基于指令集随机化的抗代码注入攻击方法

马博林[1], 张 铮[1], 陈 源[2], 邬江兴[1]

[1]中国人民解放军战略支援部队信息工程大学 郑州 中国 450001
[2]江南计算技术研究所 无锡 中国 214083

摘要 代码注入攻击是应用程序面临的一种主要安全威胁, 尤其是 Web 应用程序, 该种攻击源于攻击者能够利用应用程序存在的漏洞/后门, 向服务器端注入恶意程序并执行, 或者利用应用程序对用户输入的参数缺乏验证和过滤, 造成输入作为恶意程序执行, 从而达到攻击目的。源程序分析和输入规则匹配等现有防御方法在面对代码注入攻击时都存在着固有缺陷, 为了提高 Web 应用程序对于代码注入攻击的防御性, 提出一种基于指令集随机化的抗代码注入方法, 该防御方法不依赖于攻击者采用何种攻击方式, 能够抵御未知的代码注入攻击。基于该技术及动态、冗余构造方法, 设计一套原型系统, 采用广义随机 Petri 网(Generalized Stochastic Petri Net, GSPN)建模计算, 攻击者即使在获得随机化方法先验知识的情况下也极难突破系统的防御机制。尽管该方法需要对应用程序源代码进行随机化变换, 但处理过程是完全自动化和具有普适性的, 通过实验和现网测试表明该方法能够有效抵御大部分代码注入攻击, 实现了对攻击的主动防御。

关键词 指令集随机化; 代码注入攻击; 广义随机 Petri 网; 主动防御
中图法分类号 TP309.1 DOI 号 10.19363/J.cnki.cn10-1380/tn.2020.07.03

The Defense Method for Code-Injection Attacks Based on Instruction Set Randomization

MA Bolin[1], ZHANG Zheng[1], CHEN Yuan[2], WU Jiangxing[1]

[1] PLA Information Engineering University, Zhengzhou 450001, China
[2] Jiangnan Institute of Computing Technology, Wuxi 214083, China

Abstract Code-injection attack is one of common security threat types faced by Web applications. This attack occurs when an attacker injects malicious programs into the server by exploiting vulnerabilities and backdoors of applications, or makes use of that applications accept user input parameters without validating and filtering to make malicious input parameters executed. In addition, existing defense methods, such as application analysis and inputs analysis, have inherent defects in the face of code-injection attack. In order to counteract the trend, we present a defense method for Web application code-injection attacks based on instruction set randomization. This method does not depend on the attack methods adopted by the attacker, so it can resist unknown code-injection attacks. Based on this method, we design a prototype system, whose security is calculated by the generalized stochastic Petri net (GSPN). Even if the attacker obtains the prior knowledge of the randomized method, it is difficult to break the system. Although modifying source code of Web applications is needed, this process is fully automated and universal. Experiments show that this method can effectively resist code-injection attacks, and achieve the proactive defense.

Key words instruction set randomization; code-injection attack; generalized stochastic Petri net; active defense

1 引言

Web 应用程序设计开发技术的多样化发展, 给现代互联网用户带来丰富的网络服务同时, 也使得如何保证 Web 应用程序的安全性成为了网络空间安全领域中的焦点。代码注入攻击是 Web 应用程序面临的典型安全威胁, 攻击者利用 Web 服务各层面的漏洞或被恶意植入的后门, 向服务器端注入恶意代码并触发执行, 从而进行对特定目标的攻击。目前, 国内外学者针对代码注入攻击已经提出了广泛的解决方法, 但是其中大多数都是通过已掌握的安全策略、攻击行为、漏洞利用方法等先验知识进行被动防御, 攻击者仍能采取相应的方法进行针对性绕过, 并且防御者一方具有严重的滞后性, 因此攻击者与

通讯作者: 张铮, 博士, 副教授, Email: ponyzhang@126.com。
本课题得到国家重点研发计划(No.2018YFB0804003)资助。
收稿日期: 2020-03-01; 修改日期: 2020-06-02; 定稿日期: 2020-06-19

防御者形成了不对等的攻防态势。

本文提出一种针对 Web 应用程序的抗代码注入攻击防御方法, 该方法不对攻击者的请求、输入做检测, 不依赖于攻击者采用的攻击方式, 而是基于指令集随机化的思想, 对 Web 应用程序运行时的处理过程进行随机化变换, 使得攻击者由外部注入的恶意代码与变换后的运行时环境不匹配, 阻止注入代码执行。本文以 PHP 程序为代表实现了该方法, 并结合动态、冗余机制设计出原型系统 PHPRAND, 采用 GSPN 建模计算出系统的安全稳态概率, 即便攻击者在获得随机化方法先验知识的情况下也极难突破系统的防御机制。实验和现网测试表明该方法能够有效抵御大部分代码注入攻击, 实现对攻击的主动防御。

本文剩余部分的组织如下: 第 2 节简要介绍 Web 应用程序代码注入攻击及防御, 总结出当前防御方法的固有缺陷; 第 3 节以 PHP 程序为代表论述基于指令集随机化的抗代码注入方法的具体设计和实现; 第 4 节实现 PHPRAND 并采用 GSPN 建模计算出系统的安全稳态概率; 第 5 节进行实验和现网测试; 第 6 节探讨 PHPRAND 的局限性和改进思路; 第 7 节进行总结。

2 Web 应用程序代码注入攻击及防御

2.1 代码注入攻击

Web 应用程序代码注入攻击源于攻击者能够利用应用程序存在的漏洞/后门, 向服务器端注入恶意程序并执行, 或者利用应用程序对用户输入的参数缺乏验证和过滤, 造成输入作为恶意程序执行, 从而达到攻击目的。

图 2 给出了典型的 Web 应用架构, 其中客户端通过 HTTP 协议向服务器端发送请求, 如步骤①所示; 服务器端中的服务器软件, 例如 Apache、Nginx、IIS 等, 解析 URL 中的资源名和参数, 从文件系统中获取目标 PHP 程序, 如步骤②所示; PHP 程序通过预定义的变量接收请求中的参数, 并由 PHP 解释器完成执行, 如步骤③所示; PHP 程序若执行 include( )、require( )等具有 PHP 程序调用功能的函数, 重复步骤③; PHP 程序若执行 mysqli_query( )、oci_execute( )等具有数据库操作功能的函数, 则进行步骤④, 数据库完成操作后将结果返回至 PHP 解释器, 如步骤⑤所示; PHP 程序若执行 exec( )、system( )等具有系统命令执行功能的函数, 则进行步骤⑥, 操作系统完成调用后将结果返回至 PHP 解释器, 如步骤⑦所示; 最终服务器端通过 HTTP 协议将结果返回至客户端, 如步骤⑧所示。

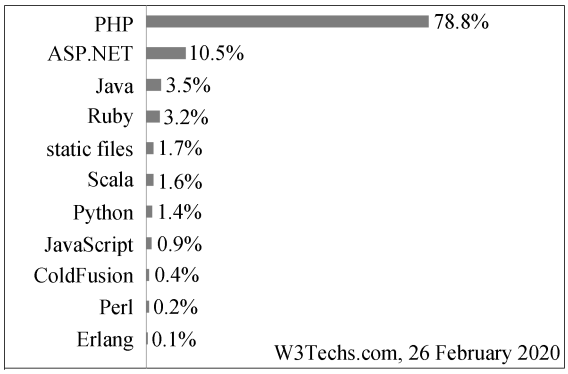


图 1 Alexa 排名前 100 万网站类型统计

Figure 1 Type Statistics by Alexa Top 1 Million Website

图 1 给出了 W^{3}Techs 根据 Alexa 排名前 100 万网站的类型统计，PHP 网站数量占比达到 78.8%，排名首位，PHP 语言已然成为目前全球范围内应用最广泛的服务器端编程语言。因此，接下来将以 PHP 应用程序为代表介绍代码注入攻击的过程和特点。

