OS Fingerprint 対策手法の実装と評価

宮本 大輔　　大江 将史　　木村 泰司　　門林 雄基
奈良先端科学技術大学院大学 情報科学研究科
E-mail: daisu-mi@is.aist-nara.ac.jp

平成 13 年 9 月 5 日

概 要

TCP/IP の規格は RFC によって定義されているが，各 OS における TCP/IP スタックの実装には差異がある．この差異は OS Fingerprint と呼ばれる．インターネットを通じて OS Fingerprint を収集し，分析することにより，悪意ある第三者はホストの OS を特定することができ，OS 特有の脆弱性を知ることができる．本研究では，外部からの OS 検出に対して，ホストで稼働している本来の OS とは異なる OS Fingerprint を返答する機構を提案する．この機構により， OS 検出ツールに誤検出を促すことができ，OS の検出行為を無力化し，ひいてはシステムの安全性を高めることができると考えられる．

1　はじめに

1.1　本研究の目的

近年，インターネットに接続されたホストの急激な増加に伴い，ホストにおけるシステムの脆弱性が指摘されるようになった．システムの脆弱性はホストの OS に密接に関係している．これは古い OS がバージョンアップされる時のリリースノート等にセキュリティに関する問題点が修正されていることで示されているように，古い OS は常に修正すべき問題を抱えているとも考えられる．すなわちホストの OS の種類やバージョンに関する情報を得ることができれば，その情報によってシステムの脆弱性が露出する可能性がある．

悪意ある第三者は，インターネットを通じて TCP/IP スタックの挙動を観察することで，ホストの OS を特定することができる．これは，TCP/IP の規格は RFC によって定義されているにも関わらず，各 OS 毎における実装には差異があるからである．この差異は OS Fingerprint と呼ばれ，攻撃者はこの 各 OS 毎に存在する Fingerprint を分析することにより，外部からホストの OS を特定することができる．このような OS の検出行為により，悪意ある第三者はシステムの脆弱性を露出させ，攻撃に役立てることができる．なお OS の検出行為には OS を特定するための様々な検査項目があり，この検査項目において OS の検出を行う個々の行為を Probe と呼ぶ．

本研究は外部から行われる Probe に対し，本来稼働している OS とは別の OS Fingerprint を偽装することにより， OS の誤検出を促すことを目的とする．その結果，OS の検出は困難になり，攻撃にかかるコストは増加するため，攻撃に必要な時間も増加する．また IDS(Intrusion Detection System:侵入検知システム）等々の問題点として，一つのトラフィックが攻撃かどうかを識別するまで時間がかかることが挙げられる．攻撃に必要な時間の増加は，攻撃行為を発見する可能性を向上させることとなり，ひいてはシステムの安全性を高めることができると考えられる．

さらに OS Fingerprint を偽装した場合は，偽装した OS に合わせた攻撃が行われる可能性が高くなる．しかし本来稼働している OS は，基本的に，偽装した OS を目標とする攻撃に対して安全であるため，この攻撃行為をデータとして収集し，攻撃者の行動分析に役立てることも可能である．

以下，まず 2 節において OS の検出を行う手法について述べ，3 節において OS Fingerprint 対策手法の設計と実装を説明する．4 節では 3 節で述べた手法の評価を行い，最後に 5 節でまとめと今後の課題について述べる．

1.2 関連研究

本研究は Deception と呼ばれる研究領域に分類される．Deception とは，悪意ある第三者の攻撃を Firewall 等で遮断する従来の技術ではなく，攻撃に対して本来返答すべき情報とは異なるいわば「偽の情報」を返答する技術である．これより，攻撃者にホストのシステムを錯覚させ，システムの脆弱性の見当を付けさせないことで，システムの安全性を高めることができると考えられる．

関連する研究としては dtk [2] や fpf [3] 等がある．dtk は各 OS を特徴付けるようなアプリケーションを偽装することにより，本来稼働している OS とは別の OS であるように攻撃者に判定させるプログラムである．これは後述する Application Banner 型の OS 検出に対して誤検出を促す．これに対し，本研究は OS Fingerprint 型の OS 検出に対して誤検出を促す．本研究で提案する手法と dtk は互いに干渉せずに動作可能であり，これらを組み合わせて用いることにより，攻撃者の錯覚を引き起こす可能性をより高められると考えられる．

また fpf は本研究と同じく OS Fingerprint の偽装を行うことができる．しかし fpf では TCP Initial Sequence Number や IP Identification の値の増加傾向等を含めた OS Probe 全体に対する対策はなされていない．このため OS 検出ツールによる検出を失敗させることはできても，本来稼働している OS を別の OS に偽装することはできない．

