ZMAP原理详细剖析

Saferman / 2018-12-20 10:30:00 / 浏览数 4163 技术文章 技术文章 顶(1) 踩(0)

---

ZMap 是一款扫描软件，由 Durumeric 领导密歇根大学研究团队开发，这种工具能令一台普通的服务器在短短 45 分钟时间里扫描 IPv4 网络空间上的每一个地址，叹为观止！扫描效率远远超过 Nmap 工具。

刚好我最近在研究 IPv6 扫描，也就产生了想深入研究这款扫描工具原理想法，也就有了这篇文章。网络上关于 ZMap 的使用教程非常之多，但是我想在这篇文章里给各位详细剖析 ZMap 这款扫描工具使用到的技术，对于未来 IPv6 网络的扫描有着重要的借鉴价值，同时知其然更要知其所以然也是白帽子应该具备的品质。

有兴趣的读者还可以自行去研究提出这款工具的论文：《ZMap: Fast Internet-wide Scanning and Its Security Applications》

## ZMap 性能

按照论文的描述，这款工具优秀的性能可以用这二点来概括：

- 指定一个端口，ZMap 只需要 45 分钟就可以在一台普通的主机上对整个 IPv4 地址空间的完成扫描
- 在默认的配置下，使用 ZMap 扫描整个 IPv4 地址空间会比 Nmap 快 1300 倍

可以看一下 ZMap 和 Nmap 在性能方面的对比：

| | Normalized Coverage | Duration (mm:ss) | Est. Internet Wide Scan |
|---|---|---|---|
| **Nmap (1 probe)** | 81.4% | 24:12 | 62.5 days |
| **Nmap (2 probes)** | 97.8% | 45:03 | 116.3 days |
| **ZMap (1 probe)** | 98.7% | 00:10 | 1:09:35 |
| **ZMap (2 probes)** | 100.0% | 00:11 | 2:12:35 |

在这表格里测试的是使用 ZMap 和 Nmap 扫描了一百万个主机的 443 端口，重复十次试验取平均测量的时间结果如上。ZMap 的测量耗时远远低于 Nmap，而且即便这样 ZMap 比 Nmap 发现更多的存活主机。在最后一列也测试了扫全网的效率，可以看见 ZMap 远远胜于 Nmap。

下面我来带领大家研究一下为什么 ZMap 具有如此高的扫描效率，通过学习它的技术也可以对我以后的研究工作有更多的帮助。

## ZMap 架构新的特点

首先来看看 ZMap 有哪些新的特点帮助它实现高性能的扫描：

（1）Optimized probe

为了避免扫描发送的探针对发送方的网络和目的网络造成压力从而导致速率下降等不好的结果出现，本文的探针采用随机探针生成技术并且广泛散发的方法，即按照随机的顺 ZMap 按照发送方 NIC 最大速率发送探针，并且跳过 TCP/IP 栈直接生成链路层的帧。

