熊本高等専門学校八代キャンパスにおける
侵入検知システムの構築と運用

小島 俊輔 1,*, 藤本 洋一 1, 岩本 舞 2

Construction and Operation of Intrusion Detection System
at National Institute of Technology, Kumamoto College, Yatsushiro Campus
Shunsuke Oshima1*, Yoichi Fujimoto1, Mai Iwamoto2

In order to detect cyber security threat at NIT Kumamoto, Yatsushiro Campus, we construct and operate an Intrusion Detection System (IDS) in our computer network since December, 2017. IDS has based on a predetermined signature and it can detect intrusions, attacks and other signs by monitoring packet-flow through the computer network. In this paper, we report introduction, construction and operation of this system.

キーワード：侵入検知、ネットワークインフラ、サイバーセキュリティ、Snort、Security Onion

Keywords: Intrusion Detection, Network Infrastructure, Cyber Security, Snort, Security Onion

1. まえがき
熊本高等専門学校八代キャンパスでは、学内ネットワークにおけるサイバーセキュリティの脅威を検出するため、2017年度より侵入検知システム（Intrusion Detection System, 以下 IDS と略）の1種であるSnortを導入している。また、2018年度よりSnortを基幹として、セキュリティイベントの可視化やログの収集・検索機能など多くのネットワークセキュリティ関係ツールを駆使したLinuxディストリビューションであるSecurity Onionを導入し実績を挙げている。本稿では、熊本高専内のネットワークにおける新設や運用状況に加え、新たな利用事例を紹介する。さらに、実際にIDSを使用したことで得られた知見についても併せて記載しておく。

2. IDS
2.1 IDSの分類
IDS は、ネットワークを通じたパケットやホストのログなどを監視し、あらかじめ設定したベースラインにマッチする軌跡を検出することで不正侵入を検知するシステムである。有名なIDSとしてはSnortやSuricataが挙げられる。

IDSは、検知する場所によって大きく2つに分類することができます。一つは、ホストコンピュータなどに常駐し、リソースやログを監視するホストベース型、もう一つは、ネットワークを流れるパケットを監視し、異常を検知するネットワーク型である。

今回導入したIDSにはネットワーク型を選択する。ネットワーク型とする理由は、1)OS タイプがWindows, Linux, MacOS, Android の場合はマルウェア対策専用のソフトウェアが存在するが、プライベート、スキャン、マイコンや各種IoT 機器では専用のマルウェア対策ソフトウェアがなく、これら機器の対策が急務であること、2) BYOD (Bring Your Own Device) によるデバイスの持ち込みや個人所有のUSBメモリの接続により、マルウェア対策が十分ではない機器から不正なコードが侵入しやすくなる可能性がある。などが挙げられる。

ネットワーク型IDSであれば、OS およびデバイスに依存せず、デバイスが不正アクセスを受けたり、デバイスからのネットワーク機器に対する異常な動きがあった場合に、早期に検知できる可能性が大きい。

2.2 Snortの概要
Snort はオープンソースのネットワーク型IDSである。1998年に開発されSourcefire社からオープンソースとして配布されてきたが、2013年にCisco Systems社が買収、その後もオープンソースとしてSnort 3が配布されている。
図1 Snort ルールの書式と例

Snort はパケットを常時監視し、Snort ルールと呼ばれる、あらかじめ定義したパターンにマッチする違反パケットを検出した場合に、警告などのアクションを実行する。図1に Snort ルールの簡単な例を示す。このルールはユーザが個別に記述し、組織独自のルールセットを作成することがでる。ルールセットが侵入検知の性能に直結するため、通常は無料の Community ルールセットを用いるか Personal や Business といった Cisco Systems 社が提供する有料のプランを選択し、組織ごとのカスタマイズを加える。

Snort ルールを最新に保つための仕組みとして、Pulled Pork と呼ばれるアップデート、および Snort ルールをダウンロードする際の認証コードである Oinkcode が必要となる。いずれも専用サイトから取得することができる。サイトから取得した Oinkcode は Pulled Pork の設定ファイルに埋め込むか、後述する Security Onion のインストール時に指定する。

2.3 Security Onion の概要


Security Onionは、インストール時にSnortまたはSuricataのどちらか一方をIDSとして選択することができる。IDS以外に以下のようなツールが標準でインストールされる。

