

深入探究文件Fuzz工具之Peach实战

overXsky

2016-11-28



共661480人围观，发现 6 个不明物体

工具

极客

***本文原创作者：overXsky，本文属FreeBuf原创奖励计划，未经许可禁止转载**

0x0 前言

本文旨在详细介绍文件Fuzz工具Peach的使用，涵盖了基于Python的2.3旧版本和基于C#的3.1新版本，讲求实践与语法并重，适合初学者和有一定其他Fuzz工具使用经验的人作为学习参考。文中若有不当之处，还望不吝指正。

说明：本文中穿插对比了3版本较于2版本有较大改动的地方，并以“V3：”特意标注。

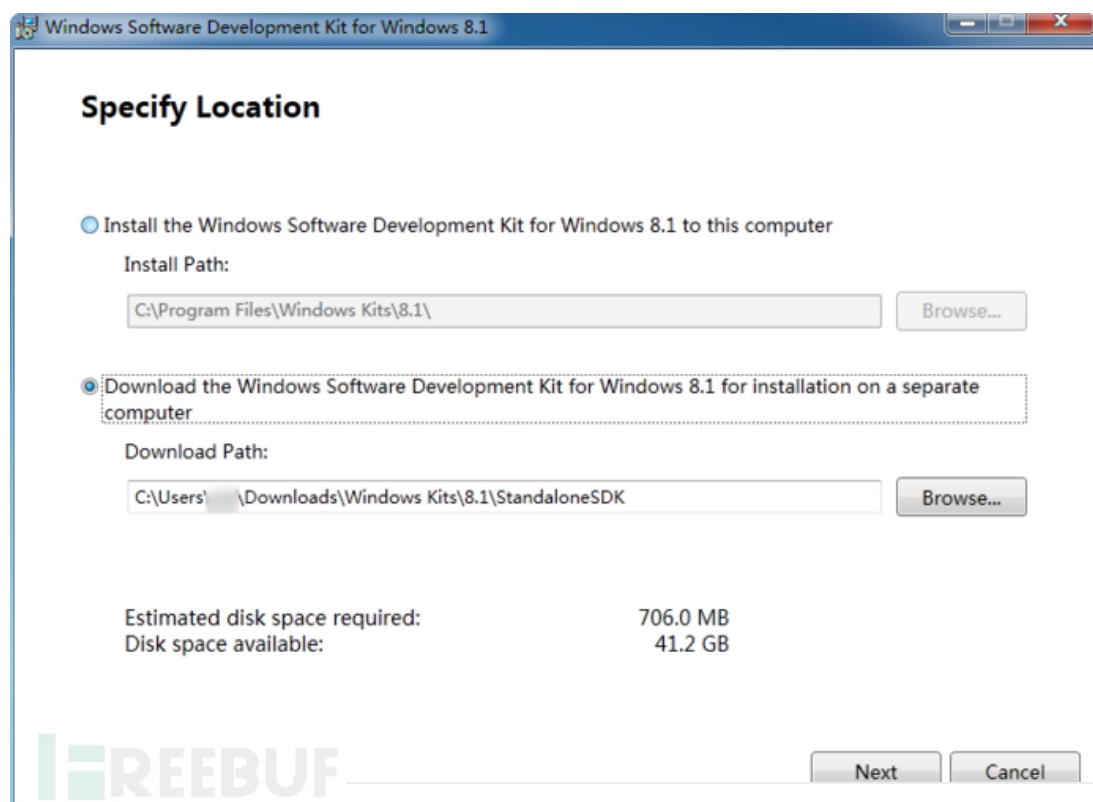
0x1 Peach介绍与安装

Peach是用python/C#写的开源Smart Fuzz工具，支持两种Fuzz方法：基于生长(Generation Based)、基于变异(Mutation Based)

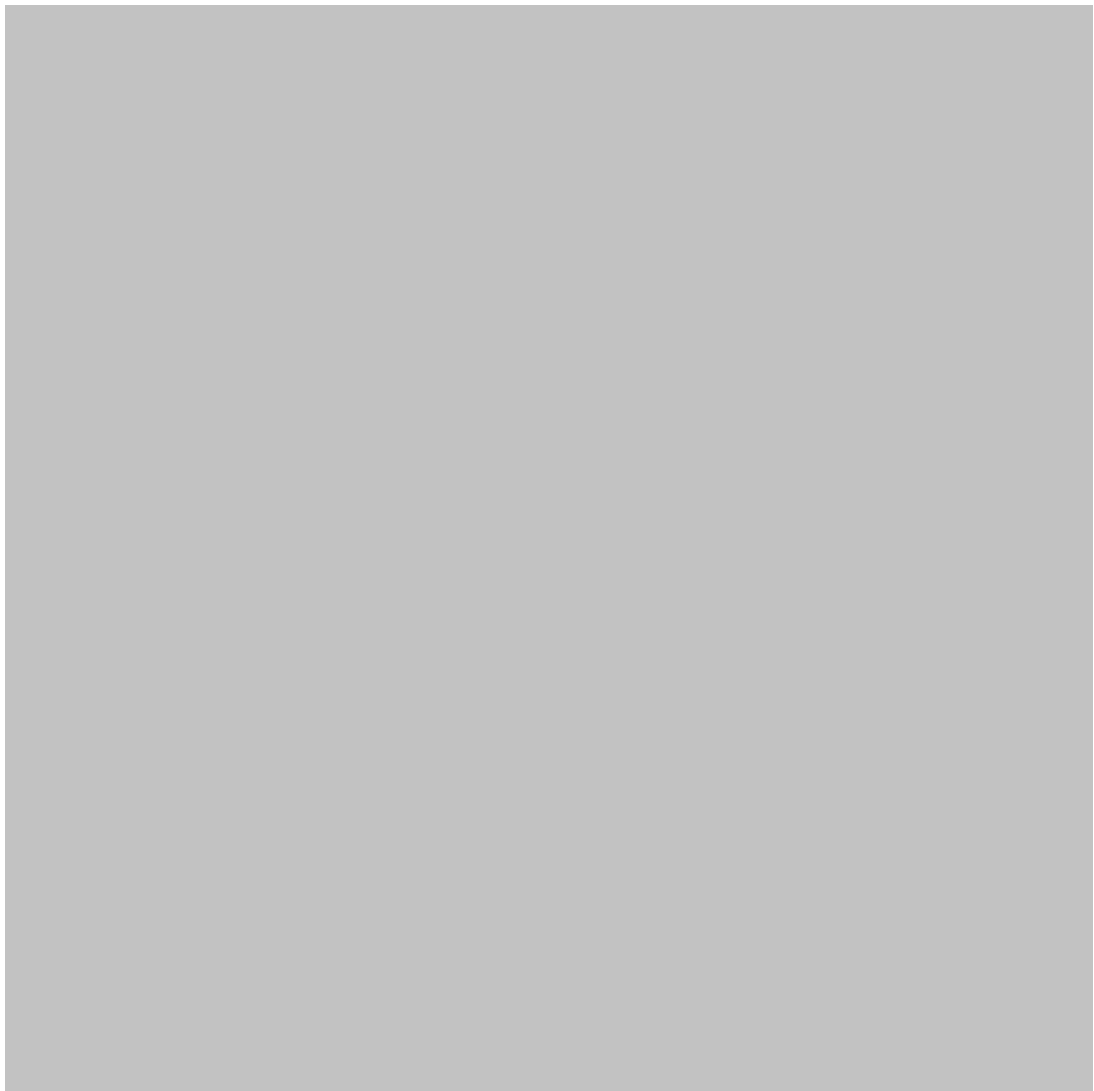
- 1.基于生长：产生随机或启发性数据填充给定的数据模型
- 2.基于变异：在给定样本文件的基础上进行修改