攻击者向服务器端注入恶意程序并执行, 或者输入恶意的数据作为程序执行, 从而实现注入攻击目的。针对 Web 应用程序的注入型攻击, 代码注入攻击通常发生在步骤②; 命令注入攻击通常发生在步骤④; 而 SQL 注入攻击通常发生在步骤⑥, 其中相较于命令注入攻击和 SQL 注入攻击而言, 代码注入攻击不需要获得相关操作系统、数据库的先验知识, 攻击方式较为多样, 攻击范围更为广泛。表 1 给出了 PHP 应用程序中代码注入攻击普遍利用的函数和关键字, 这些函数和关键字具备代码执行或者接收函数回调的功能。

2.2 主要防御方法

目前, 代码注入攻击的安全防护方法主要源于程序分析和输入规则匹配两种技术思路[1], 通常需要对利用方式、攻击行为进行分析, 并采取对应的防护措施, 但在新型的、未知的攻击行为面前都存在着固有缺陷。

程序分析包括静态源程序分析方法和动态源程序分析方法, 静态源程序分析方法能够筛选出源程序代码中的危险函数和关键字, 并对其使用方法进行检查, 代表性的 PHP 源程序静态分析工具有 phpcs-security-audit、RIPS、RATS 等; 动态源程序分析方法能够对程序源代码中的变量进行特征分析,

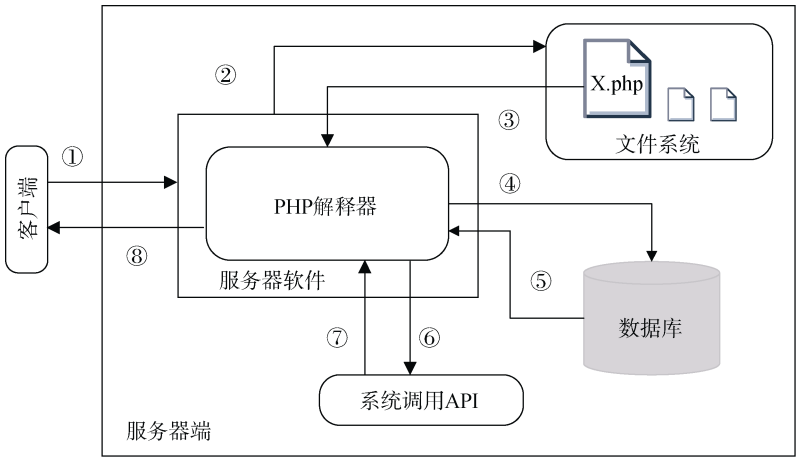


图 2　典型 Web 应用架构

Figure 2　Typical Web application architecture

表 1　PHP 代码注入攻击常利用的函数和关键字

Table 1　Functions and keywords commonly used in PHP code-injection attacks

具有代码执行功能的函数或关键字			
函数或关键字	注入位置	函数或关键字	注入位置
eval	0	assert	0
preg_replace('/.*/e', ...)	1	create_function	1
include	0	include_once	0
require	0	require_once	0
具有函数回调功能的函数或关键字			
函数或关键字	回调位置	函数或关键字	回调位置
ob_start	0	array_diff_uassoc	–1
array_diff_ukey	–1	array_filter	1
array_intersect_uassoc	–1	array_intersect_ukey	–1
array_map	0	array_reduce	1
array_udiff_assoc	–1	set_exception_handler	0
array_udiff	–1	array_uintersect_assoc	–1
sqlite_create_function	2	array_uintersect	–1
array_walk_recursive	1	array_walk	1
assert_options	1	uasort	1
uksort	1	usort	1
preg_replace_callback	1	spl_autoload_register	0
iterator_apply	1	call_user_func	0
call_user_func_array	0	register_shutdown_function	0
register_tick_function	0	set_error_handler	0

确定不同变量对于应用程序的作用, 避免不规范的变量设置造成的注入攻击, 更复杂地能够构造参数集合进行变量赋值, 并跟踪执行路径来检查应用程序在逻辑层面的注入漏洞, 例如动态符号执行、动态污点分析等技术方法。

程序分析方法对于代码注入攻击能够起到一定的防御作用, 但是随着应用程序越来越复杂, 该方法将面临着以下问题:

(1) 静态源程序分析结果无法阻止攻击发生, 若禁用危险的函数和关键字, 将会影响程序的部分功能。

(2) 动态源程序分析过程中执行路径爆炸使得该方法无法检查全部的数据流路径。

(3) 执行路径的自动发掘技术能够解决路径爆炸的问题, 但如何提高覆盖率, 检测到包含必要路径的数据流也是需要解决的。

输入规则匹配是根据安全研究人员的先验知识设置精确的规则, 持续地将用户请求与规则进行匹配, 当检测到不符合要求的用户输入时, 将丢弃整个会话或者采用类似特殊字符转义的消除机制修改用户输入, 从而抵御不正确的用户输入带来的代码注入威胁[2], 使用较为广泛的 Web 应用防火墙(Web Application Firewall, WAF)就是采用该方法设计而成。目前安全研究人员通过机器学习技术增强规则集和扩展规则定义来提高该方法的检测精度[3-5], 许多先进的机器学习解决方案已经应用到 WAF 中。尽管不断融合新型技术, 但是该方法仍存在以下固有缺陷:

(1) 规则的制定需要持续关注新型攻击, 在面对 0-DAY 攻击时, 该方法无法通过现有的规则检测到未知攻击。

(2) 在白盒环境或者攻击者已经掌握规则的情况下, 攻击者可以通过混淆攻击的方式突破精确规则, 绕过防御措施。

(3) 机器学习方法的先进性取决于训练的数量, 并且实际研究表明该方法检测率远低于 100%的理想目标。

(4) 虽然通过规则集提高了攻击行为的检测精度, 但是该方法的防御能力依然受限于方法本身的滞后反应。

3 基于指令集随机化的代码注入防御

3.1 指令集随机化思想

软件程序面临的安全威胁中, 攻击者常利用程序漏洞进行缓冲区溢出、格式化字符串等方式的攻击[6-7], 使得程序的执行流程跳转到已经注入的恶意代码位置, 执行恶意代码, 达到攻击目的。在以上攻击场景中, 注入的恶意代码指令需要与目标平台的指令集一致才可被成功执行。这种情况下, 指令集随机化(Instruction Set Randomization, ISR)通过变换被保护系统指令集的方法破坏攻击者所掌握的有关目标系统指令集信息的先验知识, 使得注入的恶意代码指令与目标系统的指令集不一致, 恶意代码无法被执行, 在不需要了解攻击者采用何种攻击方式的情况下有效抵御了代码注入攻击[8-10], 并且一定程度上能够抵御未知的代码注入攻击。

