分析自動化の取り組みから考えるSOCの将来

2019.01.24

JANOG43

NTTセキュリティ・ジャパン株式会社
野岡 弘幸

NTT Security

Today’s Abstract

- 自動化による利益は皆でシェアしよう
- 運用において「オペミス防止のための自動化」は
  非常に強力なモチベーション
- 始めることで更にできることや今の限界が見えてくる

whoami

- NTTセキュリティ・ジャパン株式会社
  セキュリティオペレーション部
- マネージドセキュリティサービス「WideAngle」のSOCにて
  高度分析サービス（リアルタイムログ分析）を担当

SOCの分析オペレーション

1. デバイスからログを収集する
2. 入手したログを分析する
3. （何かあれば）レポートを作成する
4. お客様に通知する

分析オペレーションの特徴

- 何か起きるときは一斉に起きる
  - 例：攻撃キャンペーンなどで誰かが引っかかるとそれを皮切りに皆引っかかる
- 分析が反復作業になりがち
  - 「トラブルシューティングの初動の切り分け」をずっとやってることを想像していただければ
- 複数顧客を見ているので事故が怖い
  - インシデントレポートを別の顧客に送ろうものなら重大インシデント

→ 自動化のモチベーションが高い

過去のJANOGにおける自動化関連セッション

JANOG 34
- ネットワークエンジニアとソフトウェアエンジニアの狭間で

JANOG 35
- HubotとChatOpsについて語るBoF

JANOG 36
- API/Web化によるネットワーク自動化

JANOG 38
- Zero Touch Configuration

JANOG 39
- 障害ありきで運用自動化をやってみた
- 『紙の手順書』でルータ設定を半自動化してみた with 二次元バーコードリーダ
- Ansibleコトハジメ

JANOG 40
- ShowNetでピアリングを自動化してみた
- キャリアネットワークにおける作業の自動化と課題

過去のJANOGにおける自動化関連セッション（続）

JANOG 41
- ネットワーク運用自動化BoF
- 明日からはじめるネットワーク運用自動化
- OSSとAPIで作る、AI”風”NWマネジメント/オペレーションツールとネットワーク自動化
- BDD + Pythonによるネットワーク機器のテスト自動化
- RENAT(Robotframework Extension for Network AutomationTesting)を用いた
  ネットワーク検証の自動化について

JANOG 42
- Ansible ネットワーク自動化チュートリアル
- ネットワーク運用自動化BoF 対話編
- その運用自動化では行き詰まる
  ～「つながらない」「つたわらない」「つみあがらない」を防ぐために～

JANOG43
- 自動化の行き着く先は? ← New!!