在32位win7上安装的具体步骤：

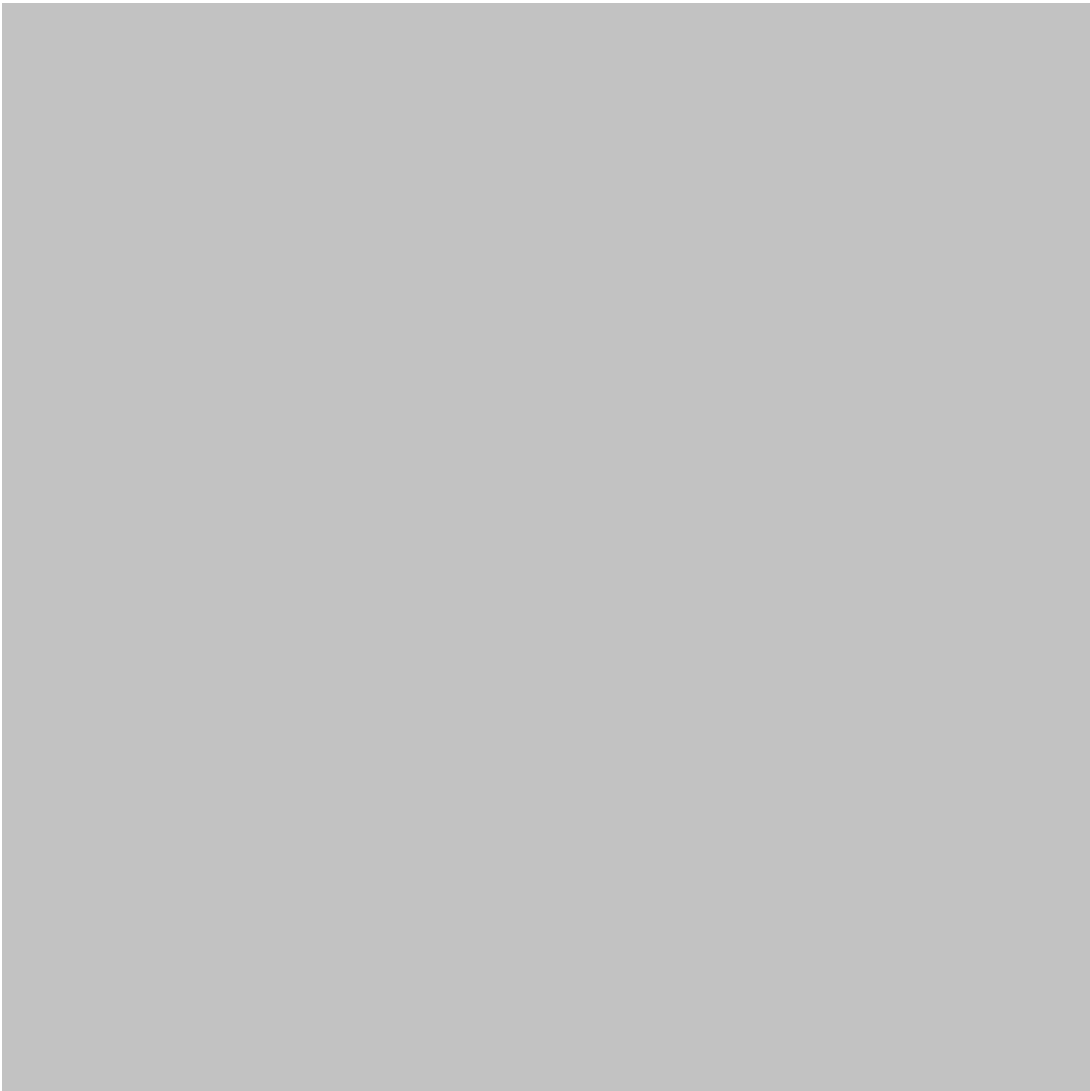
- 1.[Peach3安装指南](#)
- 2.[下载安装.NET4运行环境](#)
- 3.[下载安装Debugging Tools for Windows](#)



