DNS安全皮毛（二）

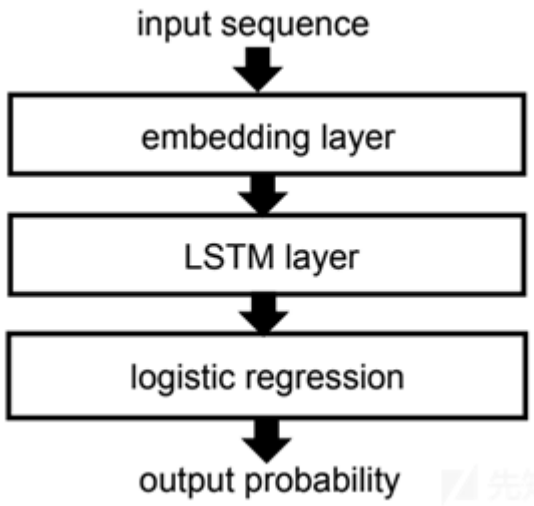
backcover7 / 2019-08-30 09:11:00 / 浏览数 4481 安全技术 WEB安全 顶(0) 踩(0)

DGA

DGA，顾名思义常用于构建恶意域名，像wannacry，xshellghost这些也都使用了DGA域名用于躲避黑名单检测。

```
void GetDgaDomain()
{
        CHAR            a3[255] = { 0 };
        int             v3;     /* eax@1 */
        int             v4;     /* STOC_4@1 */
        int             v5;     /* edi@1 */
        int             v6;     /* eax@1 */
        CHAR            v7;     /* eax@1 */
        signed int      v8;     /* ecx@1 */
        CHAR            v10 = "ab%x%x.com";
        int             v11;    /* [sp+Ch] [bp-1Ch]@1 */
        int             v12;    /* [sp+10h] [bp-18h]@1 */
        for ( int i = 1; i < 13; i++ )
        {
                srand( 20170000 + i );
                v4      = rand();
                v5      = rand();
                v6      = rand();
                printf( "2017Year%dMonth:ab%x%x.com\n", i, v6 * v5, v4 );
        }
        for ( int i = 1; i < 13; i++ )
        {
                srand( 20180000 + i );
                v4      = rand();
                v5      = rand();
                v6      = rand();
                printf( "2018Year%dMonth:ab%x%x.com\n", i, v6 * v5, v4 );
        }
        return;
}
```


先前有一篇通过使用LSTM来预测DGA域名的论文解读传遍了各大平台，基本就是通过LSTM这种基于RNN的进阶算法，可以在大跨度的时间区域范围内进行学习并预测可能



那么关键问题就是DGA域名的随机性貌似给它也带来了一些不方便。如何突破这个随机性，接下来攻击者针对这些新生的算法进行了研究和绕过。

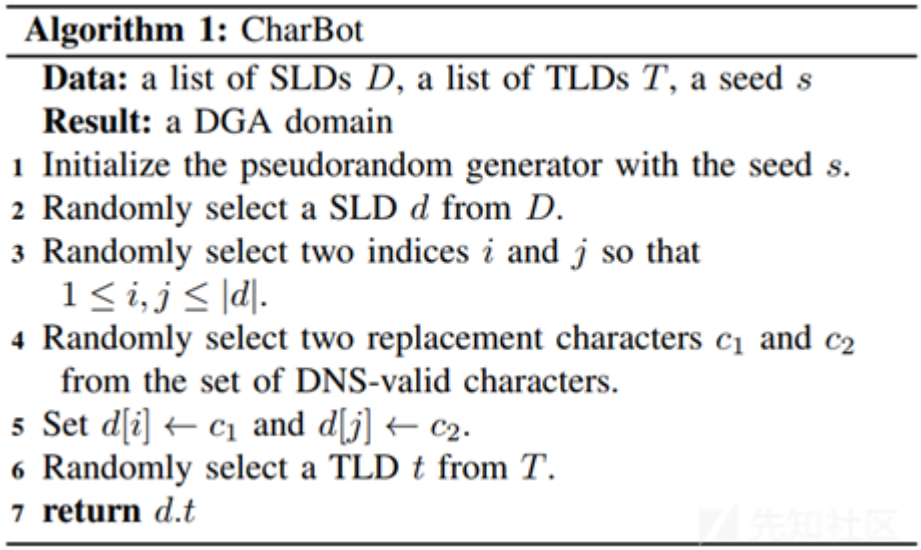
对抗升级

针对上述随机性问题，可以针对字符熵来辅助判断是否为DGA域名，为了突破随机性过于明显的特征。研究人员实现了一种伪随机的DGA算法，名为CharBot，该算法成功
classifier，例如名为FANCI的RF模型和名为LSTM.MI的DNN模型。

