示する透明性が求められ、顧客はそれを確認する合理的な理由を持つ。

事業者への確認事項 (文例) — 完了報告が得られるまでは『安全』と判定しない

- Q1. 私の VM が稼働しているハイパバイザ (ホスト) の Linux カーネルは、Linux 4.4 以降で導入されたコミット 079096f103fa のコード (要求ソケットを ehash に登録する実装) を含んでいますか。含まない場合、その根拠 (非 Linux、該当経路が存在しない等) を示してください。
- Q2. 含む場合、CVE-2026-98432 の修正 (upstream e3f33b0a1d89 相当) は、私の VM を収容する全ホスト、オフロードカード、および私が利用する管理面・共有サービス面に適用済みですか。完了日時と適用範囲を示してください。
- 回答が『該当しない』であっても、その根拠 (例: ホストにネットワークスタックが存在しない) を確認する。完了報告書 (日付・範囲・担当) を入手し、変更記録に添付する。

法令・契約の観点

- 日本の個人情報保護法が求める安全管理措置 (委託先の監督を含む) や各種ガイドラインの観点から、委託先であるクラウド事業者の脆弱性対応状況を確認することは、顧客側の合理的な注意義務の一部と位置づけられる。
- 具体的な法的義務の範囲は法令・契約・業界ガイドラインに依存する。契約上の脆弱性通知義務・報告期限の有無を確認する。
- 本資料は法的助言ではない。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

35

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (35/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

利用者のゲスト対策と、事業者が管理する下位基盤の確認を分ける点に賛成です。『未修正なら影響し得る』という条件付きの説明であり、全事業者の侵害が実証済みだとは読めません。管理網に攻撃者がいる想定も、明示された脅威モデルとして許容できます。37・38頁は非Linux層や待受経路の不在を区別しているため、本頁だけを切り出してクラウド全体の一律危険宣言と評価する必要はありません。

判定：条件を置いた説明として妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

[1] Microsoft Azure : クラウドにおける責任共有

<https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility>

[2] Azure ホストOS : Cloud Host

<https://techcommunity.microsoft.com/blog/windowsplatform/azure-host-os-%E2%80%93-cloud-host/3709528>

[3] OpenHCL : 新しいオープンソースのパラバイザ

<https://techcommunity.microsoft.com/blog/windowsplatform/openhcl-the-new-open-source-paravisor/4273172>

PaaS／SaaS で利用者ができる緩和

【Claude Mythos 5.1 の作成したスライド (36/41)】

クラウド

直撃

波及

リスク

直撃のリスク

- PaaS / SaaS では、顧客のコードやデータを処理する Linux カーネルは事業者が管理する。パケットフィルタも sysctl もカーネル更新も、顧客は触れない。
- 事業者が修正を完了するまでの間、そのカーネルで待ち受ける公開サービス (API、Web フロント、LB) は、本脆弱性による直撃を受け得る。
- 顧客にできる『技術的対策』は原理的に存在しない。できるのは、置くデータと使

- カーネルが乗っ取られると、そのカーネル上で動く全テナントのプロセス・メモリ・ストレージ接続が攻撃者の手に落ちる。共有カーネル型のコンテナ基盤では被害範囲 = ノード全体。
- 分離の設計により範囲は異なる: microVM (Firecracker)、gVisor (ユーザ空間 TCP スタックで、ホストカーネルの該当経路を使わない)、VM 単位分離は被害を限定し得る。
- ただし、どの設計が自分のテナントに使われ、その基盤カーネルが修正済みかは、事業者の表明がなければ顧客は知り得ない。

現実性

実効策

現実性のある実効策 (事業者が『基盤の Linux カーネルについて CVE-2026-98432 の対策を完了した』と責任をもって表明するまで)