本研究は Honeypot と呼ばれる，攻撃行為をデータとして収集する環境の構築も視野に入れている．本研究で提案した手法を応用すれば，本来稼働している OS をシステムの脆弱性が高い OS であるかのように誤検出を促すこともでき，攻撃者に脆弱性が高いシステムを運用しているように錯覚させ，攻撃を誘発することも可能であると考えられる．ここで，本来稼働している OS は偽装した OS に合わせた攻撃に対して基本的に安全であるため，Honeypot として攻撃行為を収集し，分析する環境を構築する際に役立てることが可能である．

2 OS の検出

2.1 OS 検出の手法

OS 検出の手法には，Application Banner を用いる手法と OS Fingerprint を用いる手法がある．前者は OS に関する情報を通知するアプリケーションを利用して Probe を行う．具体的には TELNET や HTTP の HEAD リクエスト，FTP の SYST リクエスト等で OS を特定することができる．図 1 は FreeBSD [4] において telnet サービスを行う Application Banner であるが，この図 1 の下線部を注目してやれば，この OS を FreeBSD であると特定できる．

```
> telnet 172.16.236.2 telnet
Trying 172.16.236.2...
Connected to 172.16.236.2.
Escape character is '^]'.

FreeBSD/i386 (anyhost.anywhere) (ttyp1)

login:
```

図 1: FreeBSD において telnet サービスを行うプログラムの Application Banner

後者は OS Fingerprint を収集し，分析することで OS の特定を行う．OS Fingerprint の収集手法には，ホストに対して特殊なパケットを送信し，その反応を調べる能動型の手法と，インターネット上でモニタリングしたパケットから OS 毎の特徴を見い出す受動型の手法がある．本研究では Fingerprint 収集・分析ツールとして，検出可能な OS の数が最も多い nmap [10] を用いて実装の評価を行った．nmap は能動型のツールであり，TCP スタックと IP スタックの双方に対して以下に示す Probe を行う．

TCP スタックに関する Probe

TCP FIN Probe　RFC 793 [11] では FIN フラグのセットされたセグメントを TCP コネクションの確立前に受信した場合は挙動を無視するように定義しているが，一部の OS は RST フラグのセットされたセグメントを返す．nmap は FIN フラグのセットされたセグメントを送信し，挙動を検査する．

TCP NULL Probe　TCP コネクションの確立前に TCP フラグが何もセットされていないセグメントを送信し，挙動を検査する．

TCP Xmas Probe　TCP コネクションの確立前に，通常使われないような組み合わせのフラグをセットしたセグメントを送信し，挙動を検査する．nmap は SYN，FIN，URG，PSH フラグのセットされたセグメントを送信する．

Unused Flag Probe　RFC 793 では Reserved と定義されているため使われていないフラグが TCP ヘッダにある．このフラグをセットしてセグメントを送信し，挙

動を検査する．このフラグは RFC 2481 [12] で提案された ECN という輻輳制御機構で利用されることになっており，この Probe は事実上 OS が ECN をサポートしているかどうかを検査する．

TCP Initial Sequence Number Sampling TCP コネクションの確立時に指定される Sequnce Number である Initial Sequence Number の値を収集し，その増加傾向を検査する．

TCP Initial Window Size TCP コネクションの確立時に指定される Window Size である Initial Window Size の値を検査する．

TCP Option RFC 1323 [6] 等に定義されている Timestamp Option や Window Scale Option 等の TCP Option を使用するのか，もし使用する場合はどの TCP Option をどの順序で使用するのかを検査する．

IP（IPv4）スタックに関する Probe

IP Don't Fragment Bit IP Don't Fragment Bit がセットされているかどうかを検査することにより，検出対象が Path MTU Discovery [7] を行っているかどうかを検査する．Path MTU Discovery は，IP Fragmentation にかかるコストを考慮し，End-End 間の最適な MTU を探索するための機構である．

IP Identification Number IP のフラグメント処理を行う際に各データグラムを識別するために用いられる IP Identification Number の増加傾向を検査する．

IP Type of Service IP Type of Service フィールドを利用しているか，利用している場合はその値を検査する．

ICMP Error Message Quenching 1 秒（単位秒）あたりに ICMP Error Message を送信可能な数を検査する．

ICMP Error Message Quoting ICMP Errro Message に引用された，送信したデータグラムの部分の大きさを検査する．

ICMP Error Message Echoing Integrity ICMP Error Message に引用された部分と，送信したデータグラムの差異を検査する．

nmap はこのような挙動に関する検査結果を OS 毎にまとめデータベースとして保持しており，ホストに対して行った Probe の結果と照合して OS を特定することができる．

2.2 OS 検出に対する現状

Application Banner 型の検出に対しては，各アプリケーションでシステムに関する情報を通知しないように対応したり，カーネルの持つ OS の名前やバージョン等の情報をその OS のソースファイルで変更することにより，比較的容易に OS に関する情報の流出を防いだり，他の OS に偽装することができる．

