Vol.2015-CSEC-69 No.18
Vol.2015-IOT-29 No.18
2015/5/21

TLS/SSLプロトコルを対象とした 汎用ハニーポットシステムの実装とHTTPSによる収集結果

佐藤 聡[1,a)]　佐藤 聖[2]　中井 央[1]　新城 靖[3]

概要：近年，セキュア通信を行うプロトコルとして TLS/SSL が広く用いられている．それゆえに攻 撃者の関心も非常に高い．我々は TTLS/SSL プロトコルに関する攻撃情報が容易に取得できる 汎用的なハニーポットシステムの設計，実装を行った．このシステムを用いて，HTTPS 用の ハニーポットを実装し，筑波大学のダークネットに設置して情報収集した結果について述べる．

キーワード：Internet/LAN 運用管理技術，トラフィック解析/管理，侵入検出・検知，不正・異常検出

Implementation of a general-purpose honeypot for TLS / SSL protocol and experimental results by HTTPS

Abstract: In recent years , TLS / SSL is widely used as a protocol for secure communication. It is interested in it for attackers. We designed and implementated a generic honeypot system that can easily collect attack information about TTLS / SSL protocol. Using this system, we implements an HTTPS honeypot. And we describe results of the collected information to be installed in darknet at University of Tsukuba.

Keywords: Internet/LAN operation and management technology,Traffic analysis/management,Intrusion detection,Fraud/error detection

1. はじめに

近年，機密性が高い情報を扱う TLS(Transport Layer Security)/SSL(Secure Socket Layer) に対して，攻撃者の関心は非常に高い．最近では，OpenSSL の Heartbleed 脆弱性が重大なインシデントを引き起こした [2]．このインシデントにおいて，多くの攻撃者が攻撃対象としたのが，HTTPS サーバが動作するため開放されている 443 番ポートである [3]．しかし，HTTPS 以外にも TSL/SSL を用いているプロトコルは多数存在する．ゆえに，TLS/SSL に関する に対する攻撃パターンの収集・解析を行うことは，TLS/SSL に対する攻撃の早期発見や対処のため重要である．

攻撃パターン収集のための手法として，ハニーポットがある．現在 HTTPS を対象としたハニーポットは，本物の OS と Web サーバなどのアプリケーションを用い，パケットを記録するものが主流である．このような実現方法のハニーポットを，高対話型ハニーポット，特に仮想機械を用いたものを仮想ハニーポットと呼ぶ．これらの手法は，高度な情報を得ることができる反面，大規模なネットワーク上に設置する場合コストが増大する．対して，OS やアプリケーションをエミュレートするものを低対話型ハニーポットと呼び，機能や収集可能な情報が劣るものの，比較的低コスト，安全に設置することが可能である．

現在，HTTPS ハニーポットとして用いられるものの多くは，実物の HTTPS サーバとパケットキャプチャを組み合わせた高対話型，仮想ハニーポットである．しかしながら，この手法を筑波大学のダークネットを例とする大規模なネットワークに用いると，複数の問題が生じる．まず，

[1] 筑波大学 学術情報メディアセンター
Academic Computing and Communications Center, University of Tsukuba
[2] 筑波大学 情報学群情報科学類
College of Information Science, The School of Informatics, University of Tsukuba
[3] 筑波大学 システム情報工学域 情報工学域
Division of Information Engineering, Faculty of Engineering, Information and Systems, University of Tsukuba
[a)] akira@cc.tsukuba.ac.jp

仮想機械を用いるため設置コストが大きくなる上，複数ホストにまたがっての証明書管理が煩雑となる．Apache バーチャルホストなどのバーチャルホストを用いる手法も存在するが，これらは IP アドレス単位もしくはドメイン名単位でのみ設定が可能である．故に，ネットワークに設置する場合，バーチャルホストの定義に加え，IP エイリアスを用いて 1 つのネットワークカードに複数の IP アドレスを割り当てるなど複雑な設定を要する．また，パケットキャプチャを用いた高対話型ハニーポットの多くは，プロトコルに関わりなく時系列順にまとまったパケットを記録する．しかしながら，TLS はメッセージ単位での送受信が行われるため，解析の際にパケットを分割する必要が生ずる．そのため，特定のメッセージに焦点を当てて，ログを横断的に解析を行うことに労力を要する．

本研究では，大規模なネットワークに低コストで設置することが可能な，TLS/SSL プロトコルを対象とした汎用ハニーポットシステムを設計し，Poohunt と名付け実装を行った．Poohunt は，ネットワークセグメントのエミュレートを行うことが可能なハニーポットである Honeyd[10] をバーチャルホストとして用いることにより，ネットワークへの設置が容易である．また，パケットを TLS メッセージごとに分割し，データベースへの記録を行うことにより，TLS プロトコルに対する脅威の発見，解析が迅速に行うことが可能である．今回は特に TLS/SSL プロトコルの上位プロトコルとして，HTTPS を対象に実装を行った．

