

IP Time-to-Live Attacks

Related PCAP File(s):

- ip_ttl.pcapng

Time-to-Live attacks are primarily utilized as a means of evasion by attackers. Basically speaking the attacker will intentionally set a very low TTL on their IP packets in order to attempt to evade firewalls, IDS, and IPS systems. These work like the following.



1. The attacker will craft an IP packet with an intentionally low TTL value (1, 2, 3 and so on).
2. Through each host that this packet passes through this TTL value will be decremented by one until it reaches zero.
3. Upon reaching zero this packet will be discarded. The attacker will try to get this packet discarded before it reaches a firewall or filtering system to avoid detection/controls.
4. When the packets expire, the routers along the path generate ICMP Time Exceeded messages and send them back to the source IP address.

Finding Irregularities in IP TTL

For starters, we can begin to dump our traffic and open it in Wireshark. Detecting this in small amounts can be difficult, but fortunately for us attackers will most times utilize ttl manipulation in port scanning efforts. Right away we might notice something like the following.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.5	192.168.10.1	TCP	60	2801 → 80 [SYN] Seq=0 Win=512 Len=0
2	0.000039	192.168.10.5	192.168.10.1	TCP	60	2802 → 80 [SYN] Seq=0 Win=512 Len=0
3	0.000050	192.168.10.5	192.168.10.1	TCP	60	2803 → 80 [SYN] Seq=0 Win=512 Len=0
4	0.000061	192.168.10.5	192.168.10.1	TCP	60	2804 → 80 [SYN] Seq=0 Win=512 Len=0
5	0.000070	192.168.10.5	192.168.10.1	TCP	60	2805 → 80 [SYN] Seq=0 Win=512 Len=0
6	0.000076	192.168.10.5	192.168.10.1	TCP	60	2806 → 80 [SYN] Seq=0 Win=512 Len=0
7	0.000082	192.168.10.5	192.168.10.1	TCP	60	2807 → 80 [SYN] Seq=0 Win=512 Len=0
8	0.000093	192.168.10.5	192.168.10.1	TCP	60	2808 → 80 [SYN] Seq=0 Win=512 Len=0
9	0.000114	192.168.10.5	192.168.10.1	TCP	60	2809 → 80 [SYN] Seq=0 Win=512 Len=0
10	0.000122	192.168.10.5	192.168.10.1	TCP	60	2810 → 80 [SYN] Seq=0 Win=512 Len=0
11	0.000129	192.168.10.5	192.168.10.1	TCP	60	2811 → 80 [SYN] Seq=0 Win=512 Len=0

However, we might also notice a returned SYN, ACK message from one of our legitimate service ports on our affected host. In doing so, the attacker might have successfully evaded one of our firewall controls.

Resources

Table of Contents

Introduction

Intermediate Network Traffic Analysis Overview

Link Layer Attacks

- ARP Spoofing & Abnormality Detection
- ARP Scanning & Denial-of-Service
- 802.11 Denial-of-Service
- Rogue Access Point & Evil-Twin Attacks

Detecting Network Abnormalities

- Fragmentation Attacks
- IP Source & Destination Spoofing Attacks
- IP Time-to-Live Attacks
- TCP Handshake Abnormalities
- TCP Connection Resets & Hijacking
- ICMP Tunneling

Application Layer Attacks

- HTTP/HTTPs Service Enumeration Detection
- Strange HTTP Headers
- Cross-Site Scripting (XSS) & Code Injection Detection
- SSL Renegotiation Attacks
- Peculiar DNS Traffic
- Strange Telnet & UDP Connections

Skills Assessment

- Skills Assessment

My Workstation

OFFLINE

host. In doing so, the attacker might have successfully evaded one of our firewall controls.

No.	Time	Source	Destination	Protocol	Length	Info
11	0.000129	192.168.10.5	192.168.10.1	TCP	60	2811 → 80 [SYN] Seq=0 Win=512 Len=0
12	0.000135	192.168.10.5	192.168.10.1	TCP	60	2812 → 80 [SYN] Seq=0 Win=512 Len=0
13	0.000140	192.168.10.5	192.168.10.1	TCP	60	2813 → 80 [SYN] Seq=0 Win=512 Len=0
14	0.000147	192.168.10.5	192.168.10.1	TCP	60	2814 → 80 [SYN] Seq=0 Win=512 Len=0
15	0.000157	192.168.10.5	192.168.10.1	TCP	60	2815 → 80 [SYN] Seq=0 Win=512 Len=0
16	0.000164	192.168.10.5	192.168.10.1	TCP	60	2816 → 80 [SYN] Seq=0 Win=512 Len=0
17	0.000171	192.168.10.5	192.168.10.1	TCP	60	2817 → 80 [SYN] Seq=0 Win=512 Len=0
18	0.000176	192.168.10.5	192.168.10.1	TCP	60	2818 → 80 [SYN] Seq=0 Win=512 Len=0
19	0.000186	192.168.10.5	192.168.10.1	TCP	60	2819 → 80 [SYN] Seq=0 Win=512 Len=0
20	0.000195	192.168.10.5	192.168.10.1	TCP	60	2820 → 80 [SYN] Seq=0 Win=512 Len=0
21	0.000201	192.168.10.5	192.168.10.1	TCP	60	2821 → 80 [SYN] Seq=0 Win=512 Len=0
22	0.000207	192.168.10.5	192.168.10.1	TCP	60	2822 → 80 [SYN] Seq=0 Win=512 Len=0
23	0.000213	192.168.10.5	192.168.10.1	TCP	60	2823 → 80 [SYN] Seq=0 Win=512 Len=0
24	0.000219	192.168.10.5	192.168.10.1	TCP	60	2824 → 80 [SYN] Seq=0 Win=512 Len=0
25	0.000228	192.168.10.5	192.168.10.1	TCP	60	2825 → 80 [SYN] Seq=0 Win=512 Len=0
26	0.000236	192.168.10.5	192.168.10.1	TCP	60	2826 → 80 [SYN] Seq=0 Win=512 Len=0
27	0.000242	192.168.10.5	192.168.10.1	TCP	60	2827 → 80 [SYN] Seq=0 Win=512 Len=0
28	0.000255	192.168.10.5	192.168.10.1	TCP	60	2828 → 80 [SYN] Seq=0 Win=512 Len=0
29	0.000261	192.168.10.5	192.168.10.1	TCP	60	2829 → 80 [SYN] Seq=0 Win=512 Len=0
30	0.000267	192.168.10.5	192.168.10.1	TCP	60	2830 → 80 [SYN] Seq=0 Win=512 Len=0
31	0.000278	192.168.10.5	192.168.10.1	TCP	60	2831 → 80 [SYN] Seq=0 Win=512 Len=0
32	0.000287	192.168.10.5	192.168.10.1	TCP	60	2832 → 80 [SYN] Seq=0 Win=512 Len=0
33	0.000292	192.168.10.1	192.168.10.5	TCP	60	80 → 2801 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460

So, if we were to open one of these packets, we could realistically see why this is. Suppose we opened the IPv4 tab in Wireshark for any of these packets. We might notice a very low TTL like the following.

```
Internet Protocol Version 4, Src: 192.168.10.5, Dst: 192.168.10.1
  0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x7312 (29458)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 3
    > [Expert Info (Note/Sequence): "Time To Live" only 3]
    Protocol: TCP (6)
    Header Checksum: 0xaf67 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.10.5
    Destination Address: 192.168.10.1
```

As such, we can implement a control which discards or filters packets that do not have a high enough TTL. In doing so, we can prevent these forms of IP packet crafting attacks.

← Previous

Next →

✔ Mark Complete & Next

