

A Structured and Practical Methodology For Security Evaluation of a IP Based Stack (Version 0.2)

Venkat Pothamsetty
Critical Infrastructure Assurance Group
Cisco Systems, Austin, TX
vpothams@cisco.com

Andrew Balinsky
Security Technologies Assessment Team
Cisco Systems, Austin, TX
balinsky@cisco.com

19th June 2003

Abstract

Any operating system that uses an IP based protocol stack for its networking purposes will have an IP address at its network layer and will rely on its transport layer for reliable or unreliable transportation of the IP datagrams. It is well known that security is not one of the primary components in the initial design specifications IP/TCP/UDP protocol based stack and is vulnerable to classic attack techniques including sniffing, spoofing and flooding [Morris-TCP/IP].

TCP/IP is ubiquitous now, despite these security problems. In fact, the TCP/IP usage is expanding rapidly into new devices and new areas such as house hold devices and critical infrastructure areas. The more secure IP version 6 has not seen widespread deployment yet. Instead of totally replacing the present V4 stack, what we are seeing today is a cover up at the application layer (with VPNs, SSH, and SSL) to diminish the effect of those exploits. There are innumerable number of machines, features, and protocols using TCP/IP and therefore there is a need for structured methodology for evaluating the robustness of a given TCP/IP protocol stack. The IP stack code is the probably one of the first code that gets to process the packets hitting an interface in a given device and therefore needs to be robust enough to handle any kind of malformed packet.

The objective of this document is to develop a methodology for checking the robustness of an IP based stack. We cover TCP, UDP and ICMP, the three most popular protocols implemented over an IP stack. Technically, we focus on the information needed for a product testing team to systematically check and estimate the robustness of a given stack. Our primary goal is to structuralize the evaluation process in such a way that a typical product testing team would have the knowledge of *what tools to run, the best way to run those tools, what should they look for measuring the impact of the attacks*, and possibly *how to debug the causes of the impact*, which is what the present security literature is lacking. While there are click-able GUI tools that are capable of running various tools described in this document, we focus on analyzing the output of various tools, explaining the “nuts and bolts” of various attack techniques, network setups and metrics to observe. We also explain various mitigation techniques and corresponding procedures.

We start out the evaluation task list with operating system identification through TCP/IP signatures and various application based signatures. We continue the task list by evaluating the randomness of Initial Sequence Numbers of the TCP/IP stack. We then extend it to the more important areas of TCP resource exhaustion and testing the stack’s robustness to various forms of flooding, including malformed packets. We then move on to analyzing the impact of various classic TCP/IP exploits against the stack. We end the document by describing a general process for testing robustness of stack implementations.

Contents

1	Introduction	4
1.1	Disclaimer	4
1.2	Need for, and Objective of the Document	4
1.3	Intended Audience	4
1.4	Structure of the Document	4
1.5	Network Layout	5
1.6	Attacking and Security Testing	5
1.7	Terminology	6
1.7.1	General English Terms	6
1.7.2	Technical	6
1.8	Evaluation Task Flowchart	6
2	Attack Techniques	6
2.1	Reconnaissance	7
2.2	Hijacking	7
2.3	Flooding	7
2.3.1	Arbitrary Packet Flooding	7
2.3.2	TCP Resource Exhaustion	7
2.4	Malformed Messages	7
3	Metrics	7
3.1	Responsiveness of the IP Layer	7
3.2	Responsiveness of the TCP Layer	8
3.3	Application Responsiveness	8
3.4	Device Responsiveness	9
3.5	CPU and Memory Utilization	9
3.6	Network Responsiveness	10
4	Before Jumping In	10
5	OS Identification and TCP/IP Signatures	10
5.1	Technique one : Through Nmap	10
5.1.1	What the output means	10
5.2	Technique Two - Through ICMP	10
5.3	Technique Three - Through Application Fingerprinting	12
5.4	Mitigation Techniques	12
6	TCP Initial Sequence Number Randomness	13
6.1	Technique One - Using Nmap	13
6.1.1	What the Output Means	14
6.2	Technique Two - Using Hping and GNUPlot	14
6.2.1	Sample PNG Plots	14
6.3	What the Output Means	14
6.4	Mitigation Techniques	14
7	Scanning	16
7.1	TCP Scans	16
7.1.1	SYN Scans	16
7.1.2	ACK Scans	16
7.2	UDP Scans	18
7.2.1	What It Means	18
7.2.2	Mitigation Techniques	18
7.2.3	Miscellaneous Scans	19

7.3	Protocol Scans	19
8	TCP Resource Exhaustion	19
8.1	Various States and How Can They be Attacked	20
8.2	Test Technique using Naptha	21
8.3	Applying Metrics	22
8.4	Sample Runs	22
8.5	Mitigation Techniques	22
9	Robustness of the Stack Against Malformed packets and Flooding	25
9.1	Introduction	25
9.2	Applying Metrics	26
9.3	How To Use ISIC Suite	26
9.4	Mitigation Techniques	29
10	Classic TCP/IP Attacks	29
10.1	Land Attack	29
10.2	Teardrop	29
10.3	Smurf	29
10.4	Jolt	29
10.5	Ping of Death	29
10.6	Technique One - Using C code and Targa2	29
10.7	Technique Two - Using Nessus and NASL	29
11	Miscellaneous	30
11.1	Application Banners	30
11.1.1	Metrics and Impact	30
12	Developing a Generic Evaluation Strategy	30
12.1	Analyzing Various Components and Architecture in the Device	31
12.2	Following the Life of a Packet	31
12.3	Developing Metrics	31
12.3.1	Metric at Each Layer	31
12.3.2	Resource Consumption by Each Component	31
12.4	Developing Attack Locations and Metrics - Through an Example	32
13	Acknowledgements	33

1 Introduction

The objective of this document is to come up with a task list for evaluating a TCP/IP stack through exposing the stack implementation to a set of known exploits and then to analyze the impact. It is meant to be extended and expanded as new techniques and better tools come out. The tools that are needed for evaluation, the metrics that should be observed while doing the attacks, and the process behind each evaluation task are discussed in detail. This document does not encompass threats in lower layer protocols (like ARP spoofing) and does not discuss application layer issues (like DNS spoofing). While the inherent vulnerabilities in the TCP/IP stack still exist, we focus on problems that can and should be solved, such as the errors that crop up due to misimplementation of TCP/IP stacks.

1.1 Disclaimer

This document is based solely on our own experiences. We, the authors and our employer, Cisco Systems Inc will not be liable for any indirect, special, or consequential damages arising from the use of, failure to use or improper use of any of the procedures, tools that we describe and recommend in this document.

1.2 Need for, and Objective of the Document

The ever-increasing trend of cyber attacks and relatively slow-growing pace of security professional population raises the need for developing a methodology for structured security testing aimed at non security professionals. Our goal is to divide the complex process of security testing into various pieces of simple evaluation task lists, which can be performed by a person having reasonable knowledge of networking and not necessarily an in-depth knowledge in the area of security. We tried to put our experience in security testing into explaining to the readers what to look for and what would be the false positives while performing a TCP/IP stack evaluation.

Strong encouragement for frequent vulnerability assessments in various National Strategy [NS-Cyberspace] documents only emphasizes the need for a structuralizing security testing process. We believe that our attempt will be useful for evaluating devices of various networking technologies. Emerging trends in industrial networking, including plans to move various industrial protocols over IP and the related security fears is one of the inspirations for writing this document. The malfunction or unstable performance of TCP/IP in such devices could result in safety issues or damage to the physical world, not just loss of data. This paper tries to document a comprehensive TCP/IP stack evaluation methodology. This paper goes through various areas of stack that need to be evaluated, explains various tools and attack techniques involved in the evaluation, and outlays various symptoms to look for determining the impact of the attacks.

We focus on ubiquitous protocols that most of the devices are likely to run such as TCP, UDP and ICMP and is not an exhaustive list of protocols by any means.

1.3 Intended Audience

This document is intended for any vendor who makes devices based on an IP stack. This document intends to give their testing teams a structured methodology so that they can make sure that their stack is reasonably robust before shipping the product to the customers.

This document is also intended for users of those devices who want to have some level of confidence on the device's security before plugging them into production networks.

This document assumes that the reader has reasonable knowledge of networking and TCP/IP and relatively less knowledge of security concepts, tools and attack techniques.

1.4 Structure of the Document

This document is divided into various sections based on various evaluation tasks that need to be done. Because each evaluation task can be achieved using multiple techniques, each of the sections are again

sub-divided into various techniques that are laid-out in the following way

1. Describe the corresponding technique in detail
2. Explain various tools, which can execute the technique and provide the best possible ways to run those tools
3. Provide sample runs of those tools
4. Explain various symptoms to look for on the device and on the network to estimate the impact of the attack and list out various possible culprits to help debug the causes of the impact
5. Suggest various mitigation techniques for each evaluation task

1.5 Network Layout

In this document, we refer the machine from which the attacks are conducted as the “attack machine” and the target stack as the “victim stack/machine”. We also refer to the machine from which the metrics are observed as the “metrics machine” or “client machine”. It must be noted that each of those machines should be selected so that those physically different machines. For example, the metrics machine should be a different machine from that of attack machine. It is preferable to place the attack machine and the victim machine on the same network. This takes care of flooding issues over the network gateways and also issues involving some routers not forwarding some illegitimate packets. Care should be taken that all the machines and network devices involved in the testing are not on anykind of production network.