設計，実装を行なったシステムの実証実験を行うため，筑波大学のダークネットに Poohunt を設置し，TLS/SSL ネゴシエーション及び HTTP リクエストの収集を行った．未使用 IP アドレスは 60,541 個に渡るが，Poohunt は 1 台のホストで要求された全ての HTTPS 接続に応答することが可能であった．TLS ネゴシエーションの解析の結果から，自己署名証明書を用いているのにも関わらず約 36%の接続が TLS セッションを確立したことが判明した．加えて，Heartbleed 脆弱性を対象としたスキャンが頻繁に行われていることを観察した．

2. 関連研究

2.1 ハニーポットを用いた HTTPS に対する攻撃の解析

Zakir らの研究 [3] では，Heartbleed 脆弱性への攻撃を追跡するため，調査対象の一つとして Amazon EC2[11] 上に設置したハニーポットのフルパケットトレースによるログをを用いている．その結果，脆弱性が公表される前に攻撃が観測されなかったことや，試行された攻撃に複数のバリエーションが存在することなどが明らかになった．また，ハニーポット以外のネットワークの調査において，103 件の攻撃成功が観測された．ハニーポットへの攻撃成功は脅威になりえないため，リスクの低い観測を行うことができる．

本研究で設計した HTTPS ハニーポットは，TLS ネゴシエーションを各メッセージに分割して記録することにより，将来的な脅威に対しての発見・解析を容易にした．

2.2 ダークネットを用いた攻撃検知

福島らの研究 [12] ではネットワーク上のインシデントの傾向を把握するため，ダークネット宛のトラフィックに注目し，トラフィックの収集を行った．そこで福島らは，観測データの送信元アドレス数と平均送信パケット数に注目し，長期間にわたり少しのパケットしか送信をしない気付かれにくい攻撃の検知手法を提案している．

ダークネット宛パケットを解析する利点として，基本的に不正なパケットのみを収集できること，低コストでかつ広域に渡り観測が可能であることが挙げられる．また，既存のネットワークに影響を与えず，利用者が不利益を被ることがない．このような特徴のため，標的型攻撃を受けやすい大学ネットワークのような自律システムへの攻撃検知にも優れる．

本研究においても，ダークネット宛ての攻撃パケット収集を可能としているが，TLS に特化してパケットを収集している点が異なる．

3. 汎用ハニーポットシステム Poohunt の設計

3.1 Poohunt の概要

Poohunt は TLS/SSL を用いた通信に対して，フルパケット及び上位プロトコルにおける通信を記録することができるハニーポットである．実装には Python2.7 を用い，TLS/SSL ライブラリとして，OpenSSL を用いている．上位のプロトコルを実現するモジュールのインタフェースを定義した．モジュールの実装を変更することにより，HTTPS や POP3s，IMAPs 等に対応可能とした．単体でも，動作するホストの IP アドレスにバインドして設置することが可能だが，オープンソースのハニーポットソフトウェアである Honeyd[13] の仮想ホストに外部スクリプトとしてバインドすることにより，ネットワークレベルでの設置が可能となる．

図 1 に，Poohunt が Honeyd のサービスとして動作する際の内部構成を示す．Honeyd Service は，Poohunt を Honeyd と連携させる際のフロントエンドであり，単体で動作させる際は別のモジュールを用いる．TLS Honeypot は，パケットを Upper Layer Honeypot に送信するのと共に，TLS トラフィックとして記録を行うモジュールであり，独立したスレッドで動作する．Upper Layer Honeypot は，TLS 層上で用いられるプロトコルのためのハニーポットであり，パケットにおいて暗号化されるトラフィックの記録を行う．今回は，上位プロトコルとして HTTPS を実装した．実際の応答は，HTTP Honeypot で表される独立