这种基于字符的DGA算法主要使用了如下三个数据集作为基本输入：

1. 合法二级域名（SLD）；
2. 顶级域名（TLD）；
3. 一个用作伪随机化种子的日期数值；

那么CharBot的基本算法过程如下（具体算法伪代码如图所示）：



Algorithm 1: CharBot

Data: a list of SLDs D, a list of TLDs T, a seed s
Result: a DGA domain

Initialize the pseudorandom generator with the seed s.
Randomly select a SLD d from D.
Randomly select two indices i and j so that $1 \leq i, j \leq |d|$.
Randomly select two replacement characters c_1 and c_2 from the set of DNS-valid characters.
Set $d[i] \leftarrow c_1$ and $d[j] \leftarrow c_2$.
Randomly select a TLD t from T.
return $d.t$

1. 选择一个data set中的domain；
2. 选择SLD中两个字符（替换的字符越多，DGA classifier检测率越高，越少则已注册越多。2个为平衡这两种状态的最佳中间值）；
3. 选择两个替换字符（算法保证替换字符与原有字符不一样）；
4. 添加com,uk,biz,fr,jp这些后缀然后成为顶级域名。

这种算法很大程度降低DGA
classifier对随机化的检测率。但是该算法唯一的实现阻碍在于软件在内置该算法时需要内置大量的域名数据集，导致软件本身的文件大小大大提升。但是根据实际测试，一个

检测CharBot算法生成的域名有如下一些方法：

1. 与Alexa样本比较发现是否存在几个字符替换的现象（Alexa样本巨大，且攻击者可能不使用Alexa样本，防御成本过高）；
2. 使用更复杂的模型来训练处理，但是也是不太可行，因为需要定期重新训练，且需要处理的对象过多；
3. 重复的白盒对抗训练，但对抗学习的参数难调；
4. 使用DGA域名以外的信息进行辅助判断，例如IP地址，域查询频率和时间等。

关于DNS类攻击的解决方法

常见的一些对恶意域名的官方拉黑办法有大家熟知的DNS
Sinkhole，NXDomain，或者是将其映射到实验性IP地质类（D类地址和E类地址）等。但是这些办法都是已经捕捉到恶意域名后再对其拉黑的方式，那么如何捕捉恶意域名

PDNS

这里的PDNS（Process-DNS system）与之前提到的PDNS（Passive
DNS）不一样。这里的PDNS是可以针对进程级别的DNS流量进行监测的系统，其监测能力主要是是基于两大特征：