Its preferable to have the metrics machine in a network different from that of the attack or victim machine, because we want the metrics machine to be a “third party” machine, accessing the victim machine. But during some attacks, like flooding for example, we might want to put the metrics machine on the same network as the attack and victim machines, we can observe the impact on the victim machine “as close as possible.” This will remove any impact on the network gateway from our observations.

In the given examples, we assume that the test is being run against a network device with the generic IP address of 1.2.3.4 . Linux command lines for attacks are given, and include the appropriate prompt (% or #) depending on whether the attack can be run as a regular user or requires root access.

1.6 Attacking and Security Testing

An attacker has to find a single vulnerability to compromise a device and has all the time in his world, a security tester has to go through all possible attacks and usually has limited time. The task list executed by an attacker and a security testing team may appear the same for a third party person. While this is partially true in network penetration testing, things are quite different in product security testing. In a penetration testing scenario, the success of the attacks can be measured, it is primarily getting access into the device or the network being tested. There is no easy measure of success in a product security testing scenario. Though the primary focus is the effect on the product’s core functionality and its manage-ability, a lot of metrics should be observed for knowing and measuring the impact of security testing. While black box testing makes sense in a network testing scenario, testing the product with-out the knowledge of inner details and architecture of a product is almost useless.

The security testing team needs to go study through various product documents and understand how the device works before starting the evaluation. He needs access to all the commands at all levels on the device. While the evaluation should be started with a initial plan in mind, the tester has to constantly think about whether various symptoms observed during any evaluation task can impact any other functionality of the device. He will have to investigate the disease causing the symptom and investigate whether the disease can be caused on any other corner of the device.

Many of these attacks listed in this document, like flooding with ISIC for example, are very uncommon over the Internet and does not make sense in a penetration testing scenario. But in a product security scenario, the impact observed during those attacks are symptoms for a disease on the product and can be exploited through any one or cumulative effect of other means. For example if the stack of a network device does not respond to pings after flooding with IP packets, the reason may be a single malformed packet at any layer

above IP layer or a design problem in an application which is designed to process and store all IP packets that it gets.

This type of analysis will help the testing team to narrow down onto a set of tests that have the highest potential to cause a problem instead of exhaustively running all the tests on all the components of the device.

1.7 Terminology

In this section, we explain about what is actually on our mind when we wrote some of the terms in this document.

1.7.1 General English Terms

Unless otherwise explained more after these terms,

1. **Catastrophic/Very Bad:** When we say that an impact of a particular attack is *very bad* or *catastrophic*, we mean that the impact has a potential to be a very serious security vulnerability and recommend that the problem should be fixed immediately and the product should not be shipped to customers without fixing the problem.
2. **Bad:** When we say that an impact of a particular attack is *bad*, we recommend that the problem should be fixed, but it is not critical enough to stop the customer shipment. The reader should take impact on other areas into consideration, like product's core functionality and management before making the decision.
3. **Negative:** The term *negative* means that the impact is certain to lead to a security vulnerability.
4. **Tolerable:** The term *tolerable* means that though the impact does have the potential to be a security vulnerability, there is nothing that can be done about it or the impact is so minimal that it can be ignored.
5. **Needs to be investigated:** This means that the cause of the impact may pose other security risks. The cause of the corresponding impact needs to be found and fixed.

1.7.2 Technical

1. **Stack:** An IP based stack, with IP at network layer and TCP/UDP at transport layer.
2. **Memory:** Where the OS stores its volatile information, usually Random Access Memory.
3. **Connection Tables:** Any header information that the OS has to store when it processes the packet.

1.8 Evaluation Task Flowchart

Our evaluation task list starts out with OS detection. Next we evaluate the PRNG of the stack. We then find out all the open TCP/UDP ports and implemented protocols on the machine. We use the information gathered in scanning to conduct the resource exhaustion, malformed and flooding attacks. We then move on to test the robustness of the stack towards legacy stack attacks. We then attempt to generalize all the task lists so that it can be applied to any stack.

2 Attack Techniques

There are several attack techniques that attackers use, depending on their goals in attacking a system. Techniques relevant to this document are described below.

2.1 Reconnaissance

This is a technique to discover and derive information about a target. On a target stack, it usually involves techniques such as scanning for various types(TCP/UDP) of open ports and implemented protocols. It also involves probing open ports for banners and passive sniffing of network traffic. This is usually the first and foremost “attack” that a typical attacker will conduct against any stack that he finds.

2.2 Hijacking

In a TCP sense, this is a technique to take over an established session to either monitor the TCP stream or insert malicious data into the stream. The hijacker has to be in the data path of the TCP stream or be able to predict the Initial Sequence Numbers of both the parties.

2.3 Flooding

This technique involves sending large amounts of data in an attempt to exhaust resources, such as network bandwidth, CPU time, memory, or storage. We divided flooding into two areas, though TCP resource exhaustion, depending on the implementation, may not require large rates of packet flooding.

2.3.1 Arbitrary Packet Flooding

Generic packet flooding relies on sending a large rate of arbitrary packets to the stack hoping to overwhelm it. The impact on the stack occurs primarily in IP queuing buffers.

2.3.2 TCP Resource Exhaustion

This technique, as the name suggests, involves tying up resources on the stack by utilizing the fact that TCP has to store state information for each connection request that it gets. The impact of these attacks is primarily on the CPU and memory usage.

2.4 Malformed Messages

This involves sending data that doesn't conform to the rules or standards and that tests boundary conditions. The goal of this attack is to see whether controls can be subverted, or simple Denial-of-Service. It can be used to exploit coding errors such as unchecked buffers (buffer overflows).

3 Metrics

Metrics are the set of symptoms that the evaluator has to look for in order to determine the impact of the attacks. Its very important to examine the metrics carefully because that is what determines whether or not the attacks had a negative impact on the device.

Without proper metrics the evaluator might miss a valid problem or get bogged down with a false positive.

Its very important to note that all the metrics that involve observing the responsiveness of stack should be done from a machine different from the attack machine.

3.1 Responsiveness of the IP Layer

Since IP layer code is probably one of the first pieces of code that gets to process the packets reaching the stack, responsiveness of the IP layer is the metric that needs to be monitored *continuously* for almost all the attacks.

The differences in performance of any IP based protocol before and after the attack can be used for measuring the impact on IP layer. The easiest to measure would be ICMP (IP protocol (0x01)) Echo latencies.

Sample comparisons of pings on a normal machine and an victim machine
Observe that the latencies would be in single digit milli seconds and would be uniform

```
%ping 1.2.3.4

PING 1.2.3.4 (1.2.3.4): 56 data bytes
64 bytes from 1.2.3.4: icmp_seq=0 ttl=255 time=1.6 ms
64 bytes from 1.2.3.4: icmp_seq=1 ttl=255 time=1.6 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=255 time=1.6 ms

(Notice that the latencies are in single low digit MS and uniform)

%ping 1.2.3.4

PING 1.2.3.4 (1.2.3.4): 56 data bytes
64 bytes from 1.2.3.4: icmp_seq=0 ttl=255 time=3980.4 ms
64 bytes from 1.2.3.4: icmp_seq=1 ttl=255 time=2192.8 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=255 time=1004.6 ms

(Latencies to the corresponding machine would be much higher under attack, and inconsistent too)
```

Table 1: Sample Run of Pings

Run a continuous ping to the target machine, and watch how the response time varies before, during, and after the attack.¹

A slow, uneven ICMP echo response while the attack is going on indicate a negative impact on the machine. A slow, uneven ICMP echo response after the attack is completed indicates a bad impact and while a complete absence of an ICMP echo response after the attack is completed indicates a catastrophic impact (For sample runs, see Table 1)

3.2 Responsiveness of the TCP Layer

Measuring the impact on the TCP layer in general and the victim port in particular gives a direct estimation of the impact of the attack on the service listening behind the port. For measuring the responsiveness of the TCP layer, latency of the SYN_ACK response for a SYN simulation is a very good measure. Hping2, a TCP specialized tool has a very useful functionality [See Table 2]. It sends a SYN packet and shows the packet it got back and the response time in the way ping shows for ICMP.

Slow and uneven SYN_ACK response latencies while the attack is going on indicates a negative impact on the machine. Continued slow and uneven SYN_ACK after the attack is completed indicates a catastrophic impact while no SYN_ACK response from a previously listening port even after the attack is completed indicates a catastrophic impact.

3.3 Application Responsiveness

The responsiveness of the application that is servicing at the port being attacked is the ultimate metric for an attack, particularly denial of service attacks.

The responsiveness of both an existing application, started before the attack and the ability of the stack to accept new connection attempts should be tested. While measuring the impact of the attack, its very important to understand the design of the application and the corresponding protocol, whether it needs persistent TCP connections, whether it uses heartbeats, whether it is CPU intensive or memory intensive etc.