根据官网的提示，如果想要独立安装，那么在安装SDK的中途做出选择即可

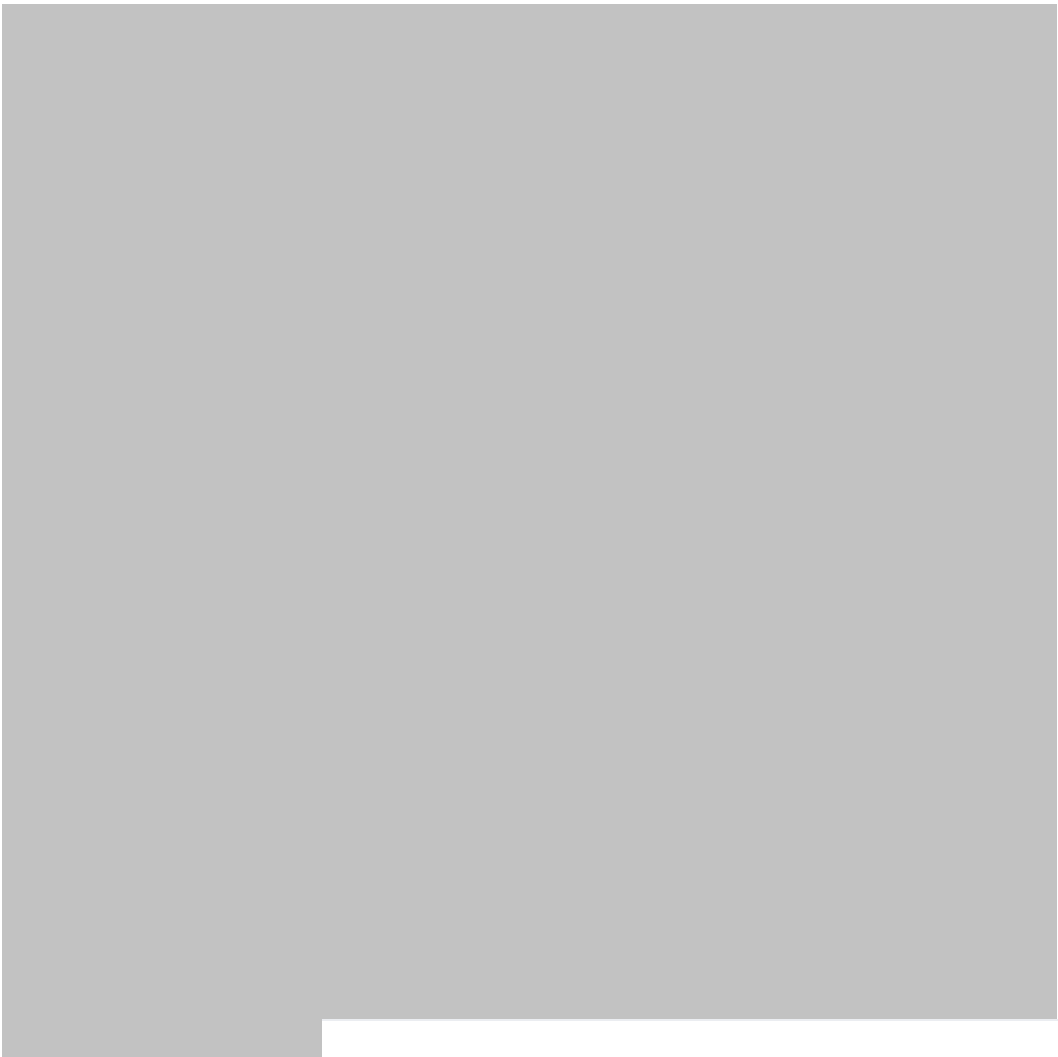


下载完后根据提示找到文件保存位置





- 4.在[Peach Community](#)上找到[Peach 3的x86-release版本下载入口](#)
- 5.解压源码到任意文件夹
- 6.运行以下命令，测试自带的HelloWorld能否正常运行。发现cmd开始不停滚动，成功！



0x2 定义Peach Pit

为了开始进行文件 Fuzz，首先必须创建一个该格式的 Peach Pit 文件。Peach Pit 文件使用XML语言定义数据结构，里面记录了文件的组织方式，我们需要如何进行 Fuzz，Fuzz 的目标等信息

Peach Pit，包含5个模块：

- 1.GeneralConf(V3:General Configuration)
- 2.DataModel(V3:Data Modeling)
- 3.StateModel(V3:State Modeling)
- 4.Agents and Monitors(V3:不变)
- 5.Test and Run Configuration(V3:Test Configuration，去掉了Run)

一个简单的XML框架：

```
<?xml version="1.0" encoding="utf-8"?>

<Peach xmlns=http://phed.rig/2008/Peach xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://phed.org/2008/Peach ../peach.xsd">

<!-- add elements here -->

</Peach>
```

V3:需要将以上包含...2008...的字段改为以下内容：

```
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">
```

1.GeneralConf / General Configuration

定义基本配置信息，包括

- 1.Include——要包含的其他Peach Pit文件
- 2.Import——要导入的python库或自定义模块
- 3.PythonPath——要添加的python库的路径
- 4.所有的Peach Pit文件都要包含default.xml文件

在HelloWorld中：

```
<Include ns="default" src="file:defaults.xml" />
```

V3:强制包含default.xml这一限制在版本3中已经去掉，改为Defaults属性包含，[参考](#)

2.DataModel

DataModel 元素是 Peach 根元素的子节点，常用属性主要是以下两个：

1.name——相当于起一个“变量名“

2.ref——引用标签，是否继承于某一个已定义的DataModel（这里继承的含义类似于面向对象中的继承，被引用的 DataModel 将为新的 DataModel 的基数据，可以重用或更新子元素）

DataModel定义了某一格式类型文件的数据模型，包括数据结构、数据关系等，常用的子元素有以下几种：

1.String——字符串型，包含了name，length等常用属性（V3:需要特别注意，在版本2中表示字符串不可变的isStatic属性在版本3中名为token）

