## 开始

1 今天给大家带来的议题是0day快速分析，大家都知道在0day在野外被广泛利用时，安全厂商都是需要在第一时间进行及时的响应，更新自己的安全防御产品相应规则以防御0day攻击。在传统的IPS/IDS安全产品中，更新规则是个麻烦事，必须首先对出现的0day漏洞进行根本原因的分析，从而写出相应的规则进行拦截，一般都是有专门的漏洞签名团队。本议题介绍几种技巧和方法来帮助分析人员快速的进行漏洞根本原因的定位分析。

2 简单介绍下我自己，我是来自南京翰海源代码实验室成员， binvul组织核心成员，binvul主要关于二进制漏洞的论坛，是一个poc分享平台，感兴趣的可以玩玩。我工作中主要从事xxxx apt防御研究及实现，业余进行二进制漏洞的挖掘、分析、利用。

3 前面也说到了，如何快速的对出现的0day进行分析预警也是非常有意义的，为厂商赢得更多的关注 ，实力输出

更快速的响应产生防御规则及应对措施，保护客户免受攻击

4 漏洞分析会有很多种不同的目的，有的需要找到根本原因，有的是分析利用。漏洞分析的本质 就是找到输入数据和崩溃点或者不符合正常逻辑点的关系。只有理解了本质，才能更好的为漏洞进行命名，同时才能深刻理解漏洞的产生。常见的方法动态调试分析

Windbg/Od

静态分析

BIN IDA

Source Code

输入数据格式分析

已有010脚本

正常样本以及POC样本流程不一致

, 通过分析正常样本和poc样本的数据字段差异，以及导致的程序流程差异从而进行猜测定性漏洞原因。

5 一般情况下遇到0day需要分析，会有2中情形，一种是你已经知道了漏洞名 漏洞产生的原因描述，或者其他人一带而过的简略分析，这个时候这个漏洞分析基本上就没有什么太大的难度了。但是通常的情况下 你只有一个poc ，不知道是什么漏洞名，甚至连原始的正常文件样本都没有，这个时候 它是一个有挑战性的活。

## 漏洞分析难点

6 我们先来看下漏洞分析的一些难点

7 漏洞基本上可以分成2类，一种是内存破坏类型漏洞，其中包括我们常见的堆、栈溢出漏洞，内存未初始化，释放后重用，任意地址读写等，另外一种就是逻辑漏洞，比如java的沙盒逃离漏洞。这里的漏洞分类也对应了漏洞的三大归类，安全功能，安全策略，安全实现。

8 内存破坏漏洞的过程 一般都是污染源数据被使用，然后进行传播，不断的感染，扩散到程序的其他逻辑块，都会反复的进行感染传播，最终导致触发了异常的事件。这个过程一般会很长，存在上百万、上千万条指令的运行传播过程。

9 内存破坏漏洞的定位通常的方法就是通过数据流反塑

从触发异常的污染传染节点回溯到被污染源

从被污染源回朔到污染源

从污染源查找到漏洞代码

从被污染源反朔到污染事件

查找到污染源的初始数据分配,正向分析数据流越界

10 从之前的内存破坏流程图中可以看到内存破坏漏洞分析的诸多难点，

首先是污染源数据动态分配不确定性，传播扩散过程中 各个污染模块的动态分配也是不确定的， 在传播的过程中，部分的污染源会被释放消失，无法直接追溯原始污染源。

由于堆、栈的动态分配特性,导致数据流污染的路径上带有多个难以确认的因素,使得通过数据流污染路径反朔内存漏洞存在更多的困难，我们常见的方法都是通过静态IDA 反汇编、windbg 栈回溯进行逐一查看 速度非常慢。而且对于复杂的漏洞，常见的windbg断点命令都是非常复杂的。对于漏洞分析人员来说 记住这些复杂的命令也是是个非常麻烦的事。

11 我们看 第一次程序的崩溃点，在看第二次程序的崩溃点，可以看到内存地址都是完全不一样的，触发崩溃点的数据也可能是不一样的，(当然了，分析内存破坏漏洞之前需要先把heap page开启)。

12 另外在我们进行数据流反向追溯的过程中，遇到内存中的值是否来污染源的也么有很好的办法得知，如果某个值导致了程序流程出现变化出现问题 那么这个值是不是来自污染源的数据就显得非常重要，可能就直接找到了根本的漏洞原因，还有可能离真相很远。