- Erasticsearch: 全文検索エンジン
- Logstash: ログイベントの監視と保管
- Kibana: プラウザベースのログ可視化ツール
- Sguil/Square: データ分析ツール
- Bro: 振る舞い指向フォレンジックツール
- Wireshark: パケット指向フォレンジックツール
- netstun-ng: 生伝統計ツール

これらのツールは単独で起動して使用する場合もあるが、SguilやSquertなどのデータ分析ツールのドロップダウンメニューから、相互に連携し補完することで様々なサイバーセキュリティの脅威をシミュレドに解析できるよう設計されている。

3. Snort の導入と構築

3.1 Snort 引入時の検討事項

熊本高専八代キャンパスでは、2017年度に学内外の通信の異常を検知する目的で、Proxy サーバを通じたパケットの監視をする Snort を導入した。導入の際には、Snort の Business プランの費用を 2017年度の予算で確保している。

Snort はネットワーク型 IDS であり、センサーノードを複数個所に設置することもできるが、監視対象の数が多いとランニングコストも高くなるので、今回は熊本高専八代キャンパスの Proxy サーバ1台のみに設置した。

次に必要となるのは Snort を稼働するサーバである。先の Proxy サーバは HTTP Proxy 以外に、学内 (LAN) と学外 (WAN) を接続するゲートウェイとして動作しており、学内外の全パケットが通過する。2017年度のサーバ費用が拡大できなかったため、Snort を別サーバではなく Proxy サーバに搭載することとした。全体の構築を図2に示す。この図からもわかるように、Snort 側の設定は監視対象のネットワークインターフェースを指定する方式であり、Proxy サーバ内で構築することで Proxy や Default Gateway の設定変更は必要ない。Default Gateway の設定変更が必要ないということは、たとえば Snort の利用時間帯を変更した場合でも、Snort プロセスを緊急停止すればすぐに元に戻るという大きなメリットがある。

3.2 サーバスペックの決定

Proxy サーバは、Intel Xeon 4Core 3.0GHz の Linux をホスト OS とする2CPU12000BogoMips、Mem4GB の KVM 仮想マシンとして構築している。日中の通信量はおよそ 100Gbps であり、Proxy のみの状態で CPU 負荷は数パーセントである。この Proxy サーバ内に Snort を導入すると CPU 負荷が高くなることが予想されるが、Proxy サーバは仮想マシンであり、高負荷の際には仮想マシンのスケジュールを上げることを検討すべきである。

結果、Snort を稼働した際の CPU 負荷は 10〜30%程度となり、外部との接続性に影響を及ぼすトラブルは確認できなかった。そのため、今回は仮想マシンのスペックをそのまま変更することなく本稼働に移行することができた。
図3 Snortによる異常検知の例

3.3 異常検知の例

検知したSnortが検知した異常の例を図3に示す。セキュリティ上の理由により、IPアドレスは一部加工してある。

Snortの異常検知は、この図のように違反バケット1個につき1行のログ、およびパケットダンプを出力する。ここで特筆すべきは、pcap形式として出力されたパケットダンプは違反したパケットのみであるということだ。管理者が異常を知るには、まずSnortのログを目視で観察し、怪しいと思われるパケットをWiresharkなどのパケット解析ツールで検索する必要がある。通常、一つの通信セッションは複数のパケットから構成されており、保存された1パケットのみから異常かも判断することは難しいが、Snortのままでは違反パケットの周囲のパケットまで含めた検査をすることはできない。

また、本校でSnortを稼働してわかったことは、違反パケットのログの個数が意外に多いということである。図3の例では約1分おきにログが出力されており、違反パケットはこのように頻繁に発生している。そのため、内容をすべて観察することは時間的に無理があると判断し、メールによりログを自動通知する仕組みを構築した。また、SnortがPriority 3以上の付与されたログやパケットダンプに絞って詳細に解析するなど実働レベルの工夫をした。

4. Security Onionの導入と構築

4.1 Security Onion導入時の検討事項

前章で述べた通り、Snortの異常では違反パケットのログとpcap形式のパケットダンプ1個1個を手作業で解析するのが限界がある。そこで、2018年度のネットワークの監視用PCを導入するための予算措置を行い、2019年3月にSnortをIDSとしてSecurity Onionを稼働した。なお、Snortは2017年度のBusinessプランを継続することとし、OinkcodeはSecurity Onionにそのまま引き継いだとした。