网络特征

域名的whois信息（域名使用时长，域名注册信息，域名注册地址）；

IP whois信息和IP物理定位（AS自治系统编号，IP物理定位）；

授权解析服务器（域名TTL值）

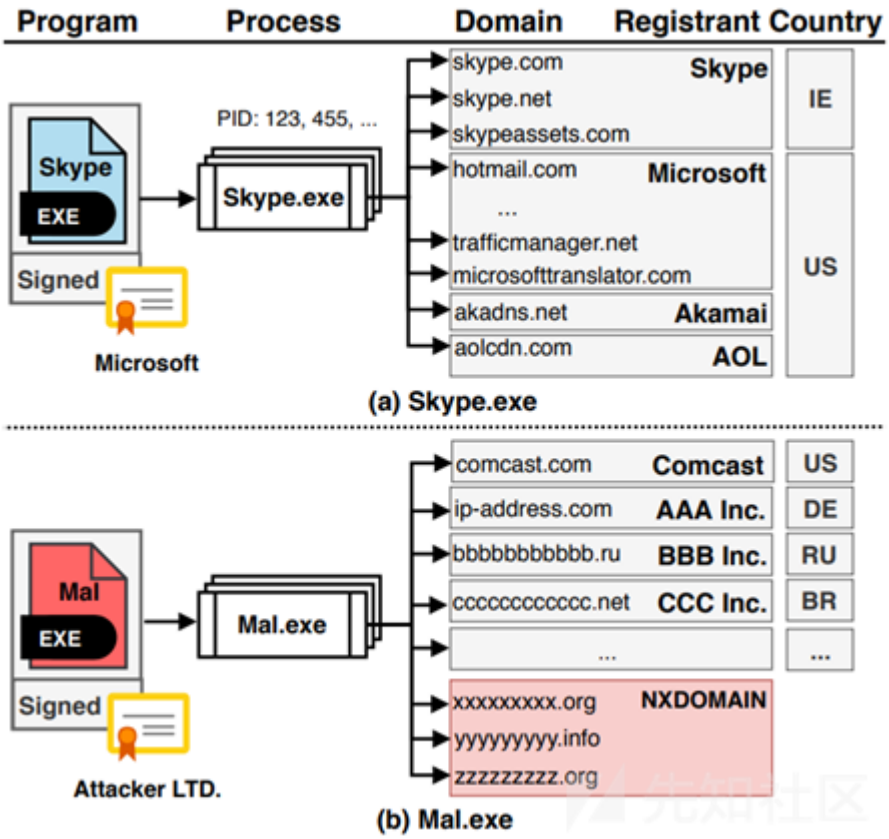
进程特征

DNS活动信息（域名和主机名，域名解析失败率）；

进程信息（加载dll，code signing）；

该系统是部署在一个后台服务器和各个需要被监控的终端主机上的，它弥补了传统DNS监测系统的溯源定位不精确的缺陷：

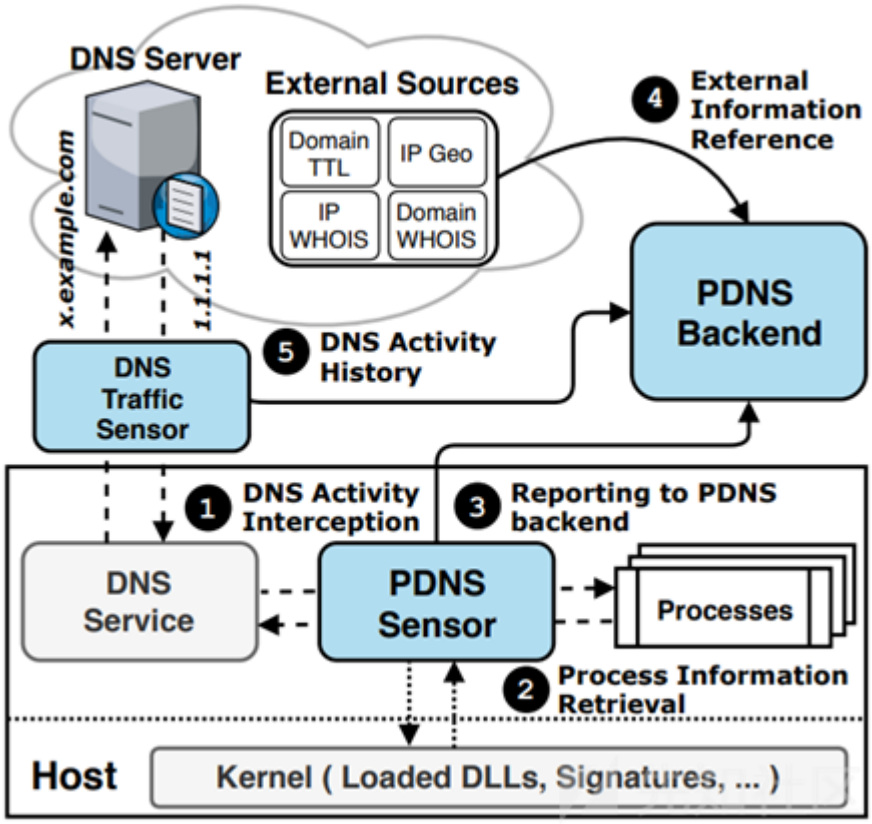
1. PDNS分析细粒度级别由主机级别到了线程级别，收集的数据更加敏感更利于模型建立；
2. 流程为中心可以立即确定恶意进程；
3. 终端部署方式可以提高对DNS-over-TLS/HTTPS的可见性；



那么为什么PDNS需要将这两个特征（网络特征和进程特征）绑定在一起进行分析使用呢？这里首先区分定义一下什么是软件，什么是进程。软件可以启动多个进程，进程是

如图所示，Skype.exe启动了多个进程，每个进程都连接到Skype或Microsoft等相关的域，Skype启动的这些进程收集到的29K个DNS查询最终只对应到19个域，这其中17
Software Sarl”。相应地，DNS
Whois记录确认大多数dns查询都有“skype”或“microsoft”作为注册者，而注册者最终是同一家公司。Skype软件以图形界面运行，允许用户交互。所以从这些信息可以看得

而Mal.exe就表现出截然相反的特征，其访问了大概20个不同的域，且所有域都是在不同地理位置的不同组织下注册的，而且某些域为NXDOMAIN。虽然恶意软件是正确签

