雷火齐鸣 最燃公测 2020 06/10 闭幕技术沙龙 TECHNICAL SALON ON THE CLOSING CEREMONY

腾讯安全平台部 Tencent Security Platform Dpt.

纵深防御之WebShell攻防

演讲人

腾讯TEG安全平台部lake2 / 杜海章

目录 CONTENTS

雷火齐鸣 最燃公测 2020 06/10 闭幕技术沙龙 TECHNICAL SALON ON THE CLOSING CEREMONY

腾讯安全平台部 Tencent Security Platform Dpt.

01

洋葱简介

腾讯自研EDR – 洋葱

融合腾讯自身十几年生产网络攻防经验，**管、控、审于一体**

终端资产一体化管理

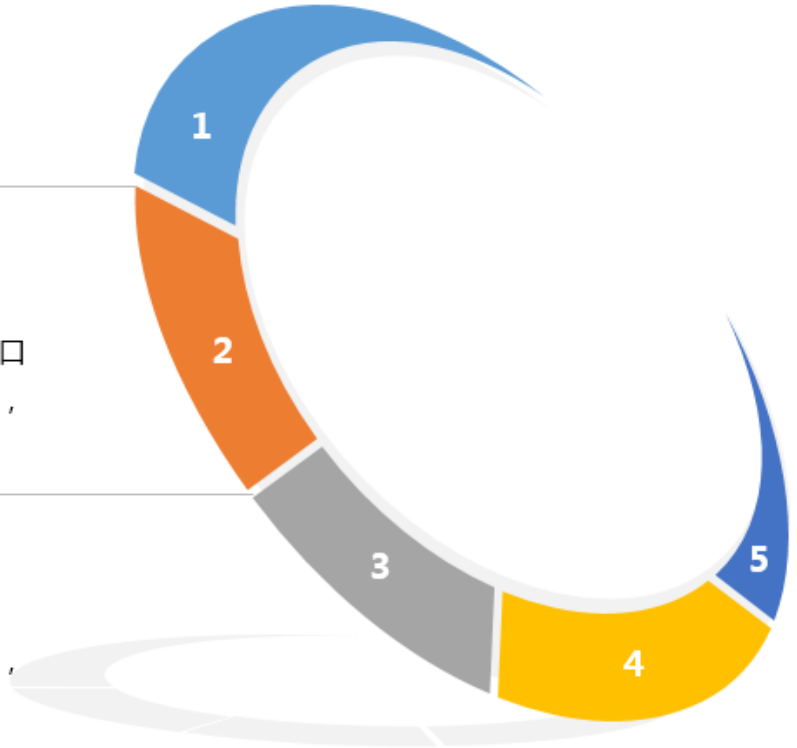
针对服务器安全检测能力，并实现了对全网服务器端点资产的全面管控。

端点安全的基线检测

依据腾讯自身业务防护实践，提供了端口配置、口令配置、账号配置的基线检测，可实现企业面临安全威胁的实时检测。

安全事件的精准溯源

通过对服务器端的登录信息、操作信息，建立行为分析模型，实现对异常操作进行异常操作溯源分析。