指令集随机化思想被提出后, Barrantes 等人[11]基于 Valgrind 虚拟化环境首次设计实现了 RISE 指令集随机化系统, 该系统基于伪随机密钥序列对代码指令片段的每个字节进行随机化。相继大量的学者投入到了指令集随机化研究领域中, 由于指令集随机化是针对攻击者由外部注入恶意代码的防御方法, 因此新型的代码复用攻击[12-15]可以绕过传统的随机化机制, 例如 Return-into-lib 攻击、Return-Oriented Programming(ROP)攻击等。

面对新型攻击, 指令集随机化技术也在攻防博弈间不断丰富发展, Sinha 等人基于 SPARC32-Leon3 FPGA 设计了新型的指令集随机化系统 Polyglot[16], 能够降低性能负担的同时有效抵御(JIT-)ROP 攻击。Guanciale 等人[17]通过设计加密后代码指令的随机访问方法, 解决代码复用攻击的绕过问题。Werner 等人[18]基于指令集随机化提出了具有缓存集随机化的缓存架构 SCATTERCACHE, 该架构改进了缓存关联方法, 消除了完全重叠的缓存集, 能够有效抵御基于驱逐的缓存攻击。

同时指令集随机化方法还可以抵御针对脚本语言或解释语言的代码注入攻击, 美国加利福尼亚大学 Angelos 教授团队提出了 Perl 语言和 shell 脚本的随机化方法[19], 能够抵御代码注入攻击。该团队基于 SQL 语句的随机化设计了 SQLrand[20]系统, 能够有效抵御 SQL 注入攻击。以上 Angelos 教授团队研究成果的防御有效性需要满足的条件是: 攻击者处于完全不了解随机化方法的黑盒环境。攻击者一旦掌握了以上相关系统的随机化方法, 调整注入代码, 便能实现有效的攻击。

本文就是以指令集随机化为基础, 创新地结合动态、冗余构造方法, 以 PHP 语言为代表设计实现了 PHPRAND 原型系统, 使得攻击者即使在获得随机化方法先验知识的情况下也极难突破系统的防御机制。

3.2 Web 服务形式化建模

Web 网站服务形式化建模[21], 可以表示为从网络请求和请求参数到代码解释器、程序源文件、响应状态码以及响应结果的映射:

$$request \times parameter \rightarrow interpreter \times file \times status \times response \tag{1}$$

映射关系式(1)左侧的 *request* 是用户通过 HTTP 协议向服务器端发出的网络请求; *parameter* 是指用户在发起网络请求时向服务器端传递或提交的参数数据。映射关系式(1)右侧的 *interpreter* 表示服务器端的代码解释器, 即为图 2 中的 PHP 解释器, 具有词法分析和语法分析两个关键功能; *file* 为用户请求的应用程序, 即为图 2 中服务器端文件系统中的 PHP 程序源文件; *status* 是指服务器端完成用户请求后返回的状态码, 当服务器端成功处理了请求后返回状态码 200, 若程序不符合代码解释器的解析规则, 产生内部执行错误则返回状态码 500; 最后 *response* 是指服务器端完成用户请求后返回的响应体内容。

本文提出的基于指令集随机化的防御思想是修改服务器端代码解释器的词法或语法, 并对程序源代码进行一致性的随机化变换。攻击者不了解当前服务器端采用何种随机化变换规则, 因此由外部注入的恶意代码与可执行代码不一致, 无法成功执行。PHPRAND 中将表 1 列出的 PHP 关键字和高危函数进行随机化处理, 在 PHP 源码文件中关键字和函数的起始端(或者末端)添加由字母和数字混合组成的多位字符串标签, 图 3 给出了随机化变换前后的 PHP 示例代码。该防御方法的原理如公式(2), 其中 $f()$为服务器端执行程序代码的函数, *interpreter* 和 *file* 分别为公式(1)中代码解释器和应用程序, *status* 为公式(1)中响应状态码。

$$f(interpreter, file) = status \tag{2}$$

不考虑攻击者采用何种方式进行代码注入, 恶意代码的存在形式主要表现为以下两种情景:

情景 1: 攻击者直接向服务器端的文件系统中注入完整独立的恶意程序文件, 例如常见的一句话木

马、反弹式木马等。该情景的服务器端中应用程序可表示为:

```
<?php
    echo("Hello World");
?>
```

(a) 随机化变换前的PHP示例代码
(a) PHP sample code before randomizing transformation

```
<?php
    echo1234("Hello World");
?>
```

(b) 随机化变换后的PHP示例代码
(b) PHP sample code after randomizing transformation

图 3　随机化变换前后的 PHP 示例代码
Figure 3　PHP sample code before and after randomizing transformation

$$file \in OriginalFiles \cup InjectionFiles \tag{3}$$

其中 *OriginalFiles* 为网站原始程序或通过合法途径更新的程序经过随机化变换后的程序集合, 并且随机化变换规则与代码解释器一致, *InjectionFiles* 为注入的恶意程序集合。

根据指令集随机化的防御思想, *InjectionFiles* 集合中的程序未经随机化变换, 不符合代码解释器的随机化规则, 因此攻击者直接请求触发恶意程序时, 程序执行失败, 防御过程可以表示为:

$$\begin{cases} f(interpreter, file) = 500 \\ file \in InjectionFiles \end{cases} \tag{4}$$

情景 2: 开发人员在设计开发网站时, 受自身能力和水平影响, 网站存在程序功能设计不规范, 访问权限设计不合理, 参数检查设计不完善等问题, 攻击者可以利用应用程序的漏洞或预留的后门接口注入恶意代码。该情景的服务器端中应用程序可表示为:

$$\begin{gathered} file \in OriginalFiles \\ AcceptFiles \subseteq OriginalFiles \end{gathered} \tag{5}$$

其中 *OriginalFiles* 为网站原始程序或通过合法途径更新的程序经过随机化变换后的程序集合, 并且随机化变换规则与代码解释器一致, *AcceptFiles* 为 *OriginalFiles* 集合中能够接受攻击者参数 *parameter*, 触发代码解释器去执行注入参数的程序子集。

根据指令集随机化的防御思想, 恶意参数 *parameter* 作为注入的代码未经随机化变换, 不符合代码解释器的随机化规则, 因此攻击者无法通过构造恶意参数成功执行恶意代码, 程序漏洞利用失败, 防御过程可以表示为:

$$\begin{cases} f(interpreter, file) = 500 \\ file \in AcceptFiles \end{cases} \tag{6}$$

通过以上形式化建模分析, 攻击者无论在情景 1 还是情景 2, 只要向 Web 应用程序中注入表 1 中的关键字或函数, 并且不符合代码解释器的随机化规则, 注入代码便无法执行, 攻击失败。实际应用中, 服务商或安全厂商可根据真实需求, 删减或扩展表 1 的内容, 调整关键字和函数随机化变换的范围, 在安全性要求极高的情况下, 可将程序使用的所有函数和关键字进行随机化变换。

3.3　PHP 程序随机化方法

图 4 给出了 PHP 程序在代码解释器中 Zend 虚拟机的执行流程, 程序执行开始的入口函数为 compile_file(), 该函数将 PHP 程序打开; 然后, 调用 zendparse()进行词法/语法分析, zendparse()函数中不断地调用 zendlex()对程序进行切割; 匹配语法, 生成抽象语法树 AST; 再由 zend_compile_top_stmt()将 AST 生成 opcode, zend_emit_op()将生成的 opcode 加入到 opcode 列表中; pass_two()设置 opcode 类型及 handler 处理器, 当解释器调用 opcode 时, zend_execute()会执行与其对应的处理函数, 例如常见的 echo 语句对应的 opcode 为 ZEND_ECHO。