¹From a Windows machine, run ping -t to get a continuous ping

Sample run of hping to a normal machine

Note that '-S' tells the tool to send a SYN packet and '-p' is the destination port. If the port is not specified, the destination port will be zero.

```
# hping2 -S 64.101.184.1 -p 22
```

```
HPING 64.101.184.1 (eth0 64.101.184.1): S set, 40 headers + 0 data bytes
```

```
len=46 ip=64.101.184.1 ttl=255 id=0 sport=22 flags=SA seq=0 win=4128 rtt=0.6 ms
```

```
len=46 ip=64.101.184.1 ttl=255 id=0 sport=22 flags=SA seq=1 win=4128 rtt=0.5 m
```

```
len=46 ip=64.101.184.1 ttl=255 id=0 sport=22 flags=SA seq=2 win=4128 rtt=0.5 ms
```

Table 2: Hping Sample Run

For example, while testing a web server, because it requires multiple TCP connections, its understandable if it is not responding during a SYN flood attack. On the other hand, if an existing SSH connection does not work during or after a SYN flood, its a bad impact. This will be product-dependent too, and could include, for example, making telephone calls from an IP telephone that is under attack.

3.4 Device Responsiveness

Here are some metrics for testing the device responsiveness

- The behavior of the console over a medium or port that is not being attacked, serial port for example.
- Whether any processes died on the operating system of the device
- Anomalies in any of the device's non-core functionality. This can be the ability of the device to log, to be administered or to interact with any other device on the network.

3.5 CPU and Memory Utilization

Both the CPU and the memory utilization on the device being attacked should be *continuously* monitored during the whole process of testing. Depending on the underlying operating system, there should be commands to show the status of free memory and CPU utilization. Care should be taken to turn off all logging and debugging, as these can seriously impact the performance of the machine, and affect the test.

The following would help to measure the impact on CPU and memory levels

- If the attack involves huge amounts of packet flooding, CPU increase, even if affects the products core functionality is tolerable(The environment in which the device operates must be taken into consideration, the rate of packets is huge and such a flooding is un-usual inside an enterprise), if it gets back to normal immediately after the flooding is stopped
- If the amount of flooding is small, large amounts of CPU usage means the attack had a very bad impact on the device and needs to be investigated more
- Various states, including TCP states typically have timeout periods for a few minutes. We recommend that a reasonable amount or recovery time be given before jumping into estimating the impact on memory. If all the memory is recovered after that period its good otherwise it needs to be investigated.

3.6 Network Responsiveness

Often knowing the impact of the attacks on a given network is important. If the device being attacked is a perimeter network device and is serving a particular network (Firewall and VPN-like access devices), the responsiveness of that network towards a client accessing the network should be monitored.

A ping *through the device* to a machine on the other side of the network is a good measure of this.

4 Before Jumping In

System vitals need to be collected before starting any attacks to create a baseline for the victim machine. Comparing the attack data to the baseline data after the attack will give a good indication about the impact of the attack. The following should be noted down, under “normal conditions”, before starting the testing.

- Normal CPU usage
- Normal memory statistics
- Process listing and connection table listing
- ICMP echo latencies, SYN_ACK latencies and various application response times under normal conditions

5 OS Identification and TCP/IP Signatures

OS identification is the first and foremost on the mind of an attacker whenever he finds a stack which has a IP address. This testing forms the starting point for all further tests. A TCP/IP signature is a unique footprint that most operating systems have, which can be used to identify the code base. This includes such things as: which ports are open, how the stack deals with IP or TCP options, and ICMP responses. See [Fyodor-OSFingerprint] for an excellent description of such techniques.

5.1 Technique one : Through Nmap

The Network Mapper (Nmap) tool, best known for its port scanning abilities, also includes Operating System detection [Fyodor-OSFingerprint]. It has over 500 signatures for different operating systems, and can often determine them to 3 levels of version numbers (e.g. Linux 2.0.35). Sample run is boxed in Table 3

5.1.1 What the output means

Note that nmap guessed the Operating System in the example below, so this OS Fingerprint matches an item in nmap’s database. If it fails to guess the OS, then your stack identity is unknown to nmap at this time, which is good, until somebody submits the signature to the database. If it guesses incorrectly, the device’s signature matches closely with some other device and therefore the attacker will not be able to identify the device until the device’s fingerprint is added to the database.

5.2 Technique Two - Through ICMP

The remote operating system can also be reliably guessed by the differences in the ICMP replies of various operating systems. Xprobe is the tool [Look at Table 4 for sample run] we recommend for the identification of a device based on ICMP replies. It is based on research by Ofir Arkin and Fyodor. The corresponding Phrack article can be found at <http://www.sys-security.com/archive/phrack/p57-0x07>

Sample Run of Nmap for OS Detection

Root access is required. The -O (Note: this is a capital O) flag tells it to do OS detection, and the two -v flags tell it to be very verbose. The resulting output (from which we've removed some output lines)

Starting nmap V. 3.10ALPHA4 (www.insecure.org/nmap/)

Host 1.2.3.4 appears to be up ... good. Initiating SYN
Stealth Scan against 1.2.3.4 Adding open port 80/tcp

The SYN Stealth Scan took 0 seconds to scan 1605 ports.
For OSScan assuming that port 22 is open and port 1 is closed
and neither are firewal led
Interesting ports on 1.2.3.4: (The 1597 ports scanned but not
shown below are in state: closed)

Port	State	Service
22/tcp	open	ssh
25/tcp	open	smtp
80/tcp	open	http
111/tcp	open	sunrpc
113/tcp	open	auth
981/tcp	open	unknown
3306/tcp	open	mysql
6000/tcp	open	X11

Remote operating system guess: Linux Kernel 2.4.0 - 2.5.20 OS
Fingerprint: TSeq(Class=RI%gcd=1%SI=4A13F1%IPID=Z%TS=100HZ)
T1(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(Resp=Y%DF=Y%W=0%ACK=0%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=0%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=C0%IPLen=164%RIPTL=148%RID=E%RIPCK=E%UCK=
E%ULEN=134%DAT=E)

Uptime 73.042 days (since Tue Jan 14 12:41:51 2003) TCP Sequence
Prediction: Class=random positive increments
Difficulty=4854769 (Good luck!) TCP ISN Seq. Numbers: E0D8A353
E19D23D3 E0EC0DB9 E1006998 E155631A E13C562C IPID Sequence
Generation: All zeros

Nmap run completed -- 1 IP address (1 host up) scanned in 4.644 seconds

Table 3: Run of Nmap for OS detection

Sample Run of Xprobe

```
# xprobe 1.2.3.4
X probe ver. 0.0.2
-----
Interface: eth0/a.b.c.d

LOG: Target: 1.2.3.4
LOG: Netmask: 255.255.255.255
LOG: probing: 1.2.3.4
LOG: [send]-> UDP to 1.2.3.4:32132
LOG: [98 bytes] sent, waiting for response.
FINAL:[ 3Com SuperStack II Switch SWNBBSI-CF,11.1.0.00S38
Nokia IPSO 3.2-2.3.1 releng 783-849
Ricoh Aficio AP4500 Network Laster Printer
Linux 2.0.x/2.2.x/2.4.x
Shiva AccessPort Bridge/Router Software V.2.1.0 ]
```

Table 4: Sample run of XProbe

Fingerprinting Using Telnet

```
# telnetfp 1.2.3.4
telnetfp0.1.2 by palmers / teso
DO: 255 251 1 255 251 3 255 253 24 255 253 31
DONT: 13 10 13 10 85 115 101 114 32 65 99 99 101 115 115 32
86 101 114 105 102 105 99 97 116 105 111 110 13 10
Found matching finger print:
Warning: fingerprint contained wildcards! (integrity: 50)
probably some cisco
```

Table 5: Sample run of Telnetfp

5.3 Technique Three - Through Application Fingerprinting

Differences in the implementation of various applications can be used to fingerprint the operating systems that they are running on. Tools exist for fingerprinting Telnet, `Telnetfp` for example [See Table 5 for sample run], which uses difference in the option negotiation in Telnet protocol to fingerprint various operating systems.

Though we did not find any tools on other applications, various other application layer protocols including IPSec can be used for fingerprinting devices and stacks.

5.4 Mitigation Techniques

Usually OS identification is not an attack by itself, unless the device is a Firewall, in which case it matters a lot whether or not the attacker can identify it. For all other devices, the product teams should focus on “hardening the device” rather than “hiding the device.” If the product team decides to expend effort on hiding a device they would make changes should be made to the stack code, changing how it responds to various packets in various Nmap tests. Changes can be made so that it matches some other device in the database. There is a possibility that changes in the stack code will break some other functionality of the device.

TCP ISN Analysis Using Nmap

Options used here are:

'-O' - The OS identification scan.

'-vv' - The verbose option is used twice here. Nmap only reports ISN randomness if at least one 'v' verbose option is specified. This will result in output like the following:

```
# nmap -Ovv 1.2.3.4
```

```
Remote operating system guess: Cisco router running IOS 12.1.5-12.2(6a) TCP Sequence Prediction: Class=truly random Difficulty=9999999 (Good luck!)
```

or like this:

```
Remote operating system guess: Windows Millennium Edition (Me), Win 2000, or WinXP TCP Sequence Prediction: Class=random positive increments Difficulty=11443 (Worthy challenge)
```

or like this:

```
Remote operating system guess: SonicWall SOHO firewall, Enterasys Matrix E1, or Accelerated Networks VoDSL TCP Sequence Prediction: Class=64K rule Difficulty=1 (Trivial joke)
```

Table 6: Sample outputs of NMap for various levels of randomness of ISNs

Additionally there are numerous other ways that the attacker can guess what he is talking to, like characteristic open ports, data in the packets sent, or proprietary implementation of protocols. So its almost impossible to hide a network device, particularly an access device.