OS Fingerprint 型の検出に対しては，OS の検出ツールでしか用いられないような特定のパケットを破棄する機構が一部の OS で実装されている．しかしながらこれは個々の Probe に対する個別の対策でしかなく，Probe 全体に対する対策はなされていない．

3 OS Fingerprint 対策手法の設計と実装

3.1 OS Fingerprint 対策手法

本研究では，ホストの OS が特定可能な場合において，悪意ある第三者はこのホストに対して何らかの被害を与えられ得ると思われる攻撃を選別することにより，ホストに対して攻撃を効率よく行うことができると考える．

そこで OS Fingerprint を偽装し，本来稼働している OS とは別の OS Fingerprint を攻撃者に応答することにより，悪意ある第三者に OS の特定を誤らせ，攻撃にかかるコストを増加させる．これより，攻撃に必要な時間を増加させ，ひいてはシステムの安全性を高められると考える．

3.2 設計

本研究では TCP/IP スタックにおけるネットワークへの入出力を処理する部分を改造することにより，本来稼働している OS とは別の OS が持つ Fingerprint へ偽装する．以下，図 2 を用いて説明する．

設計上留意した点は，偽装する対象 OS をユーザ空間から選択することを可能にした点である．これにより OS Fingerprint をシステムの稼働中に選択でき，OS の特定をより困難にすることができると考えられる．

偽装の対象となる OS は，その種類をユーザ空間より sysctl システムコールを用いて設定し，その値をカーネル空間に格納する．カーネルは格納した値を保持し，

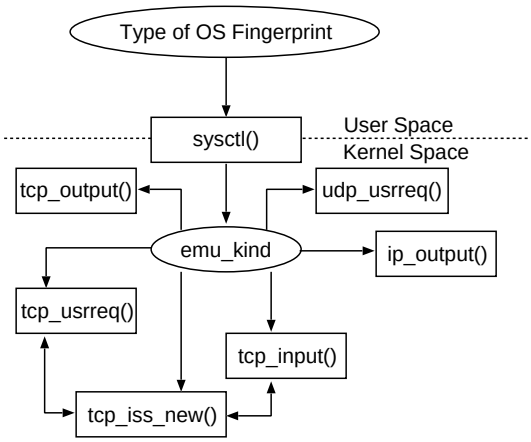

図 2: OS Fingerprint 対策手法の設計

表 1: プログラムを追加したファイル群

tcp_seq.h	tcp_seq.h
tcp_input.c	tcp_output.c
tcp_subr.c	tcp_usrreq.c
ip_output.c	udp_usrreq.c

TCP/IP におけるネットワークへの入出力を行うモジュール，TCP/IP の各パラメータを生成・変更するモジュール等から参照することで，各モジュールにおいて OS Fingerprint を偽装する．

TCP Initial Sequence Number の生成に関しては，他の OS の生成アルゴリズムを移植した．ソースファイルが公開されていない OS に対しては，観測によって得られた生成アルゴリズムを新たに実装した．これら TCP Initial Sequence Number の生成に関するモジュールは，新たに `tcp_iss_new()` 関数を作成し，`tcp_input` 関数等から必要に応じて呼び出されるように設計した．

3.3 実装

実装は FreeBSD 4.3-RELEASE（以下単に FreeBSD と記す）上で行い，開発には C 言語を用いた．開発の対象は FreeBSD カーネルであり，開発したプログラムは表 1 に示す `/usr/src/sys/netinet/` ディレクトリにおけるファイル群に追加した．

この実装では FreeBSD を NetBSD 1.4-RELEASE [9]（以下単に NetBSD と記す）や Windows 95/98/NT4.0（以下単に Windows と記す）として偽装することが可能である．

以下に FreeBSD を NetBSD や Windows に偽装するために必要な Fingerprint の変更点を示す．

IP Don't Fragment Bit

デフォルトの状態の FreeBSD や Windows では Path MTU Discovery を行うため IP Don't Fragment Bit をセットする．NetBSD では Path MTU Discovery を行わないため IP Don't Fragment Bit をセットしない．

TCP Null Probe に対する挙動

デフォルトの状態の FreeBSD は TCP Null Probe を無視するが，NetBSD や Windows ではこれに無視せず，応答する．

TCP Option

デフォルトの状態の FreeBSD や Windows は RFC 1323 で定義された Window Scale Option や Timestamp Option を使用しないのに対し，NetBSD ではこれらの Option を使用する．

TCP Xmas Probe に対する挙動

FreeBSD や NetBSD では，閉じたポートに対して行われた TCP Xmas Probe に対し，応答して送出するセグメントの ACK 番号に Probe で用いられたセグメントのシーケンス番号と等しい値を使用する．Windows では Probe で用いられたセグメントのシーケンス番号に 1 加えた値を ACK 番号に使用する．