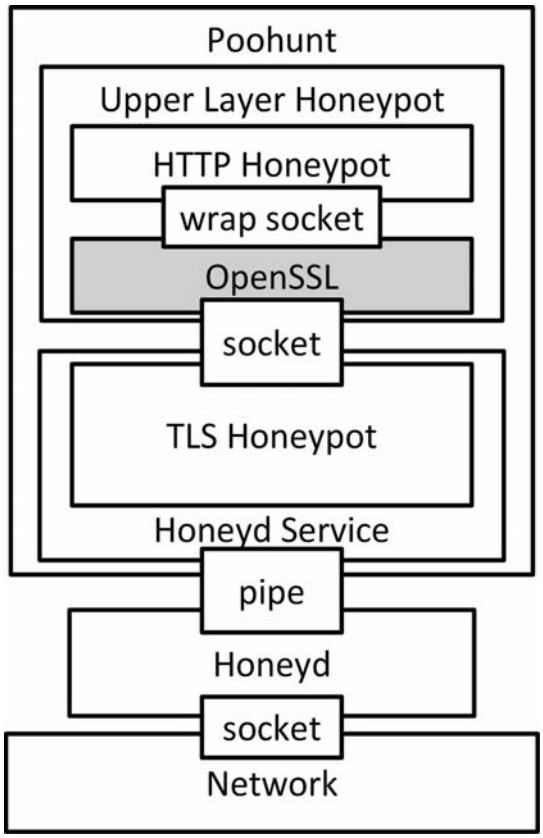

図 1 Poohunt の内部構成

したモジュールが行う．このモジュールを拡張する事により，HTTPS サーバとしてのより複雑な動作や，TLS 上で動作する別プロトコルの実装が可能となる．

Poohunt に求められる要件を示す．

- 大規模なネットワークへの容易な設置が行えること：Honeyd を用いる事により，低コストな設置を可能とする．
- 証明書の管理が行えること：宛先 IP アドレスごとに提示するサーバー証明書を変化させることにより，別個の HTTPS サーバが動作しているかのように見せかけることを可能とする．
- TLS トラフィックの解析を容易とすること：パケットを TLS メッセージごとに分割し，その種類を合わせて MySQL データベースに記録する事によって，TLS プロトコルのフォーマットに合わせた解析を容易とする．

3.2 Honeyd との連携

Honeyd は，Niels Provos らによって開発されたオープンソースのハニーポットである [10]．1 台のサーバ上で，複数のホストをネットワークレベルでエミュレートすることができ，それぞれの IP アドレス，ポート上でサービスが運用されているかのように振る舞う．その際，オペレーティングシステムや NIC のフィンガープリントを偽装することにより，IP スタックレベルでホストの擬態を行う．実際のオペレーティングシステムやソフトウェアを用いないため，再現性に欠けるが比較的安全に運用することができる．

仮想ホスト上のサービスは，Honeyd と同じサーバ上で動作する外部スクリプトを用いるか，プロキシ機能を用いて別ホストへ通信を中継することによって実現する．HTTP サーバや SMTP サーバなどは，そのための外部スクリプトが予め用意されているが，TLS/SSL プロトコルを用いる HTTPS サーバ等の外部スクリプトは存在しない．そのため，HTTPS ハニーポットを実現したい場合，Stunnel[14] などの TLS/SSL トンネリングサービスを使用する必要がある．しかし，この手法を大規模なネットワークに適用すると，TLS/SSL ネゴシエーションの記録やサーバ証明書の管理などが複雑になる．

本研究では，TLS/SSL ネゴシエーションと上位プロトコルの通信との双方を包括的に収集できる汎用ハニーポットを外部スクリプトとして実装し，Poohunt と名付けた．Poohunt は送信元 IP アドレス，送信元ポート番号，宛先 IP アドレスを Honeyd から引数として受け取る．これらの動作は，図 1 における Honeyd Service に相当する部分で行われる．

3.3 TLS に対する動作

3.3.1 サーバ証明書の管理

大規模なネットワークに HTTPS ハニーポットを設置する際，証明書が広域に渡り使いまわされていると，攻撃者に不信感を与える可能性が増加する．そのため，複数の証明書を管理，提示を行う必要が生じる．

Poohunt は，別個の証明書を持つホストが複数存在しているように見せかけるため，宛先 IP アドレスに応じてサーバ証明書を設定を行える様に設計した．また，Honeyd のフィンガープリントを併用することにより，図 2 に示すような環境を構築することが可能となる．これらの動作は，図 1 における Upper Layer Honeypot にて行われる．

3.3.2 セッション情報と TLS トラフィックの記録

通信時刻，セッション情報およびクライアントが送信した TLS 通信のパケットを記録する TLSpot テーブルを表 1 に示す．この動作は，図 1 における TLS Honeypot にて行われる．本研究においては，サーバ側が送信したパケットの記録は行わない．また，クライアント証明書を要求した場合，表 1 に格納されたパケットを解析して取得を行う．

図 3 に，一般的な TLS ハンドシェイクの例を示す．TLS メッセージは，下位プロトコルである TLS レコードプロトコル [15] により TLSPlaintext レコード単位で送受信される．

TLSPlaintext レコードの MSB(Most Significant Byte) は，自身が格納する上位プロトコルの種類を表す．RFC 5246 として制定された TLS1.2 における，上位プロトコルを表 2 に示す．この他に，TLS 拡張である Heartbeat(0x18)

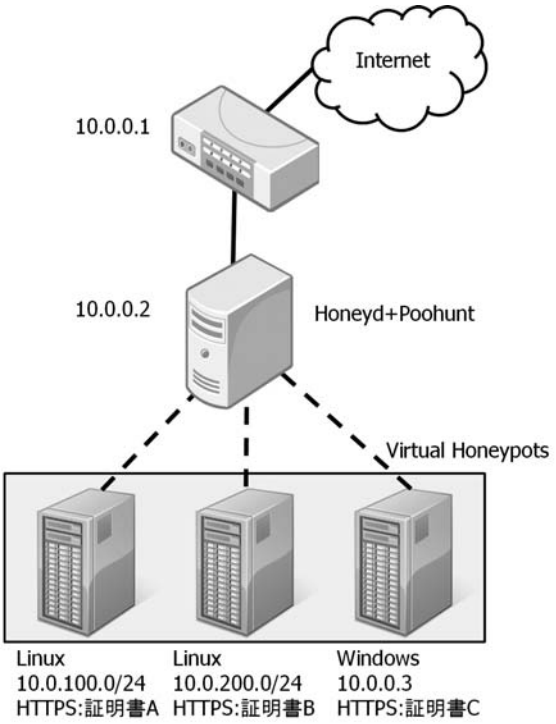

図 2 証明書の提示例

表 1 セッション情報と TLS トラフィックのテーブル

TLSpot テーブル		
カラム	データ型	意味
ID	int unsigned	id 番号
DATE_TIME	datetime	接続日時
D_IP	int unsigned	宛先 IP アドレス
S_IP	int unsigned	接続元 IP アドレス
S_PORT	int unsigned	接続元ポート
NUMBER	int unsigned	メッセージの順番
TYPE	binary(1)	メッセージの種類
RAW	varbinary(4096)	TLSPlaintext レコード