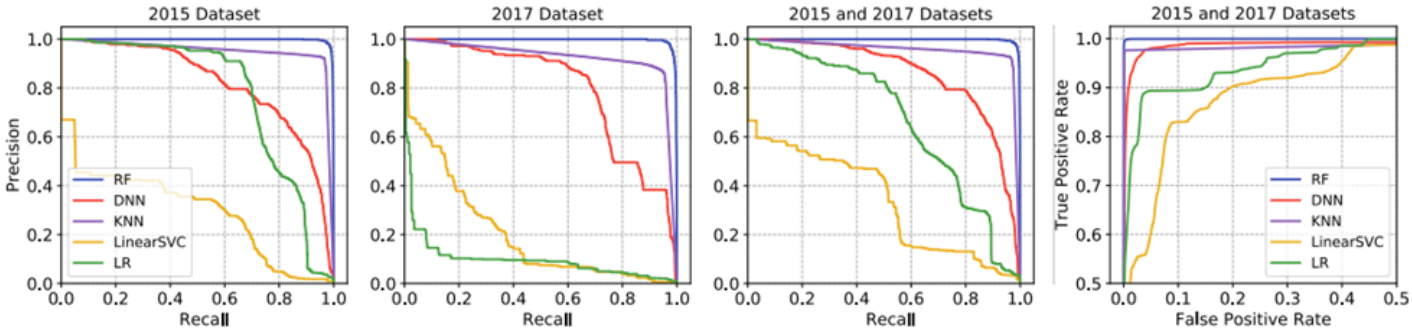


PDNS系统的基本运行过程如下：

1. 装载监控客户端的agent拦截监测DNS流量中的活动行为；
2. agent也会从内核获取进程信息（加载的dlls, code signing）；

3. agent将获取到的信息传递给backend服务器；
4. backend服务器获取外部网络特征信息；
5. backend服务器获取DNS历史活动信息；

PDNS的后台服务器获取到上述复杂信息后会丢给内置的机器学习算法进行分析判断，最后给出结果是否存在恶意软件传出或传入的DNS流量，并且能准确溯源该恶意软件的



(a) Precision-recall (b) ROC

上述过程存在两个难题，一是如何捕获到所有的DNS活动，二是如何将捕获到的DNS活动与进程联系在一起。

软件通常通过两个信道进行域名解析活动：

与解析服务器之间的UDP通信；

针对这个信道，使用网络跟踪工具即可，比如普林斯顿的Hone开源项目（该项目本身就可以将每个捕获到的包与进程ID联系在一起）。

软件将该解析认为直接委派给系统本身的DNS解析功能；

捕获进程与解析服务器之间的通信数据或者读取本地服务产生的日志文件。

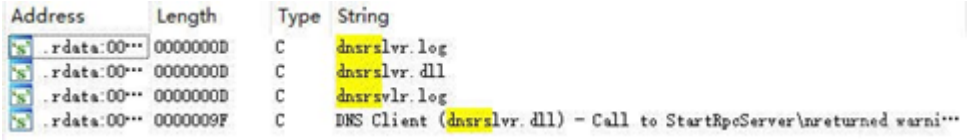
针对Linux主机，一般开发者不会使用Linux内部的DNS解析服务，所以直接通过Hone即可捕获读取。而针对Windows系统而言，开发者一般都会使用Windows自带的DNS Client Service，所以在Windows 8及以下版本，PDNS会拦截LPRC通信进行分析，而在Windows 8.1及以上版本可以直接从DNS Client Service的日志中读取即可。

那么为什么DNS Client Service日志在Windows 8及以下版本不可行呢？这是因为该日志功能存在一定的bug。

DNS log问题

针对Windows平台针对DNS Client Service的日志记录的问题，scz曾在博客中写关于为何Windows的某些低版本无法正常记录DNS log。我依样画葫芦分析了存在问题的dll组件，即dnsrslvr.dll。基本过程不再详述，可以查看scz的博客。

简单来说有两个问题，第一个在通过反编译在Strings中直接搜索关键字"dnsrs"可以看到两个略有不同的日志文件的文件名，dnsrslvr.log和dnsrsvlr.log，很明显dnsrsvlr.lo




```
v1 = a1;
g_ResolverStatus = 213909504;
g_InitState = 0;
g_StopFlag = 0;
g_hStopEvent = 0i64;
g_fServiceControlHandled = 0;
v17 = 0;
_InterlockedExchange(&g_dwServiceControlFlags, 0);
DnsLogInit_0((__int64)"dnsrsvlr.log", (__int64)"Logging for DNS resolver service");
DnsLogIt_0((__int64)"dnsrslvr.log", (__int64)"DNS Caching Resolver Service - ResolverInitialize");
if ( WPP_GLOBAL_Control != &WPP_GLOBAL_Control && *((_BYTE *)WPP_GLOBAL_Control + 28) & 1 )
  WPP_SF_(*((_QWORD *)WPP_GLOBAL_Control + 2), 20i64, &unk_7FF724433D0);
ServiceStatusHandle = 0i64;
ServiceStatus.dwServiceType = 32;
ServiceStatus.dwCurrentState = 2;
ServiceStatus.dwControlsAccepted = 0;
ServiceStatus.dwCheckPoint = 0;
ServiceStatus.dwWaitHint = 5000;
ServiceStatus.dwWin32ExitCode = 0;
ServiceStatus.dwServiceSpecificExitCode = 0;
ServiceStatusHandle = RegisterServiceCtrlHandlerExW(L"dnscache", ResolverControlHandler, 0i64);
```