自動化を「やってみた」で終わらせないために

「自動化に抵抗のあるチーム/上司」の壁を越える方法を考えてみる

1. 自動化の必要性をもって説得する
2. 運用の自動化を推進する

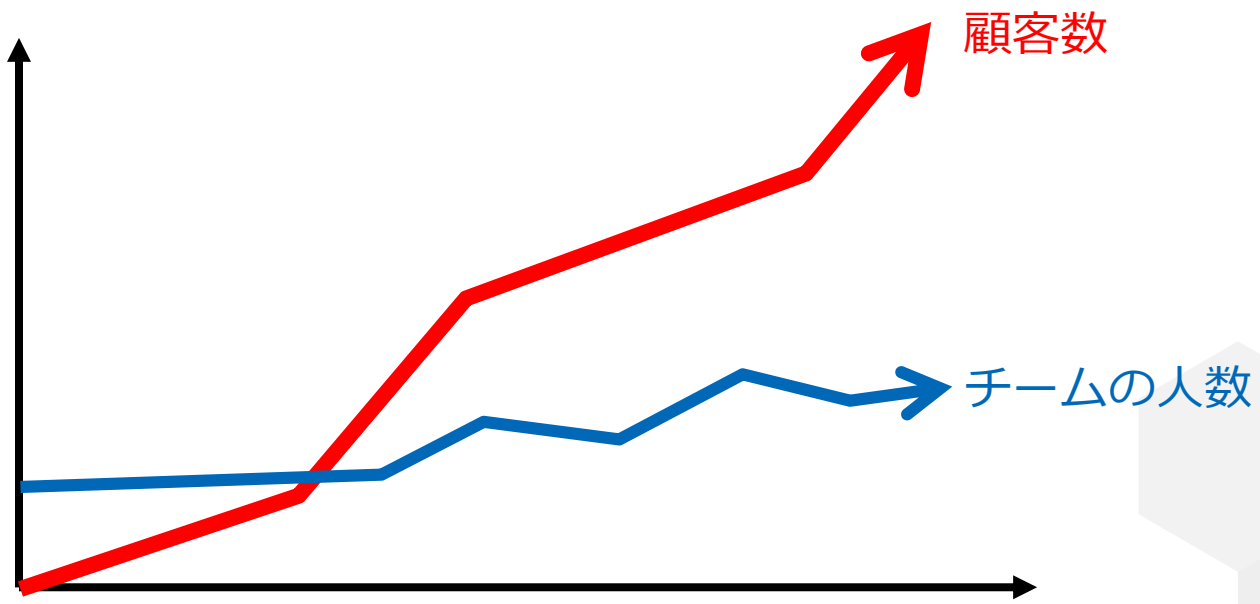
1. 自動化の「組織的な」取り組みの必要性

チームの維持拡大に関する3つの課題

1. チーム拡大の過渡期における個人負担増
   - サービスの拡大速度 > チームの拡大速度
2. 「運用を任せられるレベル」に達する技術者の確保
   - 高スキル技術者の絶対数は少ない
3. 技術者の定着
   - 「不幸な」業務は人が離れる

サンプル1：チームの拡大が追い付かない

サービス開始以降，顧客数は順調に増加
顧客が x 倍に増えてもチームの人数は x 倍になってない
→ いつまでも同じような運用では物量的に間に合わない！



サンプル2：人が集まらない

専門技能が必要なメンバーをどうやって揃えるか

1. 高スキル技術者の採用
   - 上手くいけば苦労はない
2. 要求を変える
   - × 要求するレベルを下げる
   - ○ 従来と異なるスキル保有者にも目を向ける
3. 育成する
   - やり方教えてください

サンプル3：なんとかしたい「不幸な」アナリスト業務

- 単純で面倒な「面白くない」分析
  - 診断会社からの脆弱性診断やTor通信

    → リストにマッチしたものを機械的に通知するだけ
  - 特定のIPアドレスからの「継続的な」スキャン

    → 本当に継続してる？
- 機器のバグによるログの崩壊への対応
  - パース処理の修正や別のフィールドでの読み替え

    → ひたすらに量が多い

もっとシンプルに

「運用自動化の歴史」は「オペミス対策の歴史」

- 人間はミスをする
- 定型化された作業においては機械のほうが人間より正確
- オペミスをシステムで防ぐことに反対する人は少ない
  - システムによる不便を被る可能性については今回のスコープ外
- L3はそうだった，皆さんの組織は？

2. 運用の自動化を推進する

一気に全自動化するギャップは心理的にも技術的にも大きい

段階を踏んで自動化~~の沼に引きずり込む~~に慣れさせる

1. 入力のvalidation
2. 入力の自動化
3. 単純な判断のロジック化
4. 複雑な判断のロジック化

ステップ1：入力のvalidation

- 入力の人力チェックは必ず形骸化・破綻する
  - ダブルチェックしようがトリプルチェックしようが一緒
- 見逃す確率は（多分）独立じゃない
  - 一人が見逃す状況は他の人も見逃す状況
  - 具体的には夜勤明け直前の2人がチェックしてどれだけ防げるか

→ 機械でチェックしたほうが確実

「出力の正規化」という概念が身につく

ステップ2：入力の自動化

- そもそも人間が入力するから間違える
  - コピペですら間違えるときは間違える

→ 人間の入力は最小限にする
　同じ入力を使う場所は値を共有

「入力の正規化」「構造化されたデータのほうが都合がいい」
という発想が生まれる

ステップ3：簡単な判断のロジック化

- 入力を自動化する時点で多少のロジックを使っているはず

→ 単純で機械的にできることをロジックにする

  – 何らかのリストの収集・更新

  – リストとのマッチング

「入力から出力へのデータ形式の変換」ができる

ステップ4：複雑な判断のロジック化

- ステップ3まで進めれば自動化への抵抗はほとんどないはず
  - 「構造化された入出力」「入力から出力へ変換する発想」を持っている

→ 自分達のやりたいところまで自動化する

最近のL3の取り組み
～プロジェクトLHK（仮）～

SOCの分析オペレーション（再掲）

1. デバイスからログを収集する
2. 入手したログを分析する
3. （何かあれば）レポートを作成する
4. お客様に通知する

SOCの分析オペレーション（再掲）

1. デバイスからログを収集する
2. 入手したログを分析する
3. （何かあれば）レポートを作成する
4. お客様に通知する

分析・レポート作成の
自動化/高度化

NTT Security

基本思想

- 分析オペレーションをProgrammableに
  - アナリスト自身がオペレーションを進化させる
- ライブラリ化で再開発抑制
  - 対向のインターフェース仕様が変更になっても影響を小さく