表 2 TLS 内部プロトコルの種類

プロトコル名	バイト
Change Cipher Spec Protocol	0x14
Alert Protocol	0x15
Handshake Protocol	0x16
Application Data Protocol	0x17

などが存在する．Poohunt は，この上位プロトコルの種類を TYPE カラムに格納することにより，上位プロトコルごとの解析を容易とした．

複数の TLSPlaintext レコードが連続で送信される場合，パケットがまとめられてしまう．図 3 を例とすると，Client 側から送られる TLSPlaintext レコードにおいては，Client Key Exchang,Change Cipher Spec,Finished が該当する．Poohunt はこれらを TLSPlaintext レコードごとに分割し，別々のデータベースレコードに格納する．パケットの分割は，TLSPlaintext レコードが持つ length フィールドを元

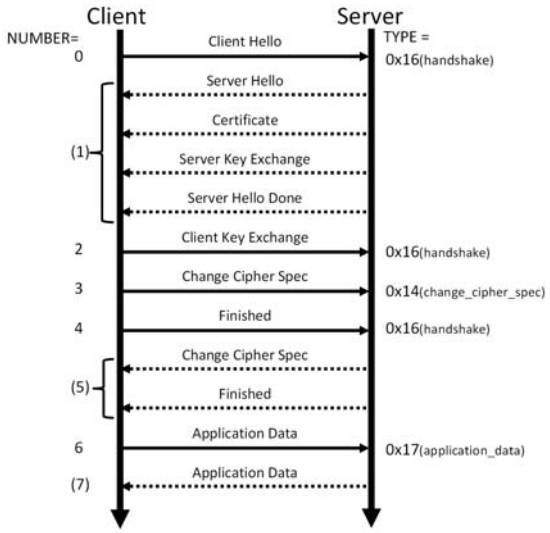

図 3 TLS ハンドシェイクの例

```
from abc import ABCMeta, abstractmethod
class UpperHoneypotBase :
        @abstractmethod
        def honeypot(self, sock): pass
```

図 4 上位層ハニーポットの抽象クラス

に行われる．パケットサイズが length に満たない場合，もしくは余剰パケットが生じた場合はそのまま 1 つのレコードに格納する．この場合，TLS プロトコルの仕様に沿わない不正な通信であると判断できる．

それぞれの TLSPlaintext レコードの順番は，NUMBER カラムに格納される．途中にサーバ側の送信が発生した場合 NUMBER は 1 つスキップされ，クライアント側の送信が区切りられたことの記録となる．このカラムによって，該当コネクションにおいて TLS ハンドシェイクがどの段階まで進行したのかを確認することが可能となる．

3.4 HTTPS に対する動作

3.4.1 HTTP ハニーポット

HTTP などの上位層のプロトコルから見て TLS は透過的であるため，実装の分離が可能である．また，モジュール化は拡張性の観点からも望ましい．

Poohunt は，HTTP ハニーポットの実装を独立したモジュールで行う設計とした．図 4 に示す抽象クラスを継承したクラスでの実装を行う．ハニーポットとして Poohunt が呼び出すメソッドは，def honeypot(self, sock) :である．このメソッドはソケットを引数として受け取り，記録を行うメッセージを戻り値として返す．実装したクラスは，Poohunt の設定ファイルにおける UPPER_SCRIPT に参照を追加することで動作する．このクラスは，図 1 における HTTP Honeypot である．

表 3　暗号方式と HTTP リクエストのテーブル

HTTPpot テーブル		
カラム	データ型	意味
ID	int unsigned	id 番号
CIPHER	char	通信路の暗号方式
BIT	smallint	鍵サイズ
VERSION	char	TLS プロトコルバージョン
MSG	varbinary(4096)	リクエスト

3.4.2　通信路の暗号方式と HTTPS 通信内容の記録

通信路の暗号方式と TLS プロトコルのバージョン，リクエストを記録する HTTPpot テーブルを表 3 に示す．このレコードは TLS ネゴシエーションが成功した場合にのみ保存される．MSG には，4 の honeypot メソッドの戻り値が格納される．HTTPpot テーブルは ID によって，TLSpot テーブルの該当するレコードに紐付けられる．紐付けられた TLSpot のテーブルにおける RAW カラムは，暗号化された HTTPpot の MSG カラムとなる．