2.Number——数据类型，可以是8、16、32或64字节的数据，并通过size属性来设定，同时也包含name属性作为名称

3.Blob——无具体数据类型，通常用来定义不明类型的数据，包含name、length、value等属性

4.Block——用于对数据进行分组，用来组合一个或者多个的其他元素，比如 Number 或者 String，有点类似于DataModel，只不过者的位置不同。

小技巧：可以把一个复杂的数据结构拆成几部分，在一个 Peach Pit 文件里定义多个不同层级的 DataModel，并通过 ref 标签层层递进，从而增强可读性和可重用性。

举例如下：

```
<DataModel name="HelloData">

<String name="ID" size="32" value="RIFF" isStatic="true" />

<Block name="TypeAndData">

<Number name="Type" size="16" />

<Blob name="Data" />

</Block>

</DataModel>
```

在HelloWorld.xml中：

```
<DataModel name="HelloWorldTemplate">

<String value="Hello World!" />

</DataModel>
```

3.StateModel

- 描述如何向目标程序发送或接受数据
- 每个StateModel至少由一个State组成，第一个State由InitialState指定，并且State只有一个属性name
- 每个State至少由一个Action组成，定义StateModel中的各种动作，动作类型由type指定
- type包括start,stop,open,close,input,output,call等

举例如下：

```

<!--输入型动作-->

<Action type="input">

<DataModel ref="InputModel" />

</Action>

<!--输出型动作，输出到文件sample.bin中-->

<Action type="output">

<DataModel ref="SomeDataMode" />

<Data name="sample" filename="sample.bin" />

</Action>

<!-- 调用动作，按照数据模型Param1DataModel产生的数据作为函数DoStuff的参数 -->

<Action type="call" method="DoStuff">

<Param name="param1" type="in">

<DataModel ref="Param1DataModel" />

</Param>

</Action>

<!-- 关闭动作 -->

</Action type="close" />

```

在Helloworld中，只需要接收数据模型HelloWorldTemplate中的数据

```

<StateModel name="state" initialState="State">

<State name="State1">

<Action type="output">

<DataModel ref="HelloWorldTemplate" />

</Action>

</State>

</StateModel>

```

4. Agents and Monitors

定义代理和监视器，可以调用Windbg等调试器来监控程序运行的错误信息等，目前已有的Agent包括Local Agent、TCP Remoting Agent、ZeroMQ和REST Json Agent

Agent下定义Monitor来监视

举例如下：


```

<Agent name="LocalAgent" Location="http://127.0.0.1:9000">

<!-- 调用WinDbg来执行notepad.exe filename命令 -->

<Mointor class="debugger.WindowsDebugEngine">

<!-- V3:class统一名为WindowsDebugger -->

<Param name="CommandLine" value="notepad.exe filename命令" />

</Mointor>

<!-- 开启页堆调试，在大多数Windows Fuzzing中很有用 -->

<Monitor class="process.PageHeap">

<!-- V3:class统一名为PageHeap -->

<Param name="Executable" value="notepad.exe" />

</Monitor>

</Agent>

```

5.Test and Run configuration / Test Configuration

版本2中包括Test和Run两个元素，版本3中只要Test一个元素即可

Test用来定义一个测试的配置，包括一个StateModel和一个Publisher（必需），以及including和excluding、Agent（可选）等信息
例子：

```

<Test name="TheTest">

<!-- V3:几乎所有Test name都命名为Default，否则会报错 -->

<Exclude xpath="//Reserved" />

<Agent ref="LocalAgent" />

<StateModel ref="TheState" />

<!-- 将生成的畸形数据写到FuzzedFile文件中 -->

<Publisher class="file.FileWriter">

<Param name="fileName" value="FuzzedFile" />

</Publisher>

</Test>

```

Publisher用来定义Peach的IO连接，可以构造网络数据流和文件流等

HelloWorld中只需要把生成的畸形数据显示到命令行，所以Publisher用的是标准输出stdout.Stdout(版本2)/Console(版本3)

```

<Test name="HelloWorldTest">

```

```
<StateModel ref="State" />

<!-- 将生成的畸形数据写到FuzzedFile文件中 -->

<Publisher class="stdout.Stdout" />

</Test>
```

(以下一段Run是版本2)

Run元素定义运行哪些测试，包含一个到多个Test元素，（可选）用Logger配置日志来捕获运行结果

举例

```
<Run name="DefaultRun">

<Test ref="TheTest" />

<!-- 把日志记录到c:\peach\logtest路径下 -->

<Logger class="logger.Filesystem">

<Param name="path" value="c:\peach\logtest路径下" />

</Logger>

</Run>
```

而版本3中，Logger作为Test的子元素，写在其中即可，如下所示：

```
<Test name="Default">

....

<Logger class="Filesystem">

<Param name="Path" value="logs" />

</Logger>

</Test>
```

在HelloWorld中:

```
<Run name="DefaultRun">

<Test ref="HelloWorldTest" />

</Run>
```

0x3 HelloWorld.xml初体验

根据之前的分析介绍，不难写出一个简单的HelloWorld代码，各位读者不妨先自己动手写写看。

给出示例——在2版本下能够运行的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>

<Peach xmlns="http://phed.org/2008/Peach" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://phed.com/2008/Peach ../peach.xsd">

<Include ns="default" src="file:default.xml" />

<DataModel name="HelloWorldTemplate">

<String value="Hello World!"/>

</DataModel>

<StateModel name="State" initialState="State1">

<State name="State1">

<Action type="output">

<DataModel ref="HelloWorldTemplate"/>

</Action>

</State>

</StateModel>

<Test name="TheTest">

<StateModel ref="State"/>

<!-- 将生成的畸形数据写到FuzzedFile文件中 -->

<Publisher class="stdout.Stdout"/>

</Test>

<Run name="DefaultRun">

<Test ref="TheTest" />

<!-- 把日志记录到c:\peach\logtest路径下 -->

<Logger class="logger.Filesystem">

<Param name="path" value="c:\peach\logtest" />

</Logger>

</Run>

</Peach>
```

而对于3版本就需要改动一部分：

1. 在实际调试中，修改了几处，首先是把xmlns改成了<http://peachfuzzer.com/2012/Peach>，否则会出现更新提示
2. 删除了include，因为报错提示说找不到default.xml文件
3. Test name改为了Default，因为报错说找不到default

4. 删除了run元素, 报错提示说在当前命名空间下子元素非法, 应当出现的元素为: 'Include, Import, Require, PythonPath, RubyPa Python, Ruby, Defaults, Data, DataModel, Godel, StateModel, Agent, Test'

给出示例——在3版本下能够运行的代码如下:

```
<?xml version="1.0" encoding="utf-8"?>

<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<!-- add elements here -->

<DataModel name="HelloWorldTemplate">

<String value="Hello World!"/>

</DataModel>

<StateModel name="State" initialState="State1">

<State name="State1">

<Action type="output">

<DataModel ref="HelloWorldTemplate"/>

</Action>

</State>

</StateModel>

<Test name="Default">

<StateModel ref="State"/>

<!-- 将生成的畸形数据写到FuzzedFile文件中 -->

<Publisher class="stdout.Stdout"/>

</Test>

</Peach>
```

官方给出的HelloWorld.xml如下, 大家可以参考:

```
<?xml version="1.0" encoding="utf-8"?>

<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<!--

This is a very simple Hello World example.

Syntax:

peach samples\HelloWorld.xml
```

Output:

This example will display all each test case to the console, you should see lots of test data scroll along. This example should only take a couple minutes to complete.

Authors:

Michael Eddington (mike@dejavusecurity.com)

-->

<!-- Create a simple data template containing a single string -->

```
<DataModel name="TheDataModel">
```

```
<String value="Hello World!" />
```

```
</DataModel>
```

<!--

Our state model will perform actions using our data models. Actions are things like sending or receiving data. How the data is sent or received will depend on the publisher we specify later on. For example you might configure to use a FileWriter publisher that would write a file. For this example we will be displaying the output to the console via standard out (Stdout).

-->

```
<StateModel name="State" initialState="State1" >
```

```
<State name="State1" >
```

```
<Action type="output" >
```

```
<DataModel ref="TheDataModel"/>
```

```
</Action>
```

```
</State>
```

```
</StateModel>
```

<!-- Our Test element will link together our state model and publisher -->

```
<Test name="Default">
```

```
<StateModel ref="State"/>
```

<!--

The publisher we select will determin what the actions in our state model actually do. For this example we are using the "Stdout" publisher. This publisher will send any "output" actions to the console.

```
-->

<Publisher class="Console" />

</Test>

</Peach>

<!-- end -->
```

0x4 数据依赖关系

[参考官方文档](#)

Peach 允许数据间的关系建模。关系像“X 是 Y 的大小”，“X 是 Y 出现的次数”，或者“X是 Y 的偏移（以字节为单位）”。

Relation元素可以用来表示数据长度、数据个数以及数据偏移等信息，格式如下：

```
<Relation type="size" of "Data" />

<Relation type="count" of "Data" />

<Relation type="offset" of "Data" />
```

size-of Relation

这个例子里，Number 会指定 Value 字符串的大小，以字节为单位。注意，这对多字节字符（wchar）同样适用。

```
<Number size="32" signed="false">

<Relation type="size" of="Value" />

</Number>

<String name="Value" />
```

count-of Relation

这个例子里，Number 会指定 Strings 数组的个数。

```
<Number size="32" signed="false">

<Relation type="count" of="Strings" />

</Number>

<String name="Strings" nullTerminated="true" maxOccurs="1024" />
```

[参考官方文档](#)

Fixup元素可以用来表示数据校验值，包括CRC32、MD5、SHA1、SHA256、EthernetChecksum、SspiAuthentication等，格式如

```
<Fixup class="FixupClass">

<Param name="ref" value="Data" />
```

</Fixup>

以如下数据模型为例：

Offset	Size	Description
0x00	4 bytes	Length of Data
0x04	4 bytes	Type
0x08		Data
after Data	4 bytes	CRC of Type and Data

```
<DataModel name="HelloData">

  <Number name="length" size="32">

    <Relation type="size" of "Data"/>

  </Number>

  <Block name="TypeAndData">

    <String name="Type" size="32"/>

    <Blob name="Data"/>

  </Block>

  <Number name="CRC" size="32">

    <Fixup class="checksums.Crc32Fixup">

      <Param name="ref" value="TypeAndData"/>

    </Fixup>

  </Number>

</DataModel>
```

0x5 PNG格式文件Fuzz实战

目的：以版本3为例，了解如何具体使用peach来做一次完整的有针对性的Fuzz（现在互联网上流传的基本都是旧版或者不完整的片段，本文代码是经过作者亲自测试有效的）

PNG文件格式如下图（图片来源于网络）：



这里先介绍一款文件格式解析神器：010Editor，大家可以自行在官网下载安装，完成之后用它打开本地的一个png图片，如下所示。





注意红框圈住的部分，可以很清楚的看到png文件内部区块的分区和含义，进而归纳出PNG的文件格式——

PNG格式如下：

- 1.文件头：由八个字节组成，0x89504e470d0a1a0a
- 2.数据块：每个数据块由四部分构成，他们的描述依次如下：
- 3.Length：占四字节，表示数据块data域占多少个字节。（注意这里不包括length自身）
- 4.Type：占四字节，表示当前块的类型。一般是英文大小写字母的ASCII码（65~90或者97~122）
- 5.Data：数据区。大小可以是0字节
- 6.CRC：占四个字节，整个chunk的CRC校验码（Length+Type+Data）

有了之前的经验，不难写出如下的PNG DataModel：