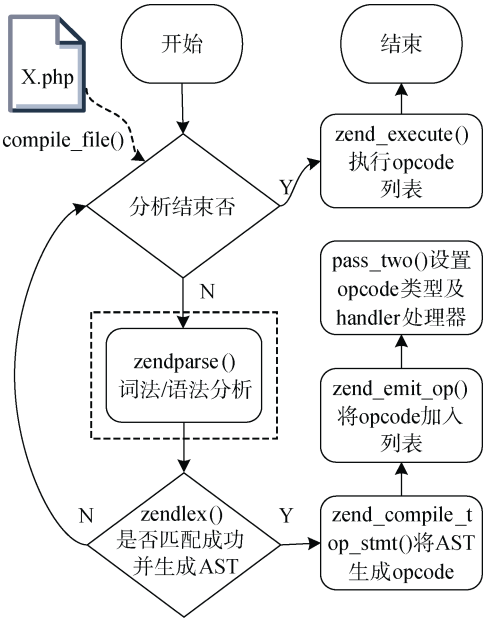


图 4　Zend 虚拟机中代码执行过程
Figure 4　Code execution process in Zend virtual machine

实现本文的随机化方法主要是修改图 4 中虚线标出的词法/语法分析阶段 zendparse(), 调整规则使其生成 AST, 保证解释器的虚拟机 Zend 能够执行随机化后的 PHP 程序。

针对 PHP 程序源代码, 基于 Zend 内核函数 zend_highlight()设计随机化处理方法。PHP 解释器的命令./php -s 就是通过调用 zend_highlight()对程序代码进行高亮显示, 修改该函数, 在 switch 选择判断中增加欲进行随机化变换 token 的 case 条件, 若匹配则将 next_color 设置为 highlight_keyword, 并调整 switch 选择判断中默认 default 条件的处理方法, 若无匹配的 case 条件, 则将 next_color 设置为 highlight_default, 调整高亮显示判断, 当 next_color 为 highlight_keyword 在目标关键字或函数后插入随机化标签。以 echo 随机化变换为目标, 修改后的 zend_highlight()部分源代码如图 5 所示, 分别执行随机化前后的./php -s test.php 命令, test.php 的源代码为图 3(a)给出的实例代码, 输出结果如图 6(a)和 6(b)所示。

```
ZEND_API void zend_highlight(... ...)
{
    zval token;
    int token_type;
    char *last_color = highlight_html;
    char *next_color;
    ... ...

    while ((token_type=lex_scan(&token, NULL))) {
        switch (token_type) {
            ... ...

            case T_ECHO:
                next_color = highlight_keyword;
            ... ...

            default:
                next_color = highlight_default;
            ... ...

        }
    }

        if (next_color != highlight_keyword) {
            ... ...
            zend_printf("1234");
            ... ...
        }

        zend_html_puts();
        ... ...

}
```

图 5 修改后的 zend_highlight()部分源代码

Figure 5 Part of the modified zend_highlight () code

```
[root@localhost bin]# ./php -s ./test.php
<code><span style="color: #000000">
<span style="color: #0000BB">&lt;?php<br />echo</span><span s
tyle="color: #007700">(</span><span style="color: #DD0000">"H
ello  World"</span><span style="color: #007700">);<
br /></span><span style="color: #0000BB">?&gt;<br /></span>
</span>
```

(a) 随机化前执行结果

(a) Execution result before randomization

```
[root@localhost bin]# ./php -s ./test.php
<?php
echo1234("Hello  World");
?>
```

(b) 随机化后执行结果

(b) Execution result after randomization

图 6 随机化前后./php -s test.php 执行结果

Figure 6 Execution results of ./php -s test.php before and after randomization

随机化标签若由 8 位数字和字母(大小写)构成, 在采用 hashcat[22]工具暴力破解, GPU 速度可达到 9000M/s 的情况下, 按每秒可以尝试 900 万种组合来计算, 攻击者黑盒静态环境下破解该标签的时间将超过 280 天。

4 PHPRAND 系统设计与安全性量化

4.1 PHPRAND 系统设计

攻击者在黑盒静态环境下破解随机化方法需要足够的计算能力和时间支撑, 但是一旦通过社会工程学等手段掌握了随机化方法, 调整注入代码, 便能实现有效的攻击。本文基于该随机化技术, 创新地结合动态、冗余构造方法, 设计了以 PHP 源程序为应用目标的原型系统 PHPRAND, 使得攻击者即使在获得随机化方法先验知识的情况下也极难突破系统的防御机制。

图 7 给出了 PHPRAND 的架构图, 主要包括分发表决模块、冗余的服务器软件、随机化 PHP 解释器、用户后台接口、解释器动态编译仓库。

分发表决模块是访问服务的真实出入口[23], 其中请求分发代理负责根据访问请求的文件后缀或上下文根将“.php”程序请求转发至随机化 PHP 解释器, 将“.jpg”、“.png”、“.css”、“.html”等静态资源请求复制为 2 份并转发至冗余的服务器软件; 响应表决器[24]负责收集比较来自冗余服务器软件的 2 份响应的一致性, 主动发现静态资源是否遭受恶意篡改, 若表决一致则将服务器软件 S_2 的响应返回至客户端, 若表决不一致则将默认的错误页面返回至客户端, 并通过内置的通信函数发送告警消息给静态资源服务模块, 响应表决器还负责收集随机化 PHP 解释器的响应, 并返回至客户端。由分发表决模块的功能可知, PHPRAND 针对静态资源的恶意篡改攻击采用了“冗余表决”的设计方法, 而针对 PHP 代码注入攻击则采用了本文实现基于指令集随机化的防御方法。

冗余的服务器软件配置独立存储在文件系统中的静态资源文件夹, 服务器软件 S_1 采用 Apache 配置, 服务器软件 S_2 采用 Nginx 配置。本文提出的基于指令集随机化的抗代码注入方法旨在提高 Web 应用程序的代码注入防御能力, 但是除了动态内容, 攻击者还会向 Web 应用程序的静态资源发起恶意篡改攻击, 植入黑链、涉政言论等非法信息, 因此考虑到原型系统 PHPRAND 的安全性, 静态资源采用如此的