TLS ハンドシェイクが行われると，通信路の暗号方式及び，付随して鍵サイズと TLS プロトコルのバージョンが確定する．それらは，TLS/SSL ライブラリから取得され，CIPHER，BIT，VERSION に格納される．

これらの動作は，図 1 における Upper Layer Honeypot にて行われる．

3.5　ブラックリスト

Poohunt の動作テストにおいて，大学や研究機関が定期的にスキャンを行っていることが判明した．このような研究用のスキャンにおいて，本研究のハニーポットの存在はノイズとなってしまうことが考えられる．また，比較的大規模なスキャンが多いため，仮想ホストを接続のたびに生成する Honeyd サーバの負荷となる可能性がある．

該当するトラフィックの記録を行わないため，IP アドレス及びネットワークのブラックリストを作成し，それに含まれる送信元の場合 Poohunt が接続を拒否することとした．Poohunt の設定ファイルにおける BLACKLIST に [”192.168.100.0/24”, ”169.254.100.0/24”, ”172.132.30.120”] のような List 形式で設定を行う．該当する送信元からの接続要求の場合，日時，送信元 IP アドレス，宛先 IP アドレスがログに記録され即座に接続がクローズする．これらの動作は，図 1 における Honeyd Service に相当する部分で行われる．

4.　実験

設計した汎用ハニーポットシステムの有効性を検証するために，筑波大学内のダークネットに設置をして実験を行った．

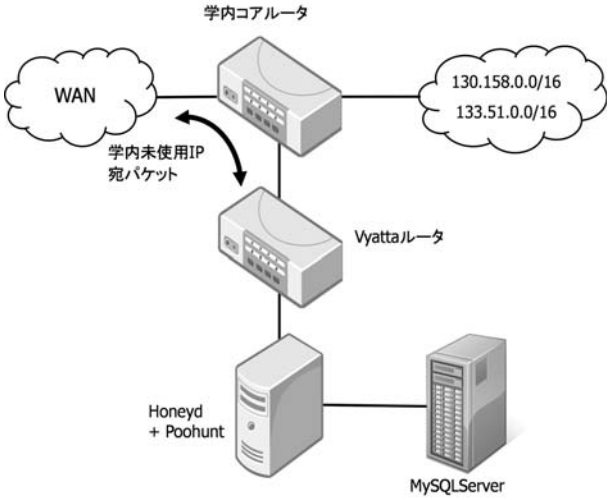

図 5　HTTPS トラフィック収集システムのネットワーク構成

4.1　ネットワーク構成

筑波大学は 2 つのクラス B ネットワーク（130.158.0.0/16 および 133.51.0.0/16）を有し，さらに小さいセグメントに分けて利用している．そのため，利用されていないセグメントが存在し，未使用セグメント宛のパケットはルータが破棄していた．本実験ではそれらのパケットをハニーポットにルーティングし，攻撃情報を収集した．

収集システム全体のネットワーク構成を図 5 に示す．これは，先行研究である佐藤らの研究 [9] を参考としたものである．ダークネット宛のパケットはすべて，学内コアルータによってソフトウェアルータである Vyatta Core に振り分けられ，ポリシーベースルーティングが行われる．以下に Vyatta のポリシーを優先度が高い順に示す．

ポリシー 1　ハニーポットからの大学ネットワーク内宛パケット：破棄する
ポリシー 2　プライベート IP アドレス宛のパケット：破棄する
ポリシー 3　大学ネットワーク内からの学内未使用 IP アドレス宛パケット：破棄する
ポリシー 4　学内未使用 IP アドレス宛てパケット：ハニーポットにルーティング

ポリシー 1 によって，仮にハニーポットサーバへの攻撃が成功した場合に，そのサーバが学内ネットワークに攻撃を加えることを阻止する．また，ポリシー 3 により大学ネットワーク利用者が誤ってハニーポットにアクセスすることを防ぐ．

4.2　Honeyd の設定

本実験では，ポート 443 宛の通信のみを Poohunt に中継する．また，OS としては Linux 2.6.35 を，NIC(Network Interface Card) としては 3Com を擬態するように設定した．

4.3 Poohunt の設定

4.3.1 Poohunt 設定ファイル

本実験は，サーバ証明書の差異による通信の変化の観測が目的ではないため，2 つのクラス B ネットワーク (130.158.0.0/16 および 133.51.0.0/16) それぞれに 1 つづつ，計 2 つのみの自己署名証明書を設定した．クライアント証明書の要求は行わなかった．

予備実験において，ミシガン大学の 2 つのクラス C ネットワークによる研究用のスキャンが，トラフィック全体の中で大きな割合を占めることが判明した．それ故に，ミシガン大学のスキャン [16](141.212.121.0/24 および 141.212.122.0/24) をブラックリストに登録した．その他の IP アドレス，ネットワークのブラックリストへの登録は行っていない．

4.3.2 HTTP ハニーポットの実装

Poohunt を HTTPS サーバとして動作させるため, HTTP リクエストの記録を行った上で簡易的に解析し，応答する機能を持つ HTTP ハニーポットの実装を行った．