漏洞风险的实时预警

结合自身服务器防护经验，针对服务器中间件、数据库、操作系统提供漏洞检测能力，实时监测发现漏洞风险。

入侵攻击的主动检测

提供主动检测、实时发现入侵检测；提供Web木马、反弹木马、异常网络通、提权攻击、弱口令等检测功能。

02

洋葱WebShell检测方案回顾

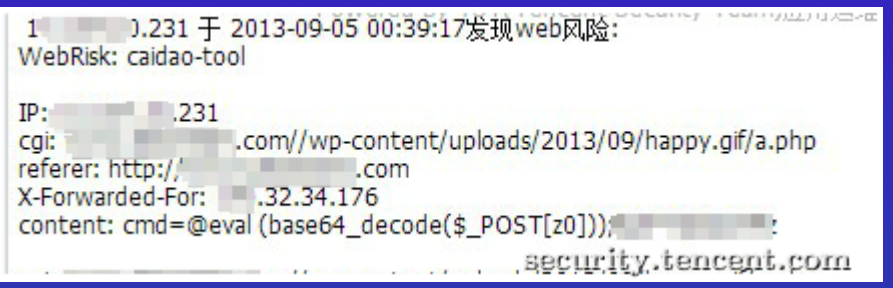
上线第一代 WebShell检测引擎

上线流量 分析检测

上线RASP动 态检测并众测

统计分析、 机器学习等

2008-2011 → 2012 → 2013 → 2014 → 2015/2016/2017



特征检测

- 维护成本高、泛化能力弱

统计分析

- 只针对混淆变形类

机器学习

- 建设需大量样本

动态检测

- 侵入性强，部署困难

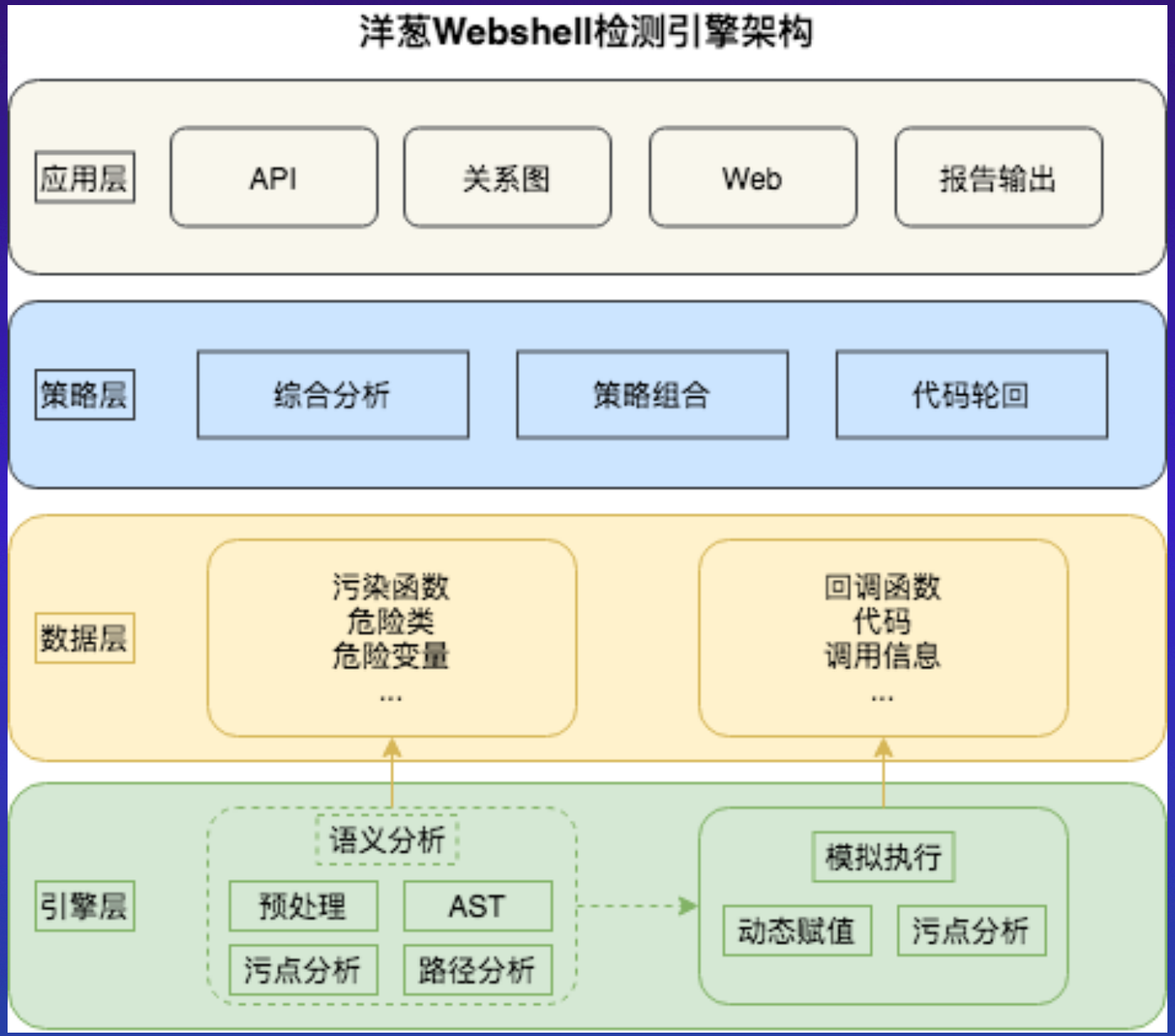
流量检测

- 成本高，漏误报比较多

03

洋葱新一代语义动态检测引擎

引擎整体框架


洋葱Webshell检测引擎架构
应用层
API
关系图
Web
报告输出