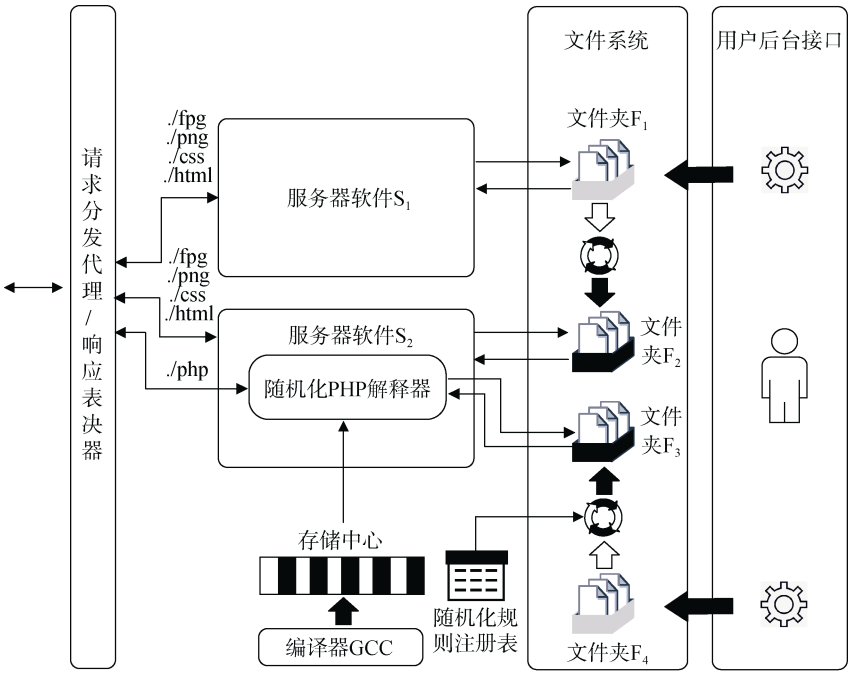


图 7　PHPRAND 架构图

Figure 7　Then architecture of PHPRAND

异构化设计能够有效提高攻击者利用服务器软件自身漏洞发起恶意篡改攻击的难度, 较全面地增强系统的防御能力。系统中 S_1 配置静态资源文件夹 F_1, S_2 配置静态资源文件夹 F_2, 由于分发表决模块只将 S_2 的响应返回至客户端, 所以 S_2 位于攻击者可达的攻击表面上, 而 S_1 不位于攻击表面, 攻击者无法对配置于 S_1 的文件夹 F_1 中静态资源实施篡改攻击。因此该系统在进行安全性评估和量化时设置以下假设:

假设 1: 服务器软件 S_1 不位于攻击表面, 攻击者不能直接攻击篡改 F_1 中静态资源。

假设 2: 文件系统已设置了访问权限, 攻击者不能间接通过突破文件系统攻击篡改 F_1 中静态资源。

F_1 与 F_2 之间建立由 F_1 向 F_2 的单向同步机制, 当出现以下任何条件时同步机制均会被触发:

(1) 服务器软件 S_1 接收到来自分发表决模块的告警消息;

(2) 用户后台接口向静态资源文件夹 F_1 中更新内容。

随机化 PHP 解释器配置独立存储在文件系统中的 PHP 源程序代码文件夹 F_3, 冗余的 F_4 中存储初始未经随机化处理的源程序代码。由于该模块在处理来自分发表决模块的“.php”程序请求时, 随机化 PHP 解释器只执行 F_3 中的代码, 所以 F_3 位于攻击者可达的攻击表面上, 而 F_4 不位于攻击表面, 攻击者无法对 F_4 中源程序代码实施注入攻击。因此该系统在进行安全性评估和量化时需增设以下假设:

假设 3: 文件夹 F_4 不位于攻击表面, 攻击者不能直接向 F_4 中注入恶意代码。

假设 4: 文件系统已设置了访问权限, 攻击者不能间接通过突破文件系统向 F_4 中注入恶意代码。

F_3 与 F_4 之间建立由 F_4 向 F_3 的单向同步机制, 并在同步过程中根据解释器动态编译仓库中的随机化规则注册表对源程序代码进行随机化处理, 当出现以下任何条件时同步机制均会被触发:

(1) 随机化 PHP 解释器产生程序代码内部执行错误, 服务器软件 S_3 返回响应状态码 500;

(2) 用户后台接口向源程序代码文件夹 F_4 中更新内容;

(3) 周期时间到达, 解释器动态编译仓库随机选择新的随机化 PHP 解释器进行部署。

解释器动态编译仓库由编译器、随机化规则注册表和存储中心组成, 该模块负责随机化 PHP 解释器的生成、维护、注销。编译器采用 GCC, 并预置编译参数, 当存储中心不足 10 个备选的随机化 PHP 解释器时, 编译器便在后台编译生成新的解释器, 并保存在存储中心。随机化规则注册表维护解释器编号与随机化规则之间的对应关系, 以便 F_4 向 F_3 的单向同步时能够准确地实施随机化变换。存储中心则是文件系统中预留存储 PHP 随机化解释器的特定

空间, 最大容量为 10 个解释器, 当到达周期时间需要替换当前解释器时, 随机选择存储中心的任一解释器, 移动到指定安装位置。

用户后台接口模块是管理用户更新系统静态资源和源程序代码的唯一途径, 同时该模块也只面向合法的管理用户开放, 攻击者无法通过该模块发起恶意攻击。因此该系统在进行安全性评估和量化时还需增设以下假设:

假设 5: 凡是通过用户后台接口对静态资源和源程序代码完成的修改或更新, 均认为是合法操作。

4.2 基于 GSPN 模型的安全性量化

Petri 网理论被广泛地应用在防御系统的安全性评估领域, 基于广义随机 Petri 网理论[25, 26]对 PHPRAND 的随机化策略与动态、冗余构造进行建模, 并利用广义随机 Petri 网与连续时间马尔科夫链同构特性, 量化计算出 PHPRAND 的逃逸概率。

建立 PHPRAND 的 GSPN 模型, 其中库所对应于遭受攻击过程中的系统状态, 变迁对应于系统遭受的攻击和防御措施。

定义 1: PHPRAND 在遭受攻击情况下的 GSPN 模型

$$GSPN=(S,T,F,K,W,M,\Lambda,\sigma) \tag{7}$$

$S=\{P_1,P_2,\cdots,P_6\}$: 系统的状态元素集合, 包括 6 个状态: P_1, P_2 含有令牌分别表示系统冗余的静态资源服务器和动态的随机化代码解释器正常工作状态; P_3 表示系统的静态资源被恶意篡改, 并由表决器发现异常; P_4 表示代码被恶意注入的尝试阶段; P_5 表示注入代码符合解释器的解析规则, 恶意注入成功; P_6 表示注入代码不符合解释器的解析规则, 恶意注入失败。

$T=\{T_1,T_2,\cdots,T_7\}$: 系统的状态变迁集合, 包括 7 个变迁: T_1 表示服务器 S_2 中静态资源被恶意篡改的时间变迁; T_2 表示代码被尝试恶意注入的时间变迁; T_3 表示代码被恶意注入成功的时间变迁; T_4 表示代码被恶意注入失败的时间变迁; T_5 表示服务器 S_2 中的静态资源还原的时间变迁; T_6 表示代码随机化方法周期性变换的时间变迁; T_7 表示清洗注入代码的时间变迁。

F: 库所与变迁之间的有向弧集合。

$K=\{1,1,0,\cdots,0\}$: 与集合 S 对应, 大小一致, 设置了状态元素的令牌容量。

W: 有向弧的权重集合, 设置为 1。

$M=\{M_0,M_1,\cdots,M_5\}$: 系统的稳定状态集合。

$\Lambda=\{\lambda,\mu_1,\mu_2,\mu_3\}$: 时间变迁 T_1,T_2,T_5,T_6,T_7 相关联的实施速率集合, λ 表示发生服务器 S_2 中静态资源被恶意篡改或代码被注入的攻击速率; μ_1 表示服务器 S_2 中静态资源由服务器 S_1 中静态资源同步还原的速率; μ_2 表示代码随机化方法变换的时间周期; μ_3 表示清洗注入代码, 还原备份代码的速率。

σ: 攻击者发起一次代码注入攻击的成功率。

表 2 PHPRAND 的 CTMC 模型稳定状态

Table 2 Stable state of abnormity CTMC model for PHPRAND

状态序号	含义
1	静态资源执行体、动态内容执行体处于正常工作状态
2	动态内容被恶意注入成功, 未出现表决异常
3	动态内容被恶意注入失败, 出现表决异常
4	静态资源被恶意篡改出现表决异常
5	静态资源被恶意篡改出现表决异常且动态内容被恶意注入成功, 未出现表决异常
6	静态资源被恶意篡改出现表决异常且动态内容被恶意注入失败, 出现表决异常

