目 録

第1章　序論

近年，ネットワークの高度化やパソコン，スマートフォンの普及によるインターネット利用者の増加により，マルウェアも増加傾向にある [1]．それに伴い，ネットワーク通信を介して情報漏洩やデータの改ざん，マルウェアなどの脅威に晒される可能性が高まってきており，それらの対策を企業などの組織だけでなく個人でも講じておくことが重要になっていると考えられる．

マルウェアに感染したコンピュータは，意図せず異常な通信を行うことが知られている．そこで，パケットの送受信状況を監視することでマルウェアの検知，早期対処を行ったり，感染時の通信状況を見ることでその感染の原因，被害状況を把握することができる．

パケットを通じて通信状況を把握する研究は盛んに．例として，独立行政法人情報通信研究機構 (NICT) が推進する NICTER(Network Incident analysis Center for Tactical Emergency Response) [2] と呼ばれる研究プロジェクトがある．このプロジェクトでは，ダークネットと呼ばれる未使用の IP アドレスに到達するパケットの情報を分析し，可視化することで直感的に分かりやすく通信状況を把握できるシステムを開発している．ただし，この NICTER は大規模なシステム向けに開発されており，個人のコンピュータの監視には向いていない．個人がパケットを観察するツールは Wireshark [3] や tcpdump [4] などが存在するが，これらは文字情報のみで分析結果を表示するため通信状況を直感的に理解しにくいことや，結果を把握するにはプロトコルや IP アドレス等の知識が必要であることから万人が通信状況を理解するのは難しいという課題もある．

これらの課題に対して，田村らの研究ではホワイトリスト方式で地球儀上にパケットを可視化させるシステムの開発を行っている [5]．ホワイトリスト方式とは，信頼できる通信のみをリスト化し，リストと一致しない場合に警告や遮断を行う異常検知手法である．この方式では，正常な通信を異常な通信とみなすフォールスポジティブなどの誤検知のリスクが高まってしまうため，検知精度に課題が残る．また，信頼できる通信を自身で判断し，ホワイトリストの設定を行う必要があるため情報工学の知識が必要となり，万人が利用するシステムには向いていない方式と言える．

そこで，本研究ではリアルタイムの通信状況からブラックリスト方式で異常通信を検知

し，その結果をパケット情報とともに可視化する個人向けシステムの開発を行う．ブラックリスト方式とは，信頼できる専門家などによって既知の攻撃パターンをリスト化し，通信時にリストとのマッチングで一致した場合に警告を出す，あるいは通信を遮断する方式である．情報工学の知識を有した専門家がブラックリストを日々更新している事が多く，それらを利用すれば設定などをする必要なく異常通信を検知することができる．

以下，2 章ではインターネットの仕組みと不正通信の検知方法について述べる．3 章ではパケットの分析と可視化に関する研究について述べる．4 章では本研究で提案するシステムの構成とシステム評価について述べる．最後に，5 章では本研究のまとめと今後の課題について述べる．

第2章　インターネットと異常通信検知手法

マルウェアに感染したコンピュータの多くは，インターネットを介した異常通信を行う．この異常通信を観測するには，インターネット通信の仕組みを知ることが不可欠である．そこで，本章ではインターネット通信の仕組みと，通信の分析に必要な情報の取得方法，異常通信の検知手法について述べる．

2.1　プロトコルの役割

コンピュータネットワークには TCP や IP をはじめとした複数のプロトコルが必要であり，TCP/IP はこれらを総称したプロトコル群のことである．現在，TCP/IP はコンピュータネットワークにおいて最も利用されているプロトコル群である．TCP/IP のプロトコルは役割ごとに階層化することができる．この階層モデルを図 2.1 に示す．

図 2.1　TCP/IP の階層モデル

TCP/IP による通信では，送信側はアプリケーション層で作成したメッセージを分割し，下位層に順番に渡す．この時，各層では上位層から渡されたデータにヘッダを付加する．こうして送信データにヘッダを重ねていったものがパケットとなる．最終的に送信するパケット

の構造の一例を図 2.2 に示す.

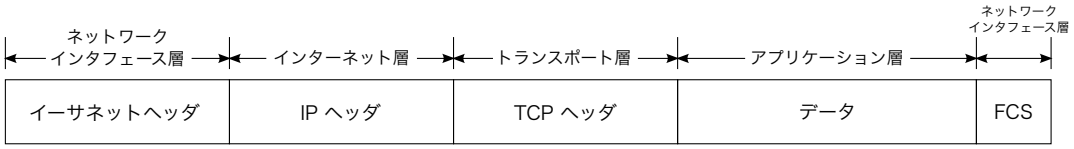


図 2.2　パケットの構造の一例

受信側は送信側からパケットを受け取り，下位層から上位層へとパケットを渡す．この時各層でパケットのヘッダを解析することで，どのようなプロトコルが使われているかがわかり，適当な上位層へデータを渡すことができる．

以下に，TCP/IP 階層モデルの各層の詳細を述べる．

ネットワークインタフェース層

ネットワークインタフェース層は，データリンクを利用して通信を行うためのインタフェースとなる階層である．ネットワークインタフェース層で扱うパケットのことを特にフレームという．データリンクの中で現在最も普及しているのが Ethernet であり，Ethernet で伝達されるフレームはすべて Ethernet フレームとして運ばれる．Ethernet の中でもいくつか異なる仕様が存在するが，そのフォーマットは 2 種類に大別される．Ethernet のフレームフォーマットを図 2.3 に示す.

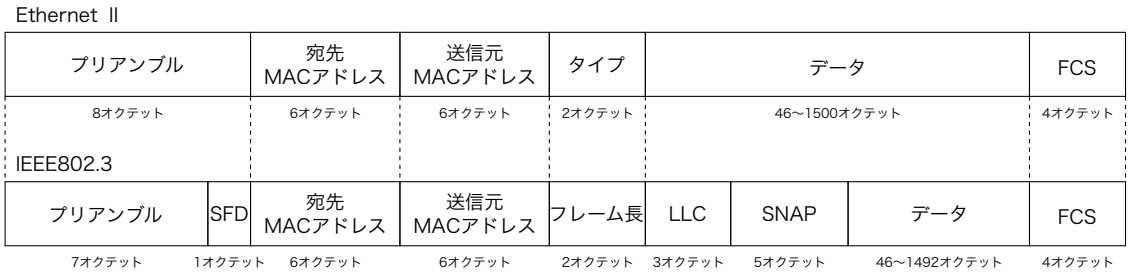


図 2.3　Ethernet フレームフォーマット

インターネット層

インターネット層は，宛先までデータを届ける役割を持つ階層である．インターネット層では，IP(Internet Protocol) が支配的に用いられる．IP はパケットが宛先に正しく届いたかを保証しない．このように保証のないパケットのことを特にデータグラムという．現在用いられている IP にはバージョン 4(IPv4) とバージョン 6(IPv6) があるが，ここでは特に主要に用

いられている IPv4 について述べる．

図 2.4　IPv4 ヘッダフォーマット

図 2.4 は IPv4 のヘッダフォーマットを示した図である．この図に示す通り，IP ヘッダ内に送信元・宛先 IP アドレスが示されており，このおかげでインターネットを介してパケットをやり取りすることができる．そのため，インターネットに接続されるすべてのホストやルータは，必ず IP の機能を備えていなければならない．

トランスポート層

トランスポート層の最も重要な役割は，アプリケーション間の通信を実現することである．これは，コンピュータ内部では複数のアプリケーションが同時に動作しており，どのプログラム同士が通信しているのかを識別する必要があるためである．これには，ポート番号という識別子が使われる．

トランスポート層では代表的なプロトコルが 2 つ存在する．

TCP(Transmission Control Protocol) は，コネクション型で信頼性を持つプロトコルである．もし通信経路の途中でデータの損失や入れ替わりが発生しても，TCP によって解決することができる．ただし，信頼性を高める代わりに制御のパケットをやり取りする必要があり，一定間隔で決められた量のデータを転送するような通信にはあまり向いていない．

UDP(User Datagram Protocol) は，コネクションレス型で信頼性のないプロトコルである．

TCP とは違い，送信したデータが宛先に届いているかの確認をしないため，データの損失や入れ替わりの確認はアプリケーション側で行う必要がある．しかし，確認しない代わりに効率よく通信を行うことができ，パケット数が少ない通信や，音声通信等の一定間隔の通信に向いたプロトコルと言える．

アプリケーション層

アプリケーション層では，アプリケーション内で行われるような処理を行う．ネットワークを利用するアプリケーションでは，アプリケーション特有の通信処理が必要である．アプリケーション特有の通信処理にあたるのがアプリケーションプロトコルである．アプリケーションプロトコルは，ブラウザとサーバー間の通信に使われる HTTP や電子メールの送受信に用いられる SMTP 等がある．

2.2　異常通信の検知手法

1 章で述べたように，マルウェアに感染したコンピュータは意図せず異常な通信を行い，次のコンピュータへとマルウェアを感染させていく事が知られている．本研究では，近年社会問題となっているマルウェア増加に対してこれらの異常通信を検知し，マルウェア感染の早期発見，対処をすることで対策を行う．

本節では，異常通信を検知するために用いる異常通信検知手法を述べる．異常通信を検知する手法は，大きくブラックリスト方式，ホワイトリスト方式，アノマリ型検知の 3 つに分類することができる [6]．

ブラックリスト方式

ブラックリスト方式は信頼できる専門家などによって既知の攻撃パターンをリスト化し，通信時にリストとのマッチングで一致した場合に警告を出す（あるいは通信を遮断する）方式である．過去の攻撃に合致したパターンのみを検知するため，正常な状態を異常とみなすこと（フォールスポジティブ）が少ないという特徴がある．しかし，未知の攻撃に対してはパターンを適用することができず，異常を見逃してしまうこと（フォールスネガティブ）が起こってしまうことがある．

ホワイトリスト方式

ホワイトリスト方式は，信頼できる通信のみをリスト化し，リストと一致しない場合に警告や遮断を行う方式である．ブラックリスト方式とは対照的に，フォールスネガティブは少なく，攻撃被害への耐性は高い．一方で，フォールスポジティブに伴う通信の遮断によってサービス停止を招いてしまうリスクも高まることから，安全性と運用上のコストは両立しにくいと言える．

アノマリ型検知

アノマリ型検知は，通常の状態のプロファイルを設定しておき，これに違反した場合に異常とみなす検知の方法である．ブラックリスト方式と比較して異常の定義が広いため，フォールスポジティブが比較的多くなってしまうが，プロファイルの閾値を調整することで誤検知を減らすことができる．また，この方法では，過去に観測されていない未知の攻撃に対しても対応できるというメリットがある．

以上が代表的な異常通信の検知手法であるが，どの方法にもメリットとデメリットがそれぞれ存在するため，互いのデメリットを補うために複数の方法を組み合わせて実装することが多い．

第3章　パケットの分析と可視化

本章では，ユーザのコンピュータ上でパケットを分析する手法と，現在研究開発が進んでいるパケット可視化に関する研究について述べる．

3.1　ツールを用いたパケットの分析手法

ユーザがコンピュータ上のパケットを分析する際には，専らパケットキャプチャツールが利用される．パケットキャプチャツールの中から，ここでは一例として Wireshark [3] と tcpdump [4] について説明する．

Wireshark

Wireshark はパケット分析において最も普及しているツールである．Wireshark は豊富な機能を持っている．その一部を以下に示す．

- 何百ものプロトコルを常に詳細に検査する
- リアルタイムでのパケットキャプチャとオフライン分析
- マルチプラットフォーム:Windows，Linux，MacOS など，多くのプラットフォーム上で動作可能
- パケットの特徴ごとに色分けして表示することで直感的にフィルタリングを理解可能にする
- XML，CSV，またはプレーンテキストに出力することが可能

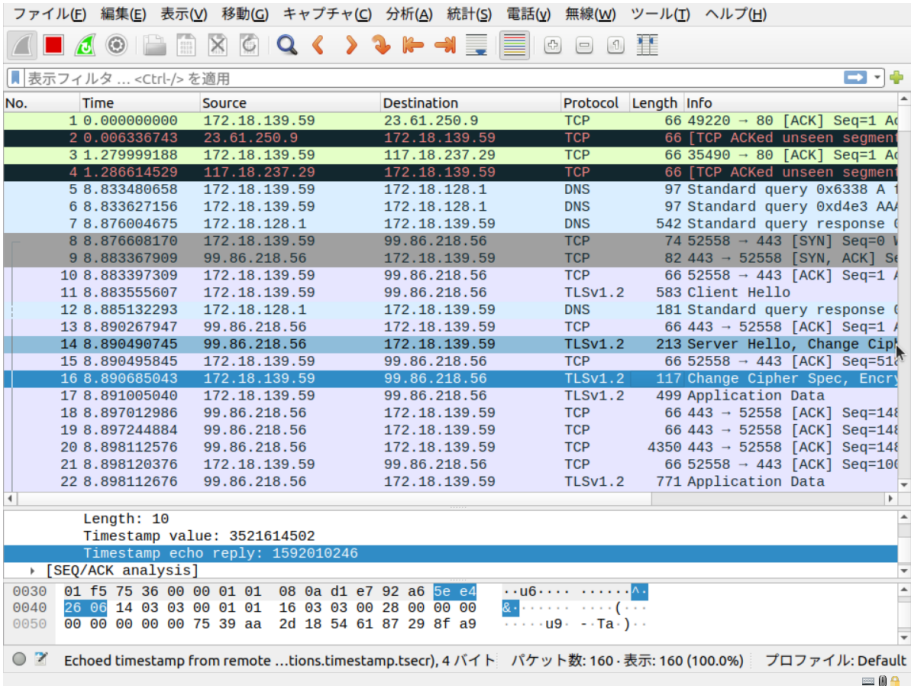


図 3.1　Wireshark を用いたパケット分析

Wireshark を起動し，ネットワークインタフェースを指定してキャプチャを開始すると，図 3.1 のようにリアルタイムに取得したパケットが表示される．パケットリストは独自の方法で色分けされている．パケットリストからパケットを選択すると，さらに詳細なパケット情報が表示される．このように，知りたいパケットを見つけて詳細情報を確認することで，通信状況の分析を行う．

tcpdump

tcpdump はネットワーク通信のデータを収集し，結果を出力する分析ツールである．

図 3.2 のようにパケット取得日時，送信元，宛先 IP アドレスだけでなく，どのようなフラグ (SYN,ACK,FIN 等) のパケットが送られたかがわかるようになっている．ほかの分析ツールと比較してリソース消費が少ないなどといった利点がある．

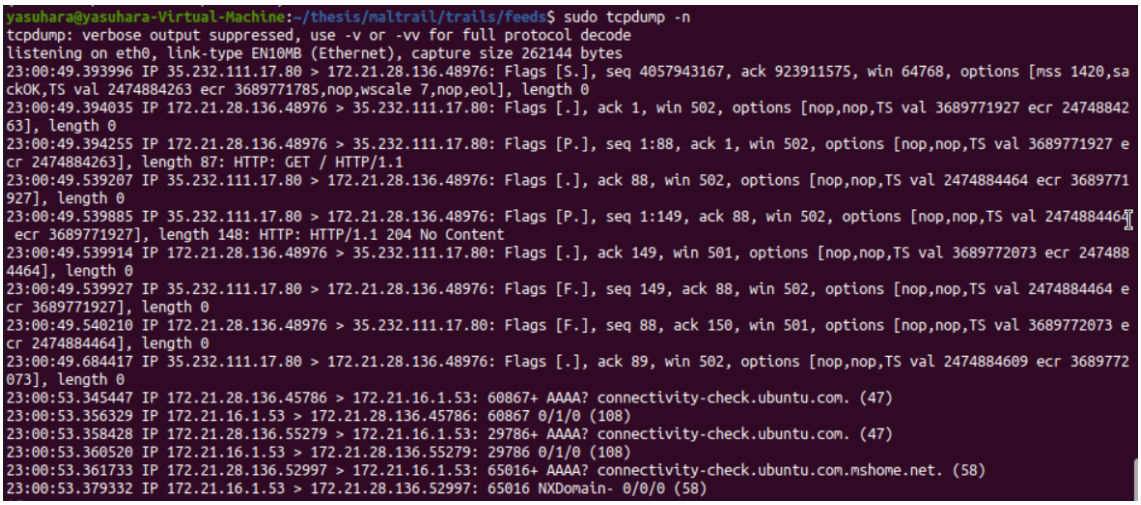


図 3.2　tcpdump を用いたパケット分析

しかし，2 つのツールともに，アドレスやプロトコルなどの専門的な用語が表示されており，これらの用語の意味を知らない人にとってはどこが異常かを判別するのは難しい．Wireshark や tcpdump に限らずほとんどのパケットキャプチャツールについても同様の事が言える．

3.2　侵入検知ツール

3.1 節で述べたツールを用いた手法は，異常の有無をユーザが能動的に調べるための手法であった．これに対して，異常通信を自動的に発見する手段として現在最も利用されているのが，侵入検知システム (Intrusion Detection System, IDS) を用いた方法である．IDS はネットワークへの異常な通信を検知し，管理者に知らせる機能を持ったソフトウェアもしくはハードウェアのことである．

ここではオープンソースのソフトウェア IDS の中から，一例として Suricata と Maltrail について述べる．

Suricata

Suricata [7] は，ブラックリスト方式で異常通信を検知する高性能のネットワーク IDS，およびネットワークセキュリティ監視ツールである．このツールは，コミュニティが運営する非営利団体である Open Information Security Foundation (OISF) によって開発されている．

使用するブラックリストとして，ルールと呼ばれるコミュニティ，およびユーザーが定義したリストを使用してネットワークトラフィックを調べ，異常を検出する．デフォルトでは

IDS として機能するが，異常を検知した場合にその通信を遮断する侵入防止システム (IPS) としても利用できるツールである．

以下に Suricata の特徴を述べる．

- 異常を検知するだけでなく，その原因となったマルウェアをダウンロードできる
- パケットよりも上の階層である TLS/SSl 証明書，HTTP リクエスト，DNS リクエストなどもログに記録できる
- クロスプラットフォームサポート-Linux，Windows，macOS，OpenBSD など

Maltrail

Maltrail [8] は，先ほど述べた Suricata と同様にブラックリスト方式で異常通信を検知するトラフィック検出システムである．ブラックリストは，Web 上で一般公開されている様々なブラックリストを使用している．

以下に Maltrail の特徴を述べる．

- 様々なアンチウイルスレポートや静的トレイルに加え，ドメイン名・URL・IP アドレス・HTTP ユーザエージェントヘッダ値の痕跡を利用する
- 新しいマルウェアなどの未知の脅威の発見に役立つ高度なヒューリスティックメカニズムやホワイトリスト方式での検知を併用できる

Maltrail は他のツールと比較して使用するブラックリストが多く，より多くのマルウェア検出に役立つと考えられる．そこで，本研究のシステムに Maltrail が取得するブラックリストを使用する．

3.3　トラフィック可視化システム

本節では，企業や非営利団体が開発を行っているトラフィック可視化システムについて述べる．

NICTER

独立行政法人情報処理研究機構 (NICT) が進めている研究に NICTER(Network Incident analysis Center for Tactical Emergency Response) [2] がある．NICTER は「インターネット

上で時々刻々と発生しているセキュリティインシデントへの迅速な対応」を目的としており，インターネット上で生起する多種多様な事象の収集および分析を実施している．また，NICTER で行われている研究の一つがパケットの可視化である．

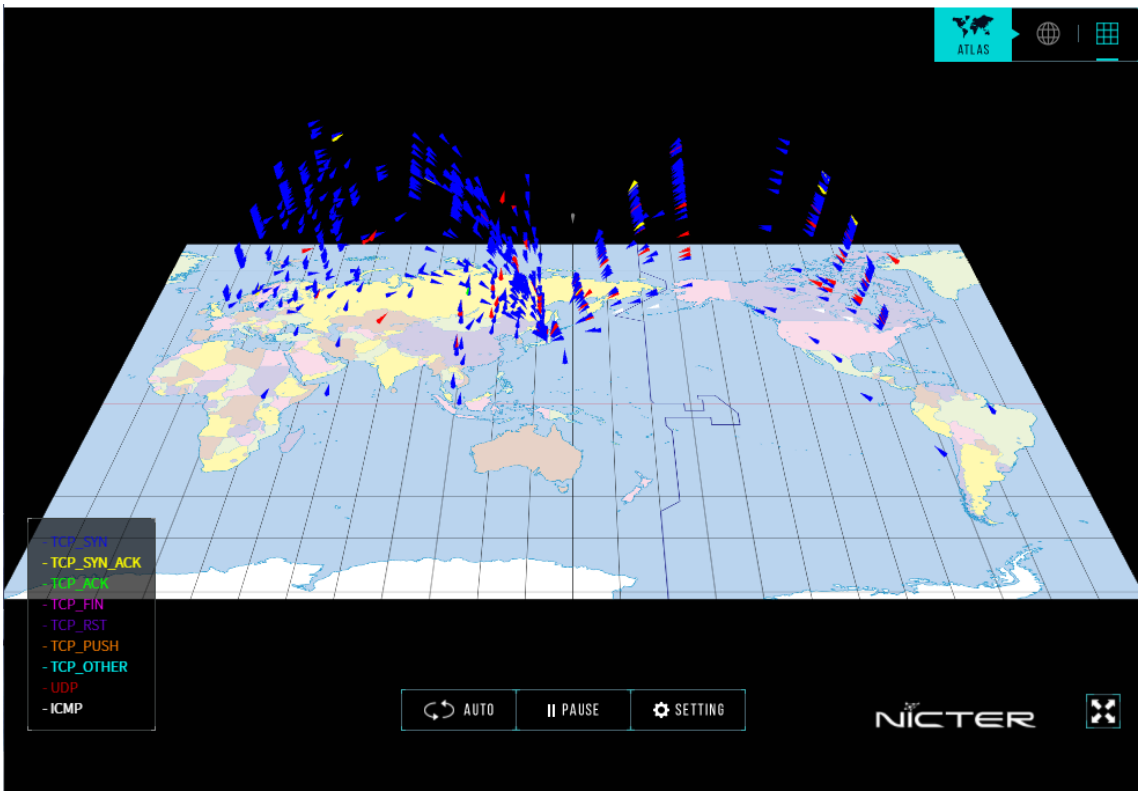


図 3.3　Atlas によるサイバー攻撃の可視化

図 3.3 は Atlas [9] と呼ばれるダークネットに届くパケットを，世界地図上でアニメーション表示する可視化システムである．図 3.3 では，日本国内のセンサに届いたパケットに対してリアルタイムに行われている攻撃を可視化しており，世界地図上の各国から日本に向けてデータが飛んでくる様子がわかる．Atlas はトラフィックを単に可視化するだけでなく，高さでポート番号，色でプロトコルを示すことで，わかりやすく，多くの情報を理解可能にしている．専門知識のない人でも見るだけで状況を把握でき，専門知識がある人ならさらに詳しい分析を行えるようになっている．

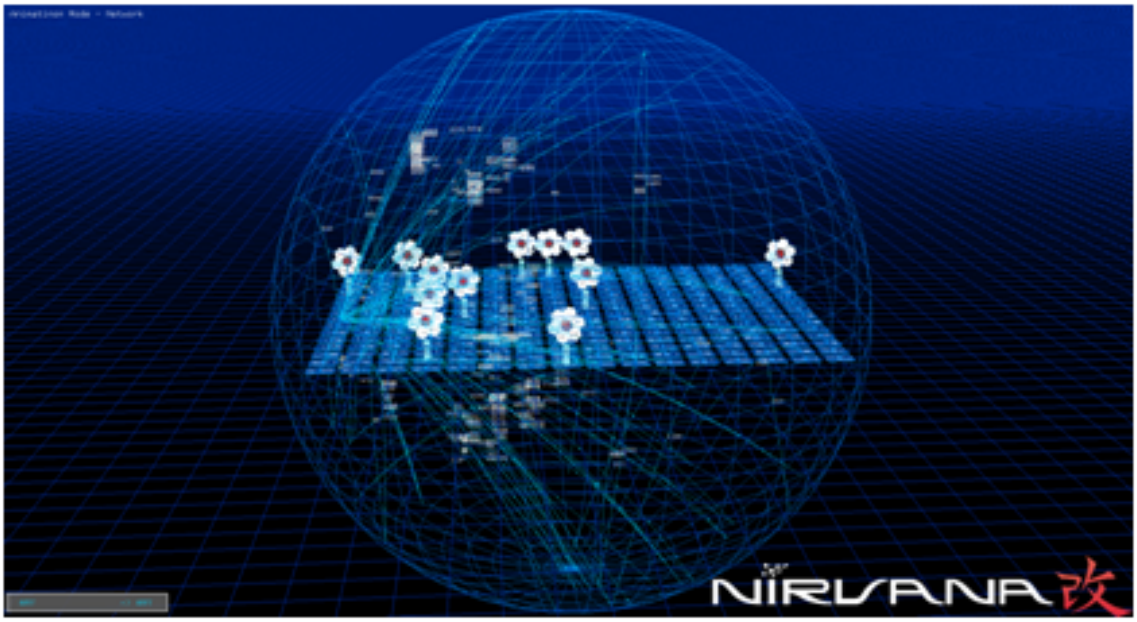


図 3.4　NIRVANA 改によるサイバー攻撃の可視化

図 3.4 は NIRVANA(NIcter Real-time Visual ANAlyzer) 改と呼ばれる NICTER の可視化技術を用いて作られたリアルタイムネットワーク可視化システムである．このシステムは組織内ネットワークの通信状況を可視化するだけでなく，サイバー攻撃に関連した異常通信を検知し，警告を表示する事も可能である．球体がインターネット全体，中央のパネルが組織内のネットワーク，白い目印が警告を表現している．通信の様子が可視化されており，通信状況を見るだけで理解することができる．

CYBERTHREAT REAL-TIME MAP

CYBERTHREAT REAL-TIME MAP [10] は，カスペルスキーが同社のセキュリティ製品を利用して収集したデータを可視化するシステムである．図 3.5 に CYBERTHREAT REAL-TIME MAP を示す.

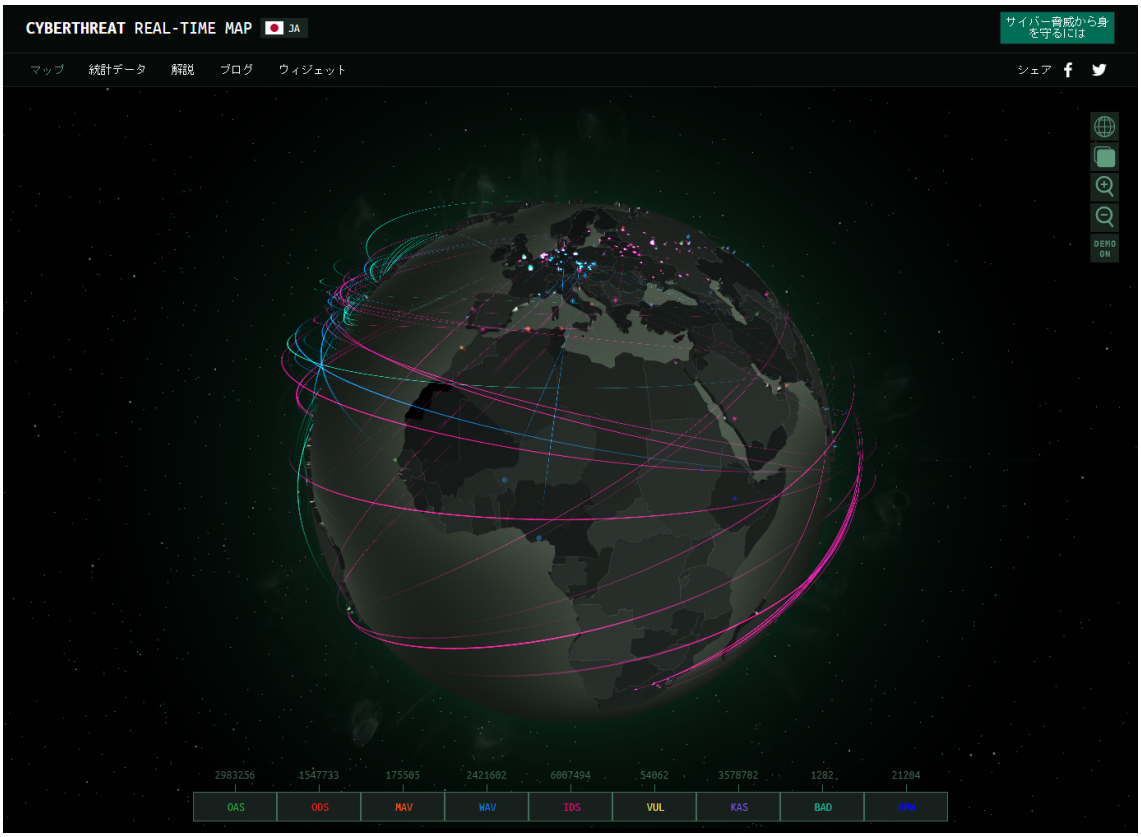


図 3.5　CYBERTHREAT REAL-TIME MAP による攻撃の可視化

図 3.5 を見ると，様々な国同士で攻撃が行われていることが分かる．色ごとに「ファイルを開く/実行/保存/コピーしたときに検知されたマルウェア」,「Web ページを開いた際に検知されたマルウェア」,「IDS で検知されたマルウェア」などが可視化されており，見るだけで攻撃の内容が分かる可視化ツールであると言える．

上述のトラフィック可視化システムから，通信や攻撃の可視化は通信状況，被害状況を把握する上で有用な手法であると言える．

第4章　パケット可視化システムの開発と評価

本研究では次の 3 つの条件を満たすパケット可視化システムを開発した.

- リアルタイムに流れるパケットを取得，分析できる.
- ブラックリストを参照し，異常を検知できる.
- 可視化システムの画面上で通信 (被害) 状況がわかる.

4.1　実装環境

本研究のパケット可視化システムの開発に使用した環境について表 4.1 に示す.

表 4.1　開発環境

ホスト OS	Windows10 Pro
メモリ	16GB
CPU	Intel(R) Core(TM)i7-7700
仮想環境	Hyper-V
ゲスト OS	Ubuntu20.04 LTS
Web ブラウザ	Google Chrome version 77.0.3865.120

4.2　システム詳細

本研究で開発したシステムは，次の 3 つの機能により構成される.

- パケット情報取得
- 異常パケット判定
- 異常パケット可視化

本システムでは，パケット情報取得機能，異常パケット判定機能を Maltrail のプログラムを利用して実装し，図 4.1 に示す流れで処理を行う．

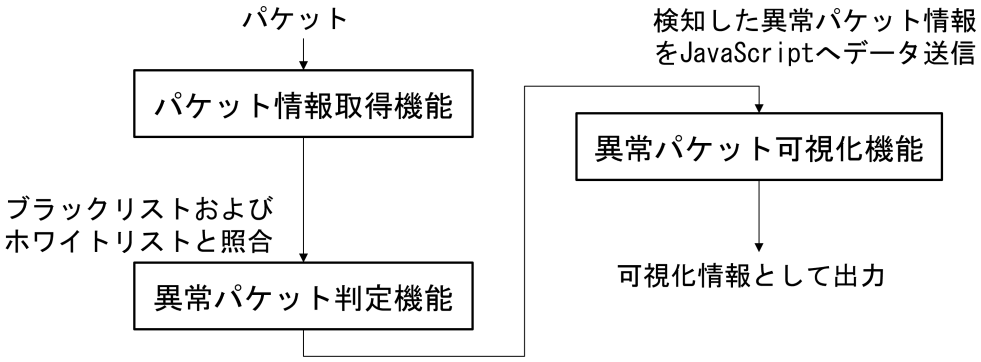


図 4.1　処理の流れ

パケット取得については 4.2.1 節で，異常パケット判定については 4.2.2 節で，パケット可視化については 4.2.3 節でそれぞれ詳細を述べる．

4.2.1　パケット情報取得

送受信されるパケットから情報を取得する．求めるパケット情報は取得日時，送信元・宛先 IP アドレス，プロトコル，送信元ポート番号 (TCP/UDP のみ)，パケット長とする．なお，本研究では検知対象とするパケットは，IPv4 パケットとする．

パケットを取得する際にはパケット解析 API である libpcap [4] を利用した．libpcap にはパケットキャプチャのための関数があり，それらを実装システムに対して用いる．

まず，pcap_findalldevs という関数で，コンピュータ上のデバイスを列挙し，配列に格納する．取得した配列からデバイスを 1 つ選択し，デバイスを開くための関数である pcap_open_live 関数に渡す．この関数では，プロミスキャスモード (パケットを無差別に取得する状態) にするかどうかを選択することができる．開発システムではすべてのパケットを取得するため，プロミスキャスモードで動作させる．デバイスを開いた後，pcap.next 関数でパケットを取得する．この pcap.next 関数を while 文によってループさせることでパケットを取得し続けるようにする．

4.2.2　異常パケット判定

本システムでは，3.2 節で述べた Maltrail が取得するブラックリストを用いた異常パケット判定を行う．

Maltrail が取得するブラックリスト

Maltrail は Web ページで公開されている様々なブラックリストを取得している．取得するブラックリストを以下の図 4.2 に示す．

360bigviktor,360conficker,360cryptolocker,360locky,360necurs,360suppobox,360tofsee,
360virut, abuseipdb,alienvault,atmos, bitcoinnodes,blocklist, botscout, bruteforceblocker, ciarmy,
cruzit, cybercrimetracker,dataplane, dshieldip,emergingthreatsbot, emergingthreatscip,
emergingthreatsdns, fareit,feodotrackerip,gpfcomics,greensnow,ipnoise,kriskinteldns,kriskintelip,
malc0de,malwaredomainkistdns, malwaredomains,maxmind,minerchk,myip,openphish,palevotracker,
proxylists,proxyspy,ransomwaretrackerdns, ransomwaretrackerip,ransomwaretrackerurl,rutgers,sblam,
scriptzteam,socksproxy,sslbl,sslproxies,statics, talosintelligence,torproject,trickbot,urlhaus,viriback,
vxvault,zeustrackermonitor,zeustrackerurl.

図 4.2　Maltrail が取得するブラックリスト

図 4.2 を見ると，様々なサイトからブラックリストを取得していることが分かる．このブラックリストの中から一例を紹介する．

MYIP.MS

MYIP.MS [11] は自身の IP アドレスや，様々な Web サイト，企業などの IP アドレスを調べることができる IP データベースサイトである．このサイトでは，独自のマルウェア，ボット検知システムを用いて異常な IP アドレスを判別する，またはサイトのコミュニティのユーザが報告することで，ブラックリストを作成している．

MYIP.MS の Web ページを図 4.3 に示す．

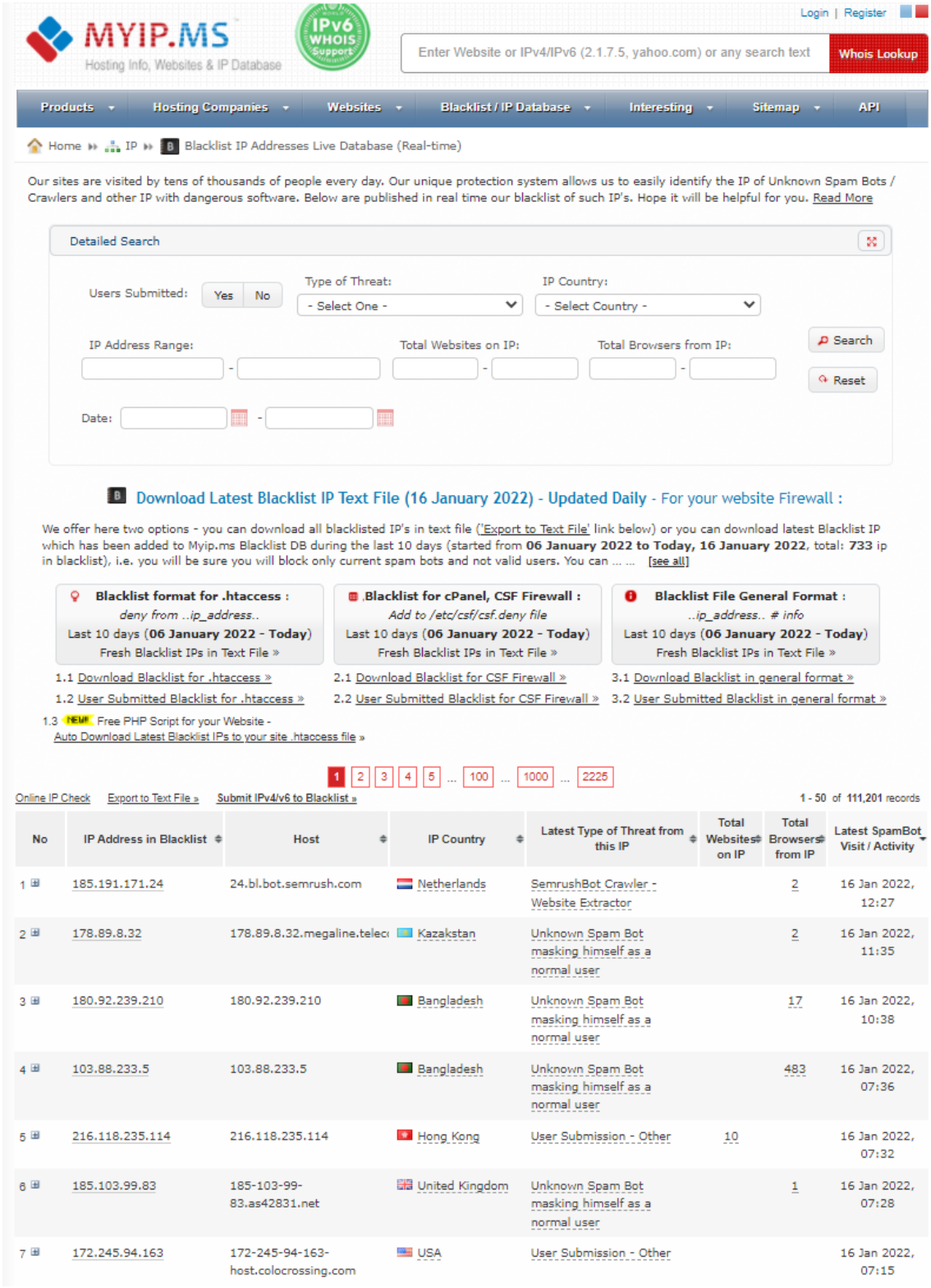


図 4.3　MYIP.MS

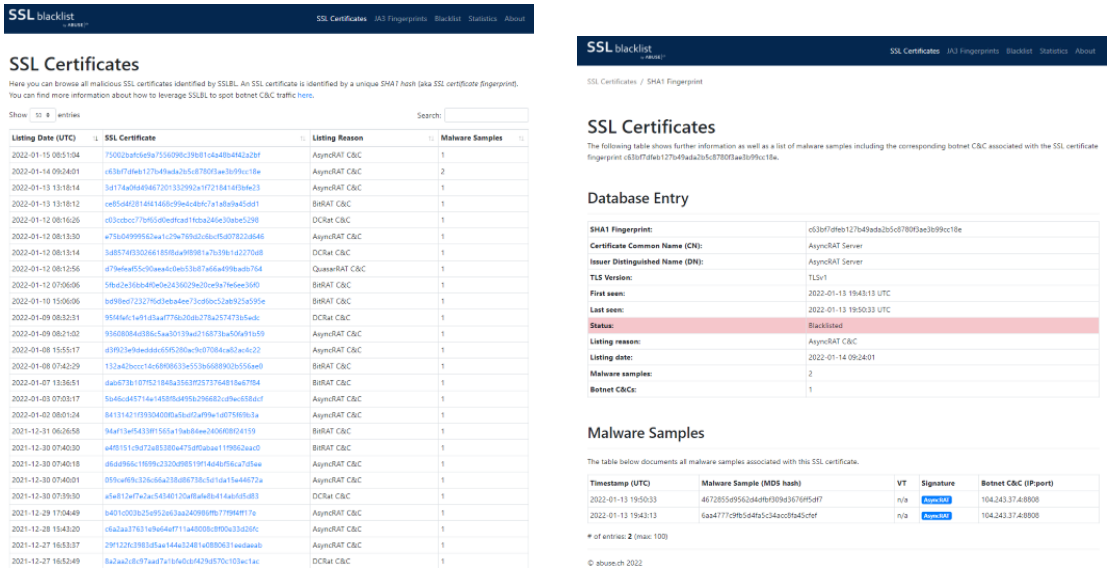
図 4.3 の下半分を見ると，リアルタイムで活動している異常な IP アドレスがまとめられ

ており，国籍やどのような種類のマルウェアなのかなどの詳細情報がわかる．

abuse.ch

abuse.ch [12] はベルン応用科学大学の研究プロジェクトで，主に異常な SSL 証明書，JA3 フィンガープリントの原因となるマルウェア，ボットの特定と追跡を行っている．このプロジェクトのデータはすでに多くの商用，およびオープンソースのセキュリティ製品に統合されており，セキュリティソフトウェアのベンダーだけでなく政府機関，インターネットプロバイダなども利用している．

abuse.ch で提供されているブラックリストを以下の図 4.4 に示す．



(a) 異常な SSL 証明書のブラックリスト　(b) ブラックリストの上から 1 行目の詳細情報

図 4.4　abuse.ch のブラックリスト

abuse.ch では図 4.4(a) のようにブラックリストがまとめられている．図 4.4(b) は図 4.4(a) のリンクをクリックした後のページを示す．IP アドレスやポート番号，どのような種類のマルウェアなのかを見ることができる．

Maltrail は上述のようなサイト以外にも様々なサイトから同様の情報を取得し，CSV ファイルへ格納して独自のブラックリストを作成している．

本システムの異常パケット判定法では，パケットを取得したとき，ドメイン名，IP アドレスをブラックリストおよびホワイトリストと照合し，ブラックリスト内に一致するものがあり，ホワイトリスト内に一致するものがなければ異常とする．それ以外の場合はすべて正

常な通信とみなす.

Maltrail にはブラックリストに書き込まれていない未知の異常通信についても検知する機能があるが，本研究では実際に未知の異常通信を検知，可視化することが困難なため利用しないこととする．異常通信を検知した場合は，パケット情報を Web ページ上に表示し，可視化する.

4.2.3　異常パケット可視化

本研究で開発したシステムでは異常パケット情報を Web ブラウザで読み込み，可視化処理を行う．本節では，可視化に用いる技術と可視化方法について述べる.

canvas 要素

Web 上での動画表現には HTML5 の，canvas 要素を用いた表現を用いることができる．canvas 要素は HTML の要素の一つであり，最新版のウェブブラウザでは標準でサポートしている．canvas 要素の編集は JavaScript によって行う．また，HTML の他の要素と連携した処理を実装したり，PHP 等の他のウェブ系のスクリプト言語と組み合わせることもでき，幅広い表現を実現することができる.

WebGL

トラフィックの可視化には 3D グラフィクス API である WebGL を用いる．WebGL はウェブブラウザ上で 3DCG を表示させるための標準仕様であり，現在の主要ブラウザの最新版なら利用することができる．別の 3DCG API である OpenGL を基本に開発されており，ブラウザ上では一般的に利用されている.

WebGL は HTML5 の canvas 要素と JavaScript を用いて表示させるため，Web 系の他のスクリプト言語とデータの交換が容易である．そのため，ローカルのファイルやデータベースとの連携も容易に行うことができる.

本システムでは 4.2.1 節で取得したパケット情報に対し，ブラックリストおよびホワイトリストを用いて異常判定を行う．その後，異常パケットを検知した場合にプロセス間通信を用いてパケット情報を JSON 形式で受信し，受信したパケット情報を WebGL を用いて可視化する.

可視化グラフィック

可視化した情報の画面について説明する (図 4.5，図 4.6)．canvas 要素を用いた表示領域上に，座標 (緯度・経度) を設定した半透明の地球儀オブジェクトを表示する．地球儀オブジェクトの中心には自分の PC に見立てたオブジェクトを表示する．パケットの IP アドレスから座標を割り出し，自分の PC と通信相手の場所を直線で結び，その上をパケットに見立てた球体オブジェクトを流すことでトラフィックを表現する．直線，球体オブジェクトともに赤色で表示する．

座標の特定には GeoIP を用いる．GeoIP は MaxMind 社が提供する，IP アドレスから地理情報を得る仕組みである．提案システムでは，パケットキャプチャの際，GeoIP 関数にパケット情報の送信元 IP アドレスおよび宛先 IP アドレスを与えることで地理情報 (緯度・経度，国コード) を取得する．可視化時に，得られた地理情報を地球儀の座標と対応させる．

4.3　実行結果

開発システムを用いてリアルタイムで異常パケットを可視化する．ping コマンドを用いてブラックリストに書き込まれている，かつホワイトリストに書き込まれていない IP アドレス”136.161.101.53”と通信を行った結果を図 4.5 に示す．

図 4.5　開発システムの実行結果

図 4.5 は中央の四角のオブジェクトが自身の PC を表したもので，IP アドレス”136.161.101.53”(ア

メリカ) と通信を行った状況を示す．図 4.5 は検知された異常通信が赤色の直線で可視化され，その直線上に赤い球体のオブジェクトが表示される．

また，左上のタブは図 4.6 のような文字ベースのパケット情報を表示するために用いる．

図 4.6　文字ベースのパケット情報のログ

図 4.6 では，過去に行った異常通信の日時，国籍，送信元 IP アドレス，宛先 IP アドレス，プロトコル，パケット長をログとして表示している．

4.4　システム評価

今回，愛媛大学工学部情報工学科ソフトウェアシステム研究室の学部生 5 名に開発したシステムを利用してもらい，自由記述でコメントを依頼し，評価を行った．

評価点

- 地球のどこと通信しているかどうかが見えて分かりやすい．
- 文字情報としてパケットの詳細情報も確認できる点が良い．

改善点

- 文字ベースでの情報表示の部分で，自身の PC の IP アドレスがどれなのか分かりにくい．

改善点の指摘にあるように文字情報の表示方法については課題が残る．

第5章　結論

本研究では個人 PC 向けのブラックリストを用いた異常パケット可視化システムの開発を行った．本研究で開発したシステムはリアルタイムでパケットを取得し，ブラックリストおよびホワイトリストを用いて異常通信を判定し，可視化を行う．可視化情報を有していることで，従来の文字情報で通信状況を分析するシステムと比較して画面を見るだけで通信状況を理解しやすい．また，本システムでは Maltrail を利用して Web 上で公開されている様々なブラックリストを取得し，ホワイトリストと併用して異常通信の検知に利用している。ブラックリストおよびホワイトリストどちらとも異常とみなした場合のみ異常通信と判定するため，田村らの研究で開発されているシステムと比較して誤検知は少ない．

しかし，異常パケットが多数観測された場合はリアルタイムにパケットの表示ができなくなることがある。パケットを多数可視化するための実装は今後の課題である。また、システム評価で得られた意見をもとにして文字情報の表示方法の改良も今後の課題である

謝辞

本研究を進めるにあたり，常日頃より丁寧かつ熱心なご指導を頂きました，高橋寛教授，甲斐博准教授，王森玲講師に深く感謝いたします.

また，本研究に際してご査読いただいた遠藤 慶一准教授，宇戸 寿幸准教授に感謝の意を表します.

最後に，本研究において多大なご協力を頂きました諸先輩方，ならびに同研究室の同期生に厚くお礼申し上げます.

参考文献

［1］ McafeeLabs, “脅 威 レ ポ ー ト:2021 年 6 月”, Mcafee.com, 2021, https://www.mcafee.com/enterprise/ja-jp/assets/reports/rp-threats-jun-2021.pdf, (2022.1.31 参照)

［2］ 中尾康二, 松本文子, 井上大介, 馬場俊輔, 鈴木和也, 衛藤将史, 吉岡克成, 力武健次, 堀良彰: インシデント分析センタ NICTER の可視化技術, 電子情報通信学会技術研究報告. ISEC, 情報セキュリティ, Vol.106, Number 176, pp.83-89 (2006).

［3］ Wireshark. https://www.wireshark.org/, (2022.1.31 参照)

［4］ TCPDUMP & LIBPCAP. https://www.tcpdump.org/, (2022. 1.31 参照)

［5］ 田村 尚規: パケット分析に基づく個人の PC のトラフィック可視化に関する研究. 愛媛大学理工学研究科電子情報工学専攻ソフトウェアシステム研究室, (2017)

［6］ Sandro Etalle, Clifford Gregory, Damiano Bolzoni, Emmanuele Zambon: Self-configuring deep protocol network whitelisting, Security Matters Whitepapers, pp.1-24 (2014).

［7］ Suricata. https://suricata.io/, (2022.1.31 参照)

［8］ Maltrail. https://github.com/stamparm/MalTrail, (2022.1.31 参照)

［9］ Atlas. https://www.nicter.jp/atlas, (2022.1.31 参照)

［10］ CIBERTHREAT REAL-TIME MAP, https://cybermap.kaspersky.com/ja, (2022.1.31 参照)

［11］ My IP Address - Shows IPv4 & IPv6 / Blacklist IP Check - Hosting Info, https://myip.ms/, (2022.1.31 参照)

［12］ abuse.ch. https://abuse.ch/, (2022.1.31 参照)

付 録A　プログラム

app.js

```
// Express モジュールを読み込む
var express = require('express'),
 http = require('http'),
 app = express();
// index.html を表示する
app.use(express.static(__dirname + '/public'));
// 10000ポートで接続を待つ
var server = http.createServer(app).listen(10000, function() {
 console.log('接続待機中...');
});
// Socket.IO モジュールを読み込む
var io = require('socket.io'),
 io = io(server);

// 接続時の処理
io.sockets.on('connection', function(socket) {
 console.log('接続開始');
 //var i = 0;

 var spawn = require('child_process').spawn,
  python = spawn('sudo', ['-E', 'python3', 'sensor2.py']);

 python.stdout.on('data', function(data) {
  //console.log(data.toString());
  console.log('python␣stdout:␣' + data.toString());
  socket.broadcast.emit('packet', data.toString());
  //i++;
 });
 python.stderr.on('data', function(data) {
  console.log('python␣stderr:␣' + data);
 });


 socket.on('capture', function() {
  console.log('yahoo');
  //startCapture(socket);
 });
 socket.on('disconnect', function() {
```

```
  console.log('接続切断');
  //python.kill();
 });

});



function startCapture(socket) {
}
```

sensor.py

```
#!/usr/bin/env python

"""
Copyright (c) 2014-2021 Maltrail developers (https://github.com/stamparm/maltrail/)
See the file 'LICENSE' for copying permission
"""

from __future__ import print_function  # Requires: Python >= 2.6

import sys

sys.dont_write_bytecode = True

import cProfile
import inspect
import math
import mmap
import optparse
import os
import platform
import re
import socket
import subprocess
import struct
import threading
import time
import traceback
import warnings
import json
import netifaces
import geoip2.database
import datetime
import time
import mysql.connector

reader = geoip2.database.Reader('./GeoIP/GeoLite2-City.mmdb')
conn = mysql.connector.connect(host="localhost", db="white_list", user="yasuhara",
```

```
    passwd="root", charset="utf8", auth_plugin="mysql_native_password")
cursor = conn.cursor(buffered=True)

from core.addr import inet_ntoa6
from core.addr import addr_port
from core.attribdict import AttribDict
from core.common import check_connection
from core.common import check_sudo
from core.common import check_whitelisted
from core.common import get_ex_message
from core.common import get_text
from core.common import is_local
from core.common import load_trails
from core.common import patch_parser
from core.compat import xrange
from core.datatype import LRUDict
from core.enums import BLOCK_MARKER
from core.enums import CACHE_TYPE
from core.enums import PROTO
from core.enums import TRAIL
from core.log import create_log_directory
from core.log import flush_condensed_events
from core.log import get_error_log_handle
from core.log import log_error
from core.log import log_event
from core.parallel import worker
from core.parallel import write_block
from core.settings import config
from core.settings import CAPTURE_TIMEOUT
from core.settings import CHECK_CONNECTION_MAX_RETRIES
from core.settings import CONFIG_FILE
from core.settings import CONSONANTS
from core.settings import DLT_OFFSETS
from core.settings import DNS_EXHAUSTION_THRESHOLD
from core.settings import GENERIC_SINKHOLE_REGEX
from core.settings import HOMEPAGE
from core.settings import HOURLY_SECS
from core.settings import HTTP_TIME_FORMAT
from core.settings import IGNORE_DNS_QUERY_SUFFIXES
from core.settings import IPPROTO_LUT
from core.settings import IS_WIN
from core.settings import LOCALHOST_IP
from core.settings import LOCAL_SUBDOMAIN_LOOKUPS
from core.settings import MAX_CACHE_ENTRIES
from core.settings import MMAP_ZFILL_CHUNK_LENGTH
from core.settings import NAME
from core.settings import NO_SUCH_NAME_COUNTERS
from core.settings import NO_SUCH_NAME_PER_HOUR_THRESHOLD
from core.settings import INFECTION_SCANNING_THRESHOLD
from core.settings import PORT_SCANNING_THRESHOLD
from core.settings import POTENTIAL_INFECTION_PORTS
```

```
from core.settings import read_config
from core.settings import REGULAR_SENSOR_SLEEP_TIME
from core.settings import SNAP_LEN
from core.settings import SUSPICIOUS_CONTENT_TYPES
from core.settings import SUSPICIOUS_DIRECT_DOWNLOAD_EXTENSIONS
from core.settings import SUSPICIOUS_DIRECT_IP_URL_REGEX
from core.settings import SUSPICIOUS_DOMAIN_CONSONANT_THRESHOLD
from core.settings import SUSPICIOUS_DOMAIN_ENTROPY_THRESHOLD
from core.settings import SUSPICIOUS_DOMAIN_LENGTH_THRESHOLD
from core.settings import SUSPICIOUS_HTTP_PATH_REGEXES
from core.settings import SUSPICIOUS_HTTP_REQUEST_PRE_CONDITION
from core.settings import SUSPICIOUS_HTTP_REQUEST_REGEXES
from core.settings import SUSPICIOUS_HTTP_REQUEST_FORCE_ENCODE_CHARS
from core.settings import SUSPICIOUS_PROXY_PROBE_PRE_CONDITION
from core.settings import SUSPICIOUS_UA_REGEX
from core.settings import VALID_DNS_NAME_REGEX
from core.settings import trails
from core.settings import VERSION
from core.settings import WEB_SCANNING_THRESHOLD
from core.settings import WHITELIST
from core.settings import WHITELIST_DIRECT_DOWNLOAD_KEYWORDS
from core.settings import WHITELIST_LONG_DOMAIN_NAME_KEYWORDS
from core.settings import WHITELIST_HTTP_REQUEST_PATHS
from core.settings import WHITELIST_UA_REGEX
from core.update import update_ipcat
from core.update import update_trails
from thirdparty import six
from thirdparty.six.moves import urllib as _urllib

warnings.filterwarnings(action="ignore", category=DeprecationWarning)       # NOTE:
    https://github.com/helpsystems/pcapy/pull/67/files

_buffer = None
_caps = []
_connect_sec = 0
_connect_src_dst = {}
_connect_src_details = {}
_path_src_dst = {}
_path_src_dst_details = {}
_count = 0
_locks = AttribDict()
_multiprocessing = None
_n = None
_result_cache = LRUDict(MAX_CACHE_ENTRIES)
_local_cache = LRUDict(MAX_CACHE_ENTRIES)
_last_syn = None
_last_logged_syn = None
_last_udp = None
_last_logged_udp = None
_done_count = 0
_done_lock = threading.Lock()
```

```
_subdomains = {}
_subdomains_sec = None
_dns_exhausted_domains = set()

class _set(set):
    pass

try:
    import pcapy
except ImportError:
    if IS_WIN:
        exit("[!] please install 'WinPcap' (e.g. 'http://www.winpcap.org/install/') and
            Pcapy (e.g. 'https://breakingcode.wordpress.com/?s=pcapy')")
    else:
        msg = "[!] please install 'Pcapy' (e.g. 'sudo pip%s install pcapy-ng')" % ('3'
            if six.PY3 else '2')

        exit(msg)

def _check_domain_member(query, domains):
    parts = query.lower().split('.')

    for i in xrange(0, len(parts)):
        domain = '.'.join(parts[i:])
        if domain in domains:
            return True

    return False

def _check_domain_whitelisted(query):
    result = _result_cache.get((CACHE_TYPE.DOMAIN_WHITELISTED, query))

    if result is None:
        result = _check_domain_member(re.split(r"(?i)[^A-Z0-9._-]", query or "")[0],
            WHITELIST)
        _result_cache[(CACHE_TYPE.DOMAIN_WHITELISTED, query)] = result

    return result

def _check_domain(query, sec, usec, src_ip, src_port, dst_ip, dst_port, proto, packet=
    None):
    if query:
        query = query.lower()
        if ':' in query:
            query = query.split(':', 1)[0]

    if query.replace('.', "").isdigit():  # IP address
        return

    if _result_cache.get((CACHE_TYPE.DOMAIN, query)) is False:
        return
```

```
result = False
if re.search(VALID_DNS_NAME_REGEX, query) is not None and not
    _check_domain_whitelisted(query):
    parts = query.split('.')

    if query.endswith(".ip-adress.com"):  # Reference: https://www.virustotal.com/
        gui/domain/ip-adress.com/relations
        _ = '.'.join(parts[:-2])
        trail = "%s(.ip-adress.com)" % _
        if _ in trails:
            result = True
            log_event((sec, usec, src_ip, src_port, dst_ip, dst_port, proto, TRAIL.
                DNS, trail, trails[_][0], trails[_][1]), packet)
            out(src_ip,dst_ip,proto,packet)
    if not result:
        for i in xrange(0, len(parts)):
            domain = '.'.join(parts[i:])
            if domain in trails:
                if domain == query:
                    trail = domain
                else:
                    _ = ".%s" % domain
                    trail = "(%s)%s" % (query[:-len(_)], _)

                if not (re.search(r"(?i)\A([rd]?ns|nf|mx|nic)\d*\.", query) and any(
                    _ in trails.get(domain, "␣")[0] for _ in ("suspicious", "
                    sinkhole"))):  # e.g. ns2.nobel.su
                    if not ((query == trail or parts[0] == "www") and any(_ in
                        trails.get(domain, "␣")[0] for _ in ("dynamic", "free␣web"
                        ))):  # e.g. noip.com
                        result = True
                        log_event((sec, usec, src_ip, src_port, dst_ip, dst_port,
                            proto, TRAIL.DNS, trail, trails[domain][0], trails[
                            domain][1]), packet)
                        out(src_ip,dst_ip,proto,packet)
                        break

    if not result and config.USE_HEURISTICS:
        if len(parts[0]) > SUSPICIOUS_DOMAIN_LENGTH_THRESHOLD and '-' not in parts
            [0]:
            trail = None

            if len(parts) > 2:
                trail = "(%s).%s" % ('.'.join(parts[:-2]), '.'.join(parts[-2:]))
            elif len(parts) == 2:
                trail = "(%s).%s" % (parts[0], parts[1])
            else:
                trail = query

            if trail and not any(_ in trail for _ in
```

```
                    WHITELIST_LONG_DOMAIN_NAME_KEYWORDS):
                    result = True
                    log_event((sec, usec, src_ip, src_port, dst_ip, dst_port, proto,
                        TRAIL.DNS, trail, "long domain (suspicious)", "(heuristic)"),
                        packet)
                    out(src_ip,dst_ip,proto,packet)
        if not result and trails._regex:
            match = re.search(trails._regex, query)
            if match:
                group, trail = [_ for _ in match.groupdict().items() if _[1] is not None
                    ][0]
                candidate = trails._regex.split("(?P<")[int(group[1:]) + 1]
                candidate = candidate.split('>', 1)[-1].rstrip('|')[:-1]
                if candidate in trails:
                    result = True
                    trail = match.group(0)

                    prefix, suffix = query[:match.start()], query[match.end():]
                    if prefix:
                        trail = "(%s)%s" % (prefix, trail)
                    if suffix:
                        trail = "%s(%s)" % (trail, suffix)

                    trail = trail.replace(".)", ").")

                    log_event((sec, usec, src_ip, src_port, dst_ip, dst_port, proto,
                        TRAIL.DNS, trail, trails[candidate][0], trails[candidate][1]),
                        packet)
                    out(src_ip,dst_ip,proto,packet)
        if not result and ".onion." in query:
            trail = re.sub(r"(\.onion)(\..*)", r"\1(\2)", query)
            _ = trail.split('(')[0]
            if _ in trails:
                result = True
                log_event((sec, usec, src_ip, src_port, dst_ip, dst_port, proto, TRAIL.
                    DNS, trail, trails[_][0], trails[_][1]), packet)
                out(src_ip,dst_ip,proto,packet)
    if result is False:
        _result_cache[(CACHE_TYPE.DOMAIN, query)] = False

def _get_local_prefix():
    _sources = set(_.split('~')[0] for _ in _connect_src_dst.keys())
    _candidates = [re.sub(r"\d+\.\d+\Z", "", _) for _ in _sources]
    _ = sorted(((_candidates.count(_), _) for _ in set(_candidates)), reverse=True)
    result = _[0][1] if _ else ""

    if result:
        _result_cache[(CACHE_TYPE.LOCAL_PREFIX, "")] = result
    else:
        result = _result_cache.get((CACHE_TYPE.LOCAL_PREFIX, ""))
```

```
    return result or '_'

def _process_packet(packet, sec, usec, ip_offset):
    """
    Processes single (raw) IP layer data
    """

    #print('packet_come')

    global _connect_sec
    global _last_syn
    global _last_logged_syn
    global _last_udp
    global _last_logged_udp
    global _subdomains_sec

    try:
        if config.USE_HEURISTICS:
            if _locks.connect_sec:
                _locks.connect_sec.acquire()

            connect_sec = _connect_sec
            _connect_sec = sec

            if _locks.connect_sec:
                _locks.connect_sec.release()

            if sec > connect_sec:
                for key in _connect_src_dst:
                    _src_ip, _dst = key.split('~')
                    if not _dst.isdigit() and len(_connect_src_dst[key]) >
                        PORT_SCANNING_THRESHOLD:
                        if not check_whitelisted(_src_ip):
                            _dst_ip = _dst
                            for _ in _connect_src_details[key]:
                                log_event((sec, usec, _src_ip, _[2], _dst_ip, _[3],
                                    PROTO.TCP, TRAIL.IP, _src_ip, "potential port
                                    scanning", "(heuristic)"), packet)
                                out(src_ip,dst_ip,PROTO.TCP,packet)
                    elif len(_connect_src_dst[key]) > INFECTION_SCANNING_THRESHOLD:
                        _dst_port = _dst
                        _dst_ip = [_[-1] for _ in _connect_src_details[key]]
                        _src_port = [_[-2] for _ in _connect_src_details[key]]

                        if len(_dst_ip) == len(set(_dst_ip)):
                            if _src_ip.startswith(_get_local_prefix()):
                                log_event((sec, usec, _src_ip, _src_port[0], _dst_ip[0],
                                     _dst_port, PROTO.TCP, TRAIL.PORT, _dst_port, "
                                    potential infection", "(heuristic)"), packet)
                                out(src_ip,dst_ip,PROTO.TCP,packet)
                _connect_src_dst.clear()
```

```
            _connect_src_details.clear()

            for key in _path_src_dst:
                if len(_path_src_dst[key]) > WEB_SCANNING_THRESHOLD:
                    _src_ip, _dst_ip = key.split('~')
                    _sec, _usec, _src_port, _dst_port, _path = _path_src_dst_details
                        [key].pop()
                    log_event((_sec, _usec, _src_ip, _src_port, _dst_ip, _dst_port,
                        PROTO.TCP, TRAIL.PATH, "*", "potential web scanning", "(
                        heuristic)"), packet)
                    out(src_ip,dst_ip,PROTO.TCP,packet)
            _path_src_dst.clear()
            _path_src_dst_details.clear()

    ip_data = packet[ip_offset:]
    ip_version = ord(ip_data[0:1]) >> 4
    localhost_ip = LOCALHOST_IP[ip_version]

    if ip_version == 0x04:  # IPv4
        ip_header = struct.unpack("!BBHHHBBH4s4s", ip_data[:20])
        fragment_offset = ip_header[4] & 0x1fff
        if fragment_offset != 0:
            return
        iph_length = (ip_header[0] & 0xf) << 2
        protocol = ip_header[6]
        src_ip = socket.inet_ntoa(ip_header[8])
        dst_ip = socket.inet_ntoa(ip_header[9])
        ############
        #out(src_ip,dst_ip,_src_port,_dst_port,protocol,ip_header)
        #time.sleep(1)
        ############
    elif ip_version == 0x06:  # IPv6
        # Reference: http://chrisgrundemann.com/index.php/2012/introducing-ipv6-
            understanding-ipv6-addresses/
        ip_header = struct.unpack("!BBHHBB16s16s", ip_data[:40])
        iph_length = 40
        protocol = ip_header[4]
        src_ip = inet_ntoa6(ip_header[6])
        dst_ip = inet_ntoa6(ip_header[7])
    else:
        return

    if protocol == socket.IPPROTO_TCP:  # TCP
        src_port, dst_port, _, _, doff_reserved, flags = struct.unpack("!HHLLBB",
            ip_data[iph_length:iph_length + 14])

        if flags != 2 and config.plugin_functions:
            if dst_ip in trails:
                log_event((sec, usec, src_ip, src_port, dst_ip, dst_port, PROTO.TCP,
                     TRAIL.IP, dst_ip, trails[dst_ip][0], trails[dst_ip][1]), packet
                    , skip_write=True)
```

```
                out(src_ip,dst_ip,PROTO.TCP,packet)
            elif src_ip in trails and dst_ip != localhost_ip:
                log_event((sec, usec, src_ip, src_port, dst_ip, dst_port, PROTO.TCP,
                     TRAIL.IP, src_ip, trails[src_ip][0], trails[src_ip][1]), packet
                    , skip_write=True)
                out(src_ip,dst_ip,PROTO.TCP,packet)
        if flags == 2:  # SYN set (only)
            _ = _last_syn
            _last_syn = (sec, src_ip, src_port, dst_ip, dst_port)
            if _ == _last_syn:  # skip bursts
                return

            if dst_ip in trails or addr_port(dst_ip, dst_port) in trails:
                _ = _last_logged_syn
                _last_logged_syn = _last_syn
                if _ != _last_logged_syn:
                    trail = addr_port(dst_ip, dst_port)
                    if trail not in trails:
                        trail = dst_ip
                    if not any(_ in trails[trail][0] for _ in ("attacker",)) and not
                         ("parking site" in trails[trail][0] and dst_port not in
                        (80, 443)):
                        log_event((sec, usec, src_ip, src_port, dst_ip, dst_port,
                            PROTO.TCP, TRAIL.IP if ':' not in trail else TRAIL.IPORT
                            , trail, trails[trail][0], trails[trail][1]), packet)
                        out(src_ip,dst_ip,PROTO.TCP,packet)
            elif (src_ip in trails or addr_port(src_ip, src_port) in trails) and
                dst_ip != localhost_ip:
                _ = _last_logged_syn
                _last_logged_syn = _last_syn
                if _ != _last_logged_syn:
                    trail = addr_port(src_ip, src_port)
                    if trail not in trails:
                        trail = src_ip
                    if not any(_ in trails[trail][0] for _ in ("malware",)):
                        log_event((sec, usec, src_ip, src_port, dst_ip, dst_port,
                            PROTO.TCP, TRAIL.IP if ':' not in trail else TRAIL.IPORT
                            , trail, trails[trail][0], trails[trail][1]), packet)
                        out(src_ip,dst_ip,PROTO.TCP,packet)
            if config.USE_HEURISTICS:
                if dst_ip != localhost_ip:
                    key = "%s~%s" % (src_ip, dst_ip)
                    if key not in _connect_src_dst:
                        _connect_src_dst[key] = set()
                        _connect_src_details[key] = set()
                    _connect_src_dst[key].add(dst_port)
                    _connect_src_details[key].add((sec, usec, src_port, dst_port))

                    if dst_port in POTENTIAL_INFECTION_PORTS:
                        key = "%s~%s" % (src_ip, dst_port)
                        if key not in _connect_src_dst:
```

```
                    _connect_src_dst[key] = set()
                    _connect_src_details[key] = set()
                _connect_src_dst[key].add(dst_ip)
                _connect_src_details[key].add((sec, usec, src_port, dst_ip))
    else:
        tcph_length = doff_reserved >> 4
        h_size = iph_length + (tcph_length << 2)
        tcp_data = get_text(ip_data[h_size:])

        if tcp_data.startswith("HTTP/"):
            match = re.search(GENERIC_SINKHOLE_REGEX, tcp_data[:2000])
            if match:
                trail = match.group(0)
                log_event((sec, usec, src_ip, src_port, dst_ip, dst_port, PROTO.
                    TCP, TRAIL.HTTP, trail, "sinkhole␣response␣(malware)", "(
                    heuristic)"), packet)
                out(src_ip,dst_ip,PROTO.TCP,packet)
            else:
                index = tcp_data.find("<title>")
                if index >= 0:
                    title = tcp_data[index + len("<title>"):tcp_data.find("</
                        title>", index)]
                    if re.search(r"domain␣name␣has␣been␣seized␣by|Domain␣Seized|
                        Domain␣Seizure", title):
                        log_event((sec, usec, src_ip, src_port, dst_ip, dst_port
                            , PROTO.TCP, TRAIL.HTTP, title, "seized␣domain␣(
                            suspicious)", "(heuristic)"), packet)
                        out(src_ip,dst_ip,PROTO.TCP,packet)
            content_type = None
            first_index = tcp_data.find("\r\nContent-Type:")
            if first_index >= 0:
                first_index = first_index + len("\r\nContent-Type:")
                last_index = tcp_data.find("\r\n", first_index)
                if last_index >= 0:
                    content_type = tcp_data[first_index:last_index].strip().
                        lower()

            if content_type and content_type in SUSPICIOUS_CONTENT_TYPES:
                log_event((sec, usec, src_ip, src_port, dst_ip, dst_port, PROTO.
                    TCP, TRAIL.HTTP, content_type, "content␣type␣(suspicious)",
                    "(heuristic)"), packet)
                out(src_ip,dst_ip,PROTO.TCP,packet)
        method, path = None, None

        if "␣HTTP/" in tcp_data:
            index = tcp_data.find("\r\n")
            if index >= 0:
                line = tcp_data[:index]
                if line.count('␣') == 2 and "␣HTTP/" in line:
                    method, path, _ = line.split('␣')
```

```
if method and path:
    post_data = None
    host = dst_ip
    first_index = tcp_data.find("\r\nHost:")
    path = path.lower()

    if first_index >= 0:
        first_index = first_index + len("\r\nHost:")
        last_index = tcp_data.find("\r\n", first_index)
        if last_index >= 0:
            host = tcp_data[first_index:last_index]
            host = host.strip().lower()
            if host.endswith(":80"):
                host = host[:-3]
            if host and host[0].isalpha() and dst_ip in trails:
                log_event((sec, usec, src_ip, src_port, dst_ip, dst_port
                    , PROTO.TCP, TRAIL.IP, "%s␣(%s)" % (dst_ip, host.
                    split(':')[0]), trails[dst_ip][0], trails[dst_ip
                    ][1]), packet)
                out(src_ip,dst_ip,PROTO.TCP,packet)
            elif re.search(r"\A\d+\.[0-9.]+\Z", host or "") and re.
                search(SUSPICIOUS_DIRECT_IP_URL_REGEX, "%s%s" % (host,
                path)):
                if not dst_ip.startswith(_get_local_prefix()):
                    trail = "(%s)%s" % (host, path)
                    log_event((sec, usec, src_ip, src_port, dst_ip,
                        dst_port, PROTO.TCP, TRAIL.HTTP, trail, "
                        potential␣iot-malware␣download␣(suspicious)", "(
                        heuristic)"), packet)
                    out(src_ip,dst_ip,PROTO.TCP,packet)
                    return
            elif config.CHECK_HOST_DOMAINS:
                _check_domain(host, sec, usec, src_ip, src_port, dst_ip,
                     dst_port, PROTO.TCP, packet)
    elif config.USE_HEURISTICS and config.CHECK_MISSING_HOST:
        log_event((sec, usec, src_ip, src_port, dst_ip, dst_port, PROTO.
            TCP, TRAIL.HTTP, "%s%s" % (host, path), "missing␣host␣header
            ␣(suspicious)", "(heuristic)"), packet)
        out(src_ip,dst_ip,PROTO.TCP,packet)
    index = tcp_data.find("\r\n\r\n")
    if index >= 0:
        post_data = tcp_data[index + 4:]

    url = None
    if config.USE_HEURISTICS and path.startswith('/'):
        _path = path.split('/')[1]

        key = "%s^%s" % (src_ip, dst_ip)
        if key not in _path_src_dst:
            _path_src_dst[key] = set()
        _path_src_dst[key].add(_path)
```

```
        if key not in _path_src_dst_details:
            _path_src_dst_details[key] = set()
        _path_src_dst_details[key].add((sec, usec, src_port, dst_port,
            path))

    elif config.USE_HEURISTICS and dst_port == 80 and path.startswith("
        http://") and any(_ in path for _ in
        SUSPICIOUS_PROXY_PROBE_PRE_CONDITION) and not
        _check_domain_whitelisted(path.split('/')[2]):
        trail = re.sub(r"(http://[^/]+/)(.+)", r"\g<1>(\g<2>)", path)
        trail = re.sub(r"(http://)([^/(]+)", lambda match: "%s%s" % (
            match.group(1), match.group(2).split(':')[0].rstrip('.')),
            trail)
        log_event((sec, usec, src_ip, src_port, dst_ip, dst_port, PROTO.
            TCP, TRAIL.HTTP, trail, "potential␣proxy␣probe␣(suspicious)"
            , "(heuristic)"), packet)
        out(src_ip,dst_ip,PROTO.TCP,packet)
        return
    elif "://" in path:
        unquoted_path = _urllib.parse.unquote(path)

        key = "code␣execution"
        if key not in _local_cache:
            _local_cache[key] = next(_[1] for _ in
                SUSPICIOUS_HTTP_REQUEST_REGEXES if "code␣execution" in _
                [0])

        if re.search(_local_cache[key], unquoted_path, re.I) is None:
            # NOTE: to prevent malware domain FPs in case of outside
            scanners
            url = path.split("://", 1)[1]

            if '/' not in url:
                url = "%s/" % url

            host, path = url.split('/', 1)
            if host.endswith(":80"):
                host = host[:-3]
            path = "/%s" % path
            proxy_domain = host.split(':')[0]
            _check_domain(proxy_domain, sec, usec, src_ip, src_port,
                dst_ip, dst_port, PROTO.TCP, packet)
    elif method == "CONNECT":
        if '/' in path:
            host, path = path.split('/', 1)
            path = "/%s" % path
        else:
            host, path = path, '/'
        if host.endswith(":80"):
            host = host[:-3]
```

```
            url = "%s%s" % (host, path)
            proxy_domain = host.split(':')[0]
            _check_domain(proxy_domain, sec, usec, src_ip, src_port, dst_ip,
                 dst_port, PROTO.TCP, packet)

        if url is None:
            url = "%s%s" % (host, path)

        if config.USE_HEURISTICS:
            user_agent, result = None, None

            first_index = tcp_data.find("\r\nUser-Agent:")
            if first_index >= 0:
                first_index = first_index + len("\r\nUser-Agent:")
                last_index = tcp_data.find("\r\n", first_index)
                if last_index >= 0:
                    user_agent = tcp_data[first_index:last_index]
                    user_agent = _urllib.parse.unquote(user_agent).strip()

            if user_agent:
                result = _result_cache.get((CACHE_TYPE.USER_AGENT,
                    user_agent))
                if result is None:
                    if re.search(WHITELIST_UA_REGEX, user_agent, re.I) is
                        None:
                        match = re.search(SUSPICIOUS_UA_REGEX, user_agent)
                        if match:
                            def _(value):
                                return value.rstrip('\\').replace('(', "\\("
                                    ).replace(')', "\\)")

                            parts = user_agent.split(match.group(0), 1)

                            if len(parts) > 1 and parts[0] and parts[-1]:
                                result = _result_cache[(CACHE_TYPE.
                                    USER_AGENT, user_agent)] = "%s␣(%s)" % (
                                    _(match.group(0)), _(user_agent))
                            else:
                                result = _result_cache[(CACHE_TYPE.
                                    USER_AGENT, user_agent)] = _(match.group
                                    (0)).join(("(%s)" if part else "%s") % _
                                    (part) for part in parts)
                        if not result:
                            _result_cache[(CACHE_TYPE.USER_AGENT, user_agent)] =
                                 False

                    if result:
                        log_event((sec, usec, src_ip, src_port, dst_ip, dst_port
                            , PROTO.TCP, TRAIL.UA, result, "user␣agent␣(
                            suspicious)", "(heuristic)"), packet)
                        out(src_ip,dst_ip,PROTO.TCP,packet)
```

```
if not _check_domain_whitelisted(host):
    path = path.replace("//", '/')

    unquoted_path = _urllib.parse.unquote(path)
    unquoted_post_data = _urllib.parse.unquote(post_data or "")

    checks = [path.rstrip('/')]

    if '?' in path:
        checks.append(path.split('?')[0].rstrip('/'))

        if '=' in path:
            checks.append(path[:path.index('=') + 1])

        _ = re.sub(r"(\w+=)[^&=]+", r"\g<1>", path)
        if _ not in checks:
            checks.append(_)
            if _.count('/') > 1:
                checks.append("/%s" % _.split('/')[-1])
    elif post_data:
        checks.append("%s?%s" % (path, unquoted_post_data.lower()))

    if checks[-1].count('/') > 1:
        checks.append(checks[-1][:checks[-1].rfind('/')])
        checks.append(checks[0][checks[0].rfind('/'):].split('?'
            )[0])

    for check in filter(None, checks):
        for _ in ("", host):
            check = "%s%s" % (_, check)
            if check in trails:
                if '?' not in path and '?' in check and post_data:
                    trail = "%s(%s␣\\(%s␣%s\\))" % (host, path,
                        method, post_data.strip())
                    log_event((sec, usec, src_ip, src_port, dst_ip,
                        dst_port, PROTO.TCP, TRAIL.HTTP, trail,
                        trails[check][0], trails[check][1]))
                    out(src_ip,dst_ip,PROTO.TCP,packet)
                else:
                    parts = url.split(check)
                    other = ("(%s)" % _ if _ else _ for _ in parts)
                    trail = check.join(other)
                    log_event((sec, usec, src_ip, src_port, dst_ip,
                        dst_port, PROTO.TCP, TRAIL.URL, trail,
                        trails[check][0], trails[check][1]))
                    out(src_ip,dst_ip,PROTO.TCP,packet)
                return

    if "%s/" % host in trails:
        trail = "%s/" % host
        log_event((sec, usec, src_ip, src_port, dst_ip, dst_port,
```

```
                    PROTO.TCP, TRAIL.URL, trail, trails[trail][0], trails[
                    trail][1]))
                out(src_ip,dst_ip,PROTO.TCP,packet)
                return

            if config.USE_HEURISTICS:
                match = re.search(r"\b(CF-Connecting-IP|True-Client-IP|X-
                    Forwarded-For):\s*([0-9.]+)".encode(), packet, re.I)
                if match:
                    src_ip = "%s,%s" % (src_ip, match.group(1))

                for char in SUSPICIOUS_HTTP_REQUEST_FORCE_ENCODE_CHARS:
                    replacement = SUSPICIOUS_HTTP_REQUEST_FORCE_ENCODE_CHARS
                        [char]
                    path = path.replace(char, replacement)
                    if post_data:
                        post_data = post_data.replace(char, replacement)

                if not any(_ in unquoted_path.lower() for _ in
                    WHITELIST_HTTP_REQUEST_PATHS):
                    if any(_ in unquoted_path for _ in
                        SUSPICIOUS_HTTP_REQUEST_PRE_CONDITION):
                        found = _result_cache.get((CACHE_TYPE.PATH,
                            unquoted_path))
                        if found is None:
                            for desc, regex in
                                SUSPICIOUS_HTTP_REQUEST_REGEXES:
                                if re.search(regex, unquoted_path, re.I | re
                                    .DOTALL):
                                    found = desc
                                    break
                            _result_cache[(CACHE_TYPE.PATH, unquoted_path)]
                                = found or ""
                        if found and not ("data␣leakage" in found and
                            is_local(dst_ip)):
                            trail = "%s(%s)" % (host, path)
                            log_event((sec, usec, src_ip, src_port, dst_ip,
                                dst_port, PROTO.TCP, TRAIL.URL, trail, "%s␣(
                                suspicious)" % found, "(heuristic)"), packet
                                )
                            out(src_ip,dst_ip,PROTO.TCP,packet)
                            return

                    if any(_ in unquoted_post_data for _ in
                        SUSPICIOUS_HTTP_REQUEST_PRE_CONDITION):
                        found = _result_cache.get((CACHE_TYPE.POST_DATA,
                            unquoted_post_data))
                        if found is None:
                            for desc, regex in
                                SUSPICIOUS_HTTP_REQUEST_REGEXES:
                                if re.search(regex, unquoted_post_data, re.I
```

```
                                        | re.DOTALL):
                                    found = desc
                                    break
                            _result_cache[(CACHE_TYPE.POST_DATA,
                                unquoted_post_data)] = found or ""
                        if found:
                            trail = "%s(%s␣\\(%s␣%s\\))" % (host, path,
                                method, post_data.strip())
                            log_event((sec, usec, src_ip, src_port, dst_ip,
                                dst_port, PROTO.TCP, TRAIL.HTTP, trail, "%s␣
                                (suspicious)" % found, "(heuristic)"),
                                packet)
                            out(src_ip,dst_ip,PROTO.TCP,packet)
                            return

                    if '.' in path:
                        _ = _urllib.parse.urlparse("http://%s" % url)  # dummy
                            scheme
                        path = path.lower()
                        filename = _.path.split('/')[-1]
                        name, extension = os.path.splitext(filename)
                        trail = "%s(%s)" % (host, path)
                        if extension in SUSPICIOUS_DIRECT_DOWNLOAD_EXTENSIONS
                            and not is_local(dst_ip) and not any(_ in path for _
                             in WHITELIST_DIRECT_DOWNLOAD_KEYWORDS) and '=' not
                            in _.query and len(name) < 10:
                            log_event((sec, usec, src_ip, src_port, dst_ip,
                                dst_port, PROTO.TCP, TRAIL.URL, trail, "direct␣%
                                s␣download␣(suspicious)" % extension, "(
                                heuristic)"), packet)
                            out(src_ip,dst_ip,PROTO.TCP,packet)
                        else:
                            for desc, regex in SUSPICIOUS_HTTP_PATH_REGEXES:
                                if re.search(regex, filename, re.I):
                                    log_event((sec, usec, src_ip, src_port,
                                        dst_ip, dst_port, PROTO.TCP, TRAIL.URL,
                                        trail, "%s␣(suspicious)" % desc, "(
                                        heuristic)"), packet)
                                    out(src_ip,dst_ip,PROTO.TCP,packet)
                                    break

    elif protocol == socket.IPPROTO_UDP:  # UDP
        _ = ip_data[iph_length:iph_length + 4]
        if len(_) < 4:
            return

        src_port, dst_port = struct.unpack("!HH", _)

        _ = _last_udp
        _last_udp = (sec, src_ip, src_port, dst_ip, dst_port)
        if _ == _last_udp:  # skip bursts
```

```
            return

        if src_port != 53 and dst_port != 53:  # not DNS
            if dst_ip in trails:
                trail = dst_ip
            elif src_ip in trails:
                trail = src_ip
            else:
                trail = None

            if trail:
                _ = _last_logged_udp
                _last_logged_udp = _last_udp
                if _ != _last_logged_udp:
                    if not any(_ in trails[trail][0] for _ in ("malware",)):
                        log_event((sec, usec, src_ip, src_port, dst_ip, dst_port,
                            PROTO.UDP, TRAIL.IP, trail, trails[trail][0], trails[
                            trail][1]), packet)
                        out(src_ip,dst_ip,PROTO.UDP,packet)

        else:
            dns_data = ip_data[iph_length + 8:]

            # Reference: http://www.ccs.neu.edu/home/amislove/teaching/cs4700/fall09
                /handouts/project1-primer.pdf
            if len(dns_data) > 6:
                qdcount = struct.unpack("!H", dns_data[4:6])[0]
                if qdcount > 0:
                    offset = 12
                    query = ""

                    while len(dns_data) > offset:
                        length = ord(dns_data[offset:offset + 1])
                        if not length:
                            query = query[:-1]
                            break
                        query += get_text(dns_data[offset + 1:offset + length + 1])
                            + '.'
                        offset += length + 1

                    query = query.lower()

                    if not query or re.search(VALID_DNS_NAME_REGEX, query) is None
                        or any(_ in query for _ in (".intranet.",)) or query.split('
                        .')[-1] in IGNORE_DNS_QUERY_SUFFIXES:
                        return

                    parts = query.split('.')

                    if ord(dns_data[2:3]) & 0xfa == 0x00:  # standard query (both
                        recursive and non-recursive)
```

```
type_, class_ = struct.unpack("!HH", dns_data[offset + 1:
    offset + 5])

if len(parts) > 2:
    if len(parts) > 3 and len(parts[-2]) <= 3:
        domain = '.'.join(parts[-3:])
    else:
        domain = '.'.join(parts[-2:])

    if not _check_domain_whitelisted(domain):  # e.g. <hash
        >.hashserver.cs.trendmicro.com
        if (sec - (_subdomains_sec or 0)) > HOURLY_SECS:
            _subdomains.clear()
            _dns_exhausted_domains.clear()
            _subdomains_sec = sec

        subdomains = _subdomains.get(domain)

        if not subdomains:
            subdomains = _subdomains[domain] = _set()
            subdomains._start = sec

        if not re.search(r"\A\d+\-\d+\-\d+\-\d+\Z", parts
            [0]):
            if sec - subdomains._start > 60:
                subdomains._start = sec
                subdomains.clear()
            elif len(subdomains) < DNS_EXHAUSTION_THRESHOLD:
                subdomains.add('.'.join(parts[:-2]))
            else:
                trail = "(%s).%s" % ('.'.join(parts[:-2]), '
                    .'.join(parts[-2:]))
                if re.search(r"bl\b", trail) is None:

                    # generic check for DNSBLs
                    if not any(_ in subdomains for _ in
                        LOCAL_SUBDOMAIN_LOOKUPS):
                                        # generic check
                        for local DNS resolutions
                        log_event((sec, usec, src_ip,
                            src_port, dst_ip, dst_port,
                            PROTO.UDP, TRAIL.DNS, trail, "
                            potential␣dns␣exhaustion␣(
                            suspicious)", "(heuristic)"),
                            packet)
                        out(src_ip,dst_ip,PROTO.UDP,packet)
                        _dns_exhausted_domains.add(domain)

                    return

# Reference: http://en.wikipedia.org/wiki/
```

```
                    List_of_DNS_record_types
                if type_ not in (12, 28) and class_ == 1:  # Type not in (
                    PTR, AAAA), Class IN
                    if addr_port(dst_ip, dst_port) in trails:
                        trail = addr_port(dst_ip, dst_port)
                        log_event((sec, usec, src_ip, src_port, dst_ip,
                            dst_port, PROTO.UDP, TRAIL.IPORT, "%s (%s)" % (
                            dst_ip, query), trails[trail][0], trails[trail
                            ][1]), packet)
                        out(src_ip,dst_ip,PROTO.UDP,packet)
                    elif dst_ip in trails:
                        log_event((sec, usec, src_ip, src_port, dst_ip,
                            dst_port, PROTO.UDP, TRAIL.IP, "%s (%s)" % (
                            dst_ip, query), trails[dst_ip][0], trails[dst_ip
                            ][1]), packet)
                        out(src_ip,dst_ip,PROTO.UDP,packet)
                    elif src_ip in trails:
                        log_event((sec, usec, src_ip, src_port, dst_ip,
                            dst_port, PROTO.UDP, TRAIL.IP, src_ip, trails[
                            src_ip][0], trails[src_ip][1]), packet)
                        out(src_ip,dst_ip,PROTO.UDP,packet)
                    _check_domain(query, sec, usec, src_ip, src_port, dst_ip
                        , dst_port, PROTO.UDP, packet)

            elif config.USE_HEURISTICS:
                if ord(dns_data[2:3]) & 0x80:  # standard response
                    if ord(dns_data[3:4]) == 0x80:  # recursion available,
                        no error
                        _ = offset + 5
                        try:
                            while _ < len(dns_data):
                                if ord(dns_data[_:_ + 1]) & 0xc0 != 0 and
                                    dns_data[_ + 2] == "\00" and dns_data[_
                                    + 3] == "\x01":  # Type A
                                    break
                                else:
                                    _ += 12 + struct.unpack("!H", dns_data[_
                                         + 10: _ + 12])[0]

                            _ = dns_data[_ + 12:_ + 16]
                            if _:
                                answer = socket.inet_ntoa(_)
                                if answer in trails and not
                                    _check_domain_whitelisted(query):
                                    _ = trails[answer]
                                    if "sinkhole" in _[0]:
                                        trail = "(%s).%s" % ('.'.join(parts
                                            [:-1]), '.'.join(parts[-1:]))
                                        log_event((sec, usec, src_ip,
                                            src_port, dst_ip, dst_port,
                                            PROTO.UDP, TRAIL.DNS, trail, "
```

```
                                sinkholed␣by␣%s␣(malware)" % _
                                [0].split("␣")[1], "(heuristic)"
                                ), packet)  # (e.g. kitro.pl,
                                devomchart.com, jebena.
                                ananikolic.su, vuvet.cn)
                            out(src_ip,dst_ip,PROTO.UDP,packet)
                        elif "parking" in _[0]:
                            trail = "(%s).%s" % ('.'.join(parts
                                [:-1]), '.'.join(parts[-1:]))
                            log_event((sec, usec, src_ip,
                                src_port, dst_ip, dst_port,
                                PROTO.UDP, TRAIL.DNS, trail, "
                                parked␣site␣(suspicious)", "(
                                heuristic)"), packet)
                            out(src_ip,dst_ip,PROTO.UDP,packet)
                except IndexError:
                    pass

            elif ord(dns_data[3:4]) == 0x83:  # recursion available,
                 no such name
                if '.'.join(parts[-2:]) not in
                    _dns_exhausted_domains and not
                    _check_domain_whitelisted(query) and not
                    _check_domain_member(query, trails):
                    if parts[-1].isdigit():
                        return

                    if not (len(parts) > 4 and all(_.isdigit() and
                        int(_) < 256 for _ in parts[:4])):  #
                        generic check for DNSBL IP lookups
                        if not is_local(dst_ip):  # prevent FPs
                            caused by local queries
                            for _ in filter(None, (query, "*.%s" % '
                                .'.join(parts[-2:]) if query.count('
                                .') > 1 else None)):
                                if _ not in NO_SUCH_NAME_COUNTERS or
                                     NO_SUCH_NAME_COUNTERS[_][0] !=
                                    sec // 3600:
                                    NO_SUCH_NAME_COUNTERS[_] = [sec
                                        // 3600, 1, set()]
                                else:
                                    NO_SUCH_NAME_COUNTERS[_][1] += 1
                                    NO_SUCH_NAME_COUNTERS[_][2].add(
                                        query)

                                    if NO_SUCH_NAME_COUNTERS[_][1] >

                                        NO_SUCH_NAME_PER_HOUR_THRESHOLD
                                        :
                                        if _.startswith("*."):
                                            trail = "%s%s" % ("(%s)"
```

```
                 % ','.join(item.
                replace(_[1:], "")
                for item in
                NO_SUCH_NAME_COUNTERS
                [_][2]), _[1:])
            if not any(subdomain in
                trail for subdomain
                in
                LOCAL_SUBDOMAIN_LOOKUPS
                ):  # generic check
                for local DNS
                resolutions
                log_event((sec, usec
                    , src_ip,
                    src_port, dst_ip
                    , dst_port,
                    PROTO.UDP, TRAIL
                    .DNS, trail, "
                    excessive␣no␣
                    such␣domain␣(
                    suspicious)", "(
                    heuristic)"),
                    packet)
                out(src_ip,dst_ip,
                    PROTO.UDP,packet
                    )
            for item in
                NO_SUCH_NAME_COUNTERS
                [_][2]:
                try:
                    del
                        NO_SUCH_NAME_COUNTERS
                        [item]
                except KeyError:
                    pass
        else:
            log_event((sec, usec,
                src_ip, src_port,
                dst_ip, dst_port,
                PROTO.UDP, TRAIL.DNS
                , _, "excessive␣no␣
                such␣domain␣(
                suspicious)", "(
                heuristic)"), packet
                )
            out(src_ip,dst_ip,PROTO.
                UDP,packet)
        try:
            del
                NO_SUCH_NAME_COUNTERS
                [_]
```

```
                                except KeyError:
                                    pass

                                break

                        if len(parts) == 2 and parts[0] and '-' not
                            in parts[0]:
                            part = parts[0]
                            trail = "(%s).%s" % (parts[0], parts[1])

                            result = _result_cache.get(part)

                            if result is None:
                                # Reference: https://github.com/
                                    exp0se/dga_detector
                                probabilities = (float(part.count(c
                                    )) / len(part) for c in set(_
                                    for _ in part))
                                entropy = -sum(p * math.log(p) /
                                    math.log(2.0) for p in
                                    probabilities)
                                if entropy >
                                    SUSPICIOUS_DOMAIN_ENTROPY_THRESHOLD
                                    :
                                    result = "entropy␣threshold␣no␣
                                        such␣domain␣(suspicious)"

                                if not result:
                                    if sum(_ in CONSONANTS for _ in
                                        part) >
                                        SUSPICIOUS_DOMAIN_CONSONANT_THRESHOLD
                                        :
                                        result = "consonant␣
                                            threshold␣no␣such␣domain
                                            ␣(suspicious)"

                                _result_cache[part] = result or
                                    False

                            if result:
                                log_event((sec, usec, src_ip,
                                    src_port, dst_ip, dst_port,
                                    PROTO.UDP, TRAIL.DNS, trail,
                                    result, "(heuristic)"), packet)
                                out(src_ip,dst_ip,PROTO.UDP,packet)
    elif protocol in IPPROTO_LUT:  # non-TCP/UDP (e.g. ICMP)
        if protocol == socket.IPPROTO_ICMP:
            if ord(ip_data[iph_length:iph_length + 1]) != 0x08:  # Non-echo request
                return
        elif protocol == socket.IPPROTO_ICMPV6:
            if ord(ip_data[iph_length:iph_length + 1]) != 0x80:  # Non-echo request
```

```
                    return

            if dst_ip in trails:
                log_event((sec, usec, src_ip, '-', dst_ip, '-', IPPROTO_LUT[protocol],
                    TRAIL.IP, dst_ip, trails[dst_ip][0], trails[dst_ip][1]), packet)
                out(src_ip,dst_ip,protocol,packet)
            elif src_ip in trails:
                log_event((sec, usec, src_ip, '-', dst_ip, '-', IPPROTO_LUT[protocol],
                    TRAIL.IP, src_ip, trails[src_ip][0], trails[src_ip][1]), packet)
                out(src_ip,dst_ip,protocol,packet)

    except struct.error:
        pass

    except Exception:
        if config.SHOW_DEBUG:
            traceback.print_exc()

    #if protocol == socket.IPPROTO_UDP or protocol == socket.IPPROTO_TCP:  # IPv4
    #print(ip_header)
    #out(src_ip,dst_ip,protocol,packet)
    #print('aaaaa')
    #time.sleep(0.2)
    #print(TRAIL.IP)

################################################################
#maltrailのパケット情報を標準出力
def out(src_ip,dst_ip,protocol,packet):
 global dev_IPv4

 TIMEOUT_MS = 1000

 today = datetime.datetime.today()
 now = today.strftime('%Y-%m-%d␣%H:%M:%S.%f')

 #デバイス名
 dev = 'eth0'
 #デバイスのIPvプロトコル番号4アドレス
 try:
  dev_IPv4 = netifaces.ifaddresses(dev)[netifaces.AF_INET][0]['addr']
 except:
  return

 if src_ip == dev_IPv4:
  direction = 0 #送信
 elif dst_ip == dev_IPv4:
  direction = 1 #受信
 else:
  direction = -1

 json_data ={'datetime': now,
```

```
  'address': (src_ip,dst_ip),
  'direction': direction,
  'protocol': protocol,
  'length': len(str(packet))}
  #'port': (src_port, dst_port)}
#json_data['port'] = (src_port, dst_port)


try:
 record = reader.city(json_data['address'][1-direction])
 json_data['country'] = record.country.iso_code
 json_data['latlng'] = (record.location.latitude, record.location.longitude)
except:
  json_data['country'] = '??'
'''
if direction == 0:
 sql = "SELECT EXISTS(SELECT * FROM IPv4 WHERE ip=INET_ATON('"+dst_ip+"'));"
elif direction == 1:
 sql = "SELECT EXISTS(SELECT * FROM IPv4 WHERE ip=INET_ATON('"+src_ip+"'));"
else:
 sql = "SELECT EXISTS(SELECT * FROM IPv4 WHERE ip=INET_ATON('"+src_ip+"') AND ip=
     INET_ATON('"+dst_ip+"'));"

try:
 cursor.execute(sql)
except:
 pass
result = cursor.fetchone()[0]
'''
json_data['white'] = 1;

#json_encode
encode_json_data = json.dumps(json_data)
#jsonデータを標準出力
sys.stdout.write(encode_json_data)
sys.stdout.flush()

def init():
    """
    Performs sensor initialization
    """

    global _multiprocessing

    try:
        import multiprocessing

        if config.PROCESS_COUNT > 1 and not config.profile:
            _multiprocessing = multiprocessing
    except (ImportError, OSError, NotImplementedError):
        pass
```

```
def update_timer():
    retries = 0
    if not config.offline:
        while retries < CHECK_CONNECTION_MAX_RETRIES and not check_connection():
            #sys.stdout.write("[!] can't update because of lack of Internet
                connection (waiting..." if not retries else '.')
            #sys.stdout.flush()
            time.sleep(10)
            retries += 1

        if retries:
            print(")")

    if config.offline or retries == CHECK_CONNECTION_MAX_RETRIES:
        if retries == CHECK_CONNECTION_MAX_RETRIES:
            print("[x] going to continue without online update")
        _ = update_trails(offline=True)
    else:
        _ = update_trails()
        update_ipcat()

    if _:
        trails.clear()
        trails.update(_)
    elif not trails:
        _ = load_trails()
        trails.update(_)

    _regex = ""
    for trail in trails:
        if "static" in trails[trail][1]:
            if re.search(r"[\].][*+]|\[[a-z0-9_.\-]+\]", trail, re.I):
                try:
                    re.compile(trail)
                except re.error:
                    pass
                else:
                    if re.escape(trail) != trail:
                        index = _regex.count("(?P<g")
                        if index < 100:  # Reference: https://stackoverflow.com/
                            questions/478458/python-regular-expressions-with-more-
                            than-100-groups
                            _regex += "|(?P<g%s>%s)" % (index, trail)

    trails._regex = _regex.strip('|')

    thread = threading.Timer(config.UPDATE_PERIOD, update_timer)
    thread.daemon = True
    thread.start()
```

```
create_log_directory()
get_error_log_handle()

msg = "[i] using '%s' for trail storage" % config.TRAILS_FILE
if os.path.isfile(config.TRAILS_FILE):
    mtime = time.gmtime(os.path.getmtime(config.TRAILS_FILE))
    msg += " (last modification: '%s')" % time.strftime(HTTP_TIME_FORMAT, mtime)

#print(msg)

update_timer()

if not config.DISABLE_CHECK_SUDO and check_sudo() is False:
    exit("[!] please run '%s' with root privileges" % __file__)

if config.plugins:
    config.plugin_functions = []
    for plugin in re.split(r"[,;]", config.plugins):
        plugin = plugin.strip()
        found = False

        for _ in (plugin, os.path.join("plugins", plugin), os.path.join("plugins", "
            %s.py" % plugin)):
            if os.path.isfile(_):
                plugin = _
                found = True
                break

        if not found:
            exit("[!] plugin script '%s' not found" % plugin)
        else:
            dirname, filename = os.path.split(plugin)
            dirname = os.path.abspath(dirname)
            if not os.path.exists(os.path.join(dirname, '__init__.py')):
                exit("[!] empty file '__init__.py' required inside directory '%s'" %
                     dirname)

            if not filename.endswith(".py"):
                exit("[!] plugin script '%s' should have an extension '.py'" %
                    filename)

            if dirname not in sys.path:
                sys.path.insert(0, dirname)

            try:
                module = __import__(filename[:-3])
            except (ImportError, SyntaxError) as msg:
                exit("[!] unable to import plugin script '%s' (%s)" % (filename, msg
                    ))

            found = False
```

```
            for name, function in inspect.getmembers(module, inspect.isfunction):
                if name == "plugin" and not set(inspect.getargspec(function).args) &
                     set(("event_tuple',␣'packet")):
                    found = True
                    config.plugin_functions.append(function)
                    function.__name__ = module.__name__

            if not found:
                exit("[!]␣missing␣function␣'plugin(event_tuple,␣packet)'␣in␣plugin␣
                    script␣'%s'" % filename)

if config.pcap_file:
    for _ in config.pcap_file.split(','):
        _caps.append(pcapy.open_offline(_))
else:
    interfaces = set(_.strip() for _ in config.MONITOR_INTERFACE.split(','))

    if (config.MONITOR_INTERFACE or "").lower() == "any":
        if IS_WIN or "any" not in pcapy.findalldevs():
            print("[x]␣virtual␣interface␣'any'␣missing.␣Replacing␣it␣with␣all␣
                interface␣names")
            interfaces = pcapy.findalldevs()
        else:
            #print("[?] in case of any problems with packet capture on virtual
                interface 'any', please put all monitoring interfaces to promiscuous
                 mode manually (e.g. 'sudo ifconfig eth0 promisc')")
            pass
    for interface in interfaces:
        if interface.lower() != "any" and re.sub(r"(?i)\Anetmap:", "", interface)
            not in pcapy.findalldevs():
            hint = "[?]␣available␣interfaces:␣'%s'" % ",".join(pcapy.findalldevs())
            exit("[!]␣interface␣'%s'␣not␣found\n%s" % (interface, hint))

        #print("[i] opening interface '%s'" % interface)
        try:
            _caps.append(pcapy.open_live(interface, SNAP_LEN, True, CAPTURE_TIMEOUT
                ))
        except (socket.error, pcapy.PcapError):
            if "permitted" in str(sys.exc_info()[1]):
                exit("[!]␣permission␣problem␣occurred␣('%s')" % sys.exc_info()[1])
            elif "No␣such␣device" in str(sys.exc_info()[1]):
                exit("[!]␣no␣such␣device␣'%s'" % interface)
            else:
                raise

if config.LOG_SERVER and ':' not in config.LOG_SERVER:
    exit("[!]␣invalid␣configuration␣value␣for␣'LOG_SERVER'␣('%s')" % config.
        LOG_SERVER)

if config.SYSLOG_SERVER and not len(config.SYSLOG_SERVER.split(':')) == 2:
    exit("[!]␣invalid␣configuration␣value␣for␣'SYSLOG_SERVER'␣('%s')" % config.
```

```
        SYSLOG_SERVER)

if config.LOGSTASH_SERVER and not len(config.LOGSTASH_SERVER.split(':')) == 2:
    exit("[!] invalid configuration value for 'LOGSTASH_SERVER' ('%s')" % config.
        LOGSTASH_SERVER)

if config.REMOTE_SEVERITY_REGEX:
    try:
        re.compile(config.REMOTE_SEVERITY_REGEX)
    except re.error:
        exit("[!] invalid configuration value for 'REMOTE_SEVERITY_REGEX' ('%s')" %
            config.REMOTE_SEVERITY_REGEX)

if config.CAPTURE_FILTER:
    #print("[i] setting capture filter '%s'" % config.CAPTURE_FILTER)
    for _cap in _caps:
        try:
            _cap.setfilter(config.CAPTURE_FILTER)
        except:
            pass

if _multiprocessing:
    _init_multiprocessing()

if not IS_WIN and not config.DISABLE_CPU_AFFINITY:
    try:
        try:
            mod = int(subprocess.check_output("grep -c ^processor /proc/cpuinfo",
                stderr=subprocess.STDOUT, shell=True).strip())
            used = subprocess.check_output("for pid in $(ps aux | grep python | grep
                sensor.py | grep -E -o 'root[ ]*[0-9]*' | tr -d '[:alpha:] '); do
                schedtool $pid; done | grep -E -o 'AFFINITY .*' | cut -d ' ' -f 2 |
                grep -v 0xf", stderr=subprocess.STDOUT, shell=True).strip().split('\
                n')
            max_used = max(int(_, 16) for _ in used)
            affinity = max(1, (max_used << 1) % 2 ** mod)
        except:
            affinity = 1
        p = subprocess.Popen("schedtool -n -2 -M 2 -p 10 -a 0x%02x %d" % (affinity,
            os.getpid()), shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE
            )
        _, stderr = p.communicate()
        if "not found" in stderr:
            msg, _ = "[?] please install 'schedtool' for better CPU scheduling",
                platform.linux_distribution()[0].lower()
            for distro, install in {("fedora", "centos"): "sudo yum install
                schedtool", ("debian", "ubuntu"): "sudo apt-get install schedtool"}.
                items():
                if _ in distro:
                    msg += " (e.g. '%s')" % install
                    break
```

```
                #print(msg)
            except:
                pass

def _init_multiprocessing():
    """
    Inits worker processes used in multiprocessing mode
    """

    global _buffer
    global _multiprocessing
    global _n

    if _multiprocessing:
        #print("[i] preparing capture buffer...")
        try:
            _buffer = mmap.mmap(-1, config.CAPTURE_BUFFER)  # http://www.alexonlinux.com
                /direct-io-in-python

            _ = b"\x00" * MMAP_ZFILL_CHUNK_LENGTH
            for i in xrange(config.CAPTURE_BUFFER // MMAP_ZFILL_CHUNK_LENGTH):
                _buffer.write(_)
            _buffer.seek(0)
        except KeyboardInterrupt:
            raise
        except:
            exit("[!] unable to allocate network capture buffer. Please adjust value of
                'CAPTURE_BUFFER'")

        _n = _multiprocessing.Value('L', lock=False)

        try:
            for i in xrange(config.PROCESS_COUNT - 1):
                process = _multiprocessing.Process(target=worker, name=str(i), args=(
                    _buffer, _n, i, config.PROCESS_COUNT - 1, _process_packet))
                process.daemon = True
                process.start()
        except TypeError:   # Note: https://github.com/stamparm/maltrail/issues/11823
            _buffer = None
            _multiprocessing = None
        else:
            #print("[i] created %d more processes (out of total %d)" % (config.
                PROCESS_COUNT - 1, config.PROCESS_COUNT))
            pass
def monitor():
    """
    Sniffs/monitors given capturing interface
    """

    #print("[^] running...")
```

```
def packet_handler(datalink, header, packet):
    global _count

    ip_offset = None
    try:
        dlt_offset = DLT_OFFSETS[datalink]
    except KeyError:
        log_error("Received␣unexpected␣datalink␣(%d)" % datalink, single=True)
        return

    try:
        if datalink == pcapy.DLT_RAW:
            ip_offset = dlt_offset

        elif datalink == pcapy.DLT_PPP:
            if packet[2:4] in (b"\x00\x21", b"\x00\x57"):  # (IPv4, IPv6)
                ip_offset = dlt_offset

        elif datalink == pcapy.DLT_NULL:
            if packet[0:4] in (b"\x02\x00\x00\x00", b"\x23\x00\x00\x00"):  # (IPv4,
                IPv6)
                ip_offset = dlt_offset

        elif dlt_offset >= 2:
            if packet[dlt_offset - 2:dlt_offset] == b"\x81\x00":  # VLAN
                dlt_offset += 4
            if packet[dlt_offset - 2:dlt_offset] in (b"\x08\x00", b"\x86\xdd"):  # (
                IPv4, IPv6)
                ip_offset = dlt_offset

    except IndexError:
        pass

    if ip_offset is None:
        return

    try:
        if six.PY3:  # https://github.com/helpsystems/pcapy/issues/37#issuecomment
            -530795813
            sec, usec = [int(_) for _ in ("%.6f" % time.time()).split('.')]
        else:
            sec, usec = header.getts()

        if _multiprocessing:
            block = struct.pack("=III", sec, usec, ip_offset) + packet

            if _locks.count:
                _locks.count.acquire()

            write_block(_buffer, _count, block)
            _n.value = _count = _count + 1
```

```
                if _locks.count:
                    _locks.count.release()
            else:
                _process_packet(packet, sec, usec, ip_offset)

        except socket.timeout:
            pass

    try:
        def _(_cap):
            global _done_count

            datalink = _cap.datalink()


#
# NOTE: currently an issue with pcapy-png and loop()
#
#            if six.PY3 and not config.pcap_file:  # https://github.com/helpsystems/
    pcapy/issues/37#issuecomment-530795813
#                def _loop_handler(header, packet):
#                    packet_handler(datalink, header, packet)
#
#                _cap.loop(-1, _loop_handler)
#            else:

            while True:
                success = False
                try:
                    (header, packet) = _cap.next()
                    if header is not None:
                        success = True
                        packet_handler(datalink, header, packet)
                    elif config.pcap_file:
                        with _done_lock:
                            _done_count += 1
                        break
                except (pcapy.PcapError, socket.timeout):
                    pass

                if not success:
                    time.sleep(REGULAR_SENSOR_SLEEP_TIME)

        if config.profile and len(_caps) == 1:
            print("[=] will store profiling results to '%s'..." % config.profile)
            _(_caps[0])
        else:
            if len(_caps) > 1:
                if _multiprocessing:
                    _locks.count = threading.Lock()
```

```
                _locks.connect_sec = threading.Lock()

            for _cap in _caps:
                threading.Thread(target=_, args=(_cap,)).start()

            while _caps and not _done_count == (config.pcap_file or "").count(',') + 1:
                time.sleep(1)

        if not config.pcap_file:
            print("[i] all capturing interfaces closed")
    except SystemError as ex:
        if "error return without" in str(ex):
            print("\r[x] stopping (Ctrl-C pressed)")
        else:
            raise
    except KeyboardInterrupt:
        print("\r[x] stopping (Ctrl-C pressed)")
    finally:
        print("\r[i] cleaning up...")

        if _multiprocessing:
            try:
                for _ in xrange(config.PROCESS_COUNT - 1):
                    write_block(_buffer, _n.value, b"", BLOCK_MARKER.END)
                    _n.value = _n.value + 1
                while _multiprocessing.active_children():
                    time.sleep(REGULAR_SENSOR_SLEEP_TIME)
            except KeyboardInterrupt:
                pass

        if config.pcap_file:
            flush_condensed_events(True)

def main():
    for i in xrange(1, len(sys.argv)):
        if sys.argv[i] == "-q":
            #sys.stdout = open(os.devnull, 'w')
            pass
        if sys.argv[i] == "-i":
            for j in xrange(i + 2, len(sys.argv)):
                value = sys.argv[j]
                if os.path.isfile(value):
                    sys.argv[i + 1] += ",%s" % value
                    sys.argv[j] = ''
                else:
                    break

    #print("%s (sensor) #v%s {%s}\n" % (NAME, VERSION, HOMEPAGE))

    if "--version" in sys.argv:
        raise SystemExit
```

```
parser = optparse.OptionParser(version=VERSION)
parser.add_option("-c", dest="config_file", default=CONFIG_FILE, help="configuration
    ␣file␣(default:␣'%s')" % os.path.split(CONFIG_FILE)[-1])
parser.add_option("-r", dest="pcap_file", help="pcap␣file␣for␣offline␣analysis")
parser.add_option("-p", dest="plugins", help="plugin(s)␣to␣be␣used␣per␣event")
parser.add_option("-q", "--quiet", dest="quiet", action="store_true", help="turn␣off
    ␣regular␣output")
parser.add_option("--console", dest="console", action="store_true", help="print␣
    events␣to␣console")
parser.add_option("--offline", dest="offline", action="store_true", help="disable␣(
    online)␣trail␣updates")
parser.add_option("--debug", dest="debug", action="store_true", help=optparse.
    SUPPRESS_HELP)
parser.add_option("--profile", dest="profile", help=optparse.SUPPRESS_HELP)

patch_parser(parser)

options, _ = parser.parse_args()

#print("[*] starting @ %s\n" % time.strftime("%X /%Y-%m-%d/"))

read_config(options.config_file)

for option in dir(options):
    if isinstance(getattr(options, option), (six.string_types, bool)) and not option
        .startswith('_'):
        config[option] = getattr(options, option)

if options.debug:
    config.console = True
    config.PROCESS_COUNT = 1
    config.SHOW_DEBUG = True

if options.pcap_file:
    if options.pcap_file == '-':
        print("[i]␣using␣STDIN")
    else:
        for _ in options.pcap_file.split(','):
            if not os.path.isfile(_):
                exit("[!]␣missing␣pcap␣file␣'%s'" % _)

        print("[i]␣using␣pcap␣file(s)␣'%s'" % options.pcap_file)

if not config.DISABLE_CHECK_SUDO and not check_sudo():
    exit("[!]␣please␣run␣'%s'␣with␣root␣privileges" % __file__)

try:
    init()
    if config.profile:
        open(config.profile, "w+b").write("")
```

```
            cProfile.run("monitor()", config.profile)
        else:
            monitor()
    except KeyboardInterrupt:
        print("\r[x] stopping (Ctrl-C pressed)")

if __name__ == "__main__":
    code = 0

    try:
        main()
    except SystemExit as ex:
        if isinstance(get_ex_message(ex), six.string_types) and get_ex_message(ex).strip
            ('0'):
            print(get_ex_message(ex))
            code = 1
    except IOError:
        log_error("\n\n[!] session abruptly terminated\n[?] (hint: \"https://
            stackoverflow.com/a/20997655\")")
        code = 1
    except Exception:
        msg = "\r[!] unhandled exception occurred ('%s')" % sys.exc_info()[1]
        msg += "\n[x] please report the following details at 'https://github.com/
            stamparm/maltrail/issues':\n---\n'%s'\n---" % traceback.format_exc()
        log_error("\n\n%s" % msg.replace("\r", ""))

        print(msg)
        code = 1
    finally:
        if not any(_ in sys.argv for _ in ("--version", "-h", "--help")):
            print("\n[*] ending @ %s" % time.strftime("%X /%Y-%m-%d/"))

        os._exit(code)
```

index.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
 <meta charset="utf-8">
 <title>パケット可視化システム</title>
 <link rel="stylesheet" href="style.css">
 <script src="js/jquery-3.1.1.min.js"></script>
 <script src="js/three.min.js"></script>
 <script src="js/OrbitControls.js"></script>
 <script src="js/Detector.js"></script>
 <script src="/socket.io/socket.io.js"></script>
 <script src="main3.js"></script>
</head>
<body>
```

```
<!-- 可視化上に文字情報を載せるための領域 -->
<div id="textArea">
 <div id="buttonArea">
  <input type="button" class="button" id="infoTableButton" value="パケット情報"/>
 </div>

 <div id="infoTableArea" style="display:␣none">
  <input type="button" class="button" id="initTableButton" value="履歴のクリア"/>
  <table id="infoTable">
   <thead>
    <tr>
     <th class="datetime">取得日時</th>
     <th class="country">国</th>
     <th class="s_addr">送信元IPアドレス</th>
     <th class="d_addr">宛先IPアドレス</th>
     <th class="protocol">プロトコル</th>
     <th class="length">パケット長</th>
    </tr>
   </thead>
   <tbody id="infoTableBody"></tbody>
  </table>
 </div>
</div>

<!-- 可視化領域 -->
<div id="canvasArea"></div>

</body>
</html>
```

main.js

```
// HTMLが読み込まれた後の処理
$(function(){
 "use␣strict";

 // "パケット情報テーブル"の高さ設定
 $(document).ready(function(){
  var height = $(window).height() - 190;
  $("#infoTableBody").height(height);
 });
 $(window).on("resize", function(){
  var height = $(window).height() - 190;
  $("#infoTableBody").height(height);
 });

 // "パケット情報"ボタンクリック時
 $("#infoTableButton").click(function(){
  var $area = $("#infoTableArea");
```

```
  if( $area.is(":visible") ){
   $area.fadeOut("slow");
  }else{
   $("#menuArea").fadeOut("fast");
   $area.fadeIn("slow");
  }
 });

 // "可視化メニュー"ボタンクリック時
 $("#menuButton").click(function(){
  var $area = $("#menuArea");

  if( $area.is(":visible") ){
   $area.fadeOut("slow");
  }else{
   $("#infoTableArea").fadeOut("fast");
   $area.fadeIn("slow");
  }
 });


});

(function() {
"use strict";

// ------------------------------------------- 変数宣言
var renderer;
var scene;
var camera;
var controls;
var baseTime;

var radius = 20;

var ip = [];

var flow = [];

var flag = [];

var socket = null;

var marker = 0;
var flagmarker = 0;
var MAX_FLOW = 300;  // 表示可能なパケットフローの上限
var MAX_COUNTRY = 2;


// ------------------------------------------- window イベント
```

```
// 読み込み時のイベント
$(function(){
 init();
});

// リサイズ時のイベント
$(window).resize(function() {
 renderer.setSize(window.innerWidth, window.innerHeight);
 camera.aspect = window.innerWidth / window.innerHeight;
 camera.updateProjectionMatrix();
});


// ------------------------------------------- 読み込み時の処理
function init() {
 initThree();
}


// 可視化モデルの準備
function initModel() {
 // シーン作成
 scene = new THREE.Scene();

 // ライト作成
 var ambient = new THREE.AmbientLight(0xffffff);
 scene.add(ambient);

 // テクスチャ読み込み
 var textureLoader = new THREE.TextureLoader();
 var texture = textureLoader.load('earthblack.png');
 // 地球儀(裏面)
 var earthGeometry = new THREE.SphereGeometry(radius, 36, 36);
 var earthBackMaterial = new THREE.MeshStandardMaterial({
  //color: 0xffffff,
  //color:0x008080,
  color: 0x696969,
  roughness: 1.0,
  metalness: 0.2,
  opacity:0.5,
  //wireframe: true,
  transparent: true,
  map: texture,
  side: THREE.BackSide });
 scene.add( new THREE.Mesh(earthGeometry, earthBackMaterial) );
 var earthwire = new THREE.SphereGeometry(radius, 36, 18);
 var earthwireMaterial = new THREE.MeshStandardMaterial({
  color: 0x5f9ea0,
  roughness: 1.0,
  matalness: 0.3,
  wireframe: true,
```

```
 transparent: true,
 opacity:0.4 });
scene.add( new THREE.Mesh(earthwire, earthwireMaterial) );
// 地球儀(表面)
var earthFrontMaterial = new THREE.MeshStandardMaterial({
 color: 0xffffff,
 //color:0x6699FF,
 //color:0x000080,
 roughness: 1.0,
 metalness: 0.2,
 opacity:0.5,
 //wireframe: true,
 transparent: true,
 map: texture,
 side: THREE.FrontSide });
scene.add( new THREE.Mesh(earthGeometry, earthFrontMaterial) );
// 背景が地球儀に透けてしまわないための措置
var geometry = new THREE.SphereGeometry(radius+0.1, 36, 18);
var material = new THREE.MeshStandardMaterial({
 color: 0x000000,
 side: THREE.BackSide });
scene.add( new THREE.Mesh(geometry, material) );

// 宇宙

var spaceGeometry = new THREE.Geometry();
for (var i = 0; i < 2000; i++) {
 var phi = Math.random() * Math.PI * 2;
 var theta = Math.random() * Math.PI * 2;
 spaceGeometry.vertices.push(new THREE.Vector3(
  1000 * Math.cos(phi) * Math.cos(theta),
  1000 * Math.sin(phi),
  1000 * Math.cos(phi) * Math.sin(theta)));
}
var spaceMaterial = new THREE.PointsMaterial({size: 2, color: 0x5f9ea0});
var space = new THREE.Points(spaceGeometry, spaceMaterial);
scene.add(space);

// 自分のPCに見立てた立方体
var geometry = new THREE.CubeGeometry(1, 1, 1);
var material = new THREE.MeshStandardMaterial();
var mesh = new THREE.Mesh(geometry, material);
scene.add(mesh);

return scene;
}

function initThree() {
// canvas領域の取得
var canvasFrame = document.getElementById("canvasArea");
```

```
 // レンダラ初期化
 renderer = new THREE.WebGLRenderer({ antialias: true});
 renderer.setSize(window.innerWidth, window.innerHeight);
 renderer.setClearColor(0x000000, 1);
 canvasFrame.appendChild(renderer.domElement);

 // カメラ作成
 camera = new THREE.PerspectiveCamera(60, window.innerWidth / window.innerHeight);
 camera.position.set(0, 0, 50);
 // カメラコントロール作成
 controls = new THREE.OrbitControls(camera);
 controls.autoRotate = true;

 scene = initModel();

 var loader = new THREE.JSONLoader();
 loader.load('monitor.json', function (geometry, materials)
 {
     var geo = geometry;
     var mat = new THREE.MeshFaceMaterial(materials);
     var mesh = new THREE.Mesh(geo, mat);
     scene.add(mesh);
 });

 initFlow();

 baseTime = +new Date;
 render();

 startNetwork();
}




// ------------------------------------------- レンダリング
function render() {
 requestAnimationFrame(render);
 // カメラコントロールの状態を更新
 controls.update();

 updateFlow();

 renderer.render(scene, camera);
};


// パケットフローオブジェクトをあらかじめ準備しておく
function initFlow() {
 for(var i=0; i < MAX_FLOW; i++) {
  var packet = {};
```

```
  // パケットの経路に見立てた線分
  var lineGeometry = new THREE.Geometry();
  lineGeometry.vertices.push(new THREE.Vector3(0, 0, 0));
  lineGeometry.vertices.push(new THREE.Vector3(0, 0, 0));
  var lineMaterial = new THREE.LineBasicMaterial({color: 0xffffff, linewidth: 1,
      transparent: false, opacity: 0.1});
  var line = new THREE.Line(lineGeometry, lineMaterial);

  // パケットに見立てた球体
  var sphereGeometry = new THREE.SphereGeometry(0.5);
  var sphereMaterial = new THREE.MeshStandardMaterial({color: 0xffffff});
  var sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);

  line.visible = false;
  sphere.visible = false;

  scene.add(line);
  scene.add(sphere);

  // オブジェクトに格納
  packet.line = line;
  packet.lineG = lineGeometry;
  packet.lineM = lineMaterial;
  packet.sphere = sphere;
  packet.sphereG = sphereGeometry;
  packet.sphereM = sphereMaterial;
  packet.progress = 1.0;
  packet.available = true;

  flow.push(packet);
 }
}


// パケットフローオブジェクトを追加する
function setFlow(pkt) {

 var packet = flow[marker];
 marker = (marker + 1) % MAX_FLOW;

 // 仰角
 var phi = pkt.latlng[0] * (Math.PI / 180);
 // 方位角
 var theta = pkt.latlng[1] * (Math.PI / 180) * -1;

 // ホワイトリストに存在するパケットなら色を緑
 /*if(pkt.white == 0) {
  var packetColor = 0x00ff00;
  packet.line.visible = false;
  packet.sphere.visible = false;
```

```
 }
 else {*/
 var packetColor = 0xff0000;
 packet.line.visible = true;
 packet.sphere.visible = true;
 setmark(radius * Math.cos(phi) * Math.cos(theta),radius * Math.sin(phi),radius * Math.
     cos(phi) * Math.sin(theta),phi);
  //setmark(radius * Math.cos(phi) * Math.cos(theta),radius * Math.sin(phi),radius *
      Math.cos(phi) * Math.sin(theta))
 //}

 // パケットの経路の設定
 var line = packet.line;
 var lineGeometry = packet.lineG;
 lineGeometry.verticesNeedUpdate = true;
 lineGeometry.vertices[1].x = radius * Math.cos(phi) * Math.cos(theta);
 lineGeometry.vertices[1].y = radius * Math.sin(phi);
 lineGeometry.vertices[1].z = radius * Math.cos(phi) * Math.sin(theta);
 var lineMaterial = packet.lineM;
 lineMaterial.color.setHex(packetColor);

 // パケットに見立てた球体の設定
 var sphereMaterial = packet.sphereM;
 sphereMaterial.color.setHex(packetColor);

 // シーンに追加
 /*
 if(packet.available == true) {
  packet.line.visible = true;
  packet.sphere.visible = true;
 }
 */

 // パケットフローオブジェクトに情報を追加する
 packet.phi = phi;
 packet.theta = theta;
 packet.progress = 0.0;
 packet.available = false;
 packet.direction = pkt.direction;
}


// パケットフローオブジェクトの状態を更新する
function updateFlow() {
 for(var i=0; i < MAX_FLOW; i++) {
  var packet = flow[i];

  // パケットフローが空なら何もしない
  if(packet.available == true) {
   continue;
  }
```

```
  if(packet.progress < 1.0) {
   var phi = packet.phi;
   var theta = packet.theta;
   var direction = packet.direction;
   var progress = packet.progress;

   var sphere = packet.sphere;
   if(direction == 0) {
    sphere.position.x = radius * Math.cos(phi) * Math.cos(theta) * progress;
    sphere.position.y = radius * Math.sin(phi) * progress;
    sphere.position.z = radius * Math.cos(phi) * Math.sin(theta) * progress;
   } else {
    sphere.position.x = radius * Math.cos(phi) * Math.cos(theta) * (1.0 - progress);
    sphere.position.y = radius * Math.sin(phi) * (1.0 - progress);
    sphere.position.z = radius * Math.cos(phi) * Math.sin(theta) * (1.0 - progress);
   }

   packet.progress += 0.01;
  } else {
   packet.sphere.visible = false;
   packet.line.visible = true;
   packet.available = false;
   /*
   if(packet.white == 1) {
    setmark(sphere.position.x,sphere.position.y,sphere.position.z,theta);
   }
   else {
   }
   */
  }
 }
}


function setmark(x,y,z,phi) {

 var textureLoader = new THREE.TextureLoader();
 var texture = textureLoader.load('fire.jpg');
 //for(var i=0; i < MAX_COUNTRY; i++) {
 //不正通信先をハイライトするオブジェクト
 /*
 var geometry = new THREE.CylinderGeometry(1, 1, 2, 25, 25, true);
 var material = new THREE.MeshBasicMaterial({
  color: 0xffffff,
  map: texture,
  transparent: true
  //blending: THREE.AdditiveBlending,
  //side: THREE.FrontSide
 });
 */
```

```
 var geometry = new THREE.SphereGeometry( 0.5, 16, 16);
 //var geometry = new THREE.TorusKnotGeometry(0.2, 0.5, 30, 8, 2, 5, 4);
 var material = new THREE.MeshBasicMaterial({color: 0xffB22222,transparent: true});
 var circle = new THREE.Mesh(geometry, material);

 circle.visible = true;

 circle.position.x = x*1.05;
 circle.position.y = y*1.05;
 circle.position.z = z*1.05;
 //circle.rotation.z = phi;

 //scene.add(circle);

 //オブジェクトに格納
 flag[flagmarker] = circle;

 scene.add(flag[flagmarker]);

 flagmarker = (flagmarker + 1) % MAX_COUNTRY;

//}
}


/*
function setmark(x,y,z) {
 //if( flagmarker < MAX_COUNTRY ){
 var textureLoader = new THREE.TextureLoader();
 var texture = textureLoader.load('fire.jpg');
 const geometry = new THREE.CylinderGeometry(0.5, 0.5, 0.5, 25, 25, true);
 const geometry = new THREE.TorusGeometry(0.2,0.2,2,100);
 const material = new THREE.MeshBasicMaterial({
  color: 0xffffff,
  map: texture,
  transparent: true
  //blending: THREE.AdditiveBlending,
  //side: THREE.FrontSide
 });
 var mark = new THREE.Mesh(geometry,material);


 var mark = flag[flagmarker];
 flagmarker = (flagmarker + 1) % MAX_COUNTRY;

 //通信場所に表示されるオブジェクトの位置
 mark.x = x;
 mark.y = y;
 mark.z = z;
```

```
 for(var i=0; i<MAX_COUNTRY; i++) {
  if ( mark.position.x != flag[i].x && mark.position.y != flag[i].y && mark.position.z
      != flag[i].z ) {
   flagmarker = (flagmarker + 1) % MAX_COUNTRY;
   flag[i].x = x;
   flag[i].y = y;
   flag[i].z = z;
   console.log('flag_notexsist');
   mark.visible = true;
   break;
  }
  else {
   console.log('flag_exsist');
   mark.visible = true;
  }
 }

 mark.x.NeedUpdate = true;
 mark.y.NeedUpdate = true;
 mark.z.NeedUpdate = true;
 mark.visible = true;
 //mesh.rotation.x = MATH.PI/4;
 //scene.add(mesh);

}
*/


// ------------------------------------------- socket処理
function startNetwork() {
 // 接続開始
 socket = io.connect();

 // 接続時
 socket.on('connect', function() {
  console.log('start␣socket.io');
  //socket.emit('capture');
 });

 // 受信時イベント
 socket.on('packet', function(data) {
  var packet = JSON.parse(data);

  if(data[0] != '{') {
   console.log('aaaaaaaaaaa');
  }
  else {
   if(packet.country != '??') {
    setFlow(packet);
    //console.log('country:OK')
   } else {
```

```
    setFlow({latlng:[-90, 0], white: 0, direction: packet.direction});
    //console.log('country:not␣OK')
   }
   //console.log(packet.country);
   insertInfoTable(packet);
   //console.log('aaaa');
   //sleep(0.1);
  }
 });

 socket.on('disconnect', function() {
  console.log('stopped␣socket.io');
 });
}

// パケット情報を表に追加する
function insertInfoTable(packet) {
 // 行の設定
 //console.log('aaaaf');
 var $row = $("<tr></tr>")
  .append($("<td␣class='datetime'></td>").text(packet.datetime))
  .append($("<td␣class='country'></td>></td>").text(packet.country))
  .append($("<td␣class='s_addr'></td>></td>").text(packet.address[0]))
  .append($("<td␣class='d_addr'></td>></td>").text(packet.address[1]))
  .append($("<td␣class='protocol'></td>></td>").text(packet.protocol))
  .append($("<td␣class='length'></td>></td>").text(packet.length));
  //.append($("<td␣class='s_port'></td>></td>").text(packet.port[0]))
  //.append($("<td␣class='d_port'></td>></td>").text(packet.port[1]));

 //console.log('aaaa');
 // テーブル取得
 var $table = $('#infoTable␣tbody');
 // テーブルに行を追加
 $table.append($row);

 // 最下部までスクロール
 $table.scrollTop($table[0].scrollHeight);
}


})();
```