6 TCP Initial Sequence Number Randomness

One of the potential weaknesses of the TCP protocol is whether the Initial Sequence Numbers (ISN) are guess-able. If an attacker can guess ISNs, they can blindly reset one side of the connection and possibly hijack existing connections. For a detailed description of this problem, and some solutions, refer to RFC 1948. Using cryptographic protocols for TCP channel encryption is not 100% foolproof. There are tools out there which can do Man in the Middle attacks on SSH and SSL if public certificates are not used. It should be noted that *all* IP interfaces on a product should be tested, as they may be based on different stacks. An example of this is the internal and external interfaces of a firewall or a device which has a dedicated management interface different from its IP stack interface. Also note that even if the ISNs are not easily guess-able, they can be used to fingerprint various operating systems.

6.1 Technique One - Using Nmap

The easiest method for knowing about the randomness of the Sequence numbers is to use nmap's -O option, which we met earlier in the OS identification section. Various examples are in Table 6

6.1.1 What the Output Means

Nmap tries to guess the operating system based on the patterns in the ISN numbers. It puts devices into various categories, whether the ISNs are truly random or they are incremented randomly or incremented by a constant number.

6.2 Technique Two - Using Hping and GNUPlot

This technique involves actually collecting the ISNs from a range of ports and plotting them to see any visible patterns in them. The following is a generic procedure for collecting the ISN samples:

1. Make a SYN request to a particular port
2. Collect the ISN from the SYN-ACK packet we got back
3. Send a RST and terminate the connection.

The following is a specific method using Hping and GNUPlot.

Step 1 Use Hping to collect the Initial Sequence Numbers.

```
# hping2 -I <interface> 1.2.3.4 -i u100000 -c 500 -Q -p 23 -S
```

Options used here are:

'-I' - Specifies the interface to use (often eth0)

'-i' - u100000 Specifies waiting 100000 microseconds (note the 'u') between packets

'-c 500' - Send 500 packets

'-Q' - Collect sequence numbers

'-p 23' - Target port 23 (in this case)

'-S' - Set the SYN flag.

Step 2 - Use GNUplot to plot the ISNs Gnuplot is a program that can generate graphical plots from data files. In this example, we show the output of a script [Appendix A] that plots a file (like the seqnums.txt generated in Step 1). It creates a png file, which can be viewed with various tools under Windows or Linux.

6.2.1 Sample PNG Plots

We collected three sample of sequence number plots to show what randomness means.

6.3 What the Output Means

If the Sequence numbers are easily predictable, say incremented by a fixed amount, its very bad and the PRNG should be replaced immediately. If there are patterns but it requires “lots” of packets to predict the next Sequence number, its tolerable.

6.4 Mitigation Techniques

Good random number generators are needed for generating the TCP ISNs. Depending on the device, there may be several sources of entropy, including time, temperature, and process IDs. Good references for generating software based PRNGs are at [Matreya-PRNG], [TMatthews-PRNG]

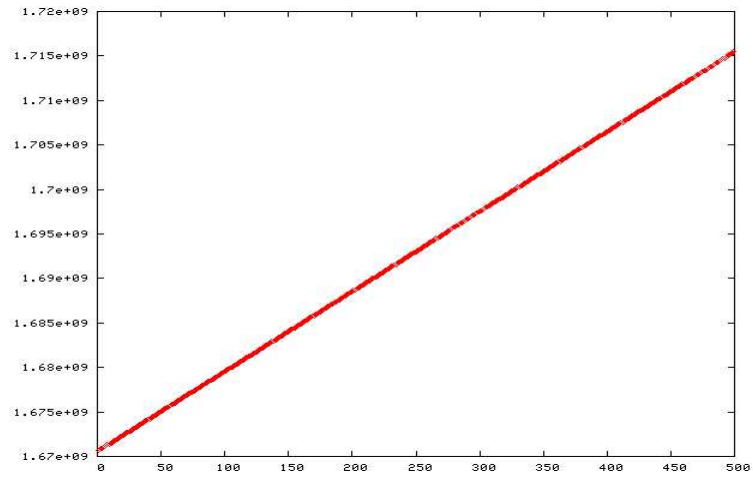


Figure 1: Very Bad Sequence number plots

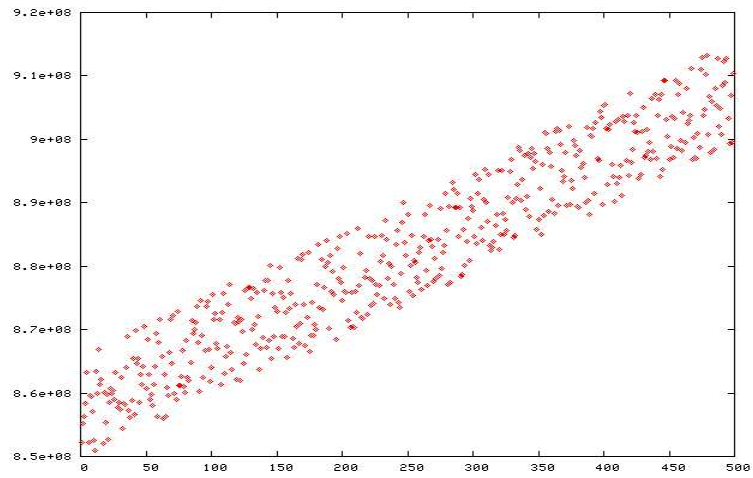


Figure 2: Bad Sequence number plots

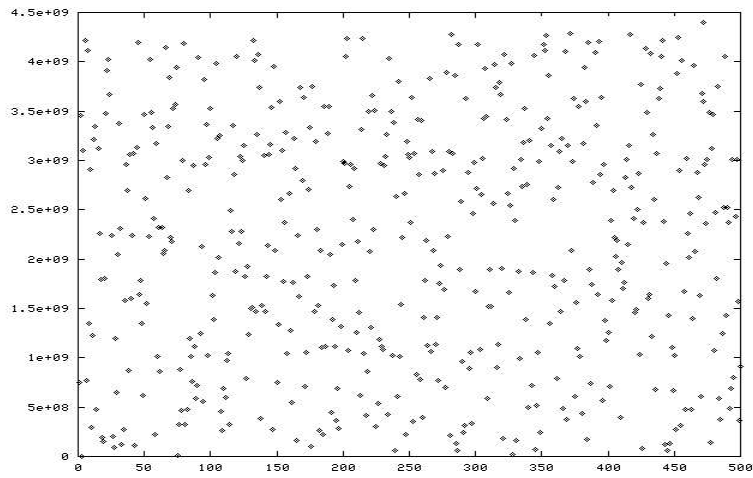


Figure 3: good Sequence number plots

7 Scanning

Scanning is a classic technique for gathering information about the services on a remote TCP/IP stack. Various techniques typically involve sending packets to a target of interest, and analyzing the responses or lack thereof.

The information about TCP ports is pretty dependable because it is connection oriented and the analysis is based on the responses that come back. The UDP port information and protocol information depends on false positives and is not dependable.

We recommend the following for all types of scans

- Conduct the scans on the whole port range, 1-65535. Devices using proprietary protocols generally use arbitrary high ports for communication.
- Conduct scans on and on all the interfaces present on the device. There is a possibility that multiple interfaces are running multiple stacks.

The following metrics should be continuously monitored during scanning.

- Pings to the device should be continuously monitored. Since the rate of packets involved in scanning is small, there should be no impact on pings.
- CPU and memory should be continuously monitored. Again, since the rate is small there should be minimal impact on CPU and memory.

7.1 TCP Scans

TCP port scanning aims to find open TCP ports. While SYN scans are the common method of finding open TCP ports, ACK and FIN scans will be useful when the SYN packets are filtered (most commercial Firewalls do this).

7.1.1 SYN Scans

This scan involves sending a SYN packet to a given port or a set of ports. If the port is open, a SYN-ACK packet will come back. If it is closed, a RESET packet will come back. If the packet is filtered by a firewall, nothing will come back.

What the Output Means Each port which NMap reported as “open” has returned a SYN-ACK packet back for the SYN packet sent. So, there is a significant chance that services are listening behind those ports.

Mitigation Techniques

1. Analyze whether each port that is reported to be open is required to be open on that particular interface and also whether or not it is required to listen “externally”.
2. All unneeded services should be stopped and the corresponding ports should be closed.

7.1.2 ACK Scans

Some Firewall based devices generally silently drop SYN packets addressed directly to them. If that device’s stack is dropping just the SYNs and letting packets with other flags up the stack, an ACK scan will help analyze that. The scan involves sending an ACK packet. If a RST comes back, we can safely assume that the port is not “blocked”. And if nothing comes back, we can assume that the machine (or any device sitting before the machine) is silently dropping all the packets.