图 8 给出了 PHPRAND 在遭受攻击情况下的 GSPN 模型, 基于该模型的可达集连续时间马尔可夫链(Continuous Time Markov Chain, CTMC)如图 9 所示, CTMC 中的所有稳定状态如表 2 所示。

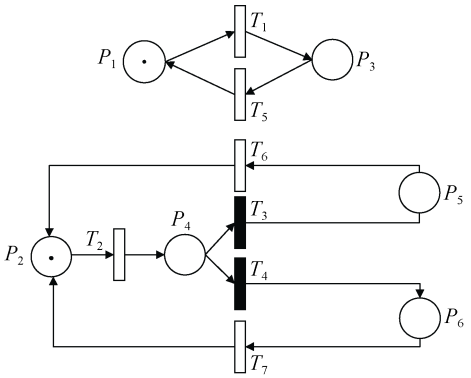


图 8 PHPRAND 遭受攻击时 GSPN 模型

Figure 8 GSPN model for PHPRAND under attacks

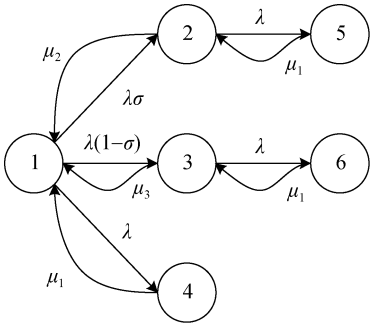


图 9 PHPRAND 遭受攻击时 CTMC 模型

Figure 9 CTMC model for PHPRAND under attacks

根据文献[26]中的稳态概率求解方法, 可得到 CTMC 的状态转移方程:

$$\begin{bmatrix}\dot{P}_{M_0}(t)\\ \dot{P}_{M_1}(t)\\ \dot{P}_{M_2}(t)\\ \dot{P}_{M_3}(t)\\ \dot{P}_{M_4}(t)\\ \dot{P}_{M_5}(t)\end{bmatrix}=\begin{bmatrix}-2\lambda & \mu_2 & \mu_3 & \mu_1 & 0 & 0\\ c\lambda & -\lambda-\mu_2 & 0 & 0 & \mu_1 & 0\\ (1-c)\lambda & 0 & -\lambda-\mu_3 & 0 & 0 & \mu_1\\ \lambda & 0 & 0 & -\mu_1 & \mu_2 & \mu_3\\ 0 & \lambda & 0 & 0 & -\mu_1-\mu_2 & 0\\ 0 & 0 & \lambda & 0 & 0 & -\mu_1-\mu_3\end{bmatrix}\begin{bmatrix}P_{M_0}(t)\\ P_{M_1}(t)\\ P_{M_2}(t)\\ P_{M_3}(t)\\ P_{M_4}(t)\\ P_{M_5}(t)\end{bmatrix} \tag{8}$$

求出各标识稳定状态概率的解:

$$\begin{cases}P_{M_0}=\dfrac{C_0}{C}\\ P_{M_1}=\dfrac{C_1}{C}\\ P_{M_2}=\dfrac{C_2}{C}\\ P_{M_3}=\dfrac{C_3}{C}\\ P_{M_4}=\dfrac{C_4}{C}\\ P_{M_5}=\dfrac{C_5}{C}\end{cases} \tag{9}$$

式中

$$\begin{aligned}C_0 &= \mu_1^3\mu_2\mu_3+\mu_1^2\mu_2^2\mu_3+\mu_1^2\mu_2\mu_3^2+2\mu_1^2\mu_2\mu_3\lambda+\\ &\quad \mu_1\mu_2^2\mu_3^2+\mu_1\mu_2^2\mu_3\lambda+\mu_1\mu_2\mu_3^2\lambda+\mu_1\mu_2\mu_3\lambda^2\\ C_1 &= \mu_1^3\mu_2\mu_3+\mu_1^2\mu_2^2\mu_3+\mu_1^2\mu_2\mu_3^2+2\mu_1^2\mu_2\mu_3\lambda+\\ &\quad \mu_1\mu_2^2\mu_3^2+\mu_1\mu_2^2\mu_3\lambda+\mu_1\mu_2\mu_3^2\lambda+\mu_1\mu_2\mu_3\lambda^2\\ C_2 &= \mu_1^3\mu_2\lambda+\mu_1^2\mu_2\lambda^2+\mu_1^2\mu_2^2\lambda-\sigma\mu_1^3\mu_2\lambda+\\ &\quad \mu_1\mu_2\mu_3\lambda^2+\mu_1\mu_2^2\mu_3\lambda+\mu_1^2\mu_2\mu_3\lambda-\\ &\quad \sigma\mu_1^2\mu_2\lambda^2-\sigma\mu_1^2\mu_2^2\lambda-\sigma\mu_1\mu_2\mu_3\lambda^2-\\ &\quad \sigma\mu_1\mu_2^2\mu_3\lambda-c\mu_1^2\mu_2\mu_3\lambda\\ C_3 &= 2\mu_2\mu_3\lambda^3+\mu_2\mu_3^2\lambda^2+2\mu_2^2\mu_3\lambda^2+\mu_2^2\mu_3^2\lambda+\\ &\quad 3\mu_1\mu_2\mu_3\lambda^2+\mu_1\mu_2\mu_3^2\lambda+\mu_1\mu_2^2\mu_3\lambda+\\ &\quad \mu_1^2\mu_2\mu_3\lambda+\sigma\mu_2\mu_3^2\lambda^2-\sigma\mu_2^2\mu_3\lambda^2\\ C_4 &= \lambda(c\mu_1^2\mu_3\lambda+c\mu_1\mu_3^2\lambda+c\mu_1\mu_3\lambda^2)\\ C_5 &= \lambda(\mu_1\mu_2\lambda^2+\mu_1\mu_2^2\lambda+\mu_1^2\mu_2\lambda-c\mu_1\mu_2\lambda^2-\\ &\quad c\mu_1\mu_2^2\lambda-c\mu_1^2\mu_2\lambda)\\ C &= C_0+C_1+C_2+C_3+C_4+C_5\end{aligned}$$

PHPRAND 的随机化标签由 4 位 0/1 编码构成, 攻击者在不知情目标系统采用了随机化方法的情况下, 代码注入成功的概率为 0, 若攻击者知情目标系统采用了随机化方法, 且获知随机化标签为 4 位的情况下, 代码注入成功的成功率为 $1/2^4$。除此之外, 结合 PHPRAND 设计, 表 3 给出了 GSPN 模型的参数设置。

表 3 PHPRAND 的 CTMC 模型参数

Table 3 Parameters of abnormity CTMC model for PHPRAND

参数	值	含义
$\lambda(h)$	6	攻击者入侵系统的平均时间为 10 分钟[26]
σ	$0,1/2^4$	分别在攻击者未获得随机化方法先验知识和已获得随机化方法先验知识的情况下恶意注入程序成功执行的概率
$\mu_1(h)$	720	静态资源被恶意篡改出现表决异常时, 静态资源还原的平均时间为 5 秒
$\mu_2(h)$	6	动态内容被恶意注入成功, 未出现表决异常时, 动态内容变换随机化规则的平均时间为 10 分钟
$\mu_3(h)$	720	动态内容被恶意注入失败, 出现表决异常时, 清洗注入代码的平均时间为 5 秒