TCP Initial Window Size

デフォルトの状態の FreeBSD における Initial Window Size は `0x402e` であるのに対し，NetBSD では `0x4000` であり，Windows では `0x2017` である．

ICMP Error Message Echoing Integrity

FreeBSD では UDP データグラムが閉じたポートに対して到着した場合，このデータグラムを引用する際，UDP チェックサムを 0 に置き換えて返信する．NetBSD や Windows は送信した UDP データグラムのチェックサムを正しく引用して返信する．

TCP Initial Sequnce Number Sampling

デフォルトの状態の FreeBSD における Initial Sequence Number は，ランダムに生成した値が使用されるのに対し，NetBSD では前回用いた値に乱数を加えた値が使用される．

Windows における TCP Initial Sequence Number の生成アルゴリズムは一般には公開されていないが，一定の時間毎に微少な値だけ増加していることが知られている [5]．なお TCP の Initial Sequence Number の予測ができれば，TCP のコネクションを乗っ取ることが可能で

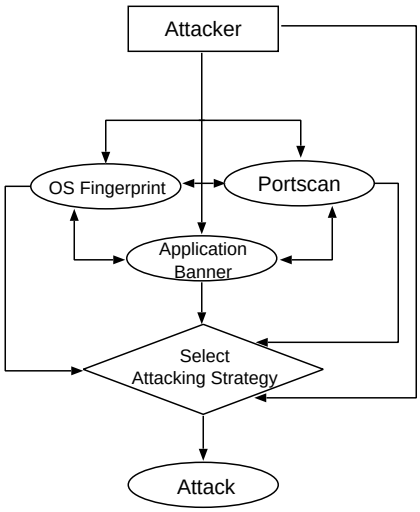

図 3: 調査により得られた，攻撃における戦略決定

あるため，[1] [8] RFC793 には Initial Sequence Number を, 少なくとも 4 マイクロ秒毎に変更するよう定められている．

IP Identification Number Sampling

FreeBSD や NetBSD では，IP Identification Number の値に，前回用いられた IP Identification Number に `0x0001` を加えた値を使用する．Windows では前回用いられた IP Identification Number に `0x0100` を加えた値を使用する．

3.4 考察

本研究では OS Fingerprint を偽装することで，OS の種類やバージョンを攻撃者に錯覚させる手法を示した．本研究では，悪意ある第三者は攻撃を行う前に，インターネットを通じてホストの弱点の調査を行うことを前提としている．この前提は著者らが 8ヶ月にわたってデータを収集し，その分析により導かれたものである．これより，図 3 に攻撃が行われる際の戦略決定のモデルを示す．

まず攻撃者はホストの脆弱性を Portscan によって調査する．Portscan とは，トランスポート層における接続要求を送信し，ホストの応答を観測し，サービスが行われているポートを調査する技術である．

次に攻撃者は OS Fingerprint や Application Banner を調べ OS を検出し，ホストの持つ脆弱性をより詳しく調べる．なお nmap は， OS Fingerprint による OS の検出と Portscan を同時に行うことが可能である．

攻撃者は，この Portscan や OS Fingerprint や Application Banner の調査結果より得られたホストの脆弱性に関する情報を得る．そしてこの情報を元に攻撃を行う戦略を決定し，攻撃を行う．

本研究では OS Fingerprint の偽装により，攻撃ではなく攻撃における戦略の無力化を行う．しかし，OS Fingerprint を偽装しても，攻撃者に OS を特定される可能性はある．例えば Application Banner による検出手法についての対策は，この偽装とは別に行わなければならない．また各 OS において，広く使われるプログラムとそうでないプログラムを検査することにより，OS が特定される可能性がある．例えば一般的な UNIX 系 OS では，メールの配送を行うプログラムを TCP の 25 番ポートに常駐させ，コネクションの要求を待つ．しかし Windows 95/98 ではその可能性は低いため，攻撃者は 25 番ポートにプログラムが常駐されているかどうかを検査することにより，OS の特定を行う事ができる．

また脆弱性の調査をすることなく，攻撃が行われる場合もある．この場合，偶然にシステムにとって致命的な攻撃が行われる可能性が残されている．

本研究の研究領域は，あくまで悪意ある第三者に，本来与えられるシステムに関する情報を応答する代わりに，異なるシステムに関する情報を応答することで，システムの脆弱性が攻撃されないよう，攻撃を行う際の戦略の方向性を誘導することである．

4 評価

実装の評価には nmap2.54 BETA25 を用い，FreeBSD 4.3-RELEASE を NetBSD1.4-RELEASE，Windows95/98/NT4.0 として誤って検出されるかどうかを実験した．実験には図 4 で示すコマンドを用い，その結果は図 5, 6 に示す．これらの図から OS が意図通りに偽装されることがわかった．さらに各 Fingerprint が意図通りに偽装されているかどうかを，ネットワーク監視ツールである tcpdump [13] を用いて検査し，その tcpdump の出力を図 7, 8, 9, 10, 11, 12 に示す．そして図 13, 14, 15, 16 に TCP Initial Sequence Number Sampling，IP Identification Number Sampling の結果を示す．

```
> nmap -sS -O -p 22 172.16.236.2
```