What the Output Means If the ACK scan reports a port as “closed” that means the port returned a RST and if the ACK reports a port as “filtered”, that means it did not get any reply back for the ACK packet sent.

Technique - Using port scanner NMap

This example uses the following command line arguments:

'-P0' - Does not first try to ping a host before scanning. This is to ensure that even if a device is configured not to answer ICMP Echo Replies, our scan will still go ahead.

'-sS' - The type of scan is a SYN scan

'-p 1-65535' - This will scan the entire TCP port range.

```
# nmap -n -vvv -sS 1.2.3.4 -p 1-65535
```

```
Starting nmap V. 3.10ALPHA4 ( www.insecure.org/nmap/ )
```

```
Host 1.2.3.4 appears to be up ... good. Initiating SYN  
Stealth Scan against 1.2.3.4 Adding open port 3306/tcp
```

```
Adding open port 113/tcp
```

```
Adding open port 6000/tcp
```

```
Adding open port 80/tcp
```

```
Adding open port 22/tcp
```

```
Adding open port 25/tcp
```

```
Adding open port 111/tcp
```

```
Adding open port 981/tcp
```

```
The SYN Stealth Scan took 22 seconds to scan 65535 ports.
```

```
Interesting ports on 1.2.3.4:
```

```
(The 65527 ports scanned but not shown below are in state: closed)
```

Port	State	Service
22/tcp	open	ssh
25/tcp	open	smtp
80/tcp	open	http
111/tcp	open	sunrpc
113/tcp	open	auth
981/tcp	open	unknown
3306/tcp	open	mysql
6000/tcp	open	X11

```
Nmap run completed -- 1 IP address (1 host up) scanned in 22.598 seconds
```

Table 7: NMap SYN Scan

Sample run of NMap ACK scan

Notice that all ports are 'UN-filtered', so RSTs came back from all the ports and therefore they are not silently dropped.

```
# nmap -nsA -vvv 1.2.3.4 -p 1-65535

Starting nmap V. 3.10ALPHA4 ( www.insecure.org/nmap/ ) Host
1.2.3.4 appears to be up ... good.

Initiating ACK Scan against 1.2.3.4

The ACK Scan took 161 seconds to scan 65535 ports.

All 65535 scanned ports on 1.2.3.4 are: UNfiltered
```

Table 8: NMap ACK Scan

Mitigation Techniques Drop all the packets from unwanted machines, so that the attackers will not be able to deduce anything through scanning, except that the device is silently dropping any packet that is sent from him.

7.2 UDP Scans

The UDP scan involves sending a 0 byte UDP packet to the specified ports and we know the port is closed if we get ICMP port unreachable back. NMap assumes that the port is open if it did not get anything back. The scan is based on lack of responses and therefore will result in many false-positives.²

There are various other stand alone UDP scan programs and modules from well-known tools like SATAN and SARA supposedly having better timeout mechanisms, which we are not familiar with. We intend to update the document when we have more experience with those tools.

7.2.1 What It Means

If the scan returns only a few open ports, chances are that there are services behind those ports. If the scan reports all of the ports as open ports then chances are that the stack did not respond to any of the UDP packets and that is a false positive.

We recommend to still do UDP scans because of the following reasons:

1. UDP scans are the easiest way to find out important protocols like IKE and ubiquitous protocols like DHCP
2. It also tests whether the implementation is robust enough to take care of a NULL UDP packet.

7.2.2 Mitigation Techniques

The following techniques will be useful in fighting UDP scans

1. Silently drop and do not return anything to UDP ports that do not have any services list

²Note that the UDP scans are generally very slow, particular when the whole range of ports. The readers can use '-T 5' option to increase the speed, but we caution that some open ports might be missed.

```

                                Sample UDP Scan Run

# nmap -nsU 1.2.3.4

Starting nmap V. 3.10ALPHA4 ( www.insecure.org/nmap/ )
Interesting ports on 64.101.184.183:
(The 1460 ports scanned but not shown below are in state: closed)
Port      State      Service
123/udp   open       ntp
135/udp   open       loc-srv
137/udp   open       netbios-ns
138/udp   open       netbios-dgm
445/udp   open       microsoft-ds
500/udp   open       isakmp
1030/udp  open       iad1
1900/udp  open       UPnP

```

Table 9: NMap UDP Scan

2. Implement application based ACLs and return a valid response only when the source address passes the ACLs. This is not fool proof because UDP is connectionless and source address can be spoofed, but it offers enough protection.
3. Make sure that the data of the UDP packet is not NULL before processing the data part.

7.2.3 Miscellaneous Scans

FIN Scan, XMAS Scan and NULL Scans The FIN, XMAS and NULL scans are useful for operating systems which ignore SYN requests and might let packets with FIN and weird flags go through. These scans are based on negative results. The device stack should be monitored for any abnormal before while conducting these scans

Fragmented Scans Nmap's `-f` option allows the user to fragment various scans. We recommend to launch these scans both as a means of getting through the device and also for any abnormal behavior on the part of the device.

7.3 Protocol Scans

Protocol scans are used to find what IP protocols are implemented on the stack. The scan involves sending a zero data IP packet, looping through all the IP protocol numbers. A sample NMap protocol scan is shown in Table 10

What the Output Means If nmap does not get a ICMP protocol unreachable back, it reports the protocol as "open". Usually if a small number of protocols are reported open, there is a significant chance that those protocols are implemented on the device. If all of the protocols are reported as open, then the device is probably "blocked".

8 TCP Resource Exhaustion

TCP is a connection-oriented protocol, and thus must keep track of the state of any valid connection request that it gets³. Generally, this is done by storing queues of connections that are in each state. To an attacker,

³There is an excellent diagram explaining this in Figure 18.12 of [Stevens-TCP/IP]

Sample Run of NMap Protocol Scan

```
# nmap -ns0 -vvv 1.2.3.4
Starting nmap V. 3.10ALPHA4 ( www.insecure.org/nmap/ )

Host 1.2.3.4 appears to be up ... good. Initiating IPProto Scan against 12.3.4
The IPProto Scan took 246 seconds to scan 255 ports.
Adding open port 2/udp
Adding open port 1/udp
Adding open port 6/udp
Adding open port 17/udp
Interesting protocols on 1.2.3.4:
(The 251 protocols scanned but not shown below are in state: closed)

Protocol  State      Name
1         open       icmp
2         open       igmp
6         open       tcp
17        open       udp

Nmap run completed -- 1 IP address (1 host up) scanned in 246.727 seconds
```

Table 10: Example Nmap Protocol Scan

these queues are resources that can be attacked. If these queues have no limits, then an attacker can keep adding connections until memory or disk space are exhausted. If the connections have memory limits, then an attacker can fill them up, denying the ability to make legitimate connections. These types of attacks are described in a Bindview white paper [Keys-Naptha].

8.1 Various States and How Can They be Attacked

Although a TCP has 11 possible states, there are 6 of these that are dependent on receiving a packet from the initiating client, and therefore vulnerable to attacks.

We will describe these in the order in which they normally come in a connection.

SYN_RECV state This is the state after the server has received a SYN, and responded with a SYN_ACK. The only ways out of this state are to receive an ACK, or to time out and start a close by sending a FIN. An attacker can send a stream of SYN packets (called a SYN flood attack), and refuse to send the ACK the server expects. In the absence of a timeout (or a less aggressive timeout), this can fill up the SYN_RECV queue indefinitely and block new connections. A pseudo example of testing various aspects of SYN_RECV state is described in Table 11

ESTABLISHED state This is the state after the server receives an ACK, completing the 3-way establishment handshake. At this point, the server is waiting for either data from the client. The only ways out of this state are to receive a FIN from the client, or to time out and start a close by sending a FIN. The difference between this attack and the previous one is that the attacker DOES answer the SYN-ACKs with ACKs. But he then refuses to communicate further, thus filling up resources.

A pseudo example of testing various aspects of ESTABLISHED state is described in Table 12

LAST_ACK state When a server has a connection in ESTABLISHED state and it receives a FIN, it sends an ACK. The server application is then supposed to close, which sends a FIN to the client, and waits for the

corresponding last ACK. An attacker can implement this attack by sending a SYN flood, and responding to the SYN-ACKs with FIN-ACKs. When the server replies with a FIN and an ACK, the client refuses to send the last ACK packet. This fills up the LAST_ACK queue on the server.

A pseudo example of testing various aspects of LAST_ACK state is described in Table 13

The next 3 states that can be attacked assume that the server decides (through a timeout, for example) to close a SYN_RCVD or ESTABLISHED connection. The server application does this by issuing a close, which causes the stack to send a FIN packet. This would likely happen if an attacker had run a SYN flood or an ESTABLISHED state flood.

These attacks are more difficult to test, because the attacker can't force the server to send the FIN. However, if a stack is NOT sending FIN packets to close out connections, then it is doing one of the following:

1. Filling up queues
2. Arbitrarily deleting connections when its queues get full
3. Closing connections with RST packets

Option 1) allows attacks, while option 2) is a violation of the TCP standard. Option 3) is legal, but not very client-friendly. Attacks on any of the following states have a variation in the command sequence, depending on whether you are trying to force the client through the ESTABLISHED state or not. It is important to test each variation, because the server may choose to exit either the SYN_RCVD or the ESTABLISHED states with a RST instead of a FIN. In that case, you will never get to test its behavior any further in the state path.