プロジェクトLHK（仮）の進め方

フェーズ0：
暫定開発メンバーによる方針の共有

フェーズ1：
それぞれのシステムを操作するモジュールを作成

フェーズ2：
フェーズ1で作成したモジュールを活用し，
ユースケースのサンプルを開発する

フェーズ3：
チームに展開し，各自がユースケースを作成

フェーズ0：プロジェクト開始直後の一幕

自動化に関心のあるアナリストが集結し開発方針について協議

- やりたいことは何か（基本思想の共有）
- コードをどこで動かすか（開発言語・実行環境）
  - Python3のDockerコンテナを想定
- 我々（暫定開発チーム）は何を作るのか
  - 共通モジュール（ライブラリ）
  - ユースケースのサンプル
  - ドキュメント
  - 実行インフラの整備

分析オペレーション関連システム群（抜粋）

分析においてアナリストが利用するシステム（自動化/高度化の対象）は多岐にわたる

JPSOC独自
分析ロジック

Bouncer
分析結果転送

Analysts

IPS/Proxy等
デバイス

Splunk

Inspector/Meta
ログ分析システム

KARTE
レポート作成システム

ポータルサイト

Filter

RTCE
SIEMエンジン

Mattermost
チャットコミュニケーション

MISP
ナレッジ共有

フェーズ1：各システムの操作ライブラリの作成

アナリストが直接操作しうるシステム（青色）について
アクションを模倣するモジュールを作成

JPSOC独自
分析ロジック

Bouncer
分析結果転送

LHK

IPS/Proxy等
デバイス

Splunk

Inspector/Meta
ログ分析システム

KARTE
レポート作成システム

ポータルサイト

Filter

RTCE
SIEMエンジン

Mattermost
チャットコミュニケーション

MISP
ナレッジ共有

フェーズ2：ユースケースの開発

開発しているユースケースの例

- スキャン・脆弱性診断の検知
- Tor通信の検知
- ログフィルタの自動投入
- チケットに連動したブラックリスト管理

ユースケースのイメージ

例えば,

1. ログフィルタを全件取得して
2. その中から一年間Hitしていないフィルタを削除し,
3. そのIDと名前を表にしてMattermostのdevnullチャンネルに投稿する

という操作は以下のようなコードとして書ける

```
from lhk import Filter, Mattermost
fc = Filter()
filters = fc.search() #(1)
not_match_filters = [filter for filter in filters if (datetime.now() - filter.date_lastmatch).days >= 365]
for filter in not_match_filters:
    fc.remove(filter.id) #(2)
headers = ['ID', 'Name']
table = [[filter.id, filter.description] for filter in not_match_filters]
Mattermost.post('devnull', headers=headers, table=table) #(3)
```

フェーズ3：アナリストへの展開

ユースケース作成を暫定開発メンバーからアナリスト全員に展開

1. Python3の動くコンテナを設置
2. LHKライブラリ，ユースケースをオンプレのgitlabから取得
3. ユースケースを実行

アナリストはユースケースのスクリプト作成と
実行スケジュールを決めるだけのFaaS的な環境（に，したい）

フェーズEX：どこまでも続く道

プロジェクト開始から9ヶ月
「動くこと」を重視してとりあえず動くレベルの物はだいたいできた

その気になればやれるとは思うこと（今はできていないこと）

- テストやエラー処理の作りこみ
- CI/CDなどのためのインフラ整備
- ユーザスクリプトのマネージメント
- 効果測定

開発負荷や保守性，展開速度などのバランスを鑑みて
どのあたりまでやるのか常に手探り

SOCの未来予想

「自動化は手段」

お客様は「自動化された業務」に
+α のお金を払ってくれるわけではない
→ 結果が同一なら過程の違いは価値の差にならない

「自動化できるチーム」は新たな価値を生む（かもしれない）
→ 「運用という名の労働力」以外の何かを売る

SOCの産業構造の変化

SOCの運用が人月産業から知識産業へ

- 運用を自動化する仕組みとそれを面倒見る人という構図になる

  But…

  - 攻撃トレンドは日々変化
  - 「システム入れて終わり」にはならない
- 運用で得た知識（分析ロジック・ノウハウ）を販売するようになる

  But…

  - 今ある知識だけを切り売りしてるといずれ底をつく

アナリストチームに求められるもの

- 日々の運用から問題を発見する
  - 自分達の業務を分析する
- 自分達の困りごとを自分達で解決する
  - 問題解決の手段としてのプログラミング
- 知識を更新し続ける
- 要するにDevOps
  - 改善の利益をチーム全体に還元できる人は重宝される
  - 手法ではなく文化としてDevOpsを身に着ける

まとめ

まとめ

- 自動化の利益をシェアして皆で幸せになろう
- 自動化によるミスが怖い？人間の方がミスするぞ
- 段階を踏んで自動化しよう
  やってみて見える世界がある

余談：上記はだいたいSRE本※の第II部に書いてありました
※ Beyer, B., Jones, C., Petoff, J., Murphy, N. R. 編
SRE サイトリアイアビリティエンジニアリング
オライリー・ジャパン