第二个是在DnsLogInit()调用关键函数DnsLogIt()中，进入该函数主体时做了一个条件判断，判断依据是一个全局变量LoggingMode，这个全局变量在DnsLogInit()中定义

DnsLogIt.c

```c
void DnsLogIt(char *Filename, char *Format, ...)
{
  char *v2; // rbx
  int v3; // eax
  FILE *v4; // rax
  FILE *v5; // rbx
  char Dest; // [rsp+20h] [rbp-438h]
  char v7; // [rsp+41Fh] [rbp-39h]
  va_list va; // [rsp+470h] [rbp+18h]

  va_start(va, Format);
  v2 = Filename;
  if ( LoggingMode )
  {
    v3 = _vsnprintf(&Dest, 0x3FFui64, Format, va);
    if ( v3 < 0 || (unsigned __int64)v3 > 0x3FF || v3 == 1023i64 )
      v7 = 0;
    if ( LineCount <= 10000 )
    {
      v4 = fopen(v2, "a");
    }
    else
    {
      v4 = fopen(v2, "w");
      LineCount = 0;
    }
    v5 = v4;
    if ( v4 )
    {
      fputs(&Dest, v4);
      ++LineCount;
      fclose(v5);
    }
  }
}
```


经过测试分析，安装了KB2956577补丁后即不存在该问题了。scz的博客中也描述了如何不安装补丁也可以使该功能正常化。

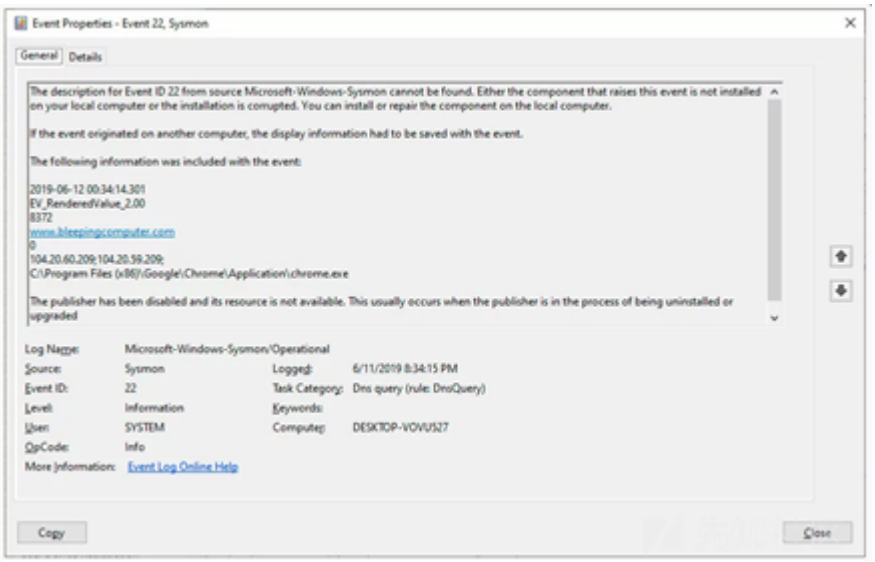
DNS记录tricks

DNS中常被用作挖掘子域名的一个漏洞DNS域传输漏洞还有个鲜为人知的一面。我们熟知DNS在TCP/IP栈中是一个比较另类的协议，因为其同时占用了TCP和UDP的53端口。53端口。一个就是在进行域传输操作时，会使用TCP 53端口进行数据同步；而DNS进行解析查询优先使用UDP
，当UDP完成不了的情况下（即查询数据过大，超过512字节），则转换TCP查询。所以监控TCP协议上的DNS数据传输，准确度会提高很多，但是捕获量并不会非常多。

```
4.2. Transport

The DNS assumes that messages will be transmitted as datagrams or in a
byte stream carried by a virtual circuit.  While virtual circuits can be
used for any DNS activity, datagrams are preferred for queries due to
their lower overhead and better performance.  Zone refresh activities
must use virtual circuits because of the need for reliable transfer.

The Internet supports name server access using TCP [RFC-793] on server
port 53 (decimal) as well as datagram access using UDP [RFC-768] on UDP
port 53 (decimal).

4.2.1. UDP usage

Messages sent using UDP user server port 53 (decimal).

Messages carried by UDP are restricted to 512 bytes (not counting the IP
or UDP headers).  Longer messages are truncated and the TC bit is set in
the header.

UDP is not acceptable for zone transfers, but is the recommended method
for standard queries in the Internet.  Queries sent using UDP may be
lost, and hence a retransmission strategy is required.  Queries or their
responses may be reordered by the network, or by processing in name
servers, so resolvers should not depend on them being returned in order.
```