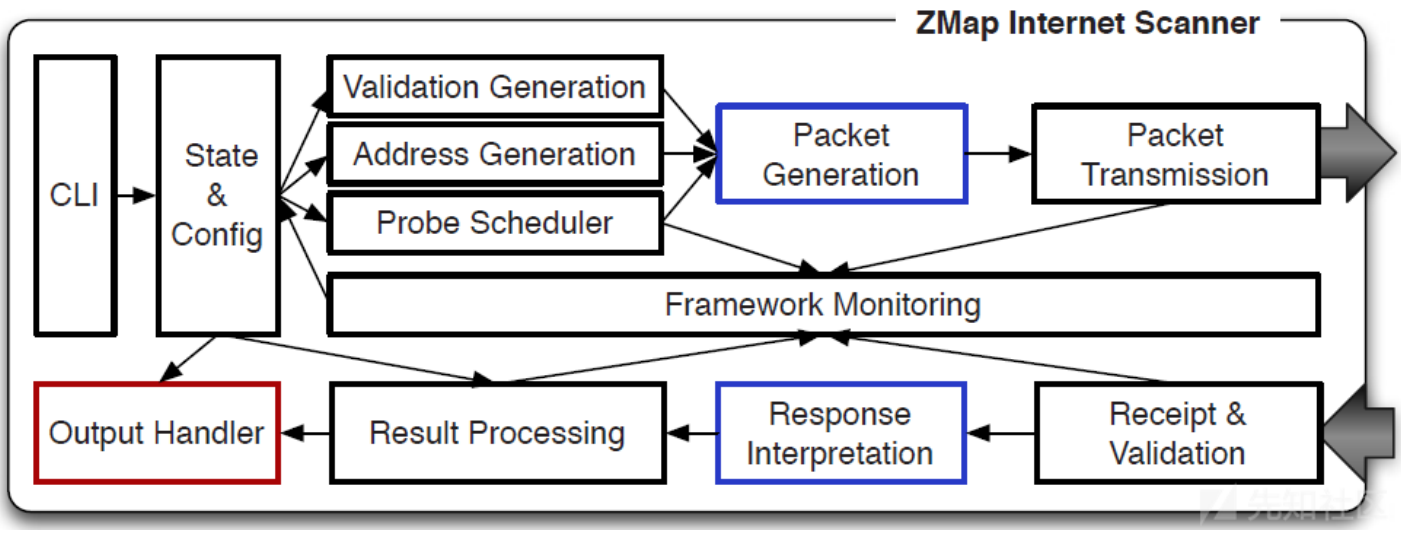
（2）No per-connection state

Nmap 会对每一个连接维护一些状态来帮助程序确认哪些主机被扫描过以及是否需要重新发送。与之不同，ZMap 不保存任何连接的这些状态。首先在解决如何不重复不遗漏扫描全部需要扫描的主机问题上，ZMap 主要使用了周期乘法生成地址的随机全排列。此外，ZMap 接受的响应数据包携带有一些状态字段，程序会从中提取尽可能多的有用信息帮助 ZMap 得到扫描结果。为了将探针导致的响应数据包从网络中的其他数据包识别分离出来，ZMap 会重载每一个发送的数据包中未被使用的值来做一些标识，这一点有点像 SYN cookies。

（3）No retransmission

ZMap 总是针对每一个目标发送相同且固定的探针数据包（默认一个），并且不会像 Nmap 那样在超时后重新发送相同探针数据包。也就是说 ZMap 不会重新发送任何数据包，主要是避免维护一些不必要的状态。放心，文章实验证明了即使在最大的扫描速率下只对每一个目标地址发送一个探针数据包也可以有效的扫描完 98% 的网络空间。