4.2 サーバーマシン

今回導入した専用PCのマシンスペックを表1に示す。導入したPCは将来の拡張性を考慮してミドルクラス型のデスクトップPCとした。HDD容量は導入予算との兼ね合いで最小限の2TBとしたが、容量が大すぎてそれだけログやパケットダンプの長期保存が可能となる。熊本高専八代キャンパスのネットワークの利用状況を調査したところ、2TBの容量は、夏休みなどの長期休暇中で1週間程度、平日であれば0.5〜1日で使い切ることが分かった。そのため、10日間のパケットダンプを保持する場合は単純計算で20〜40TBの容量が必要となる。このことから、ログ用HDDの増設に備えてSATAポートや電源容量、ケースの空きスロットはできる限り余裕をもって確保しておいたほうが良い。

また、今回はCPUのコア数を6とした。コア数を多くした理由は、Security Onionでは、SnortやBroといった処理が重いプロセスのほかに、仮想マシンを複数動作させなければならないからである。今回、6コアを確保できたため、SnortのプロセスをSecurity Onion内部で2つ起動した。これによりパケット解析を2つのコアで分散処理でき、パケット解析の取りこぼし要因を減らすことができる。また、Security Onionではそれ以外にDockerと呼ばれるコンテナ化仮想化環境が必要である。通常のLinuxディストリビューションでは一部のファームウェアをバージョンアップした上で動作不良が発生することがたびたびあるが、Dockerではコンテナと呼ばれる一粒のファームウェアリソースを提供するため、いわゆるファームウェアパッケージによる相性問題が発生しない。このDockerの特徴を利用してElasticStackの属性すべてのサービスが提供される。手元の環境では、Security Onionが起動するとcontainerd-shimが5つ、docker-proxyが5つ自動的に起動している。

構築後に、Security OnionのCPU使用率を調査した。具体的には、シェルにログイン後、topコマンド実行中に1を押下するとコアごとのリアルタイムな使用率が表示される。図4に結果を示す。

CPUは逆に続くusがユーザープロセス、syがシステムプロセス、idがアイドル時間であり、それぞれの割合（%）が表示されている。図からもわかるとおり、6コアそれぞれが均等に負荷分配されている。平均して20.0%のCPU使用率であることがわかった。CPU使用率は余裕があるように思えるが、突然発生の高負荷に対するためにはこの程度で留めておいた方が良い。また図4の下部にCPU使用率の高い順にプロセスが表示されている。

表1 Security Onion専用PCのマシンスペック

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz 6Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEM</td>
<td>16GB</td>
<td></td>
</tr>
<tr>
<td>HDD</td>
<td>2TB</td>
<td>可能なら20〜40TB</td>
</tr>
<tr>
<td>Network Interface</td>
<td>1000Base-TX × 2</td>
<td></td>
</tr>
<tr>
<td>SATA</td>
<td>6ポート (できるだけ多い方が良い)</td>
<td></td>
</tr>
<tr>
<td>ケース</td>
<td>ミドルクラス型5inchベイ5スロット</td>
<td></td>
</tr>
<tr>
<td>電源</td>
<td>600W (できるだけ多い方が良い)</td>
<td></td>
</tr>
</tbody>
</table>
4.3 全体構成

今回設置した Security Onion の設置場所および全体構成を図 5 に示す。Security Onion 用のサーバには、2つのネットワークインタフェースを用意した。一方は学内ネットワークに接続するため、もう一方は膨大な通信を監視するためのものであり、後者はプロミスキャスモードで動作させることとした。パケット監視専用のポートを用意したことで、通常のサーバとの間の通信パケットが分離でき、監視パケットの欠損を減らすことが可能となる。

4.4 異常検知の例

ここでは、Security Onion で検知した具体的な違反パケットや可視化されたログの例を示しておく。図 6 は、Squert と呼ばれるデータの分析に使用するツールのスクリーンショットである。Snort が違反パケットとして判定した情報を基に、横軸に時間、縦軸に発生件数を示したグラフが描画され、違反パケットの時間変化を確認することができる。また、図の下部には、検知された違反パケットが種類ごとに分類され表示されている。これにより、違反パケットの総数や傾向が一目でわかる。

図 7 には Kibana と呼ばれるパケット可視化ツールのスクリーンショットを示した。図は、平日のある DNS クエリについて、横軸を時間、縦軸を回数としてグラフを表示している様子である。就業時間である午前 8 時頃から上昇し、17 時を過ぎると徐々に下降していく。異常が発生したときは DNS クエリに変化を来たすことが多く、このように日頃から平常時の DNS クエリの様子を観測しておくとよい。
リケーションのアップデートパッチが提供されるなど、対策が施されているものが大半を占めていることがわかった。

また、これまでに、学生が学内に持ち込んだ端末がアドウェアに感染しており、ブラックリストとして登録されているIPアドレスと通信していたケースが5件あった。いずれも学内へのマルウェア拡散や情報漏洩などのインシデントには至っていない。発見後の対応として、違反パケットのIPアドレス情報と学内Wi-Fiの使用記録、IPアドレス払い出し情報等を問い合わせることで学生を特定することができたため、本人に注意喚起しマルウェア対策やHDDフルスキャンを求めた。また一部の学生には1時間程度の面談を実施して詳細の確認などを行った。

4.5 構築状況の調査

Security Onionの稼働状況を調査した結果を図9に示す。Security Onionは5分おきにログを出力しており、グラフの縦軸はすべて5分間あたりの平均値として示している。図9(a)の通過パケット数は、5-20kPackets/secであり、多くのパケットを処理している様子がわかる。また、図9(b)のCPU使用率の調査では20-40%程度を示しており、いずれの場合においてもCPU100%とならないことがわかった。

一方、図9(c)の違反パケットのアラート数については注意が必要である。グラフの最大値0.93件/secは5分間あたりの平均値としての表記であり、つまり5分間に発生した違反パケットのアラート数を300秒で割った1秒あたりの件数として表示している。そのため、グラフ値に300を乗じたものが実際の5分間あたりのアラート件数となり、最大で5分間あたりおよそ280件のアラートが報告されていることがわかる。ちなみにこのアラート最大件数はトップレベルドメイン（TLD）のDNS検索に関するものであった。それ以外にも多くの違反パケットのアラートが挙がっている。

図10では、2019年8月8日（木）一日当たりの処理パケット数を示している。この日本は学校では通常授業（試験発卸等）が実施されている。図から、就業開始直後に上昇し、就業終了後徐々に減少するという一日の処理パケットのおおよその変化が読み取れる。

5. まとめ

今回、SnortとSecurity Onionを導入した。導入時のコストはそれほど高くはないものの、導入後のコストが非常に高く、1件の違反パケットを調査するだけでも多くの時間を必要とすることがわかった。違反パケットの調査を進めていく中で、違反パケットの多くが誤検知で占められていることが分かった。導入当初は違反パケットのすべてについて追跡調査をしていたが、非常に多くの時間を浪費してしまうことから、過去に調査したものとよく似た違反パケットについては追跡検査しなどの工夫も必要であり、ある程度の経験が必要である。また、将来的には誤検知を減らすようSnortルールの全体的な見直しも必要となろう。
一方、誤検知の中には、SSL など暗号化された通信も含まれており、暗号化されたランダムなパケットが違法パケッ トのパケットと偶然一致したようなものも見受けられ た。1 日あたり 2TB の通信が発生する Proxy サーバの環境下 ではこのような偶然が数多く発生するが、この誤検知を減らす抜本的な解はがない。

今後の検討事項として、暗号化パケットの追跡がある。現在、熊本高専八代キャンパスの Proxy サーバでは SSL な どの暗号化通信がパケット全体の 5 割を占めている。現在 導入している Security Onion は、暗号化通信の中でまで調査 する手段がないため、単純計算で違法パケットの半数が見 過ごされていることになる。今後、暗号化通信の割合はま

すすむことになることが予想されており、SSL インスペクシ ョンなど暗号パケットを平均パケットにする新たな仕組み が必要となる。この実現には技術的な課題もさることなが ら、倫理的な問題もある。たとえば、ログインが必要なサ イトのアカウントやパスワードなどを含めすべてが Security Onion で不正に復活されてしまうため、パスワード を盗聴していることとはなくならない。実現には学内の コンセプスを得る必要があるが非常に困難であると考え る。

最後に、Security Onion は多機能であり、使いこなしには 決然と慣れると経験が必要であることを痛感している。セキュリ ティに関する教職員も勉強会などを開催するなどしていきたい。

参考文献

(10) VirusTotal Community : VirusTotal, https://www.virustotal.com/, 2019/9/21