FIN_WAIT_1 state This is the state immediately after the FIN is sent. The only ways out of this state are to receive an ACK, a FIN, or a FIN-ACK packet, or to time out. To test this, an attacker simply refuses to respond to FIN packets.

A pseudo example of testing various aspects of FIN_WAIT1 state is described in Table 14

FIN_WAIT_2 state If a server in FIN_WAIT_1 state receives an ACK for its FIN, it goes to FIN_WAIT_2 state. The only ways out of this state are to receive a FIN packet, or to time out. To test this, and attacker responds to FIN packets with an ACK, but refuses to send a FIN packet.

CLOSING state If a server is in the FIN_WAIT_1 state and it receives a FIN, it will send an ACK, and go to the CLOSING state, awaiting the final ACK. The only ways out of this state are to receive an ACK, or to time out. An attacker can test this by responding to a FIN with a FIN, but refusing to send the last ACK packet.

Example of testing FIN_WAIT2 and CLOSED states are described in Tables 15 and 16 respectively.

8.2 Test Technique using Naptha

The naptha [Keys-Naptha] tool suite consists of 2 tools, synsend, and srvr. synsend sends out TCP SYN packets as fast as it can. srvr sniffs the wire, listening for packets with certain specified TCP flags set, and responds to them with packets with another set of specified TCP flags set. The examples below will illustrate how these 2 tools can be used to implement the attacks described above.

We assume that the victim is 1.2.3.4, the attacker is 10.10.10.10, and the target port is 9876. Also, note that in some of the attacks, processes are put in the background. Be sure to kill off these processes between attacks.

For each of these attacks, it is necessary first for the attacking machine to firewall itself off from the victim machine, so that the attacker's stack does not nobly step in and respond to the victim's traffic (mostly with TCP RST packets). On Linux this is done via iptables, with this command:

```
# iptables -A INPUT -s 1.2.3.4 --protocol tcp --sport 9876 -j DROP
```

After firewalling the attack machine, `synsend` can be used to flood the victim machine with SYN packets. The `synsend` binary has options to specify source and destination addresses and the victim port. We can also tune the rate of SYNs that are sent. In the following example, 1.2.3.4 is our victim machine, 9876 is our destination (victim) port, 10.10.10.10 is our address and 1 is the sleep time in milli seconds between each SYN sent.

```
# synsend 1.2.3.4 9876 10.10.10.10 1
```

The `srvr` command will still be able to respond to traffic because it operates at link layer. Note that in the `srvr` command's options, capital letters (e.g. SA) indicate flags to match on the incoming packets, and lower case letters (e.g. fa) indicate flags to set on the outgoing packets.

The following example sends an ACK response for each and every SYN-ACK packet that it gets from 1.2.3.4.

```
# srvr -SAa 1.2.3.4
```

8.3 Applying Metrics

The resource attacks, as mentioned above depend on filling up states. Therefore the following should be monitored.

1. **Memory on the device:** The filled up states generally eat up memory, so it should be noted before the attack and should be compared against the memory during and a reasonable period after the attack to make sure there is no memory leak. Its catastrophic to have a memory leak and should be investigated immediately.
2. Connection tables on the machine. The connection table build up should approximately match up with the memory fall. Any stray connection table entry is catastrophic and needs to be investigated and fixed immediately
3. CPU on the machine. The CPU should not increase much during these attacks. If you see a CPU increase, that probably means that the stack passed on the control to the corresponding application. Any increase over 10% is abnormal for TCP resource exhaustion attacks and needs further investigation.
4. Responsiveness of the TCP stack. See the metrics section for more details.
5. Responsiveness of the application. The responsiveness should be observed both for an existing application and a new instance of an application.

8.4 Sample Runs

The following illustrates some sample runs of the above described attacks

8.5 Mitigation Techniques

The following design implementations would help to counter TCP resource exhaustion attacks

1. Accept a packet and store state only when its required. Have application and port based ACLs and return a packet only when the source address matches the ACLs.
2. Have size limits on all queues at all TCP states

SYN Flooding Attacks

SYN floods are done by sending a bunch of SYNs to the victim machine and by remaining silent without sending any further packets.

```
# iptables -A INPUT -s 1.2.3.4 --protocol tcp --sport 9876  
-j DROP
```

(Make attacker machine not to send any packets back)

```
# synsend 1.2.3.4 9876 10.10.10.10 1
```

(This sends an unending stream of unanswered SYNs.)

Table 11: SYN Flood Test

ESTABLISHED State Attack

This attack relies on both the tools, one to send SYNs and another to send ACK replies for all the SYN_ACKs the victim machine sends.

```
# iptables -A INPUT -s 1.2.3.4 --protocol tcp --sport 9876  
-j DROP
```

(Prevent attacker machine's stack from sending any packets back)

```
# synsend 1.2.3.4 9876 10.10.10.10 1
```

(Send continuous flood of SYNs. This is a continuous attack, so do this in another window or background it)

```
# srvr -SAa 1.2.3.4
```

(For all the SYN_ACK replies that you get, send a ACK)

Table 12: Testing ESTABLISHED state

LAST_ACK state attack

This attack relies on synsend, to send SYN packets and on srvr, to send FIN_ACK replies for all the SYN_ACK replies that it gets back

```
# iptables -A INPUT -s 1.2.3.4 --protocol tcp --sport 9876  
-j DROP
```

(Make attacker machine not to send any packets back)

```
# synsend 1.2.3.4 9876 10.10.10.10 1
```

(Send continuous flood of SYNs. This is a continuous attack, so do this in another window or background it)

```
# srvr -SAfa 1.2.3.4
```

(For all the SYN_ACK replies that you got, send a FIN_ACK)

Table 13: LAST_ACK state testing

FIN_WAIT_1 state attack

This attack looks EXACTLY like the ESTABLISHED state attack. The only difference is that the attacker is hoping for the victim server to start closing connections by sending FINs. The attacker will sit tight and let these FINs go unanswered.

```
# iptables -A INPUT -s 1.2.3.4 --protocol tcp --sport 9876  
-j DROP
```

(Make attacker machine not to send any packets back)

```
# synsend 1.2.3.4 9876 10.10.10.10 1
```

(Send continuous flood of SYNs. This is a continuous attack, so do this in another window or background it)

```
# srvr -SAa 1.2.3.4
```

(For all the SYN_ACK replies that you got, send an ACK. Readers should try both ways of getting to the state)

Table 14: Testing FIN_WAIT1 state

FIN_WAIT_2 state attack

```
# iptables -A INPUT -s 1.2.3.4 --protocol tcp --sport 9876  
-j DROP
```

(Make attacker machine not to send any packets back)

```
# synsend 1.2.3.4 9876 10.10.10.10 1
```

(Send continuous flood of SYNs. This is a continuous attack, so do this in another window or background it)

```
# srvr -SAa 1.2.3.4
```

(For all the SYN_ACK replies that you got, send an ACK. Readers should try both ways of getting to the state)

```
# srvr -Fa 1.2.3.4
```

(If and when the server replies with a FINs, ACK them. As in the FIN_WAIT_1 attack, this attack has another variant which leaves out the "srvr -SAa 1.2.3.4 &" command. You should test both)

Table 15: Testing FIN_WAIT2 state

CLOSING state attack

```
# iptables -A INPUT -s 1.2.3.4 --protocol tcp --sport 9876  
-j DROP
```

(Make attacker machine not to send any packets back)

```
# synsend 1.2.3.4 9876 10.10.10.10 1
```

(Send continuous flood of SYNs. This is a continuous attack, so do this in another window or background it)

```
# srvr -SAa 1.2.3.4
```

(For all the SYN_ACK replies that you got, , send an ACK. Readers should try both ways of getting to the state)

```
# srvr -Ff 1.2.3.4
```

(Answer all FINs with a FIN. As in the FIN_WAIT_1 attack, this attack has another variant which leaves out the "srvr -SAa 1.2.3.4 &" command. You should test both.)

Table 16: Testing CLOSE state

3. Have appropriate timeouts on all connections in each of the queues, so that if they are filled, they can eventually be cleared out.
4. The corresponding timeouts should be made more aggressive at the initial stages of the connection and lenient at later stages, particularly after authentication.
5. For stacks like the stack of a web server, where it should accept connection from any address, implement various SYN flood mitigation techniques like TCP Intercept or SYN Cookies [D. J. Bernstein-Syncookies].

9 Robustness of the Stack Against Malformed packets and Flooding

Due to improper parsing of TCP/IP headers and data, there are known vulnerabilities where a well-crafted (or random) TCP/UDP packet with "abnormal" header values can make the IP stack unusable and can even crash the machine. If a product's stack cannot handle malformed packets, rapid floods of packets, randomly fragmented packets, or the like, then it may crash. To an attacker, a crash may be the end goal, or may be a place to investigate further to see if the crash involves, for example, a buffer overflow that can be used to gain access to the machine.