図 4: OS の検出に用いたコマンド

```
（中略）
Remote OS guesses: NetBSD 1.3H (after 19980919) or 1.3I (before 19990119) little
 endian arch, NetBSD 1.3H-1.42 (after 19980919) or 1.3I (before 19990119) big en
dian arch, NetBSD 1.3I (after 19990119) to 1.4 x86
OS Fingerprint:
TSeq(Class=RI%gcd=1%SI=8F0D03%IPID=I%TS=100HZ)
T1(Resp=Y%DF=N%W=4000%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=N%W=4000%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(Resp=N)

TCP Sequence Prediction: Class=random positive increments
                         Difficulty=9374979 (Good luck!)
TCP ISN Seq. Numbers: 18EDDEC3 1A96B228 1C65406E 1F440B56 211F6627 2246E7AF
IPID Sequence Generation: Incremental
Final times for host: srtt: 2424 rttvar: 8174  to: 300000

Nmap run completed -- 1 IP address (1 host up) scanned in 2 seconds
```

図 5: d(NetBSD) に対する OS 検出の結果

本稿では，NetBSD に偽装した FreeBSD を，Deception の頭文字である “d” を用いて，d(NetBSD) と表記する．同様に Windows に偽装した FreeBSD を d(Windows) と表記する．

IP Don’t Fragment Bit

図 7 では，d(NetBSD) が IP Don’t Fragment Bit はセットせず，Path MTU Discovery を行わないことを示す．

TCP Null Probe に対する挙動

図 8 では，d(NetBSD) や d(Windows) が TCP Null Probe に応答する様子を示す．

TCP Option

図 9 では，d(NetBSD) が Maximum Segmentsize Op-

```
（中略）
Remote operating system guess: Windows NT4 / Win95 / Win98
OS Fingerprint:
TSeq(Class=TD%gcd=1%SI=1%IPID=BI%TS=U)
T1(Resp=Y%DF=Y%W=2017%ACK=S++%Flags=AS%Ops=M)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=2017%ACK=S++%Flags=AS%Ops=M)
T4(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=N)

TCP Sequence Prediction: Class=trivial time dependency
                         Difficulty=1 (Trivial joke)
TCP ISN Seq. Numbers: 3457B31 3457C75 3457DBA 3457EFF 3458044 3458188
IPID Sequence Generation: Broken little-endian incremental
Final times for host: srtt: 3397 rttvar: 10142  to: 300000

Nmap run completed -- 1 IP address (1 host up) scanned in 1 second
```

図 6: d(Windows) に対する OS 検出の結果

FreeBSD の場合

```
23:24:53.697963 172.16.236.1.53243 > 172.16.236.2.22: SE 1152114990:1152114990(0
) win 3072 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol>
23:24:53.700243 172.16.236.2.22 > 172.16.236.1.53243: S 3049482539:3049482539(0)
 ack 1152114991 win 16430 <mss 1460> (DF)
```

（DF Bit がセットされており，Path MTU Discovery が行われている）

d(NetBSD) の場合

```
23:22:11.234973 172.16.236.1.53176 > 172.16.236.2.22: S 4103815987:4103815987(0)
 win 2048
23:22:11.249023 172.16.236.2.22 > 172.16.236.1.53176: S 325709319:325709319(0) a
ck 4103815988 win 16384 <mss 1460>
```

（DF Bit がセットされておらず，Path MTU Discovery が行われていない）

図 7: d(NetBSD) の IP Don’t Fragment Bit に関する挙動

FreeBSD の場合

```
23:24:53.701448 172.16.236.1.53244 > 172.16.236.2.22: . win 3072 <wscale 10,nop,
mss 265,timestamp 1061109567 0,eol>
```

（Null Probe を無視する）

d(NetBSD) の場合

```
23:22:11.261479 172.16.236.1.53184 > 172.16.236.2.22: . win 2048 <wscale 10,nop,
mss 265,timestamp 1061109567 0,eol>
```

（TCP Null Probe に応答する）

```
23:22:11.262297 172.16.236.2.22 > 172.16.236.1.53184: R 0:0(0) ack 3840587912 wi
n 0
```

d(Windows) の場合

```
22:31:00.822281 172.16.236.1.43259 > 172.16.236.2.22: . win 1024 <wscale 10,nop,
mss 265,timestamp 1061109567 0,eol>
```

（TCP Null Probe に応答する）

```
22:31:00.823717 172.16.236.2.22 > 172.16.236.1.43259: R 0:0(0) ack 1976640252 wi
n 0
```

図 8: TCP Null Probe に対する挙動

FreeBSD の場合

```
23:24:53.697963 172.16.236.1.53243 > 172.16.236.2.22: SE 1152114990:1152114990(0
) win 3072 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol>
```

（TCP Option は Maximum Segmentsize Option のみが用いられている）

```
23:24:53.700243 172.16.236.2.22 > 172.16.236.1.53243: S 3049482539:3049482539(0)
 ack 1152114991 win 16430 <mss 1460> (DF)
```

d(NetBSD) の場合

```
23:22:11.258036 172.16.236.1.53183 > 172.16.236.2.22: SE 3840587912:3840587912(0
) win 2048 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol>
```

（Maximum Segmnetsize Option だけではなく，Window Scale Option，
Timestamp Option が用いられている）

```
23:22:11.260267 172.16.236.2.22 > 172.16.236.1.53183: S 358889143:358889143(0) a
ck 3840587913 win 16384 <mss 1460,nop,wscale 0,nop,nop,timestamp 1564 1061109567
>
```

図 9: TCP Option に関する挙動

FreeBSD の場合

```
23:24:53.697963 172.16.236.1.53243 > 172.16.236.2.22: SE 1152114990:1152114990(0
) win 3072 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol>
 (Window Size は 16430(0x402e))
23:24:53.700243 172.16.236.2.22 > 172.16.236.1.53243: S 3049482539:3049482539(0)
 ack 1152114991 win 16430 <mss 1460> (DF)
```

d(NetBSD) の場合

```
23:22:11.258036 172.16.236.1.53183 > 172.16.236.2.22: SE 3840587912:3840587912(0
) win 2048 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol>
 (Window Size は 16384(0x4000))
23:22:11.260267 172.16.236.2.22 > 172.16.236.1.53183: S 358889143:358889143(0) a
ck 3840587913 win 16384 <mss 1460,nop,wscale 0,nop,nop,timestamp 1564 1061109567
```

d(Windows) の場合

```
22:31:00.818791 172.16.236.1.43258 > 172.16.236.2.22: SE 1976640252:1976640252(0
) win 1024 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol>
 (Window Size は 8215(0x2017))
22:31:00.821111 172.16.236.2.22 > 172.16.236.1.43258: S 54884521:54884521(0) ack
 1976640253 win 8215 <mss 1460> (DF)
```

図 10: TCP Initial Window Size に関する挙動

FreeBSD の場合

```
23:22:11.271590 172.16.236.1.53189 > 172.16.236.2.44431: FP 3840587912:384058791
2(0) win 2048 urg 0 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol>
 (ACK 番号が Probe で用いられたセグメントのシーケンス番号に等しい)
23:22:11.272228 172.16.236.2.44431 > 172.16.236.1.53189: R 0:0(0) ack 3840587912
 win 0
```

d(Windows) の場合

```
22:31:00.833120 172.16.236.1.43264 > 172.16.236.2.32834: FP 1976640252:197664025
2(0) win 1024 urg 0 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol>
 (ACK 番号が Probe で用いられたセグメントのシーケンス番号に 1 加えた値)
22:31:00.833830 172.16.236.2.32834 > 172.16.236.1.43264: R 0:0(0) ack 1976640253
 win 0
```

図 11: TCP Xmas Probe に対する挙動

tion, Timestamp Option, Window Scale Option の順番で TCP Option をセットすることを示す.

TCP Initial Window Size

図 10 では d(NetBSD) の Initial Window Size の値が `0x4000` であり，d(Windows) では `0x2017` であることを示す.

TCP Xmas Probe に対する挙動

図 11 では， d(Windows) が TCP Xmas Probe に対して，Probe に用いられたセグメントのシーケンス番号に +1 した値を ACK 番号に用いて応答していることを示す.

ICMP Error Message Echoing Integrity

図 12 では ICMP Error Message に引用される，送信された UDP データグラムのチェックサムの部分が，

FreeBSD の場合

```
23:24:53.711846 172.16.236.1.53236 > 172.16.236.2.33764:  udp 300
                        4500 0148 46ce 0000 3c11 06b2 ac10 ec01
                        ac10 ec02 cff4 83e4 0134 4c5a 6868 6868
                        6868 6868 6868 6868 6868 6868 6868 6868
 (中略)
23:24:53.715004 172.16.236.2 > 172.16.236.1: icmp: 172.16.236.2 udp port 33764 u
nreachable
 (送信された UDP データグラムのチェックサムの値は引用せず 0000 に置き換える)
                        4500 0038 0009 0000 ff01 8b96 ac10 ec02
                        ac10 ec01 0303 a7ef 0000 0000 4500 0148
                        46ce 0000 3c11 06b2 ac10 ec01 ac10 ec02
                        cff4 83e4 0134 0000
```

d(NetBSD) の場合

```
23:22:11.272830 172.16.236.1.53176 > 172.16.236.2.44431:  udp 300
                        4500 0148 b13e 0000 3411 a441 ac10 ec01
                        ac10 ec02 cfb8 ad8f 0134 7d45 6464 6464
                        6464 6464 6464 6464 6464 6464 6464 6464
 (中略)
23:22:11.277170 172.16.236.2 > 172.16.236.1: icmp: 172.16.236.2 udp port 44431 u
nreachable
 (送信された UDP データグラムのチェックサムの値は引用する)
                        4500 0038 000a 0000 ff01 8b95 ac10 ec02
                        ac10 ec01 0303 013b 0000 0000 4500 0148
                        b13e 0000 3411 a441 ac10 ec01 ac10 ec02
                        cfb8 ad8f 0134 7d45
```

d(Windows) の場合

```
22:31:00.834444 172.16.236.1.43251 > 172.16.236.2.32834:  udp 300
                        4500 0148 44bc 0000 3d11 07c4 ac10 ec01
                        ac10 ec02 a8f3 8042 0134 951b 5050 5050
                        5050 5050 5050 5050 5050 5050 5050 5050
 (中略)
22:31:00.838119 172.16.236.2 > 172.16.236.1: icmp: 172.16.236.2 udp port 32834 u
nreachable
 (送信された UDP データグラムのチェックサムの値を引用する)
                        4500 0038 1000 0000 ff01 7b9f ac10 ec02
                        ac10 ec01 0303 3d77 0000 0000 4500 0148
                        44bc 0000 3d11 07c4 ac10 ec01 ac10 ec02
                        a8f3 8042 0134 951b
```

図 12: ICMP Error Message Integrity に関する挙動

d(NetBSD) や d(Windows) では 0 で置き換えず，正しく引用して返信することを示す.

TCP Initial Sequnce Number Sampling

表 13 は FreeBSD と d(NetBSD), d(Windows) に対して SYN フラグのセットされたセグメントを 1 秒毎に送信し，その応答の際に得られた TCP Initial Sequence Number のサンプルである.

FreeBSD がランダムに TCP Initial Sequence Number を生成するのに対し，d(NetBSD) は前回用いられた値に一定の乱数を加えた値を使用し，d(Windows) は前回用いられた値に，次に TCP Initial Sequence Number を必

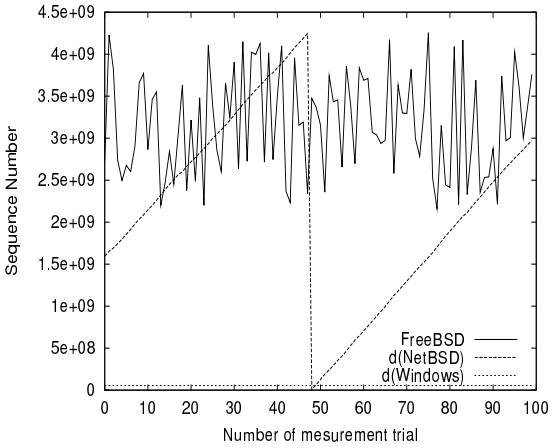

図 13: FreeBSD, d(NetBSD), d(Windows) の Initial Sequence Number

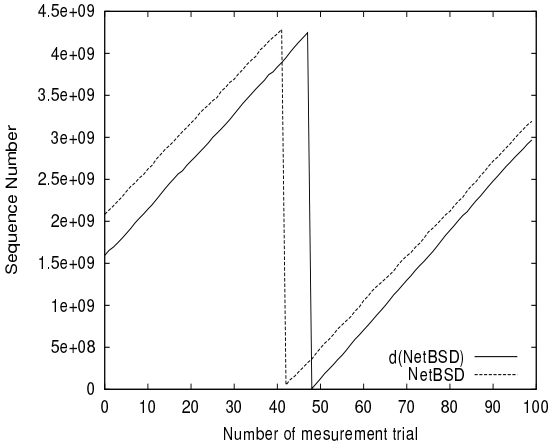

図 14: d(NetBSD) と NetBSD の Initial Sequence Number

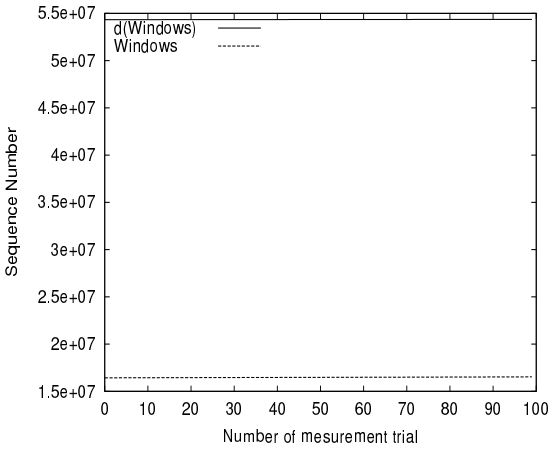

図 15: d(Windows) と Windows の Initial Sequence Number

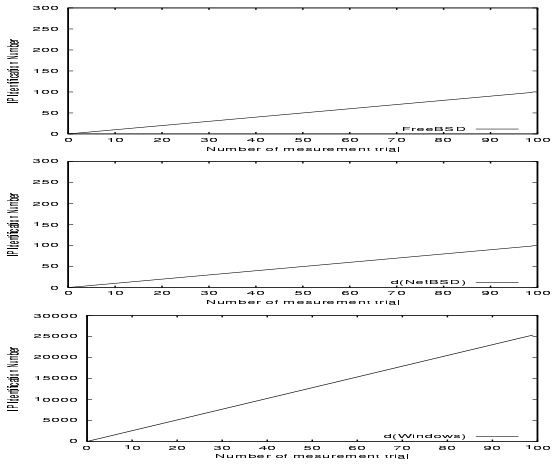

図 16: FreeBSD, d(NetBSD), d(Windows) の IP Identification Number

要とするまでの時間に依存した数値を加えた値を使用している．

偽装の精度を評価するため，d(NetBSD) と通常の NetBSD の Initial Sequence Number の値のグラフを図 14 に，d(Windows) と通常の Windows の Initial Sequence Number の値のグラフを図 15 に示す．それぞれ増加傾向にが非常に似通っていることがわかる．

IP Identification Number Sampling

表 16 は FreeBSD と d(NetBSD)，d(Windows) に 1 秒毎にデータグラムを送信し，その応答から得られた IP Identification Number のサンプルである．

FreeBSD や d(NetBSD) は IP Identification Number を前回用いられた値に `0x0001` を加えたものを使用しているのに対し，d(Windows) は前回用いられた値に `0x0100` を加えたものを使用している．

以上，各 OS における TCP/IP スタックの実装の差異である Fingerprint を調査し，各 Fingerprint の偽装によって FreeBSD を NetBSD や Windows に偽装するアーキテクチャを実装した．そして, OS 検出ツールを用いて OS の検査を行い，その結果，FreeBSD 4.3-RELEASE を図 5 にあるように NetBSD 1.4-RELEASE として，図 6 にあるように Windows 95/98/NT4.0 として，誤って検出させることに成功し，この実装が有効であることを示した．

5 おわりに

本研究では OS に関する情報を検出されることによる脆弱性の露出に着目し，OS を検出する際に用いられる OS Fingerprint を偽装することにより，OS の誤検出を促すシステムを FreeBSD カーネルに実装することで実現した．

本研究で示した技術を用いれば，OS に関する情報の流出を防ぎ，攻撃にかかるコストを増加させる．このことにより，本来稼働している OS が致命的な攻撃を回避できる可能性を高め，また IDS 等に攻撃を検出・分析させ，対応を取らせる時間をつくることが可能となると考えられる．

今後の課題としては，本研究で示した技術を応用し，システムの脆弱性が低い OS を脆弱性の高い OS に偽装した上で，攻撃者に脆弱性が高いと錯覚させ，攻撃を誘発し，攻撃行為をデータとして収集するための Honeypot 環境の実現が挙げられる．

謝辞

日ごろ御指導頂く奈良先端科学技術大学院大学の山口教授，飯田助手，知念助手に感謝致します．

参考文献

[1] S. M. Bellovin. ”Security Problems in the TCP/IP Protocol Suite”. *Computer Communication Review*, 19(2):32–48, 1989.

[2] dtk. http://www.all.net.

[3] fpf. http://www.pkcrew.org/tools.php.

[4] FreeBSD. http://www.freebsd.org.

[5] Fyodor. ”Remote OS Detection via TCP/IP Stack Fingerprinting”. http://www.insecure.org/nmap/nmap-fingerprinting, Oct 1998.

[6] V. Jacobson, R. Braden, and D. Borman. ”TCP Extensions for High Performance”. RFC 1323, May 1992.

[7] J. Mogul and S. Deering. ”Path MTU Discovery”. RFC 1191, Nov 1990.

[8] M. Morris. ”A Weakness in the 4.2BSD TCP/IP Software”, Feb 1985.

[9] NetBSD. http://www.netbsd.org.

[10] nmap. http://www.insecure.org/nmap/.

[11] Jon Postel, Editor. ”Transmission Control Protocol — DARPA Internet Program Protocol Specification”. RFC 793, Sep 1981.

[12] K. Ramakrishnan and S. Floyd. ”A Proposal to add Explicit Congestion Notication”. RFC 2481, Jun 1999.

[13] tcpdump. http://www.tcpdump.org.