TLS メッセージの収集を主眼としているため，HTTPS サーバとしての動作は最低限のもののみの実装である．HTTP リクエストと応答の組み合わせを以下に示す．

index ファイル及び/への GET メソッド	403 エラー
その他への GET メソッド	404 エラー
それ以外のリクエスト	400 エラー

5. 収集結果の解析

5.1 収集したデータの概要

収集した期間は，2014 年 11/19-12/18 に渡る 1 ヶ月間である．なお，本研究では収集システムを収集期間の前日にあたる 11/18 に稼働させた．システム稼働以前に未使用 IP アドレス上で応答するサーバは存在していない．

以下に収集データの概要を示す．なお，設置をした筑波大学のネットワーク 130.158.0.0/16 を Network A，133.51.0.0/16 を Network B と表記する．

- TLSpot の総レコード：15,005,475 レコード
- TLS アクセス：5,100,656 件 (Network A - 1,901,516 件，Network B - 3,199,140 件)
- TLS セッション確立： 1,821,601 件 (Network A - 712,805 件，Network B - 1,108,796 件)
- TCP コネクションドロップ：19,711,126 件
- ブラックリストによるクローズ：1,052,156 件
- オリジナルホスト数：1,890
- オリジナルサーバ数： 60,541 (Network A - 21,309, Network B - 39,232)

TLS アクセスとは，TLS 層でホストがパケットを送信した接続である．Poohunt の機能であるブラックリストによって拒否された接続は含まれていない．また，TLS セッション確立とは，TLS ハンドシェイクが正常に終了した接続である．このことから，自己署名証明書を用いたにも関わらず，約 36%の接続が証明書を認証したことが明らかとなった．

TCP コネクションドロップとは，TLS 層に達することなくドロップされた接続であり，主にポートスキャンであると考えられる．この接続数は，TLS 層に達した接続数と比較して約 3.2 倍と非常に多い．ブラックリストによるクローズとは，Poohunt の機能であるブラックリストにより拒否された接続の数である．全 TLS コネクション要求の内，約 17%がブラックリストからの接続要求であった．

オリジナルホスト数とは，期間内に TLS アクセスを行ったホストの数である．また，オリジナルサーバ数とは，Poohunt がエミュレートしたホストの数である．

図 6 に，日毎のアクセス数，セッション確立数ならびにそれぞれのホスト数をグラフとして示す．アクセス数とは，TLS アクセス数を示す．また，セッション確立数とは TLS セッション確立数を示す．日毎のアクセスホスト数の変動と比較し，セッション確立ホスト数は安定していることが見て取れる．なお，アクセスホスト数の標準偏差は 112.43，セッション確立ホスト数の標準偏差は 24.78 である．

表 4 TLS プロトコルの TYPE

TYPE	レコード数	ホスト数	サーバ数
0x16(Handshake)	8,973,369	841	60,541
0x14(Change Cipher Spec)	2,283,200	271	60,541
0x17(Application)	2,028,384	110	60,537
0x15(Alert)	517,670	122	60,541
0x18(Heartbeat)	345,441	20	60,533
TLS に則さない Type	857,411	1,170	60,540
計	15,005,475	1,890	60,541

表 5 Heartbeat の内訳

Type Heartbeat の内訳	レコード数	ホスト数	サーバ数
平文による Heartbleed 攻撃	329,843	4	60,532
暗号化された Heartbeat	15,133	15	13,486
Heartbeat に則さないバイト列	465	1	453
計	345,441	20	60,533

5.2 TLS メッセージの種類による解析

Poohunt は，TLS トラフィックの種類ごとに解析を行えるように，TYPE カラムがデータベースに用意されている．表 4 に，収集した TLS トラフィックの TYPE カラムごとの数を示す．

ここでは，0x18(Heartbeat) の解析を行う．

TYPE=0x18 として記録された 345,331 件のレコードの解析結果を表 5 に示す．平文で送信された Heartbeat のすべてが，Heartbleed 脆弱性を用いた攻撃であり，Client Hello と Server Hello のやりとりの直後に行われた．暗号化された Heartbeat は，暗号化された通信路での最初の通信で行われた．通常この段階で Heartbeat は行わないた

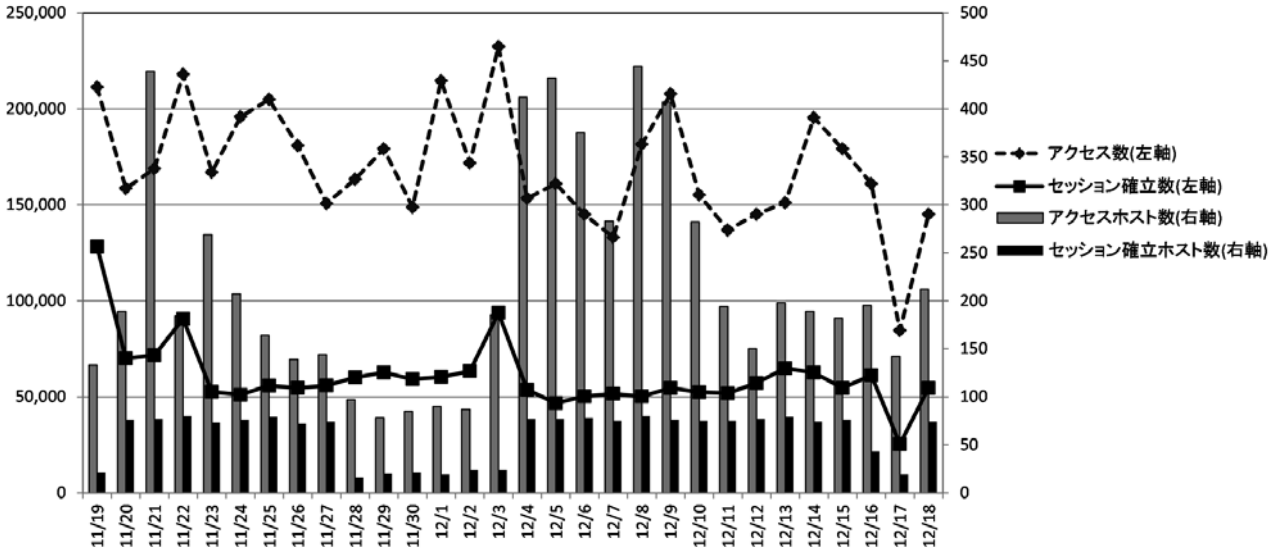

図 6 日毎の TLS アクセスの推移

表 7 HTTP リクエスト行の分類

HTTP リクエスト行	件数	割合
GET / HTTP/1.1	1,637,474	89.89%
GET /epgrec/gen-thumbnail.sh HTTP/1.1	75,036	4.12%
POST /cgi-bin/php で始まるリクエスト	27,759	1.52%
その他の HTTP メソッド	3,298	0.18%
空白	24,510	1.35%
HTTP に沿っていないリクエスト	53,524	2.94%
計	1,821,601	100%

め，Heartbleed 脆弱性を用いた攻撃である可能性が高い．

5.3 通信路の暗号化形式による解析

TLS コネクションが成立すると，Poohunt は通信路の暗号方式，鍵サイズおよび TLS プロトコルのバージョンを，HTTPpot に格納する．暗号化形式ごとのコネクション数，ホスト数，サーバ数を表 6 に示す．

ホストには複数の暗号化形式で合意したものが複数存在した．これらは Client Hello において，複数の暗号スイートリストを用いたホストが存在することを示す．

5.4 HTTP リクエストの解析

Poohunt が受け取った HTTP リクエストの内容について解析を行った．

収集した HTTP リクエストから，1 行目であるリクエスト行を以下のように分類した．

- GET / HTTP/1.1
- GET /epgrec/gen-thumbnail.sh HTTP/1.1
- POST /cgi-bin/php で始まるリクエスト
- その他の HTTP メソッド (GET，OPTIONS，HEAD)
- HTTP に沿っていないリクエスト
- 空白

表 7 に，それぞれの件数を示す．

表 8 ネットワークごとの比率

項目	Network A	Network B	A : B の比率
オリジナルサーバ数	21,309	39,232	3.52 : 6.48
TLS アクセス	1,901,516 件	3,199,140 件	3.73 : 6.27
TLS セッション確立	712,805 件	1,108,796 件	3.91 : 6.09

5.5 考察

実験結果から，Poohunt の有効性を考察する．Poohunt は，1 台のホストで 60,541 の IP アドレスに HTTPS ハニーポットとしての動作を行った．また，ネットワークに Poohunt を設置する際に必要な設定は，Honeyd および Poohunt の設定ファイルのみで完結しており，NIC などに特別な設定を必要としない．これらの事実より，Poohunt は大規模なネットワークへの容易な設置が可能である．

Poohunt を設置したネットワークごとの，サーバ数およびアクセスに関する比率を表 8 に示す．TLS アクセス数，TLS セッション確立数ともにオリジナルサーバ数に準ずる比率であった．本実験では，それぞれのネットワークに 1 つづつサーバ証明書を設定している．TLS アクセスは，サーバ証明書提示以前の状態であり，TLS セッション確立はサーバ証明書後の状態である．これらの事実から，Poohunt が適切なサーバ証明書管理を行っているものと思われる．

6. おわりに

本研究では，TLS/SSL プロトコルを対象とした汎用ハニーポットシステムである Poohunt の設計，実装を行なった．TLS/SSL の上位プロトコルとしては HTTPS を対象として実装を行った．また，筑波大学のネットワークにおけるダークネットに Poohunt を設置することによって，HTTPS トラフィックの収集を行い収集した情報を元に解析を行なった．

表 6　通信路の暗号化形式

暗号化形式	鍵長 (bit)	TLS バージョン	コネクション数	ホスト数	サーバ数
RC4-SHA	128	TLSv1/SSLv3	1,499,059	88	60,533
AES128-SHA	128	TLSv1/SSLv3	240,784	60	58,913
AES256-GCM-SHA384	256	TLSv1/SSLv3	44,442	22	36,641
AES256-SHA	256	TLSv1/SSLv3	37,316	13	19,305

本研究で設計，実装した Poohunt は TLS/SSL を用いた通信に対して，フルパケット及び HTTP リクエストを記録することができる HTTPS ハニーポットであり，オープンソースのハニーポットソフトウェアである Honeyd と連携することにより，ネットワークレベルでの設置が可能である．また，サーバ証明書を偽装する IP アドレスごとに切り替えることにより，別個の証明書を持つ HTTPS サーバが複数動作しているかのように見せかけるよう設計されている．TLS トラフィックは解析が容易となるよう分割され，MySQL データベースに保存される．その他に，HTTP サーバとしての動作を行うモジュールを独立させたことにより，拡張性を高めたことなどについて述べた．

また，Poohunt の動作，有用性の実証実験を行うための実験環境とその結果の解析について述べた．本研究では，筑波大学のダークネットに Poohunt の設置を行なった．Poohunt は，60,541 台のホストのエミュレートを行なった．また，自己署名証明書を提示したにも関わらず全アクセスのうち約 36%のホストが TLS セッションを確立させたことが明らかになった．Heartbeat メッセージにおいて，ほぼすべてのメッセージが脆弱性を狙ったものであることを示した．

本研究の今後の課題として，収集したデータのより細かい解析，自己署名証明書と認証局による署名付き証明書による違いの解析，HTTPS 以外のプロトコルに関するハニーポットの実装および収集などが挙げられる．

参考文献

[1] Let's Encrypt. https://letsencrypt.org/, accessed: 2014/12/11.
[2] CODENOMICON. Heartbleed Bug. http://heartbleed.com/, accessed: 2014/10/16.
[3] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicholas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, Vern Paxson. The Matter of Heartbleed. In *ACM Internet Measurement Conference (IMC)*, Nov 2014.
[4] Google Technical Note - TLS Next Protocol Negotiation. https://technotes.googlecode.com/git/nextprotoneg.html, accessed: 2014/12/11.
[5] RFC 7301 - Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension. http://tools.ietf.org/html/rfc7301, accessed: 2014/12/11.
[6] HTTP/2 Frequently Asked Questions. http://http2.github.io/faq/, accessed: 2014/12/11.
[7] Networking/http2 - MozillaWiki. https://wiki.mozilla.org/Networking/http2, accessed: 2014/12/11.
[8] 佐藤聡, 三田尚貴, 新城靖, 板野肯三. ハニーポットを利用した筑波大学の未使用 IP アドレス宛ての HTTP リクエストの解析. 情報処理学会研究報告. IOT, インターネットと運用技術, Vol.2013-IOT-23, No8, pp1-6, Sep 2013.
[9] 佐藤聡, 小川智也, 新城靖, 吉田健一. 筑波大学におけるハニーポットを用いた不適切な SSH アクセスの収集とその解析. 情報処理学会研究報告. IOT, インターネットと運用技術, Vol.2014-IOT-25, No17, pp1-6, May 2014.
[10] N. Provos. Honeyd- A Virtual Honeypot Daemon. In 10th DFN-CERT Workshop, Hamburg, Germany, Feb 2003.
[11] Amazon Web Services (AWS) - Cloud Computing Services. http://aws.amazon.com/, accessed: 2014/12/17.
[12] 福島祥郎, 堀良彰, 桜井幸一. ダークネット観測データに基づく攻撃挙動の特徴抽出に関する考察. 電子情報通信学会技術研究報告, ICSS, 情報通信システムセキュリティ, Vol.109, No.185, pp.37-42, Nov 2009.
[13] Developments of the Honeyd Virtual Honeypot. http://www.honeyd.org/, accessed: 2014/10/16.
[14] stunnel: Home. https://www.stunnel.org/index.html, accessed: 2014/12/11.
[15] Tim Dierks, Eric Rescorla. RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2. http://tools.ietf.org/html/rfc5246, accessed: 2014/12/15.
[16] Zakir Durumeric, James Kasten, Michael Bailey, J. Alex Halderman. In *ACM Internet Measurement Conference (IMC)*, 2013.
[17] Transport Layer Security (TLS) Parameters. http://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml, accessed: 2015/1/9.
[18] Qualys SSL Labs - Projects / SSL/TLS Deployment Best Practices. https://www.ssllabs.com/projects/best-practices/index.html, accessed: 2015/1/9.
[19] CRYPTREC, 2011 年度版　リストガイド (SSL/TLS). http://www.cryptrec.go.jp/report/c11_guide2011_TLS-f3.pdf, accessed: 2015/1/9.
[20] Security/Server Side TLS. https://wiki.mozilla.org/Security/Server_Side_TLS, accessed: 2015/1/9.
[21] AlFardan, Nadhem J. Bernstein, Daniel J. Paterson, Kenneth G. Poettering, Bertram Schuldt, Jacob C. N. . On the Security of RC4 in TLS and WPA. http://www.isg.rhul.ac.uk/tls/RC4biases.pdf, accessed: 2015/1/17.
[22] Shadowserver Foundation - Main - HomePage. https://www.shadowserver.org/wiki, accessed: 2015/1/17.
[23] Shodan. https://www.shodan.io/, accessed: 2015/1/17.