```
<!-- 定义Chunk的数据结构 -->

<DataModel name="Chunk">

  <!-- Chunk第一部分: uint32 length -->

  <Number name="Length" size="32" signed="false" endian="big">

    <Relation type="size" of="Data"/>

  </Number>

  <!-- Chunk第二和第三部分: type code, 4个英文字母; 由第一部分length指定长度的Data块-->

  <Block name="TypeAndData">

    <String name="Type" length="4"/>

    <Blob name="Data"/>
```

```
</Block>

<!-- Chunk第四部分: crc校验码 -->

<Number name="CRC" size="32" endian="big">

<Fixup class="Crc32Fixup">

<Param name="ref" value="TypeAndData"/>

</Fixup>

</Number>

</DataModel>

<!-- 将PNG文件视为一个8字节签名+若干个Chunk -->

<DataModel name="Png">

<Blob name="pngSignature" valueType="hex" value="89 50 4E 47 0D 0A 1A 0A" token="true"/>

<Block ref="Chunk" minOccurs="1" maxOccurs="1024"/>

</DataModel>
```

接下来就是StateModel部分，这里以一款名为pngcheck的轻量级可执行程序来打开png文件，[下载链接](#)

```
<!-- StateModel完成三件事: 修改文件生成畸形文件; 把该文件关闭; 调用pngcheck打开生成的畸形文件; -->

<StateModel name="TheState" initialState="Initial">

<State name="Initial">

<!-- 输出png文件 -->

<Action type="output">

<DataModel ref="Png"/>

<!-- 读入png文件 -->

<Data name="data" fileName="D:\code\peach\PNG\circle.png"/>

</Action>

<!-- 关闭png文件 -->

<Action type="close"/>

<!-- 调用pngcheck进程 -->

<Action type="call" method="D:\software\pngcheck-2.3.0-win32\pngcheck.exe" publisher="Peach.Agen

<!-- Param用来存放传递给pngcheck.exe的参数, 即畸形文件的文件名 -->

<Param name="png file" type="in">

<DataModel ref="Param"/>

<!-- 定义输出文件 -->
```

```
<Data name="filename">

<Field name="value" value="D:\code\peach\PNG\fuzzedCircle.png" />

</Data>

</Param>

</Action>

</State>

</StateModel>
```

现在我们为Peach Pit加上测试：

```
<Test name="Default">

<StateModel ref="TheState" />

<Publisher class="File">

<Param name="FileName" value="D:\code\peach\PNG\fuzzedCircle.png" />

</Publisher>

</Test>
```

到这里已经可以运行测试了，不过这样子只会生成畸形文件，对于实际应用并没有什么用处。只有加上Agent和Monitor，对分析结果行监控和日志记录才是最终需要的。

因为作者使用的是Win平台，所以调用的 class 就是 WinAgent了，其他平台的可以做出相应变换。

```
<Agent name="WinAgent">

<Monitor class="WindowsDebugger">

<Param name="CommandLine" value="D:\Software\pngcheck-2.3.0-win32\pngcheck.exe D:\code\peach\PNG

<!-- 可以用name值为WinDbgPath的Param来手动指定windbg的位置 -->

<!-- 删除LaunchViewer可以不调用GUI界面加快运行速度 -->

<Param name="StartOnCall" value="LaunchViewer" />

<Param name="CpuKill" value="true" />

</Monitor>

<Monitor class="PageHeap">

<Param name="Executable" value="D:\Software\pngcheck-2.3.0-win32\pngcheck.exe" />

</Monitor>

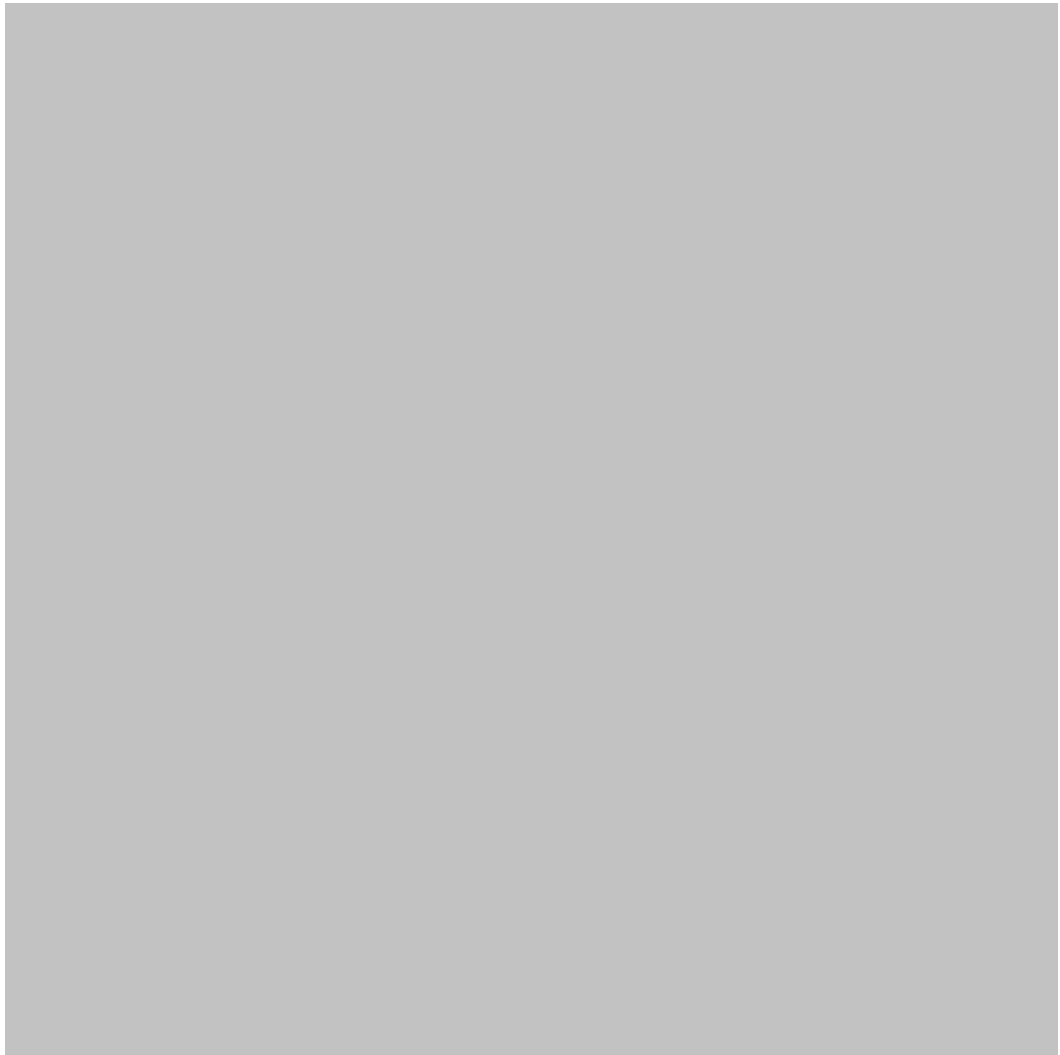
</Agent>
```

至此基本就大功告成了，运行一下看看：

0x6 举一反三：WAV格式Peach Pit编写

为了更好地帮助大家巩固以上知识，再给出一个完整的WAV Peach Pit（基于版本3）：

WAV文件格式：



Pit:

```
<?xml version="1.0" encoding="utf-8"?>

<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <Defaults>

    <Number signed="false" />

  </Defaults>

  <!-- 定义WAV文件格式 -->

  <DataModel name="Wav">

    <!-- Wav header -->

    <String value="RIFF" token="true" />

    <Number size="32" />
```

```
<String value="WAVE" token="true" />

</DataModel>

<!-- 定义一般的wave区块，作为基类，之后会经常被ref引用 -->

<DataModel name="Chunk">

<!-- RIFF -->

<String name="ID" length="4" padCharacter=" " />

<!-- Data length in bytes, 4字节，其值=总文件大小-8字节 -->

<Number name="Size" size="32">

<Relation type="size" of="Data" />

</Number>

<!-- 除去RIFF文件头和data length共8个字节后面的所有数据 -->

<Blob name="Data" />

<Padding alignment="16" />

</DataModel>

<DataModel name="ChunkFmt" ref="Chunk">

<String name="ID" value="fmt " token="true"/>

<Block name="Data">

<Number name="CompressionCode" size="16" />

<Number name="NumberOfChannels" size="16" />

<Number name="SampleRate" size="32" />

<Number name="AverageBytesPerSecond" size="32" />

<Number name="BlockAlign" size="16" />

<Number name="SignificantBitsPerSample" size="16" />

<Number name="ExtraFormatBytes" size="16" />

<Blob name="ExtraData" />

</Block>

</DataModel>

<DataModel name="ChunkData" ref="Chunk">

<String name="ID" value="data" token="true"/>

</DataModel>

<DataModel name="ChunkFact" ref="Chunk">

<String name="ID" value="fact" tok
```

```
<Block name="Data">

<Number size="32" />

<Blob/>

</Block>

</DataModel>

<DataModel name="ChunkSint" ref="Chunk">

<String name="ID" value="sInt" token="true"/>

<Block name="Data">

<Number size="32" />

</Block>

</DataModel>

<DataModel name="ChunkWavl" ref="Chunk">

<String name="ID" value="wavl" token="true"/>

<Block name="Data">

<Block name="ArrayOfChunks" maxOccurs="3000">

<Block ref="ChunkSint"/>

<Block ref="ChunkData" />

</Block>

</Block>

</DataModel>

<DataModel name="ChunkCue" ref="Chunk">

<String name="ID" value="cue " token="true"/>

<Block name="Data">

<Block name="ArrayOfCues" maxOccurs="3000">

<String length="4" />

<Number size="32" />

<String length="4" />

<Number size="32" />

<Number size="32" />

<Number size="32" />

</Block>

</Block>
```

```
</DataModel>

<DataModel name="ChunkPlst" ref="Chunk">

  <String name="ID" value="plst" token="true"/>

  <Block name="Data">

    <Number name="NumberOfSegments" size="32" >

      <Relation type="count" of="ArrayOfSegments"/>

    </Number>

    <Block name="ArrayOfSegments" maxOccurs="3000">

      <String length="4" />

      <Number size="32" />

      <Number size="32" />

    </Block>

  </Block>

</DataModel>

<DataModel name="ChunkLabl" ref="Chunk">

  <String name="ID" value="labl" token="true"/>

  <Block name="Data">

    <Number size="32" />

    <String nullTerminated="true" />

  </Block>

</DataModel>

<DataModel name="ChunkNote" ref="ChunkLabl">

  <String name="ID" value="note" token="true"/>

</DataModel>

<DataModel name="ChunkLtxt" ref="Chunk">

  <String name="ID" value="ltxt" token="true"/>

  <Block name="Data">

    <Number size="32" />

    <Number size="32" />

    <Number size="32" />

    <Number size="16" />

    <Number size="16" />
```

```
<Number size="16" />

<Number size="16" />

<String nullTerminated="true" />

</Block>

</DataModel>

<DataModel name="ChunkList" ref="Chunk">

  <String name="ID" value="list" token="true"/>

  <Block name="Data">

    <String value="adt1" token="true" />

    <Choice maxOccurs="3000">

      <Block ref="ChunkLab1"/>

      <Block ref="ChunkNote"/>

      <Block ref="ChunkLtxt"/>

      <Block ref="Chunk"/>

    </Choice>

  </Block>

</DataModel>

<DataModel name="ChunkSmpl" ref="Chunk">

  <String name="ID" value="smpl" token="true"/>

  <Block name="Data">

    <Number size="32" />

    <Number size="32" />

    <Number size="32" />

    <Number size="32" />

    <Number size="32" />

    <Number size="32" />

    <Number size="32" />

    <Number size="32" />

    <Number size="32" />

    <Number size="32" />

    <Block maxOccurs="3000">

      <Number size="32" />

      <Number size="32" />
```



```
<Number size="32" />