13 漏洞类型中另一类漏洞属于逻辑漏洞，这类漏洞各自有各自的差异，没有通用的办法进行分析，只能先透彻其逻辑才能进行分析，比如java的沙盒逃离 漏洞，你必须是首先对java的安全模型非常了解才能进行分析。比如路径遍历、杀软的安全功能绕过等。这些漏洞必须要对相应的漏洞载体各自的特征熟悉了解。

## 技巧1

14 下面对于内存破坏类的漏洞介绍2种方法技巧

第一种方法， 我做了个比喻 叫做 假如我知道未来发生什么 因为这种方法可以很好的预知漏洞分析中的关键流程

15 当我们分析漏洞时，如果每次堆分配的堆、栈地址是固定、可预知的…

我们知道这个地址的内存会被申请、释放…

我们知道这个地址的内存会被改写 这样我可以直接对关键的数据进行直接下访问断点。

我经常看到别人的漏洞分析里面下非常复杂的windbg断点，我基本上都用不到，什么非常复杂的windbg 命令都弱爆了，我们只需很简单的数据断点命令即可 ，这样对你来说漏洞分析还会难吗？ 是不是很happy？

16 以往的漏洞分析，都是分析具体的代码逻辑，通过各种回溯、复杂条件断点查找数据流的流向。转变之后的思路，将漏洞分析中 从分析代码逻辑转变到直接分析数据的流向，极大的减少繁琐的回溯层数。减少复杂度。

17 这些思路我们都可以通过VM快照来实现， 我们通常会在工作中通过vm快照来保存各种各样的环境，运用于漏洞分析是一个非常好的办法，建立合适的点，会将我们要分析的漏洞各种信息完好的保存。

18 通过VM快照来分析漏洞有一个很关键的地方 就是建立一个起始分析的点，只有建立好这个点才能确保堆、栈地址都是一致的，才能直接只跟踪数据流 不关心具体代码。我们需要一个等待堆布局基本稳定的时候建立快照，对于不同类型有不同的对待。对于文件类型，可以在应用程序打开POC处理时，不过最好是在打开POC真正处理时，很多情况下，应用程序打开POC只是简单判断下头或者大小就关闭了，我们需要找到真正开始处理的 。

对于网页类型，可以找到浏览器处理第一条脚本语句的时候，

网络类型，可以在接受应用程序数据包的时候，或者初始化连接时

很多时候上述点并不一定够，可能需要更深入的关键点，比如一些类的、接口的初始化的时候，或者外部数据初始化内部数据结构的时候。 一般情况下多试几次 就可以了

==============================================================================

19 下面通过2个例子来实战看下 这个方法带来的快捷之处

### 例子一

20 第一个例子是CVE2011-2462 是2011年10月APT攻击洛克马丁军火商被逮到的adobe reader 处理U3D格式的漏洞， 分析前没有任何细节， 由于漏洞细节较有难度，各大厂商未能及时发布细节，针对该漏洞分析方法在cansecwest2012上有微软的人通过介绍流程比较的方法进行分析。当然了 他们是在有原始正常样本和poc样本的情况下分析的。而我们没有，只有一个poc。 当时满天的附件都夹带着这个漏洞的攻击样本，翰海源首发该漏洞细节和防御规则。

21 当时分析出来时， 有人微博评论说让漏洞在飞一会，不然飞鸟尽，良攻藏。确实，现在野外广泛运用的漏洞越来越少了，漏洞攻击慢慢变成了国家层面对抗。我们这些安全人员出路在哪呢？也是值得我们思考的。

22 首先来建立初始快照，我选的是第一次readfile poc的点，然后run起来poc ，看看前后crash点的数据地址是否一样，若一样就差不多了。

23 大家可以看到，建立快照之后，每次恢复快照，run起来之后的crash点的数据地址都是完全一样的。Xxxx 一样

24 我们简单看下漏洞溯源，分析过这个漏洞的可能比较清楚，首先是从crash点向上开始分析，第一层分析的落脚点就是绘制u3d node节点时已经出现问题，可以看到ukclass=\*(node+0x68);

而\*(\*(ukclass+0x48)+0x54)=0x52520a50 此处已经已是非法值，首先我们需要跟踪这个node+0x68处的数据从哪来的，地址是dd 0352d698+0x68

25 我们恢复快照，直接对这个地址进行下数据访问断点即可 run之后 很快就找到了写入改地址的位置。 这个只是找到了第二层，在通过这种方法 经过几层的追踪，很快就找到了漏洞触发的根本原因的地方。原本这个漏洞的回溯可能要十几层到二十基层 非常的繁琐。