策略层
综合分析
策略组合
代码轮回
数据层
污染函数
危险类
危险变量
...
回调函数
代码
调用信息
...
引擎层
语义分析
预处理
AST
污点分析
路径分析
模拟执行
动态赋值
污点分析

污点分析抽象成一个三元组<sources,sinks,sanitizers>的形式。
语义污点追踪是从sinks触发，判断sinks中的变量或函数是否存在外界可控。
动态污点分析是从sources触发，判断这些污染源是否下沉至sinks点

sources：污染源

- 表示所有用户可控的输入点

sinks：危险汇聚点

- 代表所有可能产生威胁的函数或操作

sanitizers：无害处理

- 代表进行安全操作或过滤，使得样本不在具有安全风险

危险汇聚点

- 寻找万恶之源-sinks(代码执行、文件操作、动态操作、命令执行、外部组件、回调等类型)

污点传播

- 对威胁节点进行回溯追踪(变量赋值、引用传值、函数调用、函数赋值、间接传值等)

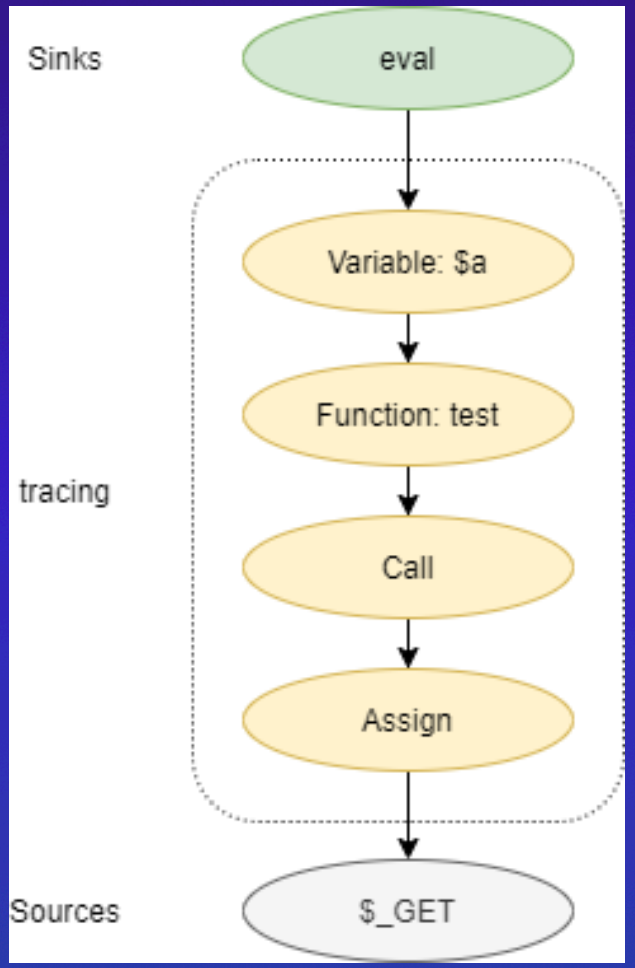
污染源

- 判断数据流参数是否外界可控(常见的污染源有预定义变量、可控函数、变量覆盖函数等)