<Number size="32" />

<Number size="32" />

<Number size="32" />

</Block>

</Block>

</DataModel>

<DataModel name="ChunkInst" ref="Chunk">

  <String name="ID" value="inst" token="true"/>

  <Block name="Data">

    <Number size="8"/>

    <Number size="8"/>

    <Number size="8"/>

    <Number size="8"/>

    <Number size="8"/>

    <Number size="8"/>

    <Number size="8"/>

    <Number size="8"/>

  </Block>

</DataModel>

<DataModel name="Wav">

  <!-- wave header -->

  <String value="RIFF" token="true" />

  <Number size="32" />

  <String value="WAVE" token="true"/>

</DataModel>

<!-- Defines the format of a WAV file -->

<DataModel name="Wav">

  <!-- wave header -->

  <String value="RIFF" token="true" />

  <Number size="32" />

  <String value="WAVE" token="true"/>

  <Choice maxOccurs="30000">
```

```
<Block ref="ChunkFmt" />

<Block ref="ChunkData" />

<Block ref="ChunkFact" />

<Block ref="ChunkSint" />

<Block ref="ChunkWavl" />

<Block ref="ChunkCue" />

<Block ref="ChunkPlst" />

<Block ref="ChunkLtxt" />

<Block ref="ChunkSmpl" />

<Block ref="ChunkInst" />

<Block ref="Chunk" />

</Choice>

</DataModel>

<StateModel name="TheState" initialState="Initial">

  <State name="Initial">

    <!-- 输出wav文件 -->

    <Action type="output">

      <DataModel ref="Wav" />

      <!-- 待读取的文件 -->

      <Data fileName="sample.wav" />

    </Action>

    <Action type="close" />

    <!-- 调用MPlayer -->

    <Action type="call" method="StartMPlayer" publisher="Peach.Agent" />

  </State>

</StateModel>

<Agent name="WinAgent">

  <Monitor class="WindowsDebugger">

    <!-- The command line to run. Notice the filename provided matched up
    to what is provided below in the Publisher configuration -->

    <Param name="CommandLine" value="c:\\mplayer\\mplayer.exe fuzzed.wav" />

    <!-- This parameter will cause the
```

```
the state model with a method="StartMPlayer" before running

program.

-->

<Param name="StartOnCall" value="StartMPlayer" />

<!-- This parameter will cause the monitor to terminate the process
once the CPU usage reaches zero.

-->

<Param name="CpuKill" value="true"/>

</Monitor>

<!-- Enable heap debugging on our process as well. -->

<Monitor class="PageHeap">

<Param name="Executable" value="c:\\mplayer\\mplayer.exe"/>

</Monitor>

</Agent>

<Test name="Default">

<Agent ref="WinAgent" platform="windows"/>

<Agent ref="LinAgent" platform="linux"/>

<Agent ref="OsxAgent" platform="osx"/>

<StateModel ref="TheState"/>

<Publisher class="File">

<Param name="FileName" value="fuzzed.wav"/>

</Publisher>

<Logger class="Filesystem">

<Param name="Path" value="logs" />

</Logger>

</Test>
```

0x7 后记

文件格式复杂多变，良好的开端是成功的一半。PeachPit的编写通常是一个细致而又复杂的过程，只有耐心+经验，才能成功写出一美的Pit文件。

***本文原创作者：overXsky，本文属FreeBuf原创奖励计划，未经许可禁止转载**


更多精彩

Peach 3

上一篇：[Geoip可视化攻击图谱：打造专属的炫酷风](#)

下一篇：[不试你可能会后悔，Windows渗透测试利器Pentest Box](#)

这些评论亮了



[okCryingFish](#) (1级)

照着官方文档抄都没抄对

回复

亮了(12)

已有 6 条评论

[qitian100](#) (2级)

2016-11-28

1楼

回

我之前粗略的研究过一下，感觉要对很多文件格式很熟悉啊

亮了 (1)

Sarshes

2016-11-28

1

@ qitian100 文件都是小事，网络协议才是大问题

亮了

河蟹

2016-11-29

2

@ Sarshes 针对远程协议的fuzz，如何monitor是关键

亮了 (1)

[BeyondHannn](#) (1级)

2016-11-29

2楼

回

厉害了我的天

亮了 (1)

[okCryingFish](#) (1级)

2018-07-24

3楼

回

照着官方文档抄都没抄对

亮了 (1)

deft

2018-09-27

4楼

回

嵌入式系统中的模糊测试怎样监视错误产生？

亮了 (1)

选择文件

未选择任何文件

欢迎 [TideSec](#) 再次光临! [退出 >>](#)

表情

插图

提交评论(Ctrl+Enter)

[取消](#)

☒ 有人回复时邮件通知我



overXsky

同在这一路上行走，只要走的比别人更久，就能够走出别人没有的距离。

1
文章数

4
评论数

最近文章

[深入探究文件Fuzz工具之Peach实战](#)

2016.11.28

浏览更多

相关阅读

[知名安全检测工具IBM Rational AppS...](#)

[工具推荐：汽车CAN总线分析框架CA...](#)

[安全扫描工具NetSparker 2.3 PRO...](#)

[ruby写的SQL注入工具-sqlcake 1.1 发布](#)

[DomLink：一款自动化的域发现工具](#)

推荐关注

FreeBuf+微信小程序

FreeBuf官方微信小程序，把安全装进口袋



扫码添加小程序

FreeBuf微信订阅号

国内领先的互联网安全新媒体，同时也是爱好者们交流与分享安全技术的社区

10月

[公开课双十一活动](#)

9月

上海

[CIS](#)


9月

上海

已结束 	已结束	已结束
扫码关注公众号		
FreeBuf企业安全服务号		
专注企业安全的精品内容分享平台		



Copyright © 2019 WWW.FREEBUF.COM All Rights Reserved [沪ICP备13033796号](#)

 阿里云 提供计算与安全服务

扫码关注公众号

FreeBuf新浪微博

国内领先的互联网安全新媒体
FreeBuf 官方微博，专属于爱好者们
交流、分享安全技术的社区



新浪微博

[PreviousNext](#)