9.1 Introduction

We don't have a tool that checks all the possible values of the TCP/IP headers to possibly exhaust all the possible "odd" packets. The best tool suite for checking this is ISIC [Frantzen-ISIC]. ISIC stands for IP Stack Integrity Checker, and consists of ESIC (for generating Ethernet frames), ISIC (IP frames), TCPSIC (TCP frames), UDPSIC (UDP frames), and ICMPSIC (ICMP frames). Each tool has several options, allowing you to specify things like packet length, IP protocol IDs, version numbers, bad checksums, and the percentage of fragmented frames. The tools can generate as many as 15,000 packets per second.

9.2 Applying Metrics

It is very important to monitor network and application responsiveness, as well as CPU and memory usage during these attacks. It also makes sense to look at various connection tables on the device, and examine whether or not various “random” packets that we are sending are able to create any connections or able to make the device store any kind of state.

During these attacks, the stack can be easily overwhelmed, just by the sheer volume of packets that it has to process. Depending on the performance specifications for the device, this may not be considered as a “significant impact” if the stack “comes back” immediately after the flooding is stopped. These kinds of scenarios can be analyzed further by changing the number of packets per second that the tool sends out. When the tool is being used with high speeds, it is quite common for our stack metrics, including ping responses, TCP responses, and applications to become quite slow.

The key metric is whether and how quickly the stack recovers when the attack is stopped.

Overall, the following would help analyze the impact of these attacks

1. **The ICMP echo latencies:** Its tolerable to have large ICMP echo latencies while flooding, but continued slow responses after the attack is very bad and needs to be further investigated
2. **CPU Increase:** Its tolerable to have increase in CPU while flooding but if the CPU drops immediately after the flooding is stopped, its OK
3. **Memory Fall:** Most of these packets are random packets, any fall in memory during the attacks is very bad and needs to be investigated

9.3 How To Use ISIC Suite

The following are the options for various tools in the suite that will be useful in the evaluation.

'-m' - Specifies the maximum number of KB/sec to generate. At the default maximum speed, packets can get dropped, so slowing down the attack can generate different results. Note that the minimum speed we have seen the tool run is about 50 packets per second, no matter how low you set the -m flag.

'-r' - Specify the random seed. We recommend picking a fixed random seed (such as 1) in order to be able to retest with the exact same sequence of packets. Although the data will be random, at least they will not vary between runs of the test, so the results can be more faithfully reproduced.

'-s' - Specify the source address (either ethernet or IP). The word 'rand' indicates this should be random.

'-d' - Destination address

'-F 40' - Specify the percentage (in this example, 40%) of all packets that will be fragmented

-V - Specify the percentage (0-100) of all packets that will have bad IP versions

-I - Specify the percentage (0-100) of all packets that will have random IP options

1. The tool should be run at full speed with no malformed options (0%) to test the impact on the stack to flooding

2. The ISIC tool should be run with no malformed versions and options and with fragmentation and less speed to test the impact of fragmentation on the stack
3. The ISIC tool should be run with either each of the malformed options or all malformed options at a time at both lesser speeds and full speeds
4. The TCPSIC tool should be run the same way as ISIC on all open TCP ports
5. The UDPSIC tool should be run the same way as ISIC on all open UDP ports. If there are no UDP ports open, the tool should be run with random ports so as to determine the impact of random UDP packets.
6. ICMPSIC should be run the same way as ISIC
7. ESIC should be run with random source and destination MAC's and with the destination MAC of the target machine.

Examples of various ISIC tools are shown in Table 17

Sample command line runs of various tools in ISIC suite

Ethernet

```
# esic -r 1 -i eth0 -s rand -d 01:02:03:04:05:06 -p rand
```

Options used in addition to those explained above are:

'-i eth0' - Specify the interface to use.

'-p rand' - Randomize the protocol field

IP

```
# isic -i eth0 -r 1 -s rand -d 1.2.3.4 -F 50 -V 50 -I 50
```

TCP

TCPSIC should be run against each of the ports that were discovered in the TCP scanning process.

```
# tcpsic -r 1 -s rand -d 1.2.3.4,23 -T 50 -u 50 -t 50
```

Options used in addition to those explained above are:

'-d 1.2.3.4,23' - Send to port 23 on the target

'-T 50' - 50% have random TCP options '-u 50' - 50% have the TCP Urgent flag set

'-t 50' - 50 % have bad TCP checksums

UDP

UDPSIC should be run against each of the ports that were discovered in the UDP scanning process.

```
# udpsic -r 1 -s rand -d 1.2.3.4,69 -F 40 -U 30
```

Options used in addition to those explained above are:

'-d 1.2.3.4,23' - Send to port 69 on the target

'-U 30' - 30 % have bad UDP checksums

ICMP

```
# icmpsic -r 1 -s rand -d 1.2.3.4 -I 50
```

Options used in addition to those explained above are:

'-I 30' - 30 % have bad ICMP checksums

Table 17: How to run various ISIC tools

9.4 Mitigation Techniques

There is no silver bullet mitigation technique against either malformed packets or flooding. Care should be taken at the design stages so that illegitimate packets are ignored at the lowest possible level in the stack. Even though stack flooding is a cat and mouse game, high speed network processors with high speed lookup tables placed as the first line of defense on the ingress side will definitely mitigate the issue.

Protocol coders should take care that each and every header field is valid at each level before passing on the data part to the next level.

10 Classic TCP/IP Attacks

The following are some of the exploits that used specific crafted malformed packets that have historically crashed many TCP/IP implementations. Though almost all of them are fixed and standard operating systems are not vulnerable to these exploits anymore, it's very important to run these exploits against the stack being tested.

10.1 Land Attack

The exploit involved sending a SYN packet with the same source and destination IP addresses. The original posting and discussion are located at <http://www.insecure.org/spl0its/land.ip.DOS.html>

10.2 Teardrop

The teardrop exploits crashed various old Windows OS and Linux OS by sending overlapping IP fragments with confusing offset values.

10.3 Smurf

Smurf attack involved sending ICMP Echo Request packets to broadcast destination addresses causing lots of replies and network congestion. A good description is at <http://www.cert.org/advisories/CA-1998-01.html>

10.4 Jolt

Jolt attack involved sending fragmented ICMP packets. This exploit primarily affected Windows machines.

10.5 Ping of Death

The ping of death exploit involved sending IP packets with data more than the usually allowed 65535 bytes.

10.6 Technique One - Using C code and Targa2

The exploit code for each of the exploits is available in all the well-known security sites, and Targa2 is a tool which has combined most of the above exploits into a single code base. The Targa2 code can be downloaded at <http://packetstormsecurity.nl/> through a simple search.

10.7 Technique Two - Using Nessus and NASL

Nessus, a general purpose vulnerability scanning tool has scripts in NASL scripting language for easier testing these classic and well-known exploits.

The corresponding scripts (called plugins for Nessus tool) can be searched at <http://cgi.nessus.org/plugins/>

Typical Command Line Arguments for Strobe

```
% strobe -b 1 -e 65535 -L 15 -T 2 <target_host>
```

The flags used here are:

-b beginning port number

-e ending port number

-L Number of lines to capture (useful for capturing multi-line banners)

-T Timeout (useful in combination with L, in the likely event that there are fewer than the number of lines specified).

Table 18: Strobe - sample arguments

Howto Run Stand-Alone NASL Scripts Here is the pseudo procedure :

1. Get the plugin code (say blah.nasl) and remove the description part until `The script code starts here.`
2. Get the NASL binary and if 1.2.3.4 is the IP address of the victim machine, run the binary as `nasl blah.nasl -t 1.2.3.4`

11 Miscellaneous

11.1 Application Banners

Many applications give away far too much information to unauthorized users. It is quite common for applications such as telnet, ftp, and mail servers to be branded with not only their own version numbers, but also the name and version of their underlying operating system.

A good example of this is ftp, which can tell you things such as: `220 VxWorks (5.4)`

By Telnetting Into The Device The most reliable method for getting a banner from a port is to use telnet, as in this example: `% telnet 1.2.3.4 25`

This example will get the banner from port 25 on the target host (1.2.3.4).

Automated Tool - Strobe Strobe is an automated tool for grabbing banners from multiple ports is strobe. However, we have discovered some reliability problems in this tool.

11.1.1 Metrics and Impact

Application banners not only typically reveal the information about the device, but also about the versions of various applications and services. Attackers have more information now about the services, can easily compromise the system if any of those services are not patched for known vulnerabilities. We recommend not to put any device or software information in application banners.

12 Developing a Generic Evaluation Strategy

IP is being put into every device imaginable for connectivity and cost saving purposes, and the architecture of these embedded devices is very different from that of PC based devices. Most of them employ IP stack on a chipset and have little CPU power. Most of these devices have different hardware and software components with varying ways of inter component communication. These also might have different memory regions and CPUs that they drag the power from, which needs to be analyzed for developing appropriate metrics to measure the impact of the attacks.