26 经过分析 找到了漏洞的最根本原因，未初始化漏洞。 这里是一个获取数据buffer地址的函数调用，从里面获取一个dword shader\_count，若该值为0的话，则就导致了绕过了下面关键指针的初始化话。 刚好这个未初始化的地址里面的数据来此污染源中的，从而顺利执行shellcode，我们通过这个就可以写出相应的防御漏洞签名规则了。

=========================================================================

### 例子二

27 第二个例子介绍的是ie的漏洞 CVE2012-4969 ，**IE execCommand function Use after free Vulnerability。这个漏洞也被用来大量的挂马攻击，是国外安全研究员eronmang不小心发现的攻击样本。我们翰海源也是首发这个漏洞的细节。**

**这个漏洞发现过程也是非常的搞笑，当时在这个漏洞出现之前有一个java挂马的漏洞，eronmang研究员分析这个挂马漏洞的网站时，不小心看到其他目录有个exploit。Html文件，然后好奇的打开看了看 然后就发现这是另一个挂马的利用的0day漏洞。**

这里其实我们可以看到一些其他的东西，apt防御检测中 检测恶意的源也是一种非常有效的办法，比如我们都知道枪击犯可能在很多次的枪击过程中都是用的同一把枪之类的，类比于APT攻击。前天fireeye发的利用刚被补掉的ie8的漏洞进行攻击的，采用的web server也是以前被利用过的恶意站点。we found that the web server has a history of hosting malware

另一个就是APT的攻击目标我们也可以进行重点检测。比如一个长受攻击的email邮箱 可以进行重点监控之类的。

28 这个例子是个浏览器漏洞，首先我们想到的是通过加入js的debug 日志来大致先确定这个漏洞的对象执行流程。 通过这个debug js语句和与之对应的断点，将漏洞过程细分 在详细的进行定位，是一个非常常用的技巧。

29 从刚才的debug 信息可以看到这个漏洞的大致执行流程，首先在html页面加载时，首先注册2个回调，一个onload会调，一个selectAll会调，页面加载后，首先调用onload会调，会调用document.execommand 函数,该函数参数为SelectAll,执行该命令时 ，又触发了selectAll会调，这个会调会通过document.write 清空页面。 可以看出这个crash是发生在document.execommand执行完毕之前的。 此时，我们对这个漏洞有了一个大致的了解和猜测。

20分钟

30 运用虚拟机快照分析法，在执行document.execommand 语句之前建立快照。

31 我们看到crash时，edi指向的地址此时已经变成了非法值，我们需要跟踪这个edi地址

哪里变成了非法值，这个非法值就是POC里面src站位的值。恢复虚拟机快照，直接下数据访问断点ba w 1 03bb4f00 即可。

### DEMO

我们简单看下演示调试过程

开虚拟机、下断点， 清楚0断点 F5 go 看栈回溯

32 通过调试，确定了最终的原因，execommand函数调用CMshtmlEd\_\_Exec

函数 ，该函数调用AddCommandTarget函数添加对应的命令，函数执行时，会申请一个MshtmlEd对象，同时会触发已经注册的selectAll会调，该会调通过document.write 清空了页面，导致了将已经申请的MshtmlEd对象通过DeleteCommandTarget函数释放掉，而execommand继续执行仍然去调用MshtmlEd对象的成员函数 导致了uaf 漏洞的产生。

33 这个方法的适用范围很广，一切的内存破坏漏洞

无论是内核还是应用层

无论是windows还是linux 系统

## 技巧2

34 前面介绍了我们通过虚拟机快照的方法来预知漏洞发生的关键数据点，下面介绍一个思路来告诉我们漏洞分析中数据是从哪来的。 我比喻成假如我知道你来自哪里

35 内存破坏漏洞本质是外界的输入数据导致程序内部逻辑、数据变量等出现异常导致出现EIP被外界数据控制。

当反向溯源时 执行过程中出现的某个数值如何判断是来自外界的数据？ 如果搞不清这个外来污染数据和触发点的关系 就无法分析出根本原因。比如下面这张图，通过虚拟机快照分析到最后时，如何确定这个shader\_count是来自文件中的 ？或者不是文件中的，那么我们还需要在进行追溯。如果知道了 那么就可以直接确定原因了

36 分析漏洞时基本都会有这2种情形，有原始poc和变异的poc，和只有poc。对于前者可以通过对比获取出现问题的字段，正向的调试跟踪这个字段的处理。或者对比2个样本的执行流程。如果只有poc的话，那就不知道出现问题的字段了，一切靠猜。

37 只有poc的情况也是有2种方法的，第一种方法就是执行过程的中的某数值影响了流程 、变量时，在污染源中进行简单的多搜索，若找到，尝试修改测试是否一致，这种方法必须是这个值很特殊，不然也没法定位，比如00000000 或者 1 之类的。肯定没有办法搜索。