WebShell示例1

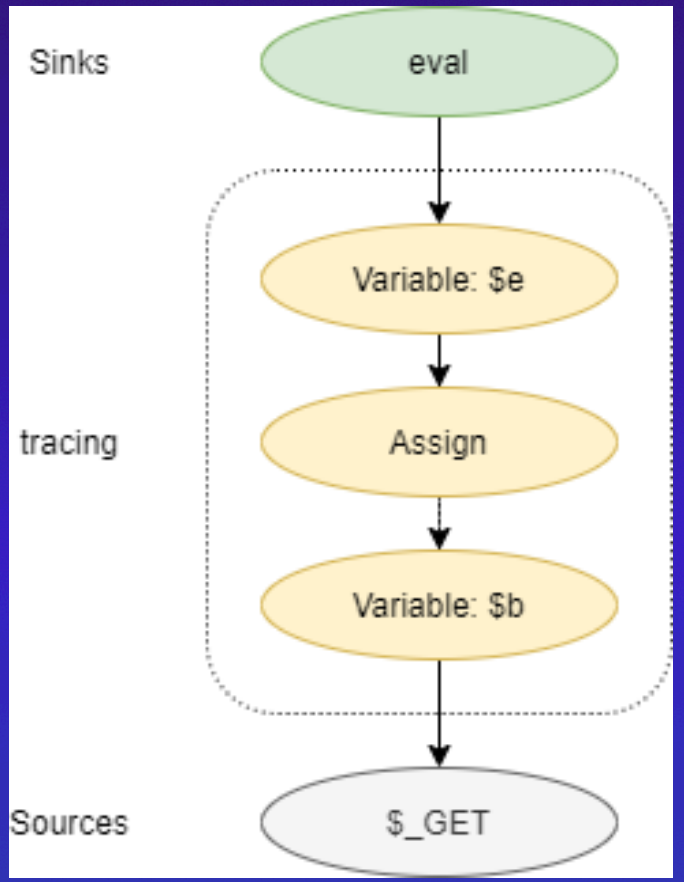
```
<?php
function test($a){
    eval($a);
}
$a = $_GET["c"];
test($a);
```

上述代码是一种函数调用型webshell，语义追踪的过程为：
首先发现eval危险函数，之后追踪eval参数$a，发现是在函数test中，然后在跟踪test函数的调用信息，最后锁定调用参数是否可控，其数据流模型图为：



WebShell示例2

```
<?php
foreach ($_GET as $b){
    if ($b == "c"){
        $e = 'e';
    }
    if ($b == "m"){
        $e .= 'val($_GET[1]);';
    }
    // ...
}
eval($e);
```



上述代码是一种通过外界输入间接控制变量进而构造出webshell文件。静态分析的过程为：

追踪$e变量，发现在赋值节点中会被$b影响，并且$b变量受到外界控制，则判断为webshell文件。在这种情况下，洋葱检测引擎会提取所有与变量相关联的节点进行回溯，确保关联节点也不会受到外界影响。