In this section, we try to help develop a generic strategy, for security testing of various types of devices, independent of their architecture and protocols that they are running. The first task to do is to analyze the various components and architecture of the device noting down the mechanisms that the components use to interact with each other and with the outside world. The next most useful task is to trace out a packet from where it enters the device to where it leaves the device, noting down various bottle necks and potential vulnerable points. The other parallel task is to start developing metrics which will help to measure the impact of the attacks

12.1 Analyzing Various Components and Architecture in the Device

Various components of the device that are needed for the device to perform its core functionality need to be identified. These components may be hardware components that are usually modules that are plugged in into a back plane or can be software-based components. Understanding how those components interact with each other will help immensely while identifying the points of attack. A single component may act as a supervisor component and other components may communicate through the supervisor component, or all the components might be peers and communicate through APIs or inter process communication or through a common shared memory location.

These components might have a single resource location, a single CPU or have multiple CPUs for each component. Likewise, each component may share a single flat memory or the memory may be divided into boundaries. The knowledge of these issues will help in resource consumption attacks, testing whether each component may consume more resources than its supposed to and whether it can write and read from each other memory locations.

12.2 Following the Life of a Packet

Tracing out a packet through various components of the device will help in identifying various attack locations and also will help out in narrowing out attack packets that might cause trouble. The packet should be followed from the time it hits the ingress interface, while it travels through various components, until it gets out through the egress interface or is processed by an application running on the device. The reader might have to choose multiple packets so that different tracings would cover all components.

12.3 Developing Metrics

The two primary components in developing metrics are identifying protocols to measure the impact at various layers and the resources that various components that the components drag their oil from.

12.3.1 Metric at Each Layer

A protocol should be identified at each layer of the protocol stack whose performance can be used to measure the impact of attacks at that layer. If packets at various layers are processed by different components, a system vitals (CPU/memory) on that component should be used to measure the impact on the component. Here is the ground rule in analyzing metrics: If a metric at level or layer(say layer A) indicates a problem, then the metrics at both a lower layer (A-1)and a upper layer(A+1) should be checked out. If the metric at the A-1 layer seems fine, then the impact is starting at A layer. If the metrics at A-1 layer show an impact, then the impact on the metrics at layer A might be just a ripple effect from layer below and therefore the metrics at one more layer below(A-2) should be checked out.

12.3.2 Resource Consumption by Each Component

Resources used by various components need to be identified and measured to measure the impact on each component. These include hardware resources like CPU/memory and software based resources like state tables.

12.4 Developing Attack Locations and Metrics - Through an Example

As a means of going through a pseudo evaluation, we take a very simple fictitious device and put up some examples of evaluation tasks. This device basically sticks on to a surface and can read the temperature and set the temperature of the surface that it is sitting on. It has a sensor component, which calculates the temperature, it has a management component from which a human can read the actual digits. It also has the capability to encapsulate the temperature data in Ethernet packets and send it over Ethernet to other devices using some protocol. It also has the ability to accept packets using the same protocol and adjust the temperature of the surface based on the data in the packet. The first step that the testing team needs to do is a complete analysis of the architecture.

This is a very simple and superficial architecture analysis that may serve as an example: This device consists of a data bus on which a supervisor component is plugged in. There is an IP component, which is a separate hardware, running some OS and is plugged into the supervisor component. The temperature sensor uses some API whenever it needs to talk to the IP component using an API. The management component runs on the IP component as a process and talks to the IP component using inter process communication.

The temperature sensor reads the temperature and uses an API to send or receive raw temperature data to the IP component. The IP component encapsulates the data into an IP packet, encapsulates a proprietary protocol header and puts the packet on the data bus. The supervisor component takes the packet from the bus and sends it through the egress interface. The management process uses a different protocol and uses the IP component for corresponding encapsulation and de-encapsulation.

On the ingress side, its symmetric, the supervisor component takes the packet from the interface and puts it on the bus. The IP component takes the packet, de-encapsulates the proprietary protocol headers and uses the API to send the data to the temperature sensor.

The supervisor component has the CPU and flat memory and all other components use the same CPU and memory. The supervisor component will also keep track of a packet going out and keeps it in memory until a “temperature set” packet comes in.

Generalizing the above methodology here are the concerns that the testing team needs to think about:

- Whether any of the protocols that are implemented on the device give out information without authentication which makes it to uniquely identifiable
- Whether any of the protocols are keeping state for any unauthenticated packet, if so whether each of those states have reasonable timeouts
- To what state can an un-authenticated attacker push the each of protocol states
- How are each of the protocol states dealing with flooding of packets of that state, with a latter state or a former state
- How does each of the protocol states dealing with malformed packets

In the above example, therefore, here is an example analysis of packet flow on the ingress side that the testing team want to follow in the process of develop a testing strategy:

1. The first component that picks up the packet is the supervisor component. It is not examining the packet, it is just putting the packet on bus. So the sending malformed packets should not have any impact on the supervisor module. The only thing that might affect the supervisor module is the number of packets that it has to pick up. The resource that it uses while it is picking up the packets is the CPU, so the CPU should be continuously monitored. The supervisor module is removing the stored state after it sees an ingress packet so a slight decrease in memory should be observed. Because it does deal with memory functions, it would make sense to keep memory as one of the metric along with the CPU. If it would not have keep state, observing just the CPU should have been sufficient.
2. The next component to gets to process the packet is the IP component. It is examining the packet, looking at its header, removing the corresponding header, possibly storing the header information.

So testing both the ability of the component against huge number of packets and malformed packets makes sense. The IP component is examining the just the IP header part, so the header to fuzz is IP header. It should not examine any latter part of the packet, so there is no need to fuzz it. The impact should be on the resources used by the IP engine. In this case it is using the same CPU/memory as the supervisor, so the possibility that the IP component taking up all the resources needs to be tested. If it were using different CPU/memory, the differences in the CPU/memory used by the supervisor and the IP component for the same set of packets will help in narrowing down the component in which the problem exists. For example for the same rate of packets if supervisor's CPU usage is the more as IP's usage, then supervisor has a problem because IP is getting all the packets that the supervisor is getting and also IP has to process these packets and supervisor does not.

3. The next component that gets to process the packet is the temperature sensor. Flooding the packets should have an impact on the sensor. The part to fuzz for testing against the malformed packets is the API data, the packets are valid IP packets with malformed data.

After this initial analysis and testing, individual protocols should be analyzed against security concepts like authentication, authorization, integrity, confidentiality etc. Each of these protocol evaluation is a comprehensive evaluation by itself, which is outside the scope of the document.

13 Acknowledgements

The authors want to thank the whole STAT and CIAG teams (especially Matt Franz) for their inputs and support.

References

- [Fyodor-OSFingerprint] Fyodor, "Remote OS detection via TCP/IP Stack FingerPrinting,<http://www.insecure.org/nmap/nmap-fingerprinting-article.html>
- [NS-Cyberspace] National Strategy for security cyber space http://www.dhs.gov/interweb/assetlibrary/National_Cyberspace_Strategy.pdf
- [Fyodor-Scanning] Fyodor, "The Art of Port Scanning" http://www.insecure.org/nmap/nmap_doc.html
- [Stevens-TCP/IP] Stevens, W. Richard, TCP/IP Illustrated, Volume 1, Addison Wesley, 1999, Reading, MA.
- [Morris-TCP/IP] R.T. Morris, "A Weakness in the 4.2BSD UNIX TCP/IP Software", CSTR 117, 1985, AT&T Bell Laboratories, Murray Hill, NJ
- [NIST-RNG] National Institute of Standards and Technology, Random Number Generation Technical Working Group, <http://csrc.nist.gov/rng/>
- [Keys-Naptha] Keyes, Robert, "The Naptha DoS vulnerabilities" http://razor.bindview.com/publish/advisories/adv_NAPTHA.html
- [NMap] Fyodor, Nmap tool, <http://www.insecure.org/nmap/>
- [Frantzen-ISIC] Frantzen, Mike, ISIC tool suite, <http://www.packetfactory.net/Projects/ISIC>
- [Matreya-PRNG] Psuedo Random Number Generators, <http://www.rsasecurity.com/products/bsafe/whitepapers/Article4-PRNG.pdf>
- [TMatthews-PRNG] , Suggestions For Random Number Generation in Software, <http://security.ece.orst.edu/koc/ece575/rsalabs/bulletn1.pdf>

[D. J. Bernstein-Syncookies] Decent writeup about SYNcookies, <http://cr.yp.to/syncookies.html>

Appendix A1 - Sample Script for Collecting Sequence Numbers

```
#!/bin/bash
# Copyright, 2003, Cisco Systems
# Author: Andrew Balinsky, balinsky@cisco.com
# Shell script to take the output of an hping sequence number plot
and graph it # using gnuplot.
# An example command line to gather these scripts is:
# hping2 -I eth0 1.2.3.4 -i u100000 -c 500 -Q -p 23 -S > seqnums.txt

if [ "x$1" = "x" ]; then
    echo "usage: $0 filename_to_plot"
    exit
fi

infile=$1
tmpfile=${1}.tmp
tmpfile2=${1}2.tmp
pngfile=${1}.png
echo "set term png" > $tmpfile
echo "set output \"${pngfile}\"" >> $tmpfile
echo "plot \"-\\" >> $tmpfile
grep -v HPING $infile > $tmpfile2
cat $tmpfile2 | awk '{FS=" "}{print nlines++ " "$1}' >> $tmpfile
gnuplot $tmpfile
/bin/rm $tmpfile $tmpfile2
gimp $pngfile &
```