- ① 機密性・完全性・可用性を要するデータを、当該 PaaS / SaaS およびそこからアクセス可能な場所に新たに保管しない。既存データは暗号化・アクセス制限・退避を検討する。
- ② 当該サービスから自組織の他システムへ到達できる認証情報 (API キー、サービスアカウント、VPN) の範囲を最小化し、横展開の経路を絞る。
- ③ 事業者に対し、対策完了の表明 (日付・範囲) と、未完了の間の補償的統制 (監視強化、ネットワーク分離) の説明を求める。契約上の報告義務を確認する。
- ④ 代替手段 (自前基盤、修正済み IaaS 上での自前運用) への一時的な切替可否を評価する。
- これは過剰反応ではなく、『顧客側に対策手段が無い』場合に残された唯一の現実的な方法である。表明が得られ次第、通常運用に戻す。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

36

基盤の修正権限と利用者の緩和手段を区別

29頁・34頁と矛盾。PaaSアプリの設定変更は可能

【ChatGPT 6 Astra Pro による意見 (36/41)】

Mythos 5.1 が作った本ページの評点: 74 点 / 100 点

言い切りは明快ですが、『顧客の技術的対策は原理的に存在しない』は少々踏み込みすぎです。PaaSではアプリのMPTCP設定を変更できる場合があり、これは29頁・34頁が既に認めている対処です。Azure App ServiceにはPrivate Endpointと公開アクセス無効化の設定もあります。こうした措置だけで事業者の共有基盤全体の安全を保証できない点はそのとおりですので、『基盤の修正は事業者に依存するが、利用者側の緩和余地は製品による』と直すのが正確です。

判定：重要な断定の訂正が必要 採点内訳：正確性 34/50、成立条件 22/30、根拠 18/20

【1】Microsoft Azure：クラウドにおける責任共有

<https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility>

【2】Azure App Service：アプリのプライベートエンドポイント

<https://learn.microsoft.com/en-us/azure/app-service/overview-private-endpoint>

【3】Go 開発チーム：MPTCP の既定無効化をめぐる検討

<https://github.com/golang/go/issues/82717>

共通注意：暫定策は更新までの措置です。更新できる場合は、修正済みカーネルへの更新を優先してください。

36 / 41

事業者別の公開仕様と実行環境

【Claude Mythos 5.1 の作成したスライド (37/41)】

主要事業者の基盤に関する技術的推定 (公開情報に基づく。事業者の正式表明が優先する)

クラウド

観点	AWS	Microsoft Azure	Google Cloud
ハイパバイザ (ホスト OS)	Nitro Hypervisor: AWS は『KVM をベースに最小化したカスタムハイパバイザ』と公表 → Linux カーネル派生。ただし『ネットワークスタックを持たず EC2 ネットワークへのアクセスが無い』『シェェル無し』と明言 → SYN を受け付ける待受経路は無いと考えられる (AWS の説明が正確である限り)。	Azure Host OS 『Cloud Host』: Windows (OneCore) ベースの専用 SKU + Hyper-V (Type-1) → Linux ではないため CVE-2026-98432 の直接の対象外。	KVM (Linux カーネル機能) + Google 独自のユーザ空間 VMM (QEMU 不使用、2017 年公表) → ホストは Linux カーネル。ホスト側の待受経路の有無は非公開。
オフロードカード / SmartNIC	Nitro Card: Annapurna Labs 設計 SoC + AWS 専用ファームウェア。AWS 求人情報は『Nitro システムの中核にある組み込み Linux 環境』に言及 → Linux ベースと推定。VPC データ面はカード側で処理し、カード自身の TCP 待受に顧客パケットは届かない設計と推定されるが、管理網側の露出は非公開 → 該当コードの有無・修正状況の確認が必要。	Azure Boost: FPGA + 専用 SoC (OS は非公開)。OpenHCL ハワバイザは各 VM 内 (VTL2) に『最小 Linux カーネル + Rust VMM』を含むが、VM 単位で動きテナント間共有ではない。	Google 製 NIC / IPU、Andromeda (ホスト上のユーザ空間データ面)。詳細は非公開。
共有サービス面 (LB / API / コンソール / IAM / ストレージ FE)	ALB/CLB は EC2 インスタンス上、NLB は Hyperplane。S3 / IAM / コンソール等のフロントエンドは Linux フリートで稼働と合理的に推定。未修正ならハイパバイザが無事でも管理面経由で波及し得る (技術的には『あり得る』)。	Azure Front Door / Load Balancer (Ananta 系 SDN) の実装 OS は非公開。多数の第一者サービスが Azure Linux (CBL-Mariner) 等の Linux 上で稼働していることは公表済み。	GFE (Google Front End) 等のフロントエンドは Linux (Borg) 上で TCP を終端。Maglev (L4 LB) はカーネルバイパス転送のため、転送経路自体は本 CVE の対象外だが、終端側は Linux。
サーバレス / PaaS 基盤	Lambda / Fargate: Firecracker (KVM ベース) microVM を Linux ホスト上で稼働。AL2023 のゲスト側は修正版あり。	App Service (Linux) / AKS / Container Apps: Linux VM。ホストは Hyper-V。	Cloud Run / App Engine: gVisor (ユーザ空間 TCP スタック) → アプリ側の TCP は該当経路を使わない設計。GKE ノードは Linux VM。
顧客が確認できること	AL2023 等ゲスト側の修正版の有無。基盤側は事業者の表明のみ。	Azure の Linux PaaS ノード / ゲストの修正状況。基盤側は事業者の表明のみ。	COS / GKE ノードイメージの修正状況。基盤側は事業者の表明のみ。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

37

公式資料はSELinux採用を公表済み

Cloud Run第2世代はmicroVM。一律gVisor判定は不可

【ChatGPT 6 Astra Pro による意見 (37/41)】

Mythos 5.1 が作った本ページの評点: 72 点 / 100 点

比較表の見通しはよいのですが、公開済みの仕様まで「非公開」にしてしまうのは惜しいところです。MicrosoftはAzure BoostのSoC上でSELinuxを使うと明記しています。またGoogle公式文書では、Cloud Run第1世代はgVisor、第2世代はLinux互換のmicroVMです。「Cloud Run／App EngineはgVisorなので該当経路を使わない」という一括判定は成立しません。世代・実行環境・実際の待受経路を分けてください。Linux採用だけで悪用可能性が確定するわけではありません。

判定：実行環境と公開事実の訂正が必要 採点内訳：正確性 33/50、成立条件 23/30、根拠 16/20

- 【1】Microsoft Azure Boost の概要とセキュリティ設計
<https://learn.microsoft.com/en-us/azure/azure-boost/overview>
- 【2】Cloud Run：サービスの実行環境の選択
<https://docs.cloud.google.com/run/docs/configuring/execution-environments>
- 【3】CodeRabbit が Google Cloud Run でAIコードレビューエージェントを構築した方法
<https://cloud.google.com/blog/products/ai-machine-learning/how-coderabbit-built-its-ai-code-review-agent-with-google-cloud-run/>

【Claude Mythos 5.1 の作成したスライド (38/41)】

よくある問い: 『Azure は Windows ベースだから安全か』『Nitro は Linux だから危険か』

クラウド

Azure のハイパバイザは本 CVE の影響を受けないか

- 結論: ホスト OS (Cloud Host = Windows) と Hyper-V 自体は Linux ではないため、CVE-2026-98432 の直接の対象ではない。
- 『ハイパバイザ層がこの CVE に該当するか』だけを切り出せば、Linux (KVM) 派生の AWS / GCP より該当性は低い。
- ただし『だから Azure の方が総合的に安全』とは言えない: (1) Windows には Windows の脆弱性がある (2) OpenHCL 内の最小 Linux、Azure Boost、Linux PaaS ノード、内部サービスなど Linux 層は存在する (3) それらの修正状況は顧客から見えない点は各社共通。

AWS Nitro は Linux 派生だから危険か

- 結論: Nitro Hypervisor は KVM ベース (Linux 派生) だが、AWS は『ネットワークスタック無し・EC2 ネットワークへのアクセス無し』と公表しており、本 CVE の発動に必要な『待受 TCP ソケットに SYN が届く』経路はハイパバイザ上に存在しないと考えられる。
- 一方、Nitro Card (組込み Linux の示唆あり)、Firecracker ホスト、共有サービス面は Linux であり、管理網側の露出や修正状況は顧客には見えない。
- 『Linux 派生 = 直撃可能』でも『Windows = 無関係』でもない。問うべきは『待受経路の有無』と『修正完了の確認可能性』。

正しい問いの立て方 — 事業者横断で同じ基準を当てる

- ① 自分が制御できない Linux カーネルはどこにあるか (ハイパバイザ / オフロードカード / 管理面 / LB / PaaS 基盤 / 内部サービス)。
- ② それぞれに『IPv4 TCP の待受ソケットへ、信頼できない所から SYN が届く経路』があるか (無ければ本 CVE の直撃は成立しない)。
- ③ 経路がある層について、該当コード (4.4 以降) の有無と修正完了を、事業者が日付と範囲を示して表明しているか。
- ④ 表明が無い層が自組織の機密データ・重要サービスに触れているなら、PaaS/SaaS の結論 (前々スライド) と同じ補償的統制を適用する。
- CVE 1 件の該当性で事業者の優劣を決めるのではなく、『確認できるか』を基準にする。確認できない層の広さそのものが、パブリッククラウド固有のリスクである。

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

38

原文・図の内容を確認

訂正を要する
赤入れ箇所は
ありません。

【ChatGPT 6 Astra Pro による意見 (38/41)】

Mythos 5.1 が作った本ページの評点: 100 点 / 100 点

ここは、製品名やOSの由来に引きずられない判断ができています。WindowsベースのCloud HostとHyper-VをLinuxの当該コードから区別し、OpenHCL・Azure Boostなど周辺のLinux層は別に確認する整理は妥当です。Linux由来でも、攻撃対象のTCP待受にパケットが到達しなければ同じ発動条件にはなりません。事業者の優劣を本CVEだけで決めず、該当コードと到達経路を確かめる説明として評価します。

判定：判断軸は妥当 採点内訳：正確性 50/50、成立条件 30/30、根拠 20/20

【1】 Azure ホストOS : Cloud Host

<https://techcommunity.microsoft.com/blog/windowsosplatform/azure-host-os-%E2%80%93-cloud-host/3709528>

【2】 OpenHCL : 新しいオープンソースのパラバイザ

<https://techcommunity.microsoft.com/blog/windowsosplatform/openhcl-the-new-open-source-paravisor/4273172>

【3】 Microsoft Azure Boost の概要とセキュリティ設計

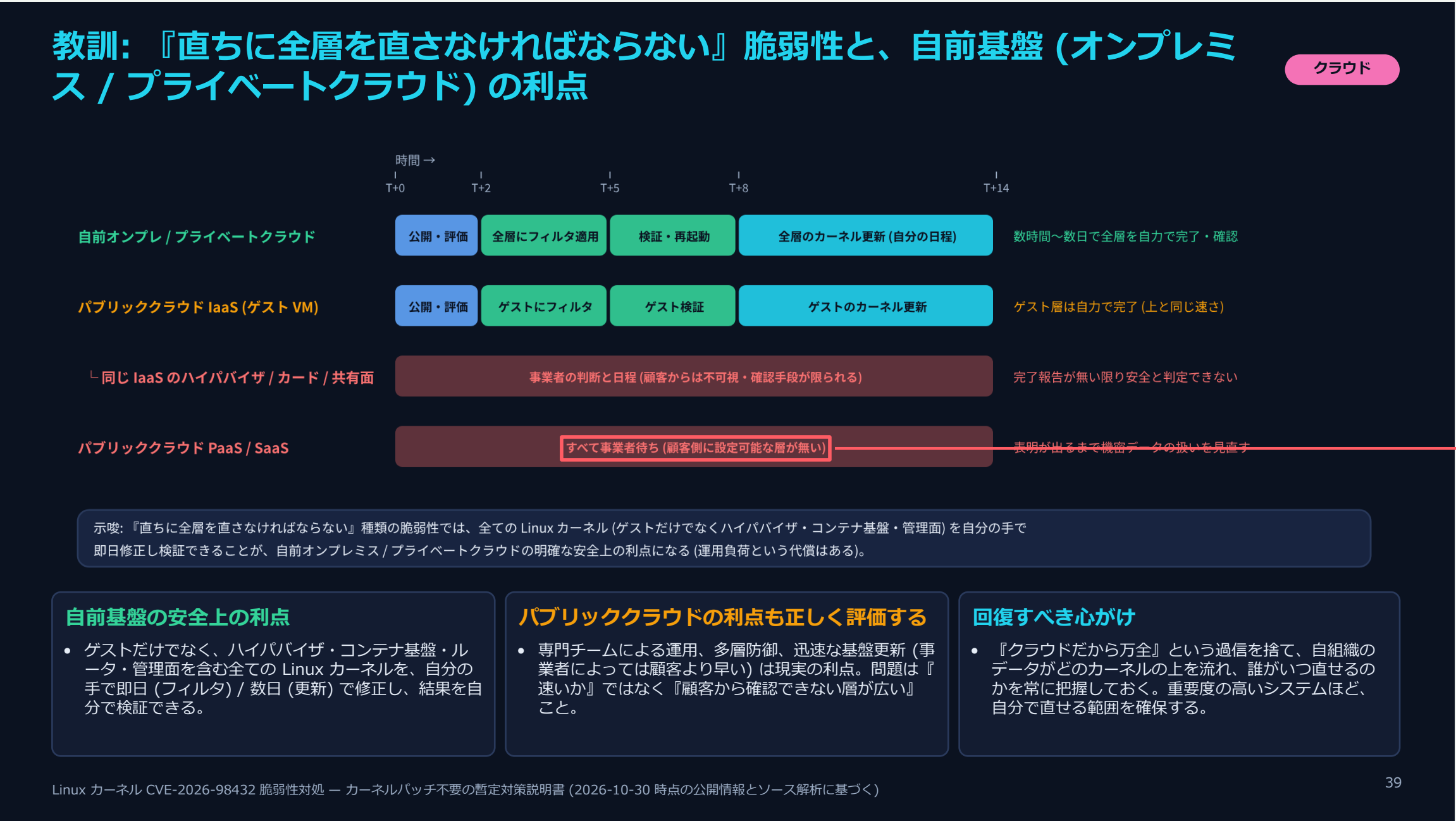
<https://learn.microsoft.com/en-us/azure/azure-boost/overview>

共通注意：暫定策は更新までの措置です。更新できる場合は、修正済みカーネルへの更新を優先してください。

38 / 41

自前運用の利点と PaaS の設定範囲

【Claude Mythos 5.1 の作成したスライド (39/41)】



実施チェックリストへの訂正反映

【Claude Mythos 5.1 の作成したスライド (40/41)】

対策チェックリスト: 今日 / 今週 / 恒久

即実行

今日 (暫定対策)

- 対策 a を全 Linux ホスト (サーバ、コンテナホスト、VM、ルータ) に適用し、counter / pkts で動作を確認
- 境界ルータにも対策 a を適用 (集約保護)、トンネルの終端ホストは個別に適用
- 対策 b (sysctl) と対策 c (Go の GODEBUG 等) を適用し、既存の MPTCP 待受を作り直す
- コンテナ: macvlan / ipvlan / SR-IOV / eBPF 経路の有無を確認し、必要なら netns 内に同じルール
- ルールを永続化し、起動順序 (ネットワーク有効化より前) を確認
- 計画再起動 (残存状態の解消)。できない場合は待受サービスの再起動
- tcpdump / counter で IP オプション付き TCP の有無と送信元を観測・記録

今週 (確認と調整)

- 各ディストリの CVE ページで修正版の公開状況を毎日確認 (Ubuntu / RHEL は 10/30 時点で未公開)
- Go 1.24+ 製サーバの棚卸し (go version -m) と、サードパーティ製品の MPTCP 利用有無の確認
- NetLabel / CIPSO を使う MLS 構成の有無を確認 (該当ホストは最優先でカーネル更新)
- クラウド事業者・ホスティング事業者へ Q1 / Q2 を送付し、完了報告書入手
- PaaS / SaaS に置く機密データの棚卸しと、事業者表明までの補償的統制の決定
- 監視: counter 急増、SYNACK 再送の異常、カーネル警告 (skb_under_panic 等) をアラート化

恒久 (根本対策と体制)

- 修正済みカーネル (安定版または各ディストリの修正版) へ更新し再起動。暫定ルールは撤去可
- RFC 7126 に沿い、境界での IP オプション付きパケット破棄を標準設定として維持
- 更新困難な Linux 機器 (NAS、アプライアンス、組込) の台帳化と、ルータ集約保護の恒常化
- 自前基盤 (オンプレ / プライベートクラウド) で『全層を自分で直せる範囲』を見直す
- 事業者との契約に脆弱性通知・対応期限・完了報告の条項があるか確認
- 本件を題材に、カーネル脆弱性時の初動手順 (フィルタ → 無効化 → 更新 → 確認) を手順書化

RFCは種類別の方針。一律の永久破棄義務ではない

BPFによる要求移行の有無も確認

28頁の残存状態の解消条件を反映

Linux カーネル CVE-2026-98432 脆弱性対処 — カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

40

【ChatGPT 6 Astra Pro による意見 (40/41)】

Mythos 5.1 が作った本ページの評点: 85 点 / 100 点

実施対象、永続化、トンネルや名前空間、NetLabelまで拾った一覧は実務向きです。ただし、26・29頁の要求移行にはBPF経路の確認を、28頁の代替手順には残存要求・子ソケットを確実に終了させる条件を反映してください。また『RFC 7126に沿ってIPオプション付きパケット破棄を恒久化』は一括りにできません。同RFCは種類別の方針を示し、EOOLやNOPだけを理由とする破棄を勧めていません。今回の暫定TCPフィルタと、恒久的な全プロトコルの境界方針は区別して記録するのが適切です。

判定：実施条件の訂正を反映する必要あり 採点内訳：正確性 44/50、成立条件 24/30、根拠 17/20

【1】Linux カーネル：IP の sysctl 設定と要求ソケット移行

<https://docs.kernel.org/networking/ip-sysctl.html>

【2】IPv4 オプションを含むパケットのフィルタリング推奨事項

<https://www.rfc-editor.org/rfc/rfc7126>

【3】Linux v7.0：再送SYNと要求ソケットの処理

https://github.com/torvalds/linux/blob/v7.0/net/ipv4/tcp_minisocks.c

共通注意：暫定策は更新までの措置です。更新できる場合は、修正済みカーネルへの更新を優先してください。

40 / 41

【Claude Mythos 5.1 の作成したスライド (41/41)】

参考文献 (一次情報を優先。URL は 2026-10-30 時点)

[CVE] Linux Kernel CVE Team, CVE-2026-98432: ipv4: free inet_opt and ireq_opt after an RCU grace period, 2026-10-15. <https://lists.openwall.net/linux-cve-announce/2026/10/15/1>

[FIX] E. Dumazet, ipv4: free inet_opt and ireq_opt after an RCU grace period, commit e3f33b0a1d89d738e2b913dab519226eaf76e15b (7.3-rc7). <https://github.com/torvalds/linux/commit/e3f33b0a1d89d738e2b913dab519226eaf76e15b>

[SRC] Linux v7.0 ソース: net/ipv4/tcp_ipv4.c, net/ipv4/af_inet.c, net/ipv4/tcp_minisocks.c, net/ipv4/inet_connection_sock.c, net/ipv4/ip_output.c, net/ipv4/ip_options.c, net/ipv4/ip_input.c, net/mptcp/subflow.c, net/netfilter/xt_u32.c, include/net/tcp.h, include/net/inet_sock.h (github.com/torvalds/linux, tag v7.0)

[ANTHROPIC] Anthropic Frontier Red Team, Discovering and exploiting vulnerabilities in critical, real-world software with Claude, 第 5 節, 2026-10-27. <https://www.anthropic.com/research/discovering-vulnerabilities-with-claude>

[GO-SEC] golang/go Issue 82717, proposal: net: Disable MPTCP by default (2026-10-29, neild 他コメント). <https://github.com/golang/go/issues/82717>

[NEWS] The Hacker News, Linux Kernel TCP Flaw With 9.8 CVSS Score Affects Ubuntu, Red Hat and Other Distributions (2026-10). Debian Security Tracker CVE-2026-98432. pwn.watch, linux-kernel-ipv4-tcp-uaf-cve-2026-98432.

[UBUNTU] Canonical Kernel Team, 26.04 LTS (Resolute Raccoon) shipping with the final 7.0 Linux kernel, 2026-04-23 (discourse.ubuntu.com)

[NFT/IPT] Netfilter Project: nft(8) マニュアル (netfilter.org); iptables-extensions(8) u32 節; xtables-nft(8); Netfilter hooks (wiki.nftables.org)

[RFC7126] F. Gont, R. Atkinson, C. Pignataro, Recommendations on Filtering of IPv4 Packets Containing IPv4 Options, BCP 186 / RFC 7126, 2014-02

[KDOC] Linux カーネル文書: Documentation/networking/ip-sysctl.rst (tcp_migrate_req), mptcp-sysctl.rst (enabled), networking/bridge.html; net/netlabel/netlabel_kapi.c (netlbl_req_setattr), net/ipv4/cipso_ipv4.c (cipso_v4_req_setattr)

[AWS] AWS Whitepaper, The Security Design of the AWS Nitro System (The Nitro System journey / The components of the Nitro System); Amazon 求人情報 Kernel/OS Engineer, AWS EC2 Nitro Operating System

~~**[AZURE]** Microsoft Tech Community, Azure Host OS - Cloud Host, OpenHCL: Evolving Azure's virtualization model (Windows OS Platform Blog)~~

~~**[CCP]** -Google Cloud Blog, 7 ways we harden our KVM hypervisor at Google Cloud (2017-01-26); D. E. Eisenbud et al., Maglev: A Fast and Reliable Software Network Load Balancer, NSDI 2016~~

Linux カーネル CVE-2026-98432 脆弱性対処 - カーネルパッチ不要の暫定対策説明書 (2026-10-30 時点の公開情報とソース解析に基づく)

41

Azure Boostの現行
公式文書も追加

Cloud Runの世代別
仕様を直接引用

【ChatGPT 6 Astra Pro による意見 (41/41)】

Mythos 5.1 が作った本ページの評点: 94 点 / 100 点

参考文献の骨格はしっかりしています。LinuxのCVE公表と修正、固定したv7.0ソース、Go開発者の議論を軸にする選択は適切です。ただし、34～39頁の製品別の説明には、Cloud Runの実行環境やAzure Boostの現行公式文書を直接結び付けたいところです。資料名やドメインだけの項目は、該当ページのURLと参照箇所まで示すと追跡が容易になります。二次報道を添えること自体は問題ありませんが、変化する修正状況と製品仕様は各ベンダーの一次資料で確かめる構成が望めます。

判定：主要一次資料は適切。追跡性に改善余地 採点内訳：正確性 50/50、成立条件 28/30、根拠 16/20

- 【1】Linux カーネルCVEチーム：CVE-2026-98432 の公表

<https://lists.openwall.net/linux-cve-announce/2026/10/15/1>
- 【2】Go 開発チーム：MPTCP の既定無効化をめぐる検討

<https://github.com/golang/go/issues/82717>
- 【3】Microsoft Azure Boost の概要とセキュリティ設計

<https://learn.microsoft.com/en-us/azure/azure-boost/overview>
- 【4】Cloud Run：サービスの実行環境の選択

<https://docs.cloud.google.com/run/docs/configuring/execution-environments>