动态模拟执行

参数赋值

- 寻找全局外部变量，如GET、POST、COOKIE等
- 通过符号计算推导出符合程序条件运行的虚拟值

污点标记

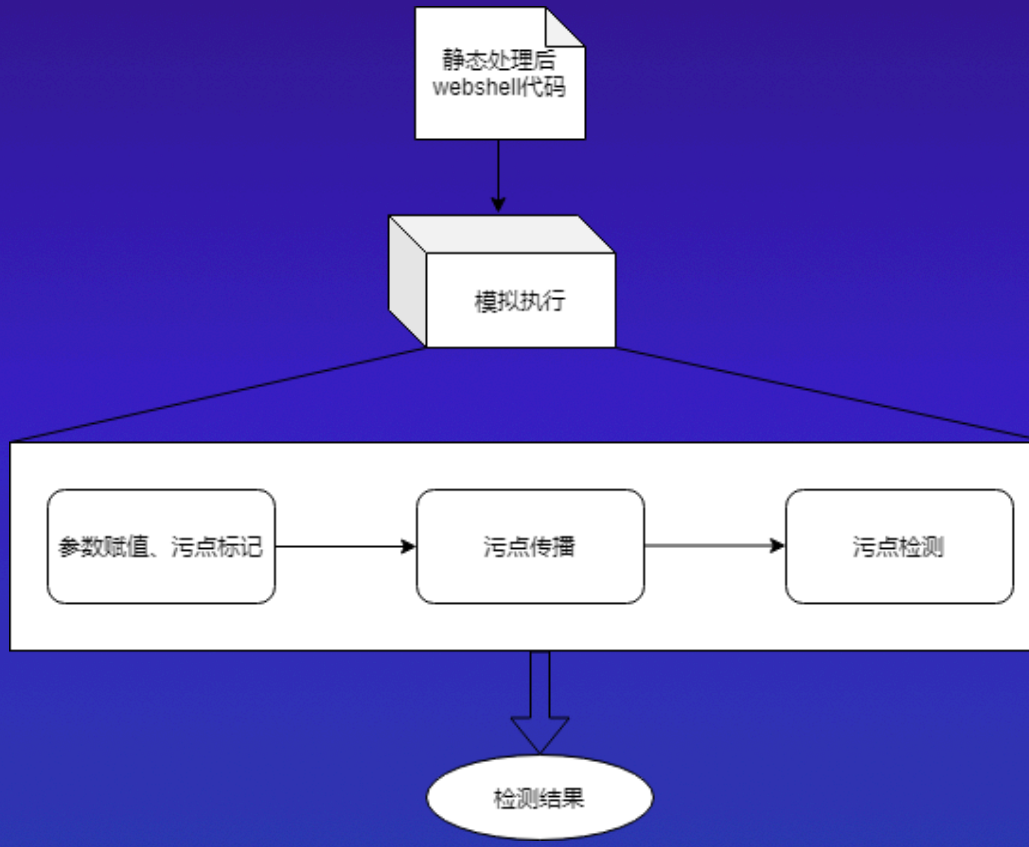
- 利用php变量的特殊标记和预留标志位，将标志位打点实现标记

污点传播

- 对字符串处理函数、加密函数和转换函数等进行处理

污点检测

- 核心的检测逻辑便是敏感函数+可控参数。



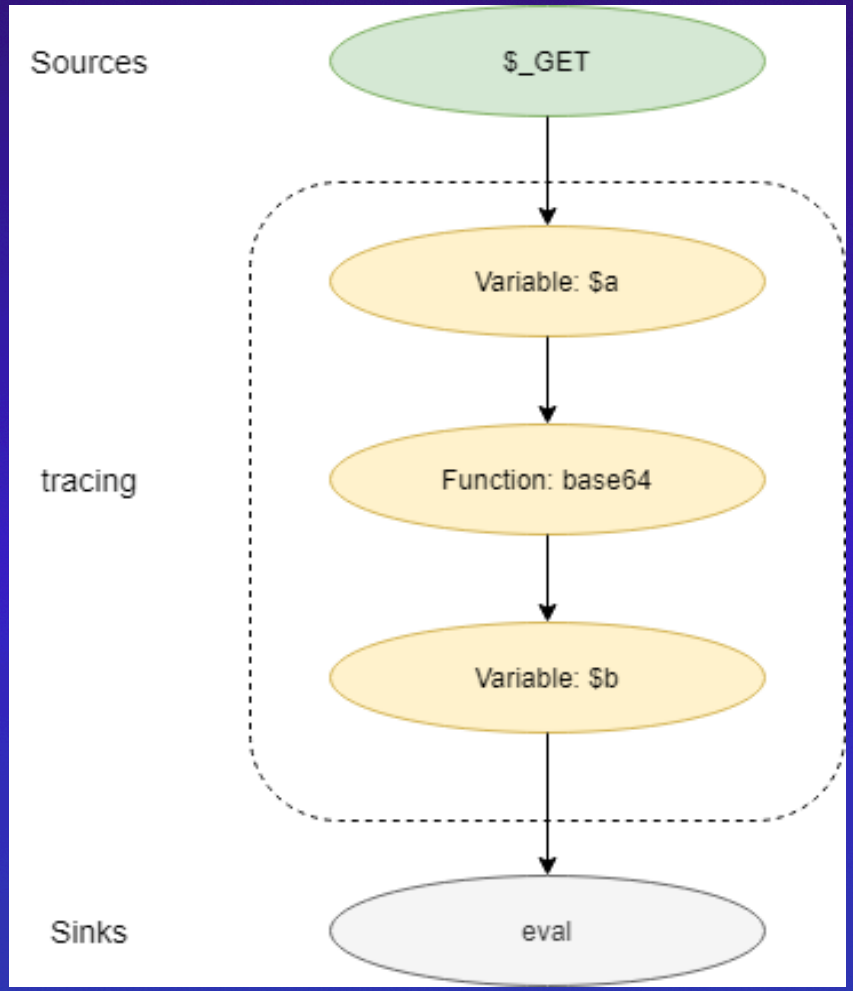
WebShell示例3

```
<?php
    $a = $_GET['code'];
    $b = base64_decode($a);
    eval($b);
?>
```