微软的新方案

新版Sysmon提供了Event ID 22: DNSEvent (DNS query)，即只要有DNS请求发出就会记录日志，配置也很简单明了。可参考https://github.com/SwiftOnSecurity/sysmon-config。



对抗升级

就在普天同庆Sysmon的新升级刚没几天，一篇关于绕过Sysmon的DNS日志记录的利用文章就公布了。

简单读了下文章，可以知道在sysmon启动后找到一个my event trace session的data collector set，这个名字曾出现在微软提供的样例代码中。

```
#define LOGFILE_PATH L"<FULLPATHTOLOGFILE.etl>"
#define LOGSESSION_NAME L"My Event Trace Session"
```

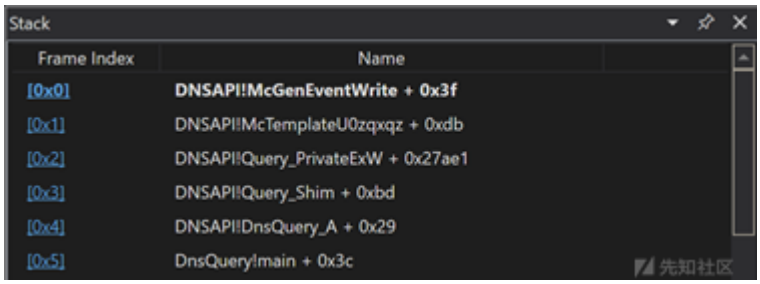
利用逆向找到该字符串所在函数进行分析。函数中提供了一个数据指针，分析发现该数据指针恰好是DNS_CLIENT所使用的的数据指针。

```
                    DAT_1400c4a10
1400c4a10 6e        ??      6Eh   n
1400c4a11 12        ??      12h
1400c4a12 95        ??      95h
1400c4a13 1c        ??      1Ch
1400c4a14 ea        ??      EAh
1400c4a15 7e        ??      7Eh   ~
1400c4a16 a9        ??      A9h
1400c4a17 49        ??      49h   I
1400c4a18 a3        ??      A3h
1400c4a19 fe        ??      FEh
1400c4a1a a3        ??      A3h
1400c4a1b 78        ??      78h   x
1400c4a1c b0        ??      B0h
1400c4a1d 3d        ??      3Dh   =
1400c4a1e db        ??      DBh
1400c4a1f 4d        ??      4Dh   M
```

```
                    DNS_CLIENT
18009bbb0 5e        ??      6Eh   n
18009bbb1 12        ??      12h
18009bbb2 95        ??      95h
18009bbb3 1c        ??      1Ch
18009bbb4 ea        ??      EAh
18009bbb5 7e        ??      7Eh   ~
18009bbb6 a9        ??      A9h
18009bbb7 49        ??      49h   I
18009bbb8 a3        ??      A3h
18009bbb9 fe        ??      FEh
18009bbba a3        ??      A3h
18009bbbb 78        ??      78h   x
18009bbbc b0        ??      B0h
18009bbbd 3d        ??      3Dh   =
18009bbbe db        ??      DBh
18009bbbf 4d        ??      4Dh   M
```

```
ulonglong FUN_140005d70(char param_1,undefined8 param_2,undefined8 param_3,undefined8 param_4)

{
  uint uVar1;
  ulonglong *_Memory;
  ulonglong uVar2;
  ulonglong uVar3;
  uint local_res10 [2];

  _Memory = (ulonglong *)malloc(0xa6);
  if (_Memory == (ulonglong *)0x0) {
    return 0x7a;
  }
  FUN_1400532d0(_Memory,0,0xa6);
  *(undefined4 *)_Memory = 0xa6;
  *(undefined4 *)((longlong)_Memory + 0x2c) = 0x20000;
  *(undefined4 *)(_Memory + 5) = 1;
  *(undefined4 *)(_Memory + 3) = 0xae44cb98;
  *(undefined4 *)((longlong)_Memory + 0x1c) = 0x4069bd11;
  *(undefined4 *)(_Memory + 4) = 0xe779380;
  *(undefined4 *)((longlong)_Memory + 0x24) = 0x128a25c9;
  *(undefined4 *)(_Memory + 8) = 0x100;
  *(undefined4 *)((longlong)_Memory + 0x3c) = 1;
  *(undefined4 *)((longlong)_Memory + 0x74) = 0x78;
  FUN_1400048f0((short *)(_Memory + 0xf));
  if (param_1 == '\0') {
    uVar1 = ControlTraceW(DAT_140109cd8,"My Event Trace Session",_Memory,1);
LAB_140005ef7:
    uVar3 = (ulonglong)uVar1;
  }
  else {
    uVar2 = StartTraceW(&DAT_140109cd8);
    uVar3 = uVar2 & 0xffffffff;
    if ((int)uVar2 == 0) {
      uVar1 = EnableTraceEx2(DAT_140109cd8,&DAT_1400c4a10,
                             CONCAT71((int7)((ulonglong)param_4 >> 8),4),0,0,0,0);
      uVar3 = (ulonglong)uVar1;
      param_4 = 0;
      _DAT_140109cd0 =
           _beginthreadex((void *)0x0,0,(_StartAddress *)&DAT_140005b70,(void *)0x0,0,local_res10);
      if (uVar1 == 0) goto LAB_140005f00;
    }
    if (DAT_140109cd8 != 0) {
      EnableTraceEx2(DAT_140109cd8,&DAT_1400c4a10,0,CONCAT71((int7)((ulonglong)param_4 >> 8),4),0,0,
                     0,0);
      uVar1 = ControlTraceW(DAT_140109cd8,L"My Event Trace Session",_Memory,1);
      goto LAB_140005ef7;
    }
    ControlTraceW(0,L"My Event Trace Session",_Memory,1);
  }
  DAT_140109cd8 = 0;
```