攻击者在未获得随机化方法先验知识的情况下的逃逸概率:

$$P_{unkown}=1-P_{M_0}(t)-P_{M_2}(t)-P_{M_3}(t)-P_{M_5}(t)=0 \tag{10}$$

攻击者在获得随机化方法先验知识的情况下的逃逸概率:

$$\begin{aligned}P_{kown} &= 1-P_{M_0}(t)-P_{M_2}(t)-P_{M_3}(t)-P_{M_5}(t)\\ &= 0.05794105\end{aligned} \tag{11}$$

由计算结果可知, 攻击者无论是否获得了随机化方法的先验知识, 突破 PHPRAND 防御机制的难度都是极大的。

5 实验和现网测试

5.1 性能测试与分析

PHPRAND 根据随机化规则对源程序代码和 PHP 解释器进行随机化处理, 实现对代码注入攻击的防御, 该过程运行在系统后台, 会带来性能负担。其中, 编译生成 PHP 解释器的过程可以通过优化 GCC 编译器降低系统性能负担。表 4 列出了对目前主流的 PHP 开发框架进行随机化处理的时间和内存

开销。

表 4 PHP 程序随机化性能

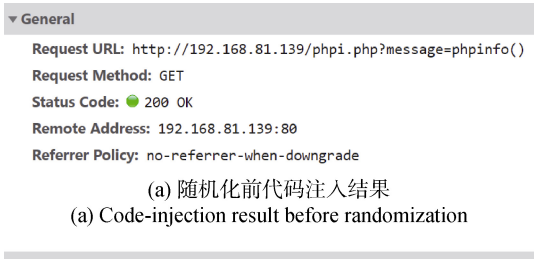
Table 4 Performance overhead of PHP applications randomization

应用框架	代码规模			处理时间(毫秒)	内存占用峰值(M)
	文件夹数量	文件数量	代码总行数		
Yii-1.1.22	262	1800	793696	138236	609.45
Wordpress-5.4.1	89	884	430209	84331	900.39
Thinkphp-5.0	78	285	69440	44361	510.50
DedeCMS-5.7	28	565	98255	97385	648.83
CodeIgniter-3.1.11	29	199	68557	31023	381.14
Cakephp-4.0.6	95	677	143948	65787	468.24

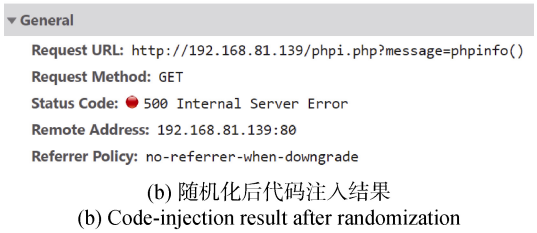
本文实现的 PHPRAND 原型系统未进行优化处理, 采用对源程序递归遍历的方式, 而在现实应用中可采用相关方法进行合理的优化, 尤其在集群和云环境, 使用 Flink、Yarn 构建任务的分布式处理方法, 能够充分利用空闲资源, 降低系统性能负担。

5.2 安全测试与分析

通过测试对比应用程序随机化前后的代码注入攻击情况来验证本文随机化变换的有效性。



(a) 随机化前代码注入结果
(a) Code-injection result before randomization



(b) 随机化后代码注入结果
(b) Code-injection result after randomization

图 10 bWAPP 应用程序随机化前后的代码注入结果

Figure 10 Results of code-injection before and after randomization

为了更全面地进行测试, 选取常见的渗透训练应用程序作为测试对象, 并以测试应用程序 bWAPP 为代表进行详细的测试描述: 首先, 将 bWAPP 部署在 Apache+PHP+mysql 实验环境中, 访问“PHP Code Injection”训练内容, 分析 phpi.php 源代码可知该训练内容可以在 URL 的 *message* 参数处注入程序代码; 然后, 使用发包工具构造 *message*=phpinfo()的 URL 请求发送至服务端, 得到代码 phpinfo()的执行结果, 如图 10(a)所示。可见, 攻击者可以该应用程序的漏洞向服务器端注入恶意程序代码, 并成功执行。

之后, 对 bWAPP 进行随机化保护,再次使用发包工具构造 *message*=phpinfo()的 URL 请求发送至服务器端, 得到状态码为 500 的错误结果, 如图 10(b)所示, 说明攻击者已经无法执行向服务器端注入的恶意程序代码, 应用程序的随机化处理后能够有效抵御代码注入攻击。

表 5 给出了对 bWAPP[27]、DVWA[28]、MCIR[29]三种应用程序在随机化变换后的攻击测试结果。

表 5 实验测试结果

Table 5 Results of experimental tests

应用程序	注入点		能否防御
	文件位置	代码片段	
bWAPP	/phpi.php	<?php @eval ("echo " . $_REQUEST["message"] . ";");?>	√
DVWA	/vulnerabilities/upload/source/low.php	<?php @eval($_POST['123']); ?>	√
MCIR	/phpwn/eval.php	$output = eval($eval_string);	√

通过上述测试表明: PHPRAND 能够有效实现对 php 程序源代码的随机化变换, 并且能够正确运行随机化变换后的程序, 攻击者通过正常的代码无法对随机化变换后的应用程序进行代码注入攻击。

5.3 现网测试与分析

为了有效评估该随机化方法在现网环境中的防御成效, 采用 PHPRAND 对郑州市景安网络科技股份有限公司(以下简称景安网络)的快云 Web 云服务进行适应性改造与测试。景安网络快云服务系统是在物理设备支撑的基础上, 使用物理设备搭建云服务集群, 在集群环境中创建虚拟机, 每个虚拟机中创建多台虚拟站点, 每台虚拟站点可为用户的一个网站提供 Web 访问服务。测试实施过程中将每台虚拟站点改造成基于 PHPRAND 的虚拟站点, 改造后的云服务集群中选取 100 台虚拟站点进行 24 小时运行情况监控。

虽然景安网络快云服务在服务入口部署了防火墙、开源云端专业版 WAF 等, 但仍有部分攻击行为突破了外挂式的被动防御措施而对用户网站造成威胁。自 2018 年 4 月 3 日至 2019 年 8 月 31 日(515 天), 100 台虚拟站点共被访问 33062237 次, 发现并抵御了绕过现有防护措施的异常访问和威胁访问共计 8274284 次, 其中代码注入攻击尝试 3911262 次, 占

总访问量的 11.83%。根据以上统计结果, 选取危害较大、具有代表性的案例进行分析。

案例一: 利用后门上传恶意程序

IP 地址为 45.116.176.244(香港)的攻击者利用某网站中的 upcache.php 脚本, 远程上传了 11.php、1122.php 等恶意脚本, 如图 11 所示。

```
45.116.176.244 - - [11/Apr/2018:11:58:31 -0400] "POST /1/plus/task/upcache.php?act=dl HTTP/1.1" 200 156 "http://wxshicai.net/1/plus/task/upcache.php?act=dl" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"
```