第一行：寻找到$a来源外部变量，对外部变量$_GET['code'] 进行参数赋值且给$_GET['code'] 打上标记，然后赋值给$a。（参数赋值、污点标记）

第二行：将$a用base64解密后赋值给$b，由于base64_decode函数hook后会检测参数是否存在污点，有则将返回值打上标记，故$b带有污点标记。（污点传播）

第三行：敏感函数（eval）执行且参数$b带有污点，故判定为恶意文件。（污点检测）



一、语义污点分析

静态的符号执行只能分析出代码所能呈现的逻辑结构，并不知道具体变量的内容。PHP复杂多变的动态特性是影响静态分析的关键性因素。

如：

（1）动态调用检测中，单纯的语义只能分析出存在动态调用函数，并不清楚具体是哪个函数调用，但直接告警很明显是不合理的，因此必须要动态的模拟执行配合才能确切的判断是否为恶意文件。

（2）检测eval等代码执行时，语义解析由于不清楚变量中的内容导致无法解析具体执行的代码内容，这一部分也是需要动态配合进行检测。

二、动态模拟执行

虽然动态检测在对抗静态混淆绕过方面具备相对优秀的检测能力，但在实现动态检测的过程中，还是遇到许多困难点，相对于传统正则引擎，这种方式也多了一些比较另类的绕过可能，主要有以下两点：

（1）打断污点传播，使得经过Sink点的参数不具备污点，绕过检测。

（2）改变程序执行流，使得程序流在不传入特定值时不经过Sink点，从而绕过检测。

静态语义分析可以保证动态污点能够准确的流向危险源
动态模拟执行能提供函数调用链用于静态语义分析

（1）动静结合可以有效的解决代码中的分支问题，抽象语法树解析可以正确的识别分支逻辑，并把可能存在风险的分支内容扔给动态去处理。
（2）报错终止问题，静态可先将代码封装为try-catch的方式，将报错代码忽略，而后交给动态，保证程序的继续进行。
（3）变量覆盖问题，则可通过hook影响变量定义或赋值的函数来覆盖。
（4）同时引擎也会提取动静态的检测结果进行整合分析。
如：静态发现一个危险函数test，但是没有找到调用信息，而动态走到了这个危险函数中，通过两个检测结果的对比与结合，便可判断其为一个Webshell文件

```
<?php
function test(){
    /*
     * eval code
     * */
}
$a = 't'.'e'.'s'.'t';
$a();
```

腾讯安全平台部

BLADE Tencent Blade

腾讯宙斯盾 DDoS防护系统

門神

腾讯蓝军 Tencent Force

TSRC 腾讯安全应急响应中心

洞犀 INSIGHT SCANNER

Onion EDR 洋葱反入侵系统

腾讯铁将军 专业·安全·便捷

金刚系统 KING KONG

Thanks