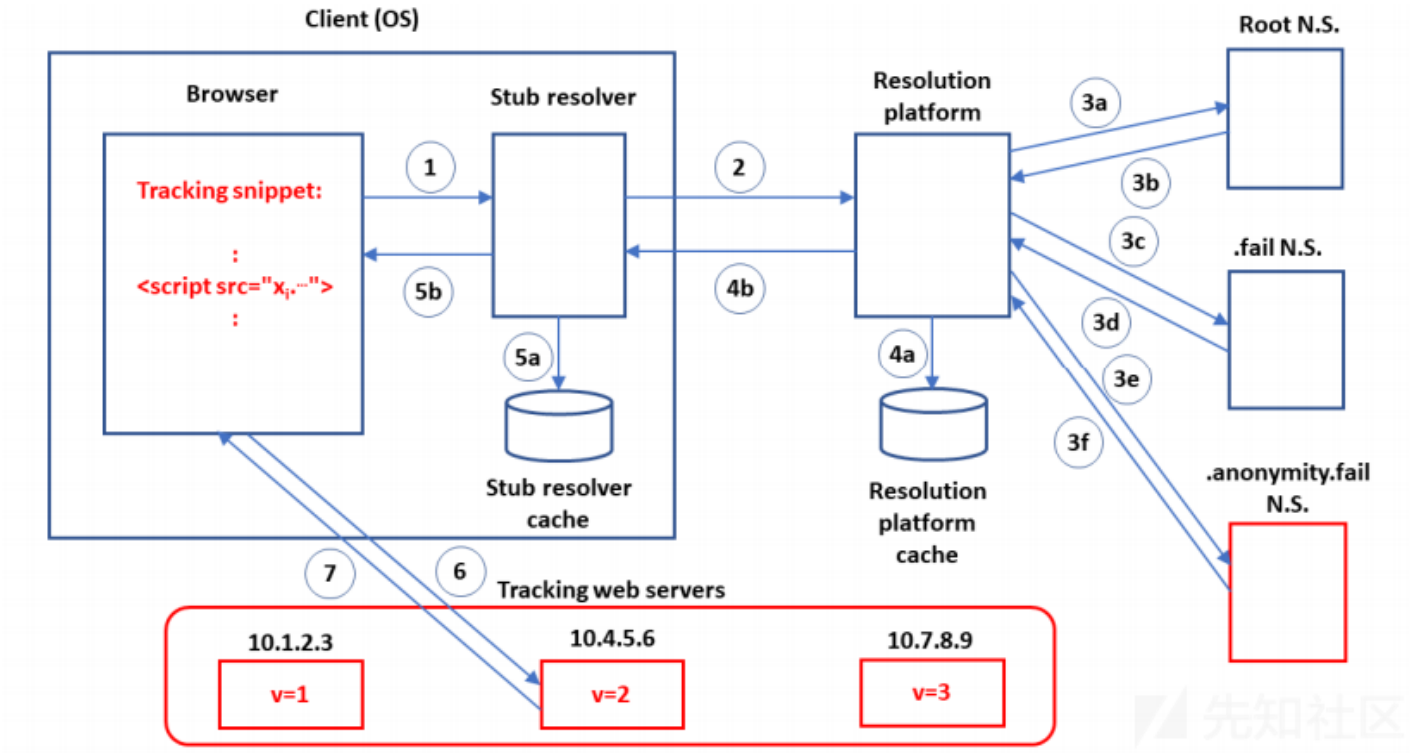
所以我们直接自定义一个简单的DNS查询函数，再对其进行动态调试即可发现其中端倪。调试中可以从堆栈信息中发现调用了evenwrite函数。



Evenwrite结束后会调用evenwritetransfer这个函数来写入日志，所以我们只要修改DLL，避开evenwritetransfer这个函数即可。

利益 OR 恶意？

使用DNS来跟踪用户行为，保持会话不中断。该技术主要解决了DNS缓存时间短的问题，并且突破cookie等此类技术在多浏览器和浏览器隐私模式下无法继续追踪的限制，



图中红色部分是跟踪者实际可控的组件部分。

1. 浏览器加载跟踪代码片段；
2. 浏览器向操作系统的Stub Resolver请求解析xi.anonymity.fail域名；
3. 操作系统向解析平台传递解析请求，而解析平台分析得知需要向域名拥有者控制的DNS解析器解析。且该DNS解析器回应了RRset，即一系列随意排序的跟踪者可控的IP
4. Stub Resolver缓存RRset，并向浏览器返回IP地址列表；
5. 浏览器向RRset中的第一个IP地址发送HTTP请求；
6. 服务器向不同的IP地址发送不同的响应内容；
7. JS收集从服务器传来的数据并组装成一个ID；

对于这种追踪方法，只有使用HTTP代理或者Tor代理才能肯定使其失效。

从这里我们可以发现一项技术本身并没有对错，只是用它的人的目的决定了是否存在恶意。该追踪技术还存在一些技术细节，我已经详细撰文发在SecQuan公众号上。https

致谢

ourren@Secwiki

sshruoshui(warethink#gmail.com)@NUTD

本文首发于安全学术圈（SecQuan）公众号。本公众号分享安全方面的论文写作、会议发表、基金申请方面的资料。

参考文献

[1] Woodbridge, J., Anderson, H. S., Ahuja, A., & Grant, D. (2016). Predicting domain generation algorithms with long short-term memory networks. arXiv preprint arXiv:1611.00791.

[2] Peck, J., Nie, C., Sivaguru, R., Grumer, C., Olumofin, F., Yu, B., ... & De Cock, M. (2019). CharBot: A Simple and Effective Method for Evading DGA Classifiers. arXiv preprint arXiv:1905.01078.

[3] Sivakorn, S., Jee, K., Sun, Y., Kort-Parn, L., Li, Z., Lumezanu, C., ... & Li, D. (2019). Countering Malicious Processes with Process-DNS Association. In NDSS.

[4] Pupeng (2015, Feb, 20). Hone [Web log post]. Retrieved August 01, 2019, from https://github.com/pupeng/hone

[5] scz. (2017, May 11). DNS系列(10)--开启DNS Client Service日志 [Web log post]. Retrieved June 24, 2019, from http://scz.617.cn/windows/201705111519.txt

[6] Microsoft. (2019, June 14). Sysmon v10.1 [Web log post]. Retrieved June 15, 2019, from https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon

[7] 红雨滴团队. (2019, June 12). 喜讯：微软发布具有DNS查询日志记录功能的Sysmon [Web log post]. Retrieved June 13, 2019, from https://mp.weixin.qq.com/s/0xKYO5NObXV9Lsyb97fhjg

[8] SwiftOnSecurity. (2019, May 10). Sysmon-config [Web log post]. Retrieved May 11, 2019, from https://github.com/SwiftOnSecurity/sysmon-config

[9] Chester A.(2019, June 15). Evading Sysmon DNS Monitoring [Web log post]. Retrieved June 16, 2019, from https://blog.xpnsec.com/evading-sysmon-dns-monitoring/

[10] DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION. (1987, November). Retrieved June 5, 2019, from https://www.ietf.org/rfc/rfc1035.txt

[11] Klein A., Benny P. (2019). DNS Cache-Based User Tracking. In NDSS.

上一篇：CVE-2015-2546 内核U... 下一篇：SUCTF2019，python源...

1. 0 条回复
   - 动动手指，沙发就是你的了！