FAQ

Q：機械学習とか使ってる？

A：作業の領域では使っていない

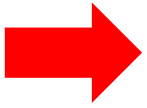
- 現在のスコープは定型化できる作業
  - ルールで書けるものに機械学習はコスパが悪い
- 結果に対する説明が難しい
  - 「AIの判断です」で納得してくれる領域ならいいんじゃないですかね
  - そうでないなら多分人類には早すぎる
- 分析においてはSIEMのアラートで一部利用し始めている
  - ドメインのパターンから怪しいサイトへのアクセスを発見，など
  - アナリストがアラートの真偽を調査

Q：分析は完全自動化できる？

A：意図的にアナリストが判断する部分を残している

- ロジックが意図通り組めていないかもしれない
  - 勿論テストはするが十分なカバレッジを保証できない
- 通知すべき真のターゲットはログの情報とは別かもしれない
  - デバイスが取得したpcapのXFFだけに情報がある，など

自動分析の結果から作成されたレポートを
ノーチェックでお客様に提出するのはまだ怖い



分析からレポートの生成までは自動で実行し，
アナリストがレポートを確認して提出する方式

Q：コード書ける人がいない（いなくなる）んだけど

A：チーム編成の計画に組み込む（ように説得する）

- 開発できるメンバーを採用する
- メンバーが開発できるようになる

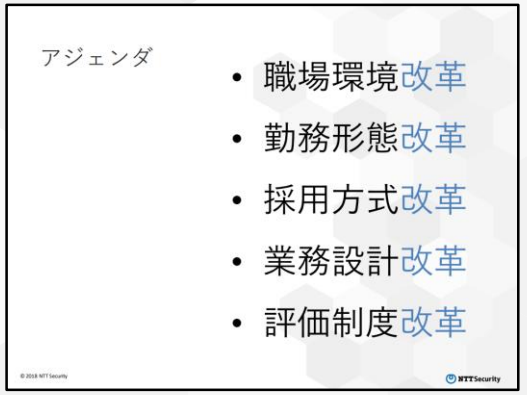
単純にソフトウェア分野の基礎力と考えた場合，教育コンテンツは既に世にある
「課題を解く」というクセ，経験を身に着ける
例：競技系コンテンツをマイペースでやっていく

- **競技プログラミング**
  AOJ，AtCoder，CheckIO，LeetCode，etc.
- **CTF**
  picoCTF，ksnctf，Cpaw ctf，akictf，etc.
- **ISUCON，Kaggle 他各種コンテスト**

Q：NSJだからできたんじゃないの？

A：そうでもないと思う

- 特殊なチーム編成になっているといったことはない
- 勿論組織としてできるように工夫した部分はある
- 「NSJのSOCを作るために何をしたか」については弊社 阿部による講演資料※を参照



※ 阿部慎司, Inside “Security Operation Center” ～セキュリティ人材を生かす5つの改革～
(ISC)^2 Secure Summit Japan 2018
https://custom.cvent.com/1B8FF20CA3284DDD9E69582158291F1D/files/3a02956101504d519f5e5d9ff219bd54.pdf

追記：当日の議論 覚えている範囲で...

Q：全員コード（python）が書ける前提？

A：何かしらコードを読むことくらいはできると思う

- そもそも一切コードを読めないと分析できない
  - 読める言語は人によりけり
- 読む能力と書く能力は別なのでそこは何かしらアシストできれば
  - 書ける人とペアで開発など

Q：開発者がいなくなった古いスクリプトの対策は？

A：スクリプトはほぼ書き捨てのように扱う

- （楽観的な意見として）攻撃トレンドが変わるとスクリプトが意味をなさなくなるから捨てられる
  - 数ヶ月くらい？
- 恒久的に動かしたいスクリプトも「賞味期限」をつけて定期的に更新する仕組みにしたい
  - 最終更新日，ライブラリのバージョンなどで判定
  - まだ構想段階
- 「自動化は自動化していることを忘れないようにする仕組みが重要」（会場にていただいたコメント）

Q：「自動化に抵抗がある」ってなんで？

A：ですよね

- 推測するに「現在回っているものに何故手を入れる必要があるのか」という意見

  → 「回らなくなってからでは遅い」が推進派（私）の意見

- 「抵抗ある人が想定する自動化による弊害」の推測
  - 自分達がどんな作業をしていたのか忘れていざというとき対応できない
  - 何かあったときにどうやって責任をとるのか

  → 「自動化」ではなく「運用方法」の問題

Q：ログが壊れる機器のバグ等にどうやって気付くか

A：現時点では人間が気付くしかない

- 機器導入・更新前の検証で気付くチャンスがある
  - 検証でのテスト自動化は自分達のチームでは取り組んでいない（範囲外）
- うっかり導入されても機器のバージョンなど広範囲に影響する場合
  他の分析をしている際に気付けるのでは