图 11 攻击者上传脚本的日志

Figure 11 Log of attacker uploading webshell

随后, 攻击者尝试触发上传的恶意脚本 11.php、1122.php 等多达 269 次, 系统对攻击进行了有效的封堵, 脚本无法成功触发, 如图 12 所示。

```
[11-Apr-2018 16:02:44] WARNING: [pool nobody] child 20640 said into stderr: "NOTICE: PHP message: PHP Fatal error:  Function name must be a string in /home/online/2_WEB/DA459957/WEB/1/plus/task/11.php on line 3"
[11-Apr-2018 16:09:21] WARNING: [pool nobody] child 20640 said into stderr: "NOTICE: PHP message: PHP Fatal error:  Function name must be a string in /home/online/2_WEB/DA459957/WEB/1/plus/task/1122.php on line 3"
```

图 12 案例一的部分防御日志

Figure 12 Part of defense log for case 1

案例二: 对恶意程序进行加密, 试图绕过

IP 地址为 66.46.80.186(加拿大)的攻击者向某网站上传加过密的 admin_channel.php 木马, 通过该木马欲获得系统权限, 控制整个 Web 服务集群。该攻击者尝试触发 admin_channel.php 木马时, 系统对本次攻击进行了有效的封堵, 如图 13 所示。

```
[26-Jul-2018 16:51:26] WARNING: [pool nobody] child 11671 said into stderr: "NOTICE: PHP message: PHP Fatal error:  Function name must be a string in /home/online/2_WEB/DA459957/WEB/1/dede/inc/admin_channel.php on line 6"
```

图 13 案例二的部分防御日志

Figure 13 Part of defense log for case 2

案例三: 程序设计存在漏洞, 注入恶意代码

IP 地址为 223.149.11.134(湖南省)的攻击者向某网站上传 cheng.lib.php 脚本至织梦框架(景安快云采用的用户默认框架)include 文件夹下的 taglib 文件夹内, 利用 index.php 中 settemplate 函数调用 makeoneTag 函数的过程中, makeoneTag 函数对 taglib 下的文件没有进行过滤, 攻击者能够加载 taglib 文件夹中的任何文件, 进而开展攻击。该攻击者尝试触发 cheng.lib.php 脚本时, 系统对本次攻击进行了有效的封堵, 脚本无法成功触发, 如图 14 所示。

```
[12-May-2018 13:00:02] WARNING: [pool nobody] child 21582 said into stderr: "NOTICE: PHP message: PHP Fatal error:  Function name must be a string in /home/online/1_WEB/DA207678/WEB/include/taglib/cheng.lib.php on line 1"
```

图 14 案例三的部分防御日志

Figure 14 Part of defense log for case 3

通过上述案例分析表明: PHPRAND 能够对部署在其中的应用程序进行有效的代码注入防护, 即使应用程序存在设计缺陷或后门, 攻击者以正常服务为攻击路径, 绕过了外挂式的被动防护, 向目标系统中注入了恶意代码, 也无法有效地对其触发执行, 代码注入攻击失败。因此, 针对代码注入攻击, 本文提出的随机化方法不再关心恶意代码如何注入到运行环境中, 而是阻止恶意代码被执行, 实现有效的防护。

6 局限性分析

PHPRAND 的局限性主要体现在以下两个方面:

(1) PHPRAND 是根据 PHP 应用程序设计而成, PHP 程序是一种解释执行语言, 本文提出的基于指令集随机化的抗代码注入方法也可以应用于其他具有解释过程的语言程序, 只需解决程序在执行过程中词法分析与语法分析的定位, 可能会涉及到程序分析技术的研究, 由于本文重点在于创新地提出这种基于指令集随机化的防御方法, 设计实现原型系统 PHPRAND, 通过实验和现网测试验证了该方法的有效性, 并未对所有程序语言类型的支持度进行研究。

(2) 本文通过 GSPN 模型计算出 PHPRAND 的安全理论值, 虽然计算结果显示攻击者在获得当前系统随机化方法的情况下, 突破系统防御的概率极小, 但是仍存在攻击者绕过随机化防御的可能。攻击者一旦获得 PHPRAND 系统中的随机化规则注册表, 便能在每次发起代码注入攻击时, 向服务器端注入符合随机化规则的恶意代码, 本节基于拟态防御架构提出 PHPRAND 的改进思路, 并未对具体的实现和应用进行深入研究。

拟态防御[30-32]以软硬件多样性为基础, 以异构性最大化为主要目标, 综合了冗余、表决、主动重构、动态迁移[33]、重配置等技术, 构建了动态异构冗余结构(Dynamic Heterogeneous Redundancy, DHR), 如图 15 所示, 由输入代理、可重构的异构组件池、异构执行体集、负反馈控制器、表决器组成。拟态防御架构下同时运行多个功能等价、结构相异的执行体, 以锁定步骤、同步执行的方式将程序的输入复制分

发至所有执行体, 不同执行体对正常的输入能够产生一致的输出结果, 但是对具有攻击行为的输入会产生不一致的输出结果, 因此在架构的表决处通过对执行体的输出结果进行裁决就能够发现执行体产生的异常结果。

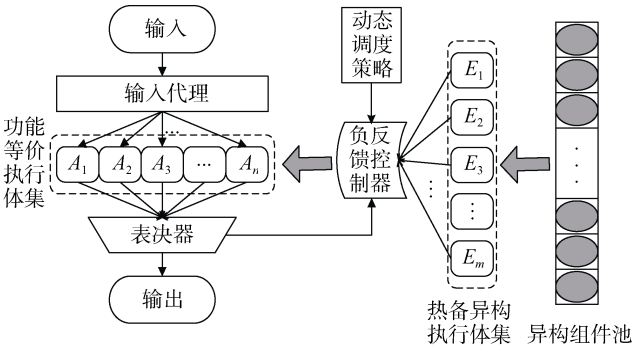


图 15 拟态防御系统 DHR 架构
Figure 15 The DHR architecture of mimic defense system

基于拟态防御架构对 PHPRAND 的改进, 增加冗余的服务器软件 S_3, 并采用与 S_2 相异的 Lighttpd, 配置源程序代码文件夹 F_5, 以及已经编译生成的随机化 PHP 解释器, 该解释器的随机化标签与 S_2 中的解释器随机化标签互不相同, 构建异构、冗余的 PHP 程序执行环境。同时分发表决模块将“.php”程序请求复制为 2 份并转发至 PHP 程序服务模块, 其中响应表决器负责收集比较来自 S_2 和 S_3 的 2 份响应的一致性, 若表决一致则将响应返回至客户端, 若表决不一致则将默认的错误页面返回至客户端, 并通过内置的通信函数发送告警消息给 PHP 程序服务模块, 更换随机化 PHP 解释器。

当基于拟态防御架构改进时, 采用 3.2 节中 $f()$ 表示程序执行的过程, $interpreter_1$、$interpreter_2$ 分别表示 S_2、S_3 中随机化解释器, 当用户请求合法文件 *file* 时, 程序执行过程可以表示为:

$$\begin{cases} f(interpreter_1, file)=200 \\ f(interpreter_2, file)=200 \end{cases} \tag{12}$$

一致的结果可以通过表决器, 响应信息返回至用户。

当攻击者注入恶意程序 *file* 符合 $interpreter_1$ 的解析规则时, 程序执行过程可以表示为:

$$\begin{cases} f(interpreter_1, file)=200 \\ f(interpreter_2, file)=500 \end{cases} \tag{13}$$

不一致的结果无法通过表决器, 错误提示页面返回至用户。

当攻击者注入恶意程序 *file* 符合 $interpreter_2$ 的解析规则时, 程序执行过程可以表示为:

$$\begin{cases} f(interpreter_1, