Web应用代码自动化审计浅谈

lyle / 2019-04-06 13:24:00 / 浏览数 3708 安全技术 WEB安全 顶(0) 踩(0)

---

0. 序

代码审计是找到应用缺陷的过程。自动化审计通常有白盒、黑盒、灰盒等多种方式。白盒指通过对源代码的分析找到应用缺陷；黑盒通常不涉及到源代码，多使用模糊测试等

1. 基础概念

1.1 输入 (Source)

Web应用的输入，可以是请求的参数（GET、POST等）、上传的文件、Cookie、数据库数据等用户可控或者间接可控的地方。

例如PHP中的 `$_GET` / `$_POST` / `$_REQUEST` / `$_COOKIE` / `$_FILES` / `$_SERVER` 等，都可以作为应用的输入。

1.2 处理函数 (Filter)

处理函数是对数据进行过滤或者编解码的函数。这些函数会对输入造成影响，为漏洞利用带来不确定性。

同样以PHP为例，这样的函数可能是 `mysqli_real_escape_string` / `htmlspecialchars` / `base64_encode` / `str_rot13` 等，也可能是应用自定义的过滤函数。

1.3 危险函数 (Sink)

危险函数又常叫做Sink Call、漏洞点，是可能触发危险行为如文件操作、命令执行、数据库操作等行为的函数。

在PHP中，可能是 `include` / `system` / `echo` 等。

1.4 问题定义

一般认为一个漏洞的触发过程是从输入经过过滤到危险函数的过程(Source To Sink)，而自动化审计就是寻找这个链条的过程。自动化审计的难点主要在于以下几个方面。

1.4.1 输入多样化

对Web应用来说，可能的输入可以来自GET/POST的参数、Cookie、Url等多个地方，这些输入格式不固定，输入空间很大。

1.4.2 过滤函数复杂

在审计的过程中，从输入没有经过任何过滤函数就到达危险函数的过程较少。但是自动化判断一些过滤数是否可以绕过较为困难。

1.4.3 代码本身的复杂度

现代Web框架，代码复杂度极高，有相当多的动态加载和调用的过程，为自动化分析带来了困难。

2. 技术基础

2.1 抽象语法树

抽象语法树，顾名思义，是一种树形的数据结构。构造AST树的基本方法是将表达式的操作数作为树结构的叶子，将表达式的操作符号作为树结构的根，依次循环迭代进行构

例如在JavaScript中， `a=1` 的抽象语法树如下：

```
{
   "type": "Program",
   "body": [
   {
       "type": "ExpressionStatement",
       "expression":
       {
           "type": "AssignmentExpression",
           "operator": "=",
           "left":
           {
```

```
                "type": "Identifier",
                "name": "a"
            },
            "right":
            {
                "type": "Literal",
                "value": 1,
                "raw": "1"
            }
        }
    }],
    "sourceType": "script"
}
```

2.2 程序控制流图

AST树依旧是比较高层次的形式，其中模块之间的调用、循环等依旧不利于数据流的处理，因此引入了更底层的程序控制流图来进行分析。

程序控制流图(Control Flow
Graph，CFG)是静态分析过程中的另一种状态，可以反映程序代码控制流程。其实，程序控制流图是由一个入口、若干个出口组成的有向图，图中的每一个节点代表一个基本

3. 解决方案

3.1 危险函数匹配

白盒审计最常见的方式是通过查找危险函数来定位漏洞，比较有代表性的工具是Seay开发的审计工具。这个工具直接找出所有危险函数的位置，这种方式没有对调用流程进行

不过同样的，这种方式在一些环境下能做到几乎无漏报，只要审计者有耐心，可以发现应用大部分的漏洞，但是在高度框架化的代码中，这种方式能找到的漏洞相对有限。

3.2 代码相似性比对

一些开发者会复制其他框架的代码，或者使用各种框架。如果事先有建立对应的漏洞图谱，则可使用相似性方法来找到漏洞。

3.3 控制流分析

在2012年，Dahse
J等人设计了RIPS，该工具引入AST进行数据流与控制流分析，结合过程内与过程间的分析得到审计结果，相对危险函数匹配的方式来说误报率少了很多，但是同样的也增加了

3.4 基于图的分析

基于图的分析是对控制流分析的一个改进，其利用CFG的特性和图计算的算法，一定程度上简化了计算，比较有代表性的是微软的Semmle QL和NDSS
2017年发表的文章Efficient and Flexible Discovery of PHP Application Vulnerabilities。

3.5 灰盒分析

基于控制流的分析开销较大，于是有人提出了基于运行时的分析方式，对代码进行Hook，当执行到危险函数时自动回溯输入，找到输入并判断是否可用。

这种方式解决了控制流分析实现复杂、计算路径开销大的问题，在判断过滤函数上也有一定的突破，但是灰盒的方式并不一定会触发所有的漏洞。fate0大佬开发的prvd就是

4. 参考资料

- [1] RIPS https://github.com/ripsscanner/rips
- [2] prvd https://github.com/fate0/prvd
- [3] PHP运行时漏洞检测 http://blog.fatezero.org/2018/11/11/prvd/
- [4] Cobra https://github.com/FeeiCN/cobra
- [5] Semmle QL https://github.com/Semmle/ql
- [6] Vulnerability hunting with Semmle QL https://blogs.technet.microsoft.com/srd/2018/08/16/vulnerability-hunting-with-semmle-ql-part-1/
- [7] Backes M , Rieck K , Skoruppa M , et al. Efficient and Flexible Discovery of PHP Application Vulnerabilities[C]// IEEE European Symposium on Security & Privacy. IEEE, 2017.
- [8] Dahse J. RIPS-A static source code analyser for vulnerabilities in PHP scripts[J]. Retrieved: February, 2010, 28: 2012.
- [9] awesome static analysis https://github.com/mre/awesome-static-analysis

5. 结语

在学习自动化审计的过程中做了一点整理，于是形成了这篇文章，水平有限，这篇文章讲得也比较浅。之后有机会再就各个技术的细节做进一步分析，不当之处还请各位师傅

点击收藏 | 0 关注 | 2

上一篇：反-反汇编patch学习（一） 下一篇：从零开始java代码审计系列(三)

1. 0 条回复
   - 动动手指，沙发就是你的了！