2007－CSEC－37 (12)
2007／5／25

Anti OS Fingerprintingシステムの実装

三村 守† 中村 康弘†

†防衛大学校 情報工学科
239-8686 神奈川県横須賀市走水 1-10-20

g45042@nda.ac.jp yas@nda.ac.jp

あらまし TCP/IP の規約は RFC に定義されているが，各 OS における TCP/IP スタックの実装には差異があり，OS Fingerprint と呼ばれる．OS Fingerprint を収集し，分析することにより，悪意ある第三者は送信元のホスト OS を特定することができる．また，多くのホストの TTL や IPid 等の情報を収集することにより，ネットワーク内部のトポロジを推定したり，稼動ホスト台数を検出することができる．本研究ではネットワーク外部での OS Fingerprint 収集・分析に対して，ネットワークの経路上で OS Fingerprint を消去し，TTL を初期化するシステムを提案し，その有効性を評価する．この機構により OS 識別行為を無力化し，ネットワーク情報の流出を防ぎ，システムの安全性を高めることができると考えられる．

Implimentation of Anti OS Fingerprinting System

Mamoru Mimura† Yasuhiro Nakamura†

†Department of Computer Science, National Defense Academy
1-10-20, Hashirimizu, Yokosuka-shi, Kanagawa 239-8686, Japan

g45042@nda.ac.jp yas@nda.ac.jp

Abstract There are some different implimentation of TCP/IP in spite of its deffinition by RFC. The difference is called OS Fingerprint. A malicious third person can identify host OS by collecting and analyzing many OS Fingerprints. And collecting TTL or IPid of many hosts, also make that estimate network topology and count number of hosts in network possible. This report proposes and evaluates the system that removes OS Fingerprint and initialize TTL on the route from inside to outseide. Assume this system defeat OS Fingerprinting, prevent outflow of network information and enhance security.

1 はじめに

1.1 研究の目的

LAN を経由してインターネットに接続するホストは急激に増加し，システムの脆弱性が指摘されることが多くなっている．クライアントホストにおいてもバッファーオーバーフロー等の脆弱性が多数報告されており，インターネットに直接接続するホストだけでなく，LAN を経由して接続するホストの脆弱性も問題となっている．システムの脆弱性はホスト OS に密接に関係している．OS のセキュリティ問題は毎日のように報告されており，修正プログラムの配布も追いつかない場合がある．また，まだ報告されていない脆弱性を利用した攻撃も多く発生するようになってきている．このように，OS は常に修正すべき問題を抱えていると考えられる．すなわち，LAN 内部で稼動する OS の種類やバージョンに関する情報が外部に漏洩すれば，その情報によってシステムの脆弱性が漏洩する可能性がある．

悪意ある第三者はネットワークを監視し，TCP/IP スタックの挙動を観察することで，ホスト OS を特定することができる．これは，TCP/IP の規約は RFC に定義されているにも関わらず，各 OS における実装に差異があるためである．この差異は OS Fingerprint と呼ばれ，攻撃者はネットワークを傍受して

取得したOS Fingerprintを既知のOS Fingerprintと比較することより，遠隔でホストOSを特定することができる[1]．また，TTLからネットワーク内部のトポロジを推定したり，IPidを用いて稼動ホスト数を検出することも可能である[2]．本研究では不正アクセスを意図してOS Fingerprintを収集する行為をOS Fingerprintingと呼ぶ．このようなOS Fingerprintingにより，悪意ある第三者はシステムの脆弱性を知り，攻撃に役立てることができる．

本研究はネットワーク外部への経路上でOS Fingerprintを一括消去し，TTLを初期化することにより，OS Fingerprintingを無力化し，システムの安全性を高めることを目的とする．

以下，第2節においてAnti OS Fingerprintingシステム(以下AOFS)の設計と実装を示し，第3節ではシステムの評価を行う．最後に第4節でまとめと今後の課題について述べる．

1.2 OS Fingerprintの流出

proxyやSOCKSを用いることで，LAN内部の構造を隠蔽することができる．この仕組みを図1を用いて説明する．ホストBのようにproxyを経由して外部に接続した場合，外部ではproxyからの接続に見えるため，これによりLAN内部の構造を隠蔽することができる．しかし，proxyやSOCKSには対応していないアプリケーションもあり，そのようなアプリケーションの外部への接続をホストAのように許可した場合には，LAN内部の構造が漏洩してしまう．また，proxyを経由する場合には，まず内部のホストがproxyに接続し，その後にproxyが外部のホストに接続する．このためLAN内部に多くのホストが存在する場合にはproxyの多くのリソースが消費され，負荷が高くなってしまう問題がある．外部への経路上のルータでNAT（Network Address Translator）[3]やNAPTを実施した場合，送信元IPアドレスや送信元ポートは変換されるが，OS Fingerprintはそのまま残るため，LAN内部の情報が流出してしまう．

1.3 関連研究

個々のホストにおけるAnti OS Fingerprinting手法として，本来のOSとは異なるOS Fingerprintを返答する機構が提案されている[4]．しかし，この方式では個々のホストごとに対策を実施する必要があり，LANに接続するホスト数が多い場合には実現は困難である．また，カーネルのTCP/IPスタックを書き換える必要があるため，ソースが公開されていない商用のOSや，汎用性が低いホストに導入することは難しい．仮に個々のホストで対策を実施したとしても，TTLやIPidにより経路情報が流出する可能性がある．このため，個々のOSに依存せず，通信経路上でOS Fingerprintを消去・偽装できることが望ましい．

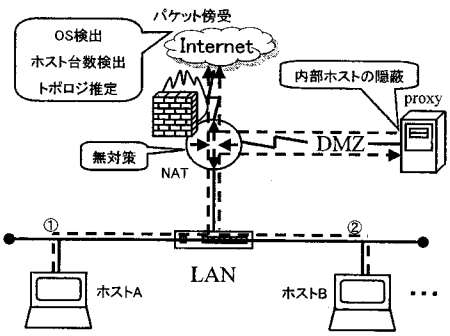

図1: proxyの機能

2 Anti OS Fingerprintingシステム(AOFS)

2.1 設計

本研究では，ネットワーク外部への経路上で経由するすべてのパケットのOS Fingerprintを消去し，TTLを初期化することでOS Fingerprintを流出させないシステムを提案する．このシステムにより，悪意ある第三者のOS FingerprintingによるOS識別，稼動ホストの台数検出やネットワークトポロジの推定を無力化し，システムの安全性を高めることができると考える．AOFSの動作を図2に示す．境界ルータ上に設置したAOFSは，内部からのすべてのパケットのヘッダを変換して外部へ転送する．

2.2 実装

境界ルータのTCP/IPスタックにおけるパケット転送処理部分を改造することにより，OS Fingerprint

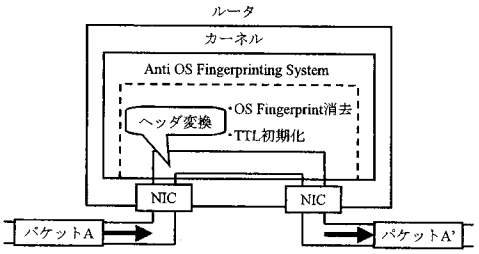

図 2: Anti OS Fingerprinting システム

の消去および TTL の初期化を行う．実装は Linux カーネル 2.6.18[7] を書き換えることによって実施した．新たに header_convert 関数を作成し，これを ip_forward 関数から呼び出すことにより，IP ヘッダおよび TCP ヘッダの変換を行う．変換処理終了後，チェックサムを再計算してパケットを送出する．以下，その細部手順について述べる．

2.2.1 IPid の変換

IPid はフラグメントパケットを再構築する際の識別子として利用されており，ひとつのパケットから分割されたパケットの IPid 値は同一となる．IPid 値については値の変化に OS 毎の規則性があり，Linux ではセッション確立まではランダム，OpenBSD では常にランダムに変化するが，多くの OS では 1 ずつ増加するように実装されている．そのため，同一送信元 IP アドレスの 1 ずつ増加する IPid を順にプロットして得られる直線を観測することで，NAT の内側の稼動ホスト台数を検出することができてしまう問題があった．これを無力化するために，IPid 値をランダムな値に変換する．IPid は OpenBSD の実装にも示されるとおり，ランダムに変化しても通信に影響はない．乱数の生成には，高速性，長周期及び高次元均等分布を併せ持つ，Mersenne Twister 乱数 [8] を使用する．ただし，パケットが本システムに到達する前にフラグメントが発生した場合には，同一の IPid 値を持つフラグメントパケット群を，すべて同一の IPid 値に変換する必要がある．よって，フラグメントパケットが到達した場合には，そのパケットの IPid 値をシードとして乱数を初期化し，新たに生成した乱数に変換する．これにより，変換されたフラグメントパケット群の IPid 値は同一となり，パケットの再構築が可能となる．

2.2.2 TCP オプションとヘッダ長の変換

TCP SYN パケットの TCP オプションは OS 毎に異なり，これも OS Fingerprint として OS 識別に利用されている．TCP オプションは必須ではないが，パフォーマンスに影響を与えるものが多い．しかし，OS Fingerprint を消去するためには，最もオプションが少ない OS に合わせて変換する必要がある．パケット長もそれに合わせて変換する．

2.2.3 TTL の初期化

TTL は OS 毎に初期値が決まっており，これも OS Fingerprint として OS 識別に利用されている．TTL 値はルータを経由するごとに 1 ずつ減算される．この性質を利用して，パケットの送信元ホストまでのホップ数を算出することができる．このため，送信元ごとのホップ数を収集することで，LAN 内部のネットワークトポロジを推定することができてしまう問題があった．これを無力化するため，本システムでは TTL 値を一定値に初期化する．

2.2.4 TOS フラグの変換

TOS 値を同一の値に変換する．TOS 値はパケットがルータで処理される際の優先順位を示す情報であるが，現在ではほとんど利用されていない．OS ごとの差異を無くすため，同一の値に変換する．

2.2.5 ウィンドウ値の変換

初期ウィンドウ値が OS の識別に利用されるため，同一の値かランダムに変換する必要がある．ウィンドウ値は許容できるバッファの大きさを通知するために利用される．よって，増加させると通信に不具合が発生する可能性があるが，減少させればパフォーマンスは低下するが，不具合は発生しないものと考えられる．AOFS では初期ウィンドウ値を最もバッファが少ない OS の値に合わせて変換する．

2.3 変換アルゴリズム

変換アルゴリズムを図 3 に示す．システム内部から外部へ中継されるすべてのパケットに対して変換処理を実施する．これにより，システム外部への OS Fingerprint の流出を防ぐ．

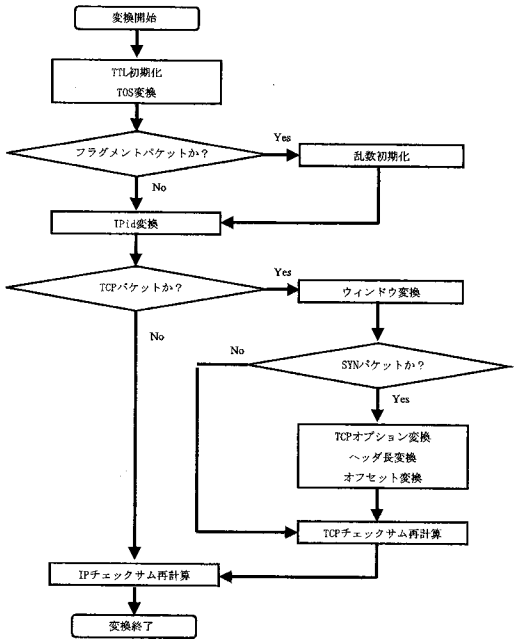

図 3: 変換アルゴリズム

3 検証実験

試作した AOFS について検証実験を行い，正常な通信が可能であり，OS Fingerprinting を無力化できていることを確認する．検証実験における AOFS の配置を図 4 に示す．各ホストは 100BASE/T イーサネットで接続されており，図中のルータ内部に AOFS を設置する．システム外部のホスト A で，システム内部に設置された各ホストのパケットを観測する．

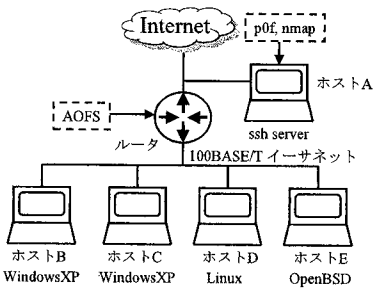

図 4: 検証実験におけるシステムの配置

3.1 アプリケーションの動作確認

LAN 内部ホストの各アプリケーションを用い，外部と通信確認を行った結果を表 1 に示す．traceroute 以外のアプリケーションについては通信に支障は生じなかった．traceroute に支障が生じたのは，IP ヘッダの TTL 値を利用してホップ数の計測を行うためである．AOFS によって TTL 値が初期化されるため，正しくホップ数を計測することができない．このように，AOFS はヘッダを利用するアプリケーションには影響がある．しかし，そのようなアプリケーションは極めて少ないのが現状である．

表 1: 通信確認結果

Application	Protocol	Result
ping	ICMP	○
traceroute	ICMP/UDP	×
nslookup	UDP/TCP	○
ssh	TCP	○
telnet	TCP	○
smtp	TCP	○
http	TCP	○
pop3	TCP	○
skype	TCP(P2P)	○

3.2 p0f による OS 識別

AOFS を経由した Linux および OpenBSD の ssh 接続を，p0f でスキャンした結果を図 5 に示す．AOFS によって OS Fingerprint は消去されており，p0f は OS 識別に失敗していることが確認できる．また，TTL も初期化されているため，ホップ数の測定にも失敗している．よって，AOFS は有効に機能しており，受動的な OS Fingerprinting を無力化し，ネットワークトポロジの推定も不能にしていると判断できる．

3.3 nmap による OS 識別

AOFS の外側から内側のホストに対し，nmap の OS 識別オプションを有効にしてスキャンした結果を図 6 に示す．AOFS によって OS Fingerprint は消去されており，nmap は OS 識別に失敗していることが

```
# ./p0f
p0f - passive os fingerprinting utility, version 2.0.8
(C) M. Zalewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN) on 'eth0', 262 sigs (14 generic, cksum 0F1F5CA2), rule: 'all'.
192.168.4.10:32777 - UNKNOWN [5840:64:1:40:.. :?:?]
  -> 192.168.4.1 :22 (link: unspecified)
192.168.4.10:21944 - UNKNOWN [5840:64:1:40:...:?:?]
  -> 192.168.4.1 :22 (link: unspecified)
```

図 5: p0f のスキャン結果

確認できる．AOFS は，能動的な OS Fingerprinting の対策としても有効に機能することが確認できた．

```
# ./nmap -O 192.168.5.4
Starting nmap 3.93 ( http://www.insecure.org/nmap/ ) at 2007-03-09 08:34 JST
Insufficient responses for TCP sequencing (1), OS detection may be less accurate
Insufficient responses for TCP sequencing (1), OS detection may be less accurate
Insufficient responses for TCP sequencing (1), OS detection may be less accurate
Interesting ports on 192.168.5.4:
(The 1666 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
111/tcp open  rpcbind
No OS matches for host (If you know what OS is running on it, see
http://www.insecure.org/cgi-bin/nmap-submit.cgi).
```

図 6: nmap のスキャン結果

3.4 IPid の観測

通常の通信と，AOFS を経由した通信の IPid 値の観測結果を図 7 に示す．通常の通信では IPid が 1 ずつ増加する直線が 3 つ観測されており，OS Fingerprint と合わせて分析することにより，ルータ内部で 4 台のホストが稼動していることが検出できる．システムを経由した場合には IPid はランダムとなり，システム内部で稼動するホスト数を検出することはできない．

3.5 通信遅延

パケットジェネレータである hping[9] を使用し，通常の通信と AOFS を経由した通信の RTT(Round Trip Time) を比較した結果を表 2 に示す．ルータ内部のホストから ssh サーバに対し，TCP SYN パケットを 10 パケット送信し，応答時間を 3 回計測した．通常のルータを経由した通信と AOFS を経由した通信の RTT にはほとんど差はなく，通信遅延はほとんど発生していないことを確認した．これは

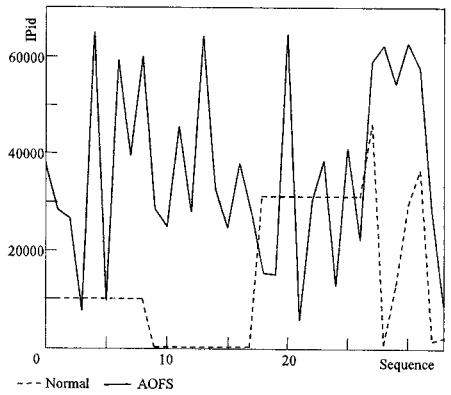

図 7: IPid 観測結果

プロセッサによる TCP/IP スタックの処理速度と，イーサネットカードによるパケットの処理速度に大きな差があるためであると考えられる．

表 2: RTT 計測結果

	Normal			AOFS		
Count	1	2	3	1	2	3
1	20.4	18.7	22.0	5.6	17.3	5.9
2	5.1	5.0	4.7	4.2	5.0	4.8
3	5.1	4.7	4.7	4.6	4.7	4.8
4	4.8	4.7	4.8	4.7	4.7	6.2
5	4.7	5.0	4.8	4.7	4.8	4.8
6	4.9	4.6	4.9	7.5	5.2	4.7
7	4.7	4.7	4.9	4.6	4.8	4.8
8	5.0	4.6	5.3	4.6	5.0	4.9
9	4.8	4.7	5.1	4.7	4.9	6.6
10	5.0	4.9	4.2	4.9	4.8	5.0
Average	6.45	6.16	6.54	5.01	6.12	5.25

ms

3.6 処理能力

大容量ファイルの転送を行い，通常のルータと AOFS の処理能力 (Throughput) を比較した結果を表 3 に示す．ルータ内部のホストから ssh サーバに対し，scp コマンドで 1GB のファイルを転送し，転

送量100MBごとの処理能力を計測した．通常のルータとAOFSの処理能力にはほとんど差はなく，処理能力はほとんど低下していないことを確認した．したがって，AOFSにおけるTCP/IPスタックへの演算処理の追加が，ルータとしての処理能力が低下するボトルネックにはならないことが言える．

表 3: Throughput 計測結果

Byte	Normal	AOFS
0-100MB	10.0	10.0
100-200MB	11.1	11.1
200-300MB	11.1	10.0
300-400MB	11.1	11.1
400-500MB	11.1	11.1
500-600MB	11.1	11.1
600-700MB	10.0	12.5
700-800MB	11.1	11.1
800-900MB	10.0	11.1
900MB-1GB	11.1	12.5
Average	10.77	11.16

B/s

4 おわりに

本研究ではネットワーク外部に流出したOS Fingerprintを収集・分析することにより，内部ホストのOS識別，ホスト台数検出やネットワークトポロジの推定が可能であることに着目した．そして，経路上の境界ルータでOS Fingerprintを消去し，TTLを初期化することによってこれを無力化する手法を提案した．さらに，AOFSをLinuxカーネルに実装し，AOFSの通信への影響，効果，通信遅延および処理能力について評価した．

本研究で提案した手法を用いれば，システム内部のネットワーク情報の流出を防ぎ，攻撃にかかるコストを増大させることができる．AOFSをNATと合わせて利用すれば，NATルータを単一のホストのように見せることも可能である．これにより，内部ホストが攻撃を回避できる可能性を高め，ネットワーク全体のセキュリティを向上させることができる．また，外部からの攻撃を検出・分析し，対策を取る時間を稼ぐこともできる．

今後の課題としては，TCPシーケンス番号やその他のOS Fingerprintの消去が考えられる．TCPシーケンス番号はランダムに変化するが，OSごとに使用している擬似乱数生成アルゴリズムが異なるため，これを分析することにより，ネットワーク内部のOSや稼動ホスト台数を検出される可能性がある．TCPシーケンス番号は応答番号と関連があるため，変換するためには変換テーブルを保持する必要がある．IPv6ヘッダについても同様に，ネットワーク内部の情報を流出させる可能性がある．また，パケットに書き込まれる情報ではなく，TCP/IPスタックの挙動を消去するためには，別の手法を検討する必要がある．

参考文献

[1] Fyodor Dostoevsky：Remote OS Detection via TCP/IP Fingerprinting(2nd Generation), http://insecure.org/nmap/osdetect/.

[2] Steven M.Bellovin：A Technique for Counting NATted Hosts, Proceedings of Internet Measurement Workshop 2002, pp. 267–272 (2002).

[3] K. Egevang, Cray Communications, P. Francis and NTT：The IP Network Address Translator(NAT) RFC1631 (1994).

[4] 宮本 大輔，大江 将史，木村 泰司，門林 雄基：OS Fingerprint 対策手法の実装と評価，Proceedings of Fourth Workshop on Internet Technology 2001, pp. 17–25 (2001).

[5] nmap http://insecure.org/nmap/.

[6] p0f http://lcamtuf.coredump.cx/p0f.shtml.

[7] The Linux Kernel Archives http://www.kernel.org/.

[8] M. Matsumoto and T. Nishimura：Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, ACM Trans on Modeling and Computer Simulation Vol. 8, No. 1, January ' pp. 3–30 (1998).

[9] hping http://www.hping.org/.