## 架构



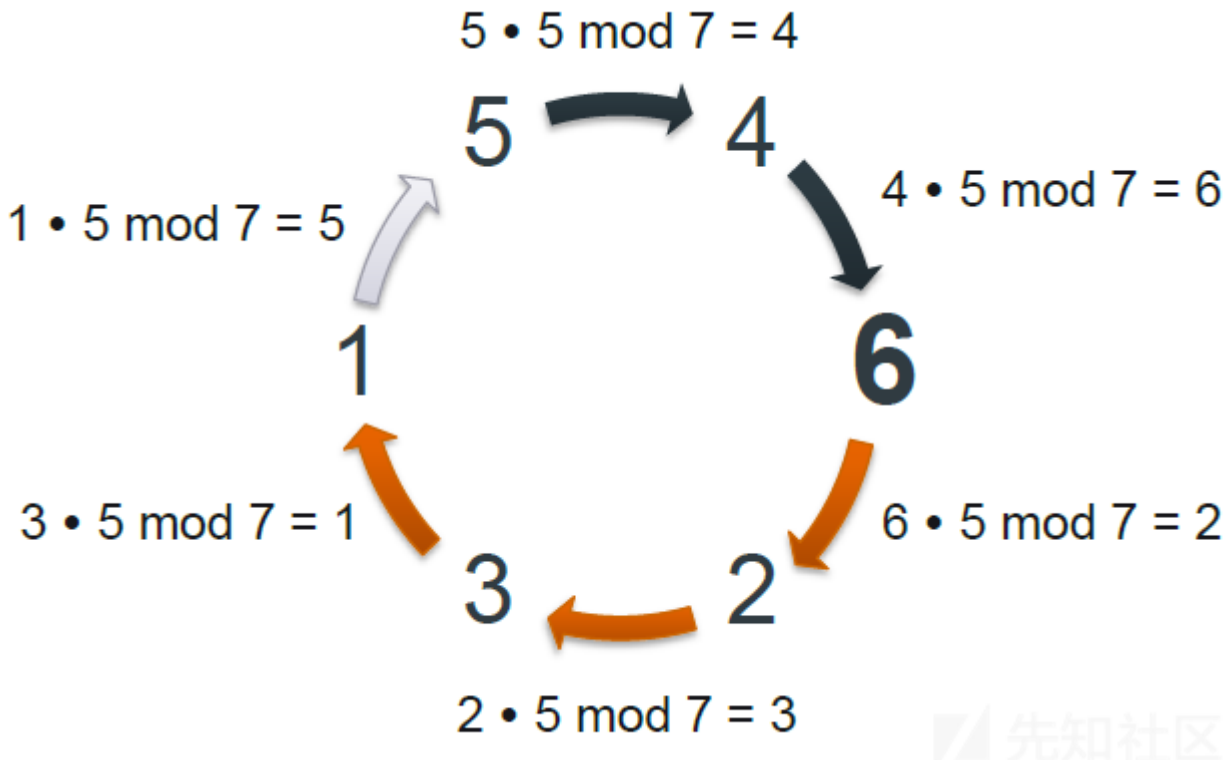

这部分介绍一下 ZMap 的架构和整体的设计思想：

- 蓝色 Packet Generation 和 Response Interpretation： 指出生成和处理多种探针数据包，包括 TCP SYN 扫描的和 ICMP echo 扫描的
- 红色 Output Handler：输出结果并且允许用户处理这些结果
- 设计思想：最重要的架构特点在于扫描过程和接受处理过程是在不同的线程完成的，彼此独立并且连续的进行

整个系统大约用了 8900 行 C 代码完成。下面我着重针对 ZMap 三个主要的点展开深入的讨论。

### Addressing Probes

首先先来看一个数学例子，如果我们期望遍历 1-6 整数，但是又不想按照顺序去遍历，想使用随机的方法遍历这些整数，一种方法如图所示，即整数模 n 乘法群（multiplicative group of integers modulo n ）。 相关参考文献：[维基百科-整数模 n 乘法群]




可以看到只要我们选定一个稍微大于需要遍历的序列最大数的奇质数 n （论文用符号 p 表示，此处为 7），那么再选定一个需要遍历的数列其中一个元素 primitive root （此处为 5，选择其他的比如 3 也可以），通过相乘取余数可以生成数列的一个全排列。

应用在 IPv4 地址中，我们可以将 128bit 的 IPv4 地址空间看成一个 1~2^32 的整数集合（128 位的二进制转换为 10 进制数），每一个整数对应 IPv4 的一个地址（0.0.0.0 地址对应的整数为 0，没有扫描的意义故排除）。然后论文选择了一个稍微大一些奇质数 2^32+15 作为 p，选择 3 作为 primitive root。这样可以用相同的数学原理得到 1~2^32 整数的全排列，每一个整数转换位 128 位二进制即可得到 IPv4 地址。

### Packet Transmission and Receipt

ZMap 被设计尽可能使用源主机的 CPU 和 NIC 能支持的最大发包速率去发送探针数据包
(probe)。程序直接在以太网层（即链路层，以太网协议是链路层很重要的一个协议）发送 raw packet，避免过多的内核层面的操作包括路由表查询、arp

缓存查询和 Netfilter 检验。这样链路层头部，包括校验码在整个扫描过程都不会变的，在内核中也不会建立任何 TCP session。

ZMap 的接受模块仍然使用
libpcap，虽然这个库很多时候会成为性能的瓶颈，但是考虑到响应的数据包速率不会有发送的那么快，在论文的测试中也验证了这一个观点，所以使用 libpcap 是可行的。

在接收到数据包后，通过源端口和目的端口直接跳过那些明显不是响应我们扫描的数据，将剩下的数据包传送到 probe module
进一步解析。由于发送和接受主线程是独立的，在发送结束后接受线程仍然额外持续 8 秒。

### Probe Modules

如果使用 TCP 端口扫描，采用的技术都是 SYN scanning 可以根据响应的数据包判断是 unreachable、close 还是 open（对应三种情况：无响应，响应 RST 以及响应 SYN-ACK）。

该模块一个很重要的任务就是识别那些响应数据包是针对我们的探针数据包响应的。按照原文的说法解决方法就是：

> Probe modules perform this validation by encoding host- and scan-invocation–specific data into mutable fields of each probe packet, utilizing fields that will have recognizable effects on fields of the corresponding response packets in a manner similar to SYN cookies.

具体哪些发送的数据包字段会在响应的时候携带发送字段的信息：在我们的 TCP 扫描技术中，我们采用源端口和序列数字；在 ICMP 扫描技术中，我们使用 ICMP identifier 和序列数字 (sequence number)。

这种验证技术很类似 SYN Cookie，这种技术主要使用与防范 SYN Flood 攻击的（SYN Cookie原理可以参考 https://zh.wikipedia.org/wiki/SYN_cookie 和 SYN cookies 机制下连接的建立）

> 在 syn 到来的时候， 服务端并不会为这个 syn 包分配资源， 免得受骗上当。只有当服务端自己的 syn/ack 收到客户端的 ack 后， 才会真正分配资源， 此时才完成正常的连接。 由此可见， 此时 syn flood 失效。

同时这一部分的 ZMap 作者参考了《UMAC: Fast and secure message
authentication》文献，也值得去阅读一边进一步了解识别响应数据包的工作原理。特别说明这部分论文提到的 MAC 不是指物理地址，而是消息摘要的意思。