生成AI時代の開発方法論と
メールサーバー運用の新しいかたち



HORNETSECURITY
BY proofpoint

Hornetsecurity株式会社
プリンシパルメッセージングエンジニア
平野 善隆

本日のトピック

- 自己紹介
- 開発方法論とは
- ソフトウェア工学がやってきたこと
- AI のプログラミングに適用すると
- 仕様を中心に考える
- 仕様を中心に置いたAIの実験
- Postfixの設定を AI で変更する実験
- Policy Runnerでの実験
- まとめ

HORNETSECURITY
BY proofpoint.

自己紹介

名前　　平野 善隆

所属　　Hornetsecurity株式会社 (旧Vade Japan)
Principal Messaging Engineer

好きな技術　　メール、DNS、Python、Go
AWS、Serverless

趣味　　長距離の自転車大会(1,200kmとか、2,000kmとか)
バンド演奏

主な活動　　M³AAWG
JPAAWG
迷惑メール対策推進協議会
Audax Randonneurs Nihonbashi





もう少し自己紹介

- 2018年 Scrum Inc.のScrum Master
- 2019年 Scrum Allianceの
  Certified Scrum Developer
- ただし更新していないので失効



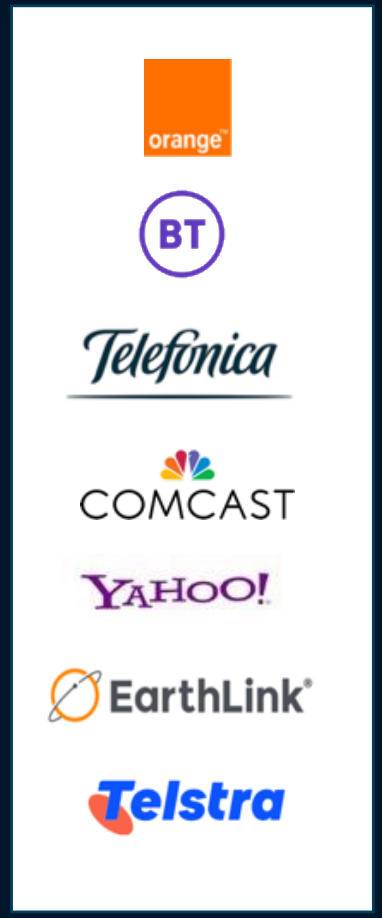
BY proofpoint.

HORNET SECURITY (VADE JAPAN)とは

設　立	2009年、フランス共和国リールにて設立 2024年 3月 ドイツ Hornetsecurityのグループとなる 2025年 12月 Proofpointのグループとなる
拠　点	ドイツ、フランス、アメリカ、カナダ、スペイン、イギリス、イタリア、北マケドニア、マルタ、日本など
社員数	約700名 ※Hornetsecurityグループ
国　内	2017年に日本市場に参入。日本国内にスレッドセンター設立
顧客数	エンドユーザ 約75,000社　／ パートナー 12,000社



保護しているメールボックス数



26億

全世界

1.5億

日本

HORNETSECURITY
BY proofpoint.

HORNETSECURITY
BY proofpoint.

コミュニティ 懇親会

肩の力を抜いて、語り合う夜に。

JPAAWG 9th General Meeting
最終日の夜に

日時

11月6日（金）
18:00～20:00

会場

大手町サンケイプラザ 3F
JPAAWGから徒歩3分
大手町駅直結

参加費

無料　申込不要

JPAAWGにご参加の方なら、途中からでも、どなたでも

HORNETSECURITY
BY proofpoint.



開発方法論とは

(本日の講演での) 開発方法論

- ウォーターフォール
- アジャイル
- スパイラル
- TDD (テスト駆動開発)
- DDD (ドメイン駆動開発)
- などなど



AIを使ったプログラミング

- Claude Code、 Codexなどの登場
- 数行の指示でアプリケーションが完成
- 想像以上に高品質





このアプリ、
ぼく、ひとりでできましたよ。
もう平野さんは、いりませんよ。

新井原さん

AIをプログラミングにどう活用するか

人間と共存

- 一部の関数を作らせる
- テストコードをかかせる
- コードのレビューをさせる
- リファクタリングをさせる

AIにまかせる

- バイブコーディング

HORNETSECURITY
BY proofpoint.

実際にAIにアプリを作らせると

- 小さなスクリプトは完璧に動作
- 変更を加えていくと
  - 関係ないところを修正する
  - さっきまで動いていた機能を壊す
  - 実装にやたらと時間がかかるようになる

うーん、昔、なんか見たことあるぞ

HORNETSECURITY
BY proofpoint.

人間での開発でも同じ

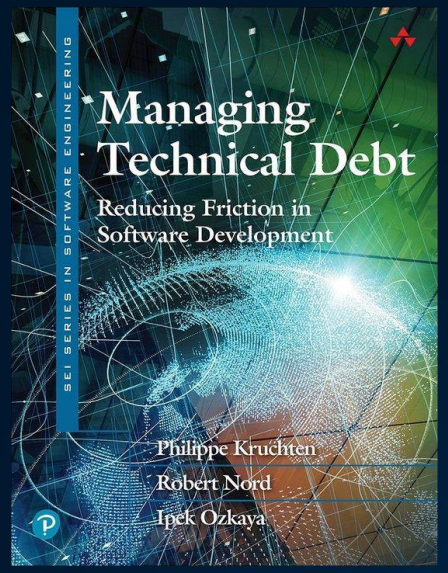
- とりあえず動くプロトタイプ開発
  ➔約2週間

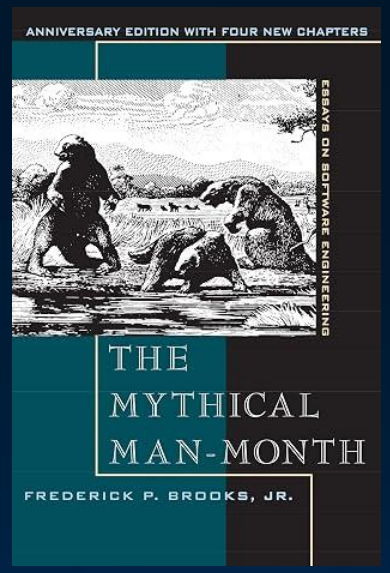
- もう少し使えるように、小さな機能を追加
  ➔約3週間

HORNETSECURITY
BY proofpoint.

みなさん経験しています

- 技術的負債の破綻
- セカンドシステム症候群
- Valley of Death





The Mythical Man-Month: Essays on Software Engineering

- MVP開発は非常に高速にできる
- その後の機能追加が極端に遅くなったり破綻する

Minimum Viable Product

顧客に価値を提供できる最小限の機能だけを備えたプロダクトやサービス

HORNETSECURITY
BY proofpoint.

技術的負債とは

- 時間はかかるがより良いアプローチを選択する代わりに、簡単ではあるが限定的な解決策を選択することで生じる、将来的な手直しにかかる暗黙のコストを示すものである

https://ja.wikipedia.org/wiki/%E6%8A%80%E8%A1%93%E7%9A%84%E8%B2%A0%E5%82%B5

HORNETSECURITY
BY proofpoint.

なぜそういうことが起きるのか

- 小さなスクリプトやMVP製品は見通しがいい
- 開発者が常に全体を把握できる
- コードをいつも最適な状態に保つことができる

★機能追加を繰り返すと・・・

- いつのまにか全体がわからなくなってくる
- 新たに作るより既存を調べる方が時間がかかる

HORNETSECURITY
BY proofpoint.

AIの場合

- 人間より多くの範囲を俯瞰できるが、限界はある
- セッションをまたぐと文脈を失いやすい
- 毎回コードの理解が必要
- 読み込み・推論・探索がコストになる

HORNETSECURITY
BY proofpoint.



HORNETSECURITY
BY proofpoint

ソフトウェア工学がやってきたこと

昔ながらの解決方法

- 複雑性を下げる
- 変更しやすくする
- 品質を維持する
- チームで開発しやすくする

AIに応用できるかな?

HORNETSECURITY
BY proofpoint.

ソフトウェア設計思想

- Dijkstra、構造化プログラミング
- モジュール化と情報隠蔽
- オブジェクト指向とSOLID
- DRYとデザインパターン
- Clean Codeという考え方
- Five Lines of Code

HORNETSECURITY
BY proofpoint.

Dijkstraと構造化プログラミング

- 1960年代、GOTO文による複雑な制御フローが問題に
- DijkstraはGOTO文の乱用を批判
- 順次・分岐・反復を基本として処理を構成
- プログラムの流れを追いやすくする
- 人間がプログラムの正しさを理解しやすくなる

HORNETSECURITY
BY proofpoint.

オブジェクト指向

- データとそれを操作する処理をひとまとめにする
- カプセル化によって内部の実装を隠す
- 継承やポリモーフィズムで共通性を表現
- 現実世界の概念をモデル化する方法として普及
- 大規模なソフトウェアの構造化に利用される

HORNETSECURITY
BY proofpoint.

SOLID原則

- Single Responsibility：単一責任の原則
  - 変更理由は一つにする
- Open/Closed：拡張に開き、修正に閉じる
  - 既存のコードをなるべく変更せずに機能を拡張できる
- Liskov Substitution：置換原則
  - 親を子に置き換えても正しく動作する
- Interface Segregation：インターフェース分離
  - 利用者に不要な機能への依存を強制しない
- Dependency Inversion：依存関係逆転
  - 具体的な実装ではなく抽象に依存する

HORNETSECURITY
BY proofpoint.

DRY原則

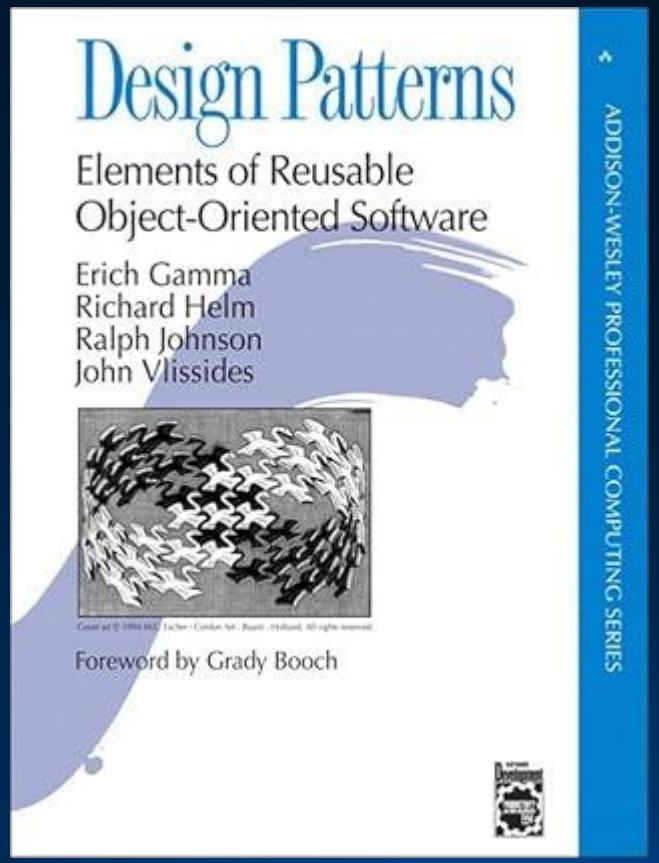
- Don't Repeat Yourself：同じ知識を重複させない
- 重複したロジックは修正漏れの原因になる
- 共通処理を一箇所にまとめる
- 修正すべき場所を減らし、整合性を維持する

- しかし、無関係な処理を無理に共通化することもある

HORNETSECURITY
BY proofpoint.

GoFデザインパターン

- 1994年GoFによって書籍化された23のデザインパターン
- 繰り返し現れる設計上の問題に解決策を与える
- Factory、Singleton、Facadeなど
- 設計者同士が共通の言葉で議論できる

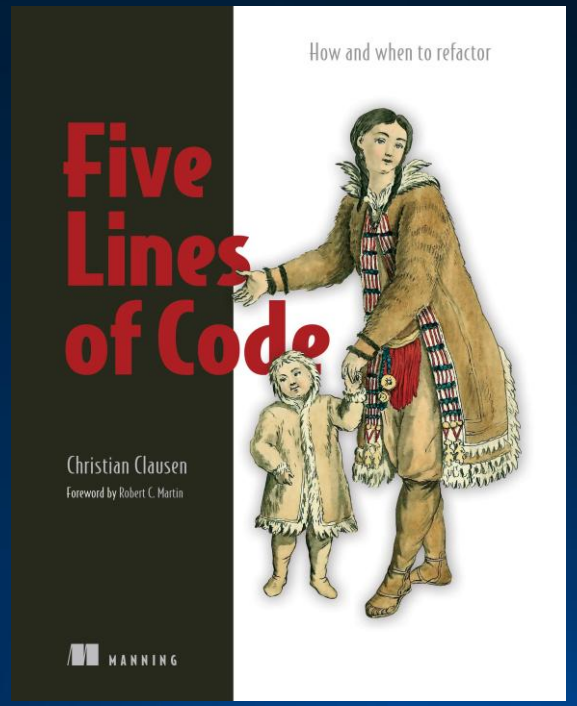


- 抽象クラスや間接参照が増えて複雑になることもある

HORNETSECURITY
BY proofpoint.

Five Lines of Code

- Christian Clausenが2021年に提唱したリファクタリング手法
- 1つのメソッドを原則5行以内にする
- 長い処理を小さなメソッドに分割する
- 名前を付けて処理の意味を表現する
- 人間が理解しやすいコードを目指す



HORNETSECURITY
BY proofpoint.



AIの
プログラミングに
適用

膨れ上がったソースコード

- 1つの関数が数百行
- 1つのファイルが数千行
- 読めなくはないが、なんとなく肌に合わないソースコード

HORNETSECURITY
BY proofpoint.

既存の理論を適用してみる

- Model View Controller に分割
  - 三層構造アーキテクチャ
- 関数の分割
- 責務の整理
- モジュール化
- リファクタリング

なんとなくいい感じに分割された

HORNETSECURITY
BY proofpoint.

きれいになったコードで機能追加

- 思ったより遅い
- いつもの3倍くらい時間とコストがかかっている

- これでは逆効果では?!

HORNETSECURITY
BY proofpoint.

1つの巨大関数とたくさんの分割された関数

10万行の関数

```
longlongfunction() {
    print("00000")
    print("00001")
    print("00002")
    print("00003")
    ....
    print("51002")
    print("51003")
    print("51004")
    ....
    print("99997")
    print("99998")
    print("99999")
}
```

100行ずつの1000個の関数

```
func_000() {
    print("00000")
    print("00001")
    print("00002")
    ....
}
```

```
func_001() {
    print("00100")
    print("00101")
    print("00102")
    ....
}
```

```
func_998() {
    print("99800")
    print("99801")
    print("99802")
    ....
}
```

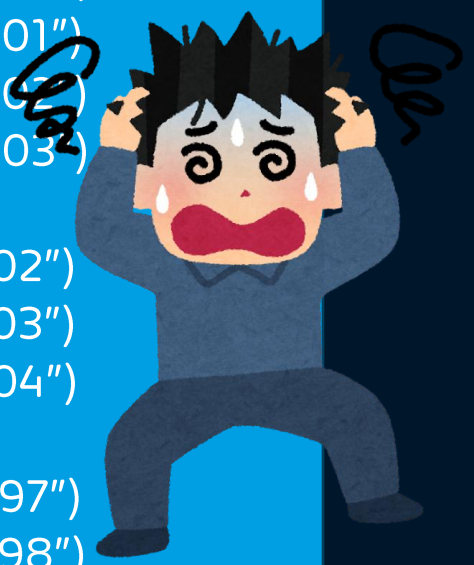
```
func_999() {
    ....
    print("99997")
    print("99998")
    print("99999")
}
```

HORNETSECURITY
BY proofpoint.

人間の場合

10万行の関数

```
longlongfunction() {
    print(“00000”)
    print(“00001”)
    print(“00002”)
    print(“00003”)
    ....
    print(“51002”)
    print(“51003”)
    print(“51004”)
    ....
    print(“99997”)
    print(“99998”)
    print(“99999”)
}
```



100行ずつの1000個の関数

```
func_000() {
    print(“00000”)
    print(“00001”)
    print(“00002”)
    ....
}
```

```
func_001() {
    print(“00100”)
    print(“00101”)
    print(“00102”)
    ....
}
```

```
func_998() {
    print(“99800”)
    print(“99801”)
    print(“99802”)
    ....
}
```



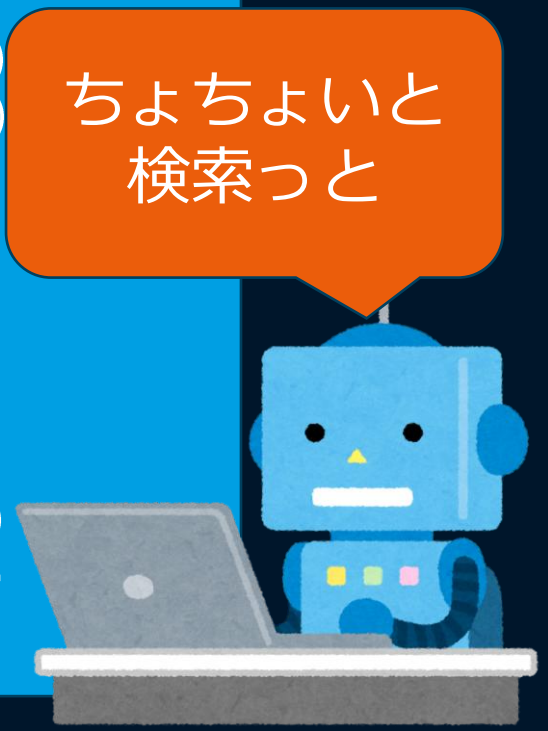

```
func_999() {
    ....
    print(“99997”)
    print(“99998”)
    print(“99999”)
}
```

HORNETSECURITY
BY proofpoint.

AIの場合

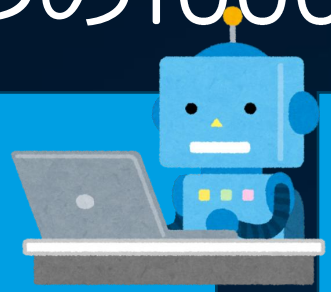
10万行の関数

```
longlongfunction() {
    print("00000")
    print("00001")
    print("00002")
    print("00003")
    ....
    print("51002")
    print("51003")
    print("51004")
    ....
    print("99997")
    print("99998")
    print("99999")
}
```

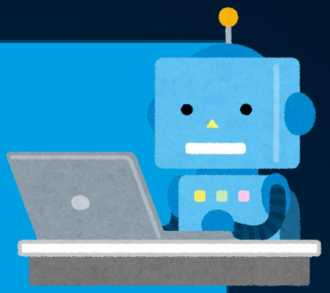


100行ずつの1000個の関数

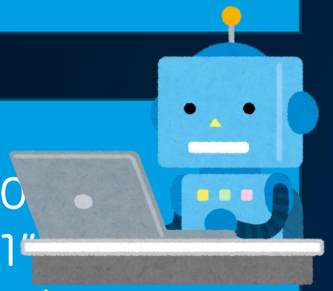
```
func_000() {
    print("00000")
    print("00001")
    print("00002")
    ....
}
```



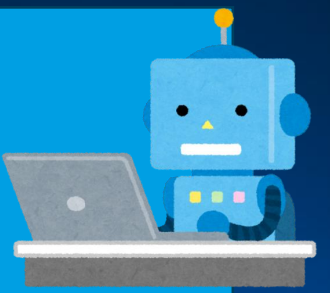
```
func_998() {
    print("99800")
    print("99801")
    print("99802")
    ....
}
```

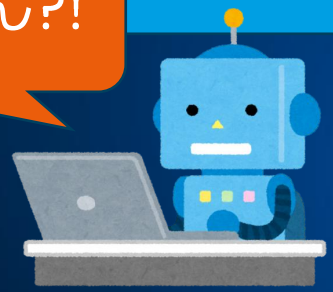


```
func_001() {
    print("00100
    print("00101
    print("00102")
    ....
}
```



```
func_999() {
    ....
    print("99997")
    print("99998")
    "99999")
```

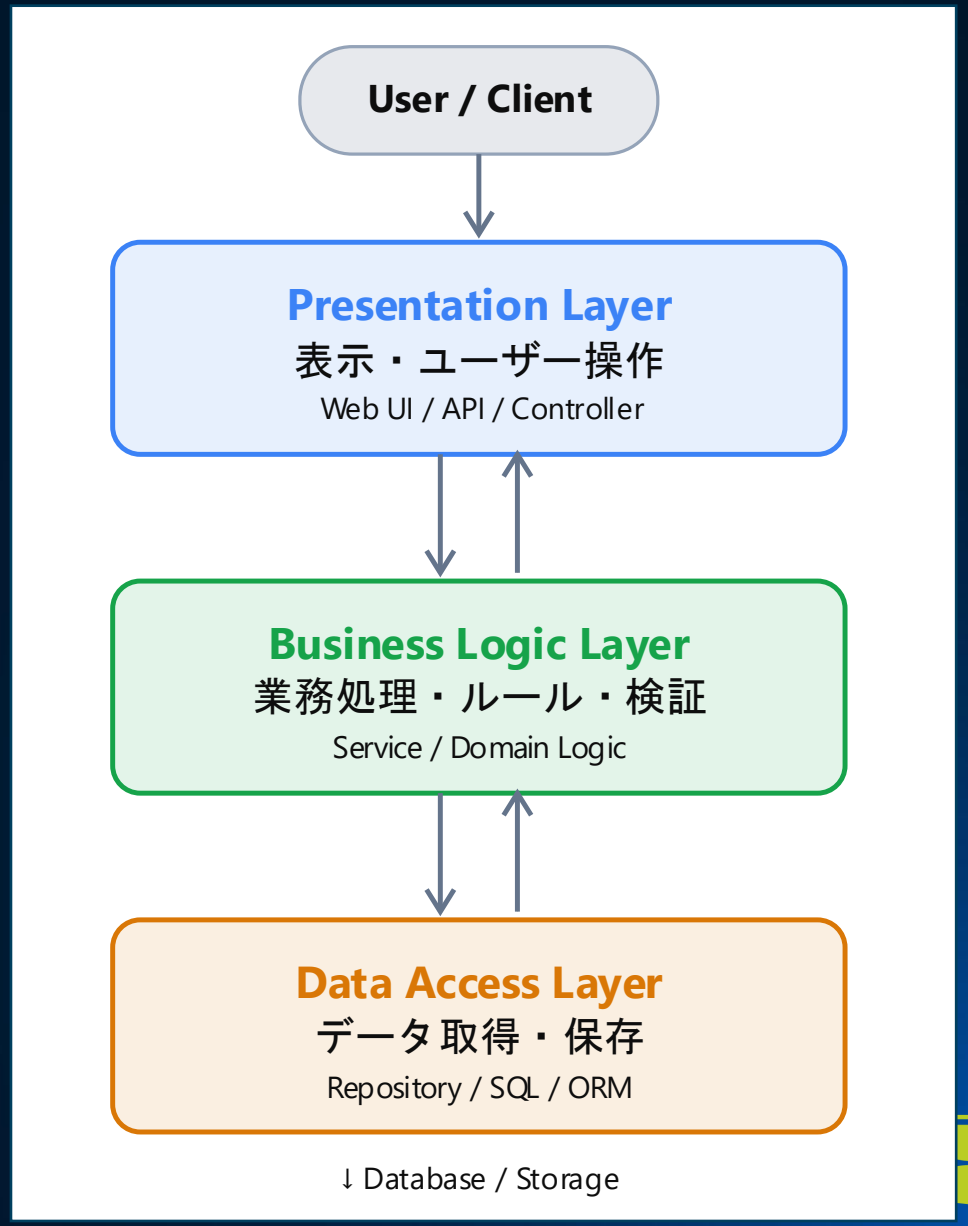




HORNETSECURITY
BY proofpoint.

三層構造Architecture

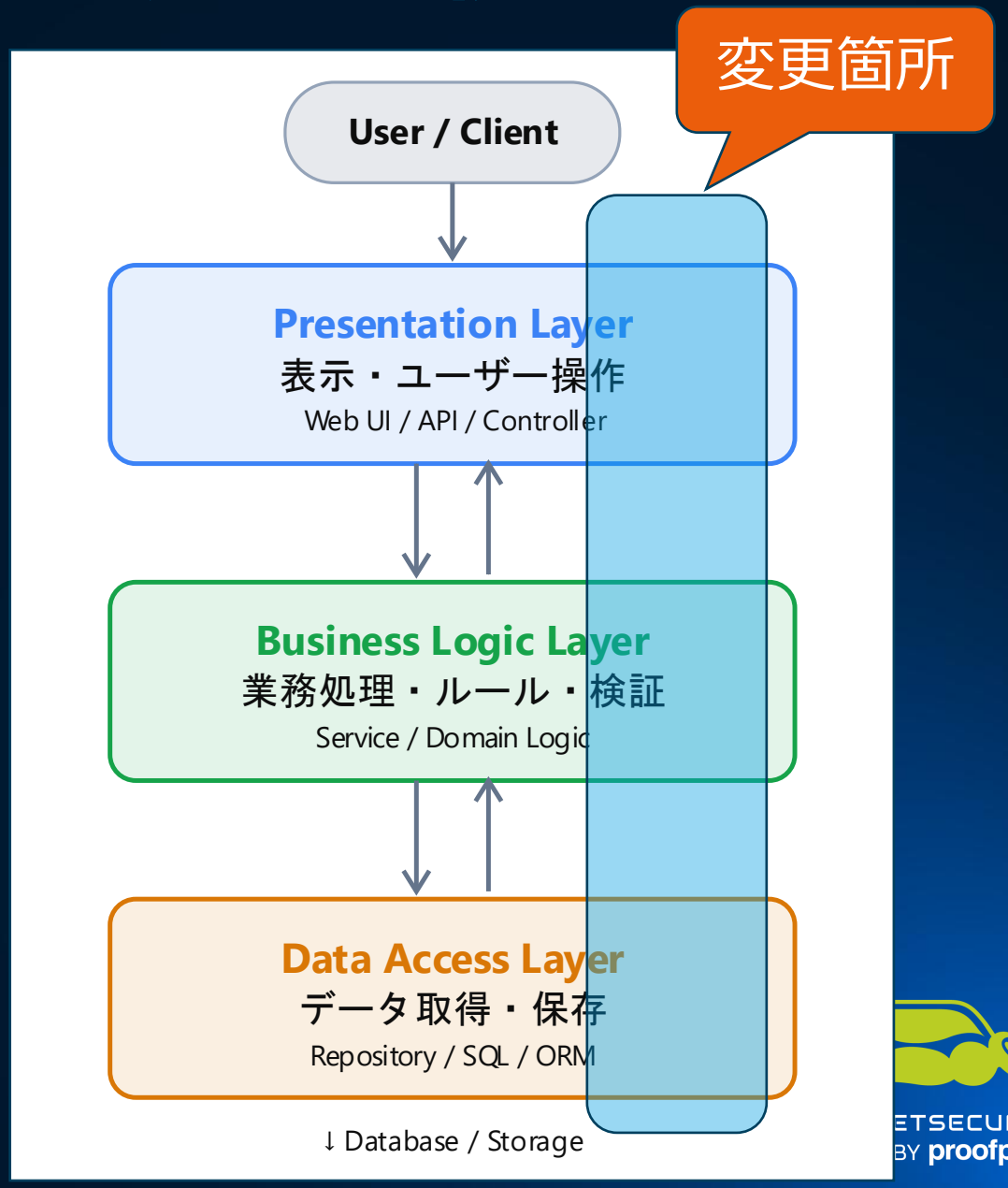
- 関心の分離
- 保守性の向上
- 再利用性の向上
- テスト容易性の向上
- 拡張性の向上



HORNETSECURITY
BY proofpoint.

AIへの変更指示はユーザー視点の機能

- 住所録に性別を追加して
- 結局AIは全箇所を読み込んで修正する
- 関係ない処理も触ってしまうかも知れない



人間とAIは似てるが違う

人間にとっての読みやすさ

- 人間の作業記憶は限られている
- 適切な名前、抽象化で詳細を隠す
- 小さな単位で理解する
- コード構造は長期的に記憶している

AIにとっての読みやすさ

- 大量のコードを一度に扱える
  - ただしコンテキストは有限
- 参照先の探索にはコストがかかる
- 暗黙の仕様や設計意図は推測しなければならない

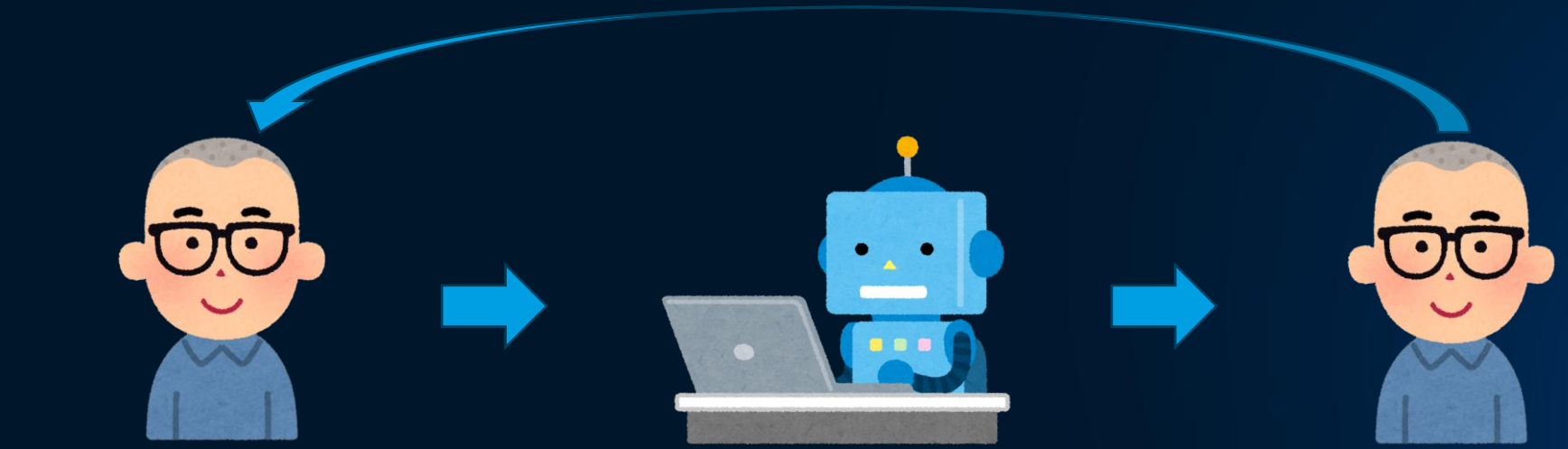
HORNETSECURITY
BY proofpoint.



そもそも人間がコードを読む必要はあるのか？

バイブコーディング

- 人間はやりたいことを指示する
- AIがソースコードを生成する
- 実際に動かして結果を確認する



人間はソースコードを見ない

HORNETSECURITY
BY proofpoint.

ソースコードを見るとき

- エラい人: 「ここの仕様どうなってるの?」
- 開発者: 「ソース見てもらったらいいですよ」
- エラい人: 「仕様書とかないの?」
- 開発者: 「あるけど、もうだいぶ変わってますよ」

ソースコード = Source of Truth

HORNETSECURITY
BY proofpoint.

ソースコードは仕様の背景まではわからない

- 新人開発者:
  「なんか、いらん分岐あったので消しときましたね」
- ベテラン開発者:
  - 「あー、それInternet Explorer7でいるやつやよ。
    まだ、使ってるところあるから戻しといて。」

HORNETSECURITY
BY proofpoint.

開発関係者の役割

- お客さま
  - 課題、困りごと、やりたいことをふわっと話す (WHY, WHAT)
- プロダクトマネージャー (PdM)
  - 課題を整理し、作るべきものと優先順を決める (WHY, WHAT)
- 開発チーム
  - 要件をプログラムなどに落とし込む (HOW, WHAT)

HORNETSECURITY
BY proofpoint.

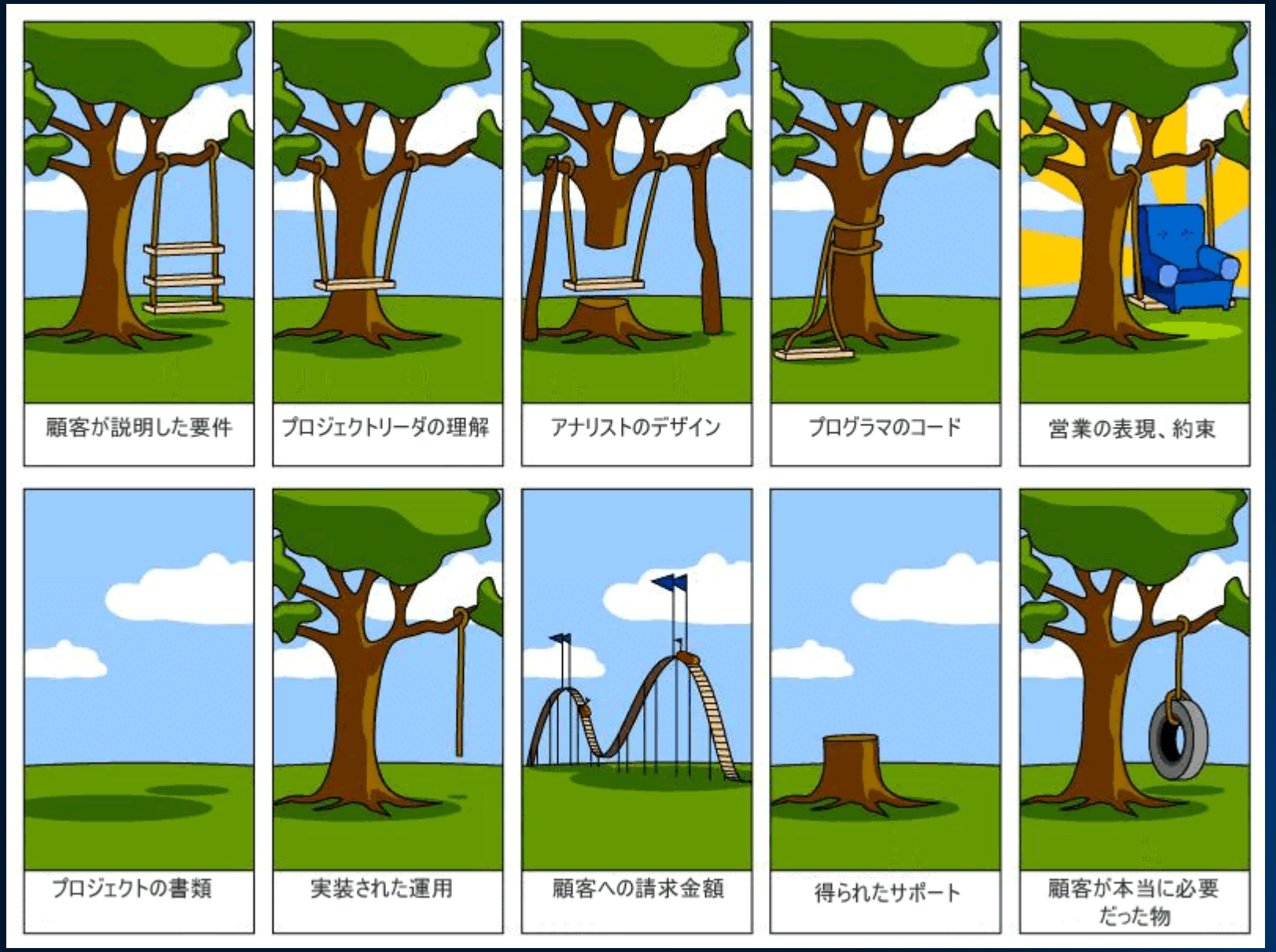
ソースコードの役割

- お客さま
  - 課題、困りごと、やりたいことをふわっと話す (WHY, WHAT)
- プロダクトマネージャー (PdM)
  - 課題を整理し、作るべきものと優先順を決める (WHY, WHAT)
- 開発チーム
  - 要件をプログラムなどに落とし込む (HOW, WHAT)

ソースコードはこの部分(の一部)

HORNETSECURITY
BY proofpoint.

よくあるやつ



HORNETSECURITY
BY proofpoint.

人間は何をやりたいのか

ここだけやりたい

- お客さま
  - 課題、困りごと、やりたいことをふわっと話す (WHY, WHAT)
- プロダクトマネージャー (PdM)
  - 課題を整理し、作るべきものと優先順を決める (WHY, WHAT)
- 開発チーム
  - 要件をプログラムなどに落とし込む (HOW, WHAT)

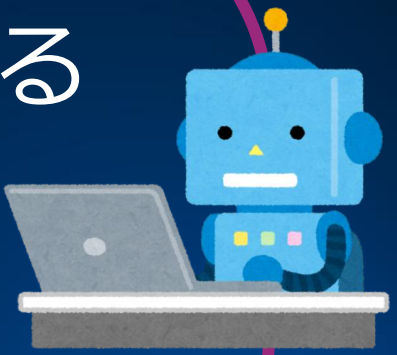
HORNETSECURITY
BY proofpoint.

こうすればいいのでは?



- お客さま
  - 課題、困りごと、やりたいことをふわっと話す (WHY, WHAT)

- プロダクトマネージャー (PdM)
  - 課題を整理し、作るべきものと優先順を決める (WHY, WHAT)
- 開発チーム
  - 要件をプログラムなどに落とし込む (HOW, WHAT)



HORNETSECURITY
BY proofpoint.



HORNETSECURITY
BY proofpoint

仕様を中心に考える

仕様駆動開発 (Spec Driven Development)

- 仕様を作成し、それをAIに実装させる
- 仕様をAIが書いてもいい

- Spec Kit
- Kiro
- など

HORNETSECURITY
BY proofpoint.

小さなSkillを書いてみた (KFD)

- いわゆる仕様のほかWHYの部分も残す
- これをKnowledgeと呼ぶ
- 小さな機能単位になるようにKnowledgeを分割
- 各Knowledgeの連携などをmapファイルに残す
- 設計で不明な点があれば必ず人間に聞く
- 思いついたものは勝手に実装せずParking Lotに残す

https://github.com/hirachan/kfd

HORNETSECURITY
BY proofpoint.



HORNETSECURITY
BY proofpoint

実験

簡単なアプリを作って試してみる (Step1)

イベント予約システム

- イベントには定員と開始日時がある
- 空席があれば申込みでき、「決済待ち」になる
- 決済すると「確定」になる
- 満席なら「キャンセル待ち」になる
- キャンセルなどで空席ができれば先着順に繰り上げ
- 誰でも申込み順に扱われる

HORNETSECURITY
BY proofpoint.

結果 (Step1)

	テスト結果	時間	トークン
スキルなし	22/22	155秒	147,021
KFD	22/22	195秒	261,390
Spec-Kit	22/22	1,093秒	5,344,630

HORNETSECURITY
BY proofpoint.

仕様追加 (Step2)

VIPを優先する

- VIPと通常の申込みを区別する
- 空席があればVIPも通常も席を確保
- 満席の時VIPは通常の「未決済」を押し出して、「キャンセル待ち」にできる
- 空席発生時の繰り上げもVIPを優先

HORNETSECURITY
BY proofpoint.

結果 (Step2)

4	テスト結果	時間	トークン
スキルなし	40/40	164秒	278,411
KFD	40/40	433秒	1,566,692
Spec-Kit	40/40	662秒	4,821,965

HORNETSECURITY
BY proofpoint.

既存仕様を壊す、仕様変更 (Step3)

- VIPの優先権は、イベント開始48時間前に失効する。
- ただし、失効前にすでに確保した枠は維持する。

HORNETSECURITY
BY proofpoint.

どういう仕様変更か

動作	変更前	変更後 (48時間前より前)	変更後 (48時間前以降)
VIP優先権	常に有効	有効	無効
満席時のVIPによる通常申込の押し出し	可能	可能	不可
キャンセル待ちの繰り上げ	VIP優先	VIP優先	全員先着順
新規VIP申込	VIPとして優先	VIPとして優先	通常申込と同じ
確保済みの席	維持	維持	維持

HORNETSECURITY
BY proofpoint.

結果 (Step1)

	テスト結果	時間	トークン
スキルなし	54/60	70秒	148,371
KFD	60/60	169秒	705,219
Spec-Kit	60/60	527秒	4,642,010

HORNETSECURITY
BY proofpoint.

どこを間違えたか

動作	変更前	変更後 (48時間前より前)	変更後 (48時間前以降)
VIP優先権	常に有効	有効	無効
満席時のVIPによる 通常申込の押し出し	可能	可能	不可
キャンセル待ちの 繰り上げ	VIP優先	VIP優先	**全員先着順**
新規VIP申込	VIPとして優先	VIPとして優先	通常申込と同じ
確保済みの席	維持	維持	維持

HORNETSECURITY
BY proofpoint.

仕様に戻せるか

	テスト結果
スキルなし	0/24
KFD	21/24
Spec-Kit	21/24

- VIP優先権が失効しても、自動キャンセル・自動返金はしない
- VIPに押し出された通常申込は、優先権失効後も自動的には元に戻さない
- などの仕様を復元できなかった

全ての仕様が復元できるわけではないが、
そこそこいける

HORNETSECURITY
BY proofpoint.



Postfixの設定をAIで管理

題材：Postfix の設定を AI で管理

- Docker 上の Postfix（Debian）
- 設定ファイルは config/ 配下（main.cf、master.cf）
- 外部宛てのメールは smtp4dev に中継する検証構成
- Codex ＋ KFD スキル
- 依頼は上司の指示のように、あえて曖昧に出す

HORNETSECURITY
BY proofpoint.

HORNETSECURITY
BY proofpoint

実験1
スパム対策

実験1：スパム対策（1/3）

最近、特定のドメインから大量のスパムが届いています。どうしたらいい？
特定のドメインは baddomain.exampleです。

画面の From: ではなく、ログの MAIL FROM ですか?
そのドメインなら check_sender_access で拒否できるが、正当なメールまで止める恐れがありますよ。

はい。MAIL FROMのドメインです。

「MAIL FROM が完全一致したら拒否する」仕様を書き、実装前に承認を求めます。

HORNETSECURITY
BY proofpoint.

実験1：スパム対策（2/3）

拒否案を撤回します。
「SPF 失敗だけ拒否」と「スコアで隔離」の方法があります。

対象ドメインのメールだけ、接続IPをDNSBLで照会し、警告ログを残すだけにします。これで実装しますか。

このドメインからは本物のメールも届いているようなので、拒否するのではなくちょっと制限を厳しくするようなことはできますか。

まずは追加ソフトの不要な方法がいいですね。

HORNETSECURITY
BY proofpoint.

実験1：スパム対策（3/3）

はい。

main.cf と sender_access に実装しました。

実装された設定

```
# config/main.cf（追記）
smtpd_restriction_classes = baddomain_ip_reputation
baddomain_ip_reputation = warn_if_reject, reject_rbl_client zen.spamhaus.org
smtpd_sender_restrictions = check_sender_access regexp:/etc/postfix/sender_access

# config/sender_access（新規）
/^.*@baddomain¥.example$/ baddomain_ip_reputation
```

HORNETSECURITY
BY proofpoint.

HORNETSECURITY
BY proofpoint

実験2
受信経路の暗号化

実験2：受信経路の暗号化（1/3）

受信経路を暗号化対応したい。TLSの種類とかいろいろあってよくわからないで、いい感じに提案してほしい。検証環境なので証明書はオレオレでいい。

STARTTLS を任意で有効にします。必須にすると平文の相手を拒否するので避けるのが無難。

量子暗号は対応しておかなくてもいいですか

検証環境なら今は不要。耐量子方式 X25519MLKEM768 には OpenSSL 3.5 が要るが、今のOSはBookworm で 3.0 系なので非対応。

HORNETSECURITY
BY proofpoint.

実験2：受信経路の暗号化（2/3）

検証環境ですが、国家レベルの機密が流れるんです。

それなら提案を撤回します。今の構成は暗号化なしで後ろに中継しているため、機密を流さないようしてください。機密用途への改造はお勧めしません。

そういう検証環境なので、まずは受信の暗号に絞って提案してください。

STARTTLS 必須・TLS 1.3・証明書固定でどうでしょう。でも耐量子にはDockerfile の変更が必要でconfigの変更だけでは実装できません。

HORNETSECURITY
BY proofpoint.

実験2：受信経路の暗号化（3/3）

TLS 1.3 と X25519MLKEM768 のみ、公開 MX にはしない、保護は受信部分だけ。
これでいいですか。

dockerfileの修正も許可します。

Dockerfile を Trixie に上げてmain.cf で TLS 必須化、master.cf で鍵交換を限定しました。

ok

HORNETSECURITY
BY proofpoint.

実験2：実装された設定

config/main.cf

```
smtpd_tls_security_level = encrypt
smtpd_tls_protocols = >=TLSv1.3
smtpd_tls_mandatory_protocols = >=TLSv1.3
smtpd_tls_cert_file = /run/postfix/tls/server.crt
smtpd_tls_key_file = /run/postfix/tls/server.key
smtpd_tls_loglevel = 1
smtpd_tls_received_header = yes
```

Dockerfile

```
FROM debian:trixie-slim
```

config/master.cf

```
smtp inet n - n - - smtpd
  -o tls_eecdh_auto_curves=X25519MLKEM768
  -o tls_ffdhe_auto_groups=
```

HORNETSECURITY
BY proofpoint.

実験2：仕様を確認

- 管理下の検証SMTPクライアント専用サービスでは STARTTLSを必須とし、TLS 1.3 と X25519MLKEM768 のハイブリッド鍵交換だけを許可する。
- 平文接続、TLS1.2 以下、従来型鍵交換へのフォールバックを許可せず、対応できないクライアントの受信処理を開始しない。
- このサービスは公開 MX として運用しない。
- Postfix は mail.test 用の自己署名証明書を提示する。クライアントは事前配布された証明書を信頼し、未確認の証明書は受け入れない。
- 暗号化の対象はクライアントから Postfix までの受信区間であり、その後の中継先や保存先の暗号化は保証しない。

HORNETSECURITY
BY proofpoint.

HORNETSECURITY
BY proofpoint

実験3
DKIM 検証

実験3：DKIM 検証（1/2）

メールのDKIM署名をverifyして、その内容をログに出力してください。

Postfix 本体では DKIM を検証できず、外部 Milter が必要です。
OpenDKIM で検証して結果をログに出します。

はい

実装前に追加の判断が必要です。
OpenDKIM はメールに Authentication-Results ヘッダーを足すが、それでいいか（おすすめ）、
ヘッダーを変えない別方式にするか。

HORNETSECURITY
BY proofpoint.

実験3：DKIM 検証（2/2）

OpenDKIM を不採用にします。python3-dkim と python3-milter で専用 Milter を作る案に切り替え、「ヘッダーや本文を変えない」ようにします。これでいいですか。

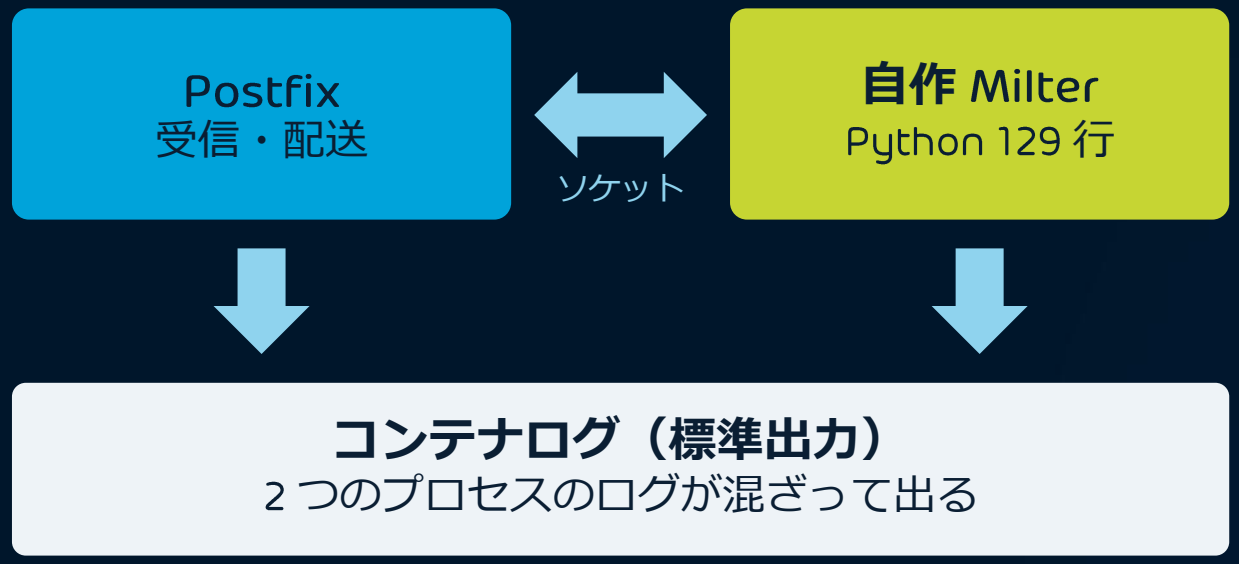
検証結果（pass／fail／署名なし／エラー）をコンテナログに出すよう実装。Docker ビルドと実メールでの確認は未実施。メール全体をメモリに持つ点も注意として報告。

ヘッダーは付けたくありません。

ok

HORNETSECURITY
BY proofpoint.

DKIM ログを出すだけでも部品が増える



ログ形式

```
DKIM_VERIFY status=pass signature=1
  domain=… selector=…
```

このために変えたもの

`Dockerfile`
パッケージを 3 つ追加、ソケット用ディレクトリを作成、
起動コマンドをシェルにして 2 プロセスを起動

`config/main.cf`
Milter への接続、障害時は受け入れ、ログを標準出力へ

`config/dkim_verify_milter.py`
新規。署名を検証してログを書く本体

Milter のログにはキュー ID がなく、どのメールの結果か Postfix のログと突き合わせにくい

HORNETSECURITY
BY proofpoint.

HORNETSECURITY
BY proofpoint

実験4
IP x Fromで制限

実験4： IP x Fromで制限

IPとEnvelope Fromドメインの組み合わせごとに件数を数えるには、ポリシーサービスなどの追加処理が必要です。
現在のKnowledgeは baddomain.example をログ観察のみとしています。レート制限を追加する場合は、そのルールを改訂してから実装します。

IPだけではなく、IP x Envelope Fromドメイン で制限をかけたい

今回はやめます



Go言語でカスタマイズできるPolicy Runnerでは?

Policy Runner の特長

Go言語でMTAを作れるフレームワーク

受信、キュー、配送までの処理をプログラムでに記述。
AIでもいつも通りのプログラミングができる。

豊富な標準モジュール

SMTP の受信と配送、キュー、分散件数カウンター、DMARC、TLS 、Authentication-Resultsヘッダなど必要な機能はすべて内包。

機能開発と運用の分離

コード内の設定ブロックに既定値を書き、設定ファイルで上書きできる。
運用用のAPIも用意。

テストフレームワークを完備

Go言語標準にテスト環境に加え、受信から配送完了までを外部からE2Eテストする環境を標準装備。

AIで開発や運用をするためのSkillを内蔵

HORNETSECURITY
BY proofpoint.



実験5
スパム対策
by Policy Runner

実験5：スパム対策（1/6）

最近、特定のドメインから大量のスパムが届いています。どうしたらいい？
特定のドメインはbaddomain.exampleです。

Fromヘッダですか、Return-Pathですか。
SMTP の MAIL FROM が baddomain.example なら、その段階でドメインを完全一致で判定し、550 5.7.1 を返す方法が直接的です。

HORNETSECURITY
BY proofpoint.

実験5：スパム対策（2/6）

envelope fromです。

MAIL FROM のドメインが baddomain.example と大文字小文字を問わず完全一致したら、SMTP の MAIL FROM 段階で 550 5.7.1 を返して拒否します。
キューには保存しません。
サブドメイン、他のドメイン、空の MAIL FROM は、このルールの対象外です。
KFD スキルは、受信動作を変える前に Knowledge Review の承認を求めています。この動作で進めてよいですか？ 承認後に policy を変更します。

HORNETSECURITY
BY proofpoint.

実験5：スパム対策（3/6）

このドメインからは本物のメールも届いているようなので、拒否するのではなくちょっと制限を厳しくするようなことはできますか。

できます。正規メールも届くなら、接続元 IP ごとに baddomain.example の受信量を制限し、超過分だけ 451 で一時保留する案を勧めます。
送信側が再送すれば、枠が空いた後に届きます。先ほどの恒久拒否案は Knowledge から取り下げました。
制限値は正規メールにも影響するため、KFD の Decision Required として確認させてください。
まず 1 IP あたり10分間に20通、超過分は 451 で進めてよいですか？

HORNETSECURITY
BY proofpoint.

実験5：スパム対策（4/6）

標準モジュールでできますか

はい。標準の database/reputation モジュールで、接続元 IP ごとの一定時間内の件数を数えられます。
MailFrom 処理で、baddomain.example に一致する場合だけその件数を参照し、上限を超えたら 451 を返す構成にできます。
また、reputation の利用にはクラスターノードの設定が必要です。
先ほどの「10分20通」はまだ提案値で、適用はしていません。

HORNETSECURITY
BY proofpoint.

実験5：スパム対策（5/6）

はい。では、それで提案してください。

- 実装には標準の database/reputation と net/clustering を使い、MailFromステージの処理で判定します。
- MAIL FROM のドメインが baddomain.example と完全一致する場合だけ、接続元 IP ごとに10分間の試行を数えます。
- 20件を超えた試行には 451 4.7.1 を返し、送信側の再送を待ちます。
- 件数を取得できないときはエラーを記録し、通常の受信処理を続けます。
- 同じ IP から届く正規メールも、上限を超えている間は遅延します。

この提案で実装に進めてよいですか？

HORNETSECURITY
BY proofpoint.

実験5：スパム対策（6/6）

はい。

承認された受信量制限を実装しました。
baddomain.example の MAIL FROM 試行を、標準の reputation モジュールで接続元 IP ごとに数え、10分間に20件を超えた分へ 451 4.7.1 を返します。
カウンターが使えない場合はエラーを記録して通常の受信処理を続けます。
対象ドメイン、上限、障害時の動作を確認するテストを追加しました。
また、既定のカウンター構成は単一ノード向けです。複数 MTA で件数を共有する場合はクラスターメンバーの設定が必要です。

Postfix ではRBLによる警告ログだけだったが、ここではPolicy Runnerの標準モジュールによる件数制限になった



HORNETSECURITY
BY proofpoint

実験6
受信経路の暗号化

by Policy Runner

実験6：受信経路の暗号化（1/6）

受信経路を暗号化対応したい。TLSの種類とかいろいろあってよくわからないで、いい感じに提案してほしい。検証環境なので証明書はオレオレでいい。

ポート25で STARTTLS を有効にする構成を勧めます。
最低 TLS 1.2 で、上限と暗号スイートは Go の既定値。
クライアント証明書は要求しません。
公開 MX では TLS を必須にしない扱いが RFC 3207 に定められています。
全件の暗号化が必要な閉じた検証環境なら、TLS 前の MAIL FROM を拒む設定を追加できます。

HORNETSECURITY
BY proofpoint.

実験6：受信経路の暗号化（2/6）

量子暗号は対応しておかなくてもいいですか

耐量子暗号を使う TLS の意味なら、専用設定は不要です。
Go 1.27 系は TLS 1.3 で耐量子方式を組み合わせた鍵交換に対応していて、
相手側も対応していれば使えます。
ただし、今の policy にはまだ受信 TLS の証明書が設定されていません。

Postfix では OS の更新が必要だった耐量子 TLS が、Go の標準ライブラリで使える

HORNETSECURITY
BY proofpoint.

実験6：受信経路の暗号化（3/6）

検証環境ですが、国家レベルの機密が流れるんです。

それなら判断を改めます。
任意の STARTTLS と自己署名証明書は採用しないでください。
今の policy は配送先 spool:25 にも TLS 必須の指定がなく、本体を /var/mail/queue-v2 に保存します。
受信口だけ暗号化しても経路全体は守れません。
送信元から spool:25 の先まで全ホップを管理できますか。
適用すべき機密取扱基準は何ですか？

HORNETSECURITY
BY proofpoint.

実験6：受信経路の暗号化（4/6）

検証環境なので、まずは受信の暗号に絞って提案してください。

送信側も設定できる閉じた検証環境として、次の構成を提案します。
・機密メール専用の受信口を内部ポートに設け、接続開始時から暗号化（ImplicitTLS）
・TLS 1.3 のみ、X25519MLKEM768 を必須
・検証用CAでサーバーと送信側の証明書を発行し、相互TLS で確認

HORNETSECURITY
BY proofpoint.

実験6：受信経路の暗号化（5/6）

はい。

Knowledge に記録しました。
既定では 127.0.0.1:2465 に専用受信口を追加し、既存の25番は維持します。
証明書や CA を読み込めない場合、平文に切り替えず policy の起動を失敗させます。
この動作で実装に進めてよいですか？

HORNETSECURITY
BY proofpoint.

実験6：受信経路の暗号化（6/6）

ok

既存の25番を維持しつつ、127.0.0.1:2465 に専用受信口を追加しました。
TLS 1.3、X25519MLKEM768、検証済みクライアント証明書を要求します。
証明書生成スクリプトと配置手順も追加し、生成した証明書は OpenSSL で検証しました。

HORNETSECURITY
BY proofpoint.

Policy Runner：実装（抜粋）

policy.go：SMTPサーバーの起動

```
    smtp.RunServer(&smtp.ServerConfig{
        Profile:     "confidential-smtp",
        Addr:        "127.0.0.1:2465",
        ImplicitTLS: true,
        TLSConfig: &tls.Config{
            MinVersion:       tls.VersionTLS13,
            MaxVersion:       tls.VersionTLS13,
            CurvePreferences: []tls.CurveID{tls.X25519MLKEM768},
            ClientAuth:       tls.RequireAndVerifyClientCert,
        },
       ....
    })
```

HORNETSECURITY
BY proofpoint.



実験7
DKIM 検証
by Policy Runner

実験7：DKIM 検証（1/4）

メールのDKIM署名をverifyして、その内容をログに出力してください。

DKIM 検証は追加できます。標準モジュールの auth/dkim に、署名ごとの検証結果を取り出す API があります。
機密メールも扱うので、ログに出す内容を実装前に決めさせてください。
おすすめは署名ごとに検証結果、d=、s=、a= を記録し、未署名なら none。検証失敗だけでは拒否しません。
この範囲でよいですか？

HORNETSECURITY
BY proofpoint.

実験7：DKIM 検証（2/4）

その範囲でokです。

Knowledge に記録しました。
25番と2465番の両方で、キュー保存前に DKIM 署名を検証します。
ログには結果、d=、s=、a= とセッション・メッセージ ID を記録します。
本文、ヘッダー全文、署名値、本文ハッシュ、エラー原文は記録しません。
この動作で実装に進めてよいですか？

HORNETSECURITY
BY proofpoint.

実験7：DKIM 検証（3/4）

エラー原文はほしいです。

エラーが返された場合は原文もログに記録するよう Knowledge を直しました。
エラー原文にはメール由来の値が含まれる可能性があるので、その点も明記しています。
セキュリティ上の意味が変わったので、再承認をお願いします。

この内容で実装に進めてよいですか？

HORNETSECURITY
BY proofpoint.

実験7：DKIM 検証（4/4）

ok

解析済みのメールを標準の DKIM 検証器に渡し、署名ごとの結果、d=、s=、a=、セッション・メッセージ ID、エラー原文を記録するようにしました。DKIM の結果だけで受信可否は変えません。

HORNETSECURITY
BY proofpoint.

Policy Runner：DKIM の実装（抜粋）

```
verifier, err := dkim.NewVerifier(&dkim.VerifierConfig{})
…
func logDKIMVerifications(logger zerolog.Logger, verifier dkim.Verifier, part email.Part) {
    verifications, err := verifier.Verifications(part)
…
    for verification := range verifications.Iter() {
        event := logger.Info().
            Str("dkim_result", strings.ToLower(verification.Result().String())).
            Str("dkim_domain", domain).
            Str("dkim_selector", selector).
            Str("dkim_algorithm", algorithm)
        if err := verification.Err(); err != nil { event = event.Err(err) }
        event.Msg("DKIM verification")
    }
```

HORNETSECURITY
BY proofpoint.



実験8
IP x Fromで制限
by Policy Runner

実験8：IP x ドメインで制限 (1/2)

IPだけではなく、IP x Envelope Fromドメイン で制限をかけたい

制限対象を baddomain.example のみに保ち、件数を 接続元IP × 大文字小文字を正規化したEnvelope Fromドメイン ごとに数えるよう変更しました。10分間に20件を超えた試行への 451 4.7.1 は継続します。キーが変わるため、旧キーの件数は引き継がず、新キーで数え直します。

この内容で進めてよいですか。

HORNETSECURITY
BY proofpoint.

実験8：IP x ドメインで制限 (2/2)

```
domain := envelopeFromDomain(s.MailFrom.Address)
if domain == "baddomain.example" {
    counts, err := h.runtime.ratePrefix.Entry(s.IP.String() + "|" +
domain).Add(1)
    ....
    if count > 20 {
        logger.Warn().Str("action", "tempfail").Uint64("rate_count", count).
            Msg("sender rate limit exceeded")
        s.WriteStrings("451 4.7.1 Rate limit exceeded")
    }
}
```

HORNETSECURITY
BY proofpoint.

HORNETSECURITY
BY proofpoint

実験9
運用にAIを使う

by Policy Runner

実験4：運用をAIする 1/2

mta-1のqueueに溜まっているメール数を教えて

mta-1のqueueには現在8通溜まっています。

届いていない宛先を一覧にして、それぞれの通数を教えて

mta-1のキューで未配信の宛先と通数は次のとおりです。
- b@example.jp：7通
- hirano@example.jp：1通。

hirano@example.jp宛のメールを今すぐ再配送して

実験4：運用をAIする 2/2

hirano@example.jp宛のメールを今すぐ再配送して

mta-1のライブAPI仕様を確認し、hirano@example.jp宛のキュー済みメールだけを今すぐ再配送する方法を調べます。対象は先ほどの確認と同じmta-1です。

再配送の対象を誤らないよう、APIの宛先フィルターで現在の該当メッセージ数とqueue IDを確認しています。

再配送リクエストは受理されました（202）。APIが返した状態確認先を確認して、配送結果を追います。

再配送を要求しましたが、mta-1の状態レポートでは対象メールが「処理中のため失敗」となっています。手動での再配送は確認できず、キューにも残っています（現在8通）。

RITY
BY proofpoint.

まとめ

- AI も機能追加を重ねると、人間と同じように破綻しやすい
- 人間向けの読みやすさが、AI にとっての読みやすさとは限らない
- ソースコードには仕様の理由（WHY）が残らないので、仕様と理由をKnowledgeとして正本にするとうまくいった
- Postfixの設定ファイルも同様に生成AIで管理できた
- Goで開発するPolicy Runnerでは、より複雑な機能追加や、メール運用まで生成AIでできた

HORNETSECURITY
BY proofpoint.

質疑・応答


HORNETSECURITY
BY proofpoint.
コミュニティ 懇親会
肩の力を抜いて、
語り合う夜に。
JPAAWG 9th General Meeting
最終日の夜に
日時
11月6日（金）
18:00～20:00
会場
大手町サンケイプラザ 3F
JPAAWGから徒歩3分
大手町駅直結
参加費
無料
申込不要
JPAAWGにご参加の方なら、途中からでも、どなたでも



質疑・応答



HORNETSECURITY
BY proofpoint

Thank you!