第二种就是我们着重讲的，通过对外界进行污染数据传播然后直接定位标记。

38 对于污染源标记传播，已经有了一些开源的框架和传播引擎代码。动态插桩、指令模拟的工具有pin、quem、dynamo\valgrind\Bochs 等等。

开源的污染标记工具比如miemu libdft dfwin

39 我这里自己实现了个传播的引擎，因为在写这个时候没有发现这些开源的引擎。有些小悲剧。我的实现是通过pin+bochs+ida脚本来实现的污染源标记的。

Pin插庄工具

非常稳定、支持多线程、支持函数hook

Bochs 指令模拟

支持绝大多数指令解析模拟、稳定、代码逻辑清晰

IDA Ptyhon 脚本

Idb中直接标记污染数据、便于逆向分析查看

40 这里简单列了下实现的架构，首先通过pin注册指令回调和相应的一些函数HOOK，指令回调调用bochs的指令解析，然后通过污染传播引擎进行传播，函数HOOK主要是HOOK 污染源的输入点，以及拷贝污染源、释放污染源操作的函数API。将这些污染的数据记录存入数据库中，然后在通过ida python脚本处理 输入相应的ida idb中，直接进行颜色标记显示。

41 这里的图就是通过监视adobe reader pdf 打开poc样本 标记3difr.X3D模块之后的显示部分，绿色的标记是表示这个是直接来自外来数据BUFFER的，浅蓝色是外来的数据进行传播的过程。0x35d是表明这个数据来自文件中的offset，10101010时表示污染的位图，表示4个字节全是污染的，后面0-0xffffffff是具体的范围。这里的赋值给esi的就是U3D 结构中的shader\_list\_count字段

42 这里标记的绿色的数据是U3D 结构中的shader\_count字段，可以看到这里jbe跳走了，来自文件中offset 0x361 偏移。

如果我们有这个标记工具，那么当我们分析前面的那个pdf漏洞时，直接就可以找到外来污染数据和漏洞的关联，直接定性根本原因，不用自己在跟踪具体的数据来源，省了很多时间。

43适用于文件类型、网络类型内存破坏漏洞

used after free类型、脚本解释类型的漏洞不适用

44 同时这个污染源标记的用处非常之多

用于漏洞挖掘，替代人工追踪数据流

用于漏洞分析，轻易展现污染的数据

用于漏洞攻击防御，检测是否外来数据控制了EIP

用于归类漏洞攻击，归类漏洞触发点

用于自动产生防御规则，归类漏洞触发字段

优点多缺点也多 性能 传播准确性等都是很有挑战的。

## 结尾

45

1. 虚拟机快照分析和污染源来源记录对于分析漏洞非常有帮助，基本上掌握了分析漏洞可以说没有什么难度了，我就是这么做的
2. 方法并不是万能的，有的时候更依赖于对漏洞本身载体的内部机制非常熟悉

3 为了漏洞分析去分析 没有意义！（

1 通过分析这一到两个漏洞，洞悉这类漏洞的本质，漏洞分析十几种类型就可以了，其他除非是非常nice exploit 可以着重分析

2 通过分析漏洞 来窥视这个软件的内部结构逻辑

盲人摸象，你模的越多 ，整体内部结构了解的越多，下次再有这个软件漏洞，手到擒来

3 举一反三，查找该软件是否存在其他相似漏洞 同一模块出现相似的漏洞的概率非常高，

例如MS11-046 和MS11-080 的afd。Sys 的iocontrol 本地提权漏洞 完全一样的。

4 思维扩展，其他软件是否存在类似漏洞 像最近的adobe pdf 沙盒逃离漏洞，本质其实是api的buffer和size的不一致导致的，adobe也只是简单的讲size 除以2进行修补， 但是聪明的人会想是否存在其他的这样的api？其他sandbox是否存在这样的问题？ 国外一研究院就发现firefox的flash sandbox也存在一模一样的问题。

最近我们发现的一个从普通用户提权到system的漏洞也是类似，从一个漏洞变成5个漏洞，非常好玩，就是一个思维扩展的过程

5 思考猜测这个漏洞大概如何通过什么方式被挖掘出来

是fuzz出来的 还是手工找出来的？

6 思考自己，若这样的代码放在你面前，你能否挖掘出来？

若不能，总结这类漏洞的模型进行提炼、汇编形态进行学习

46 感谢大家，要讲的内容就这么多，看大家对这个议题有什么疑问的地方可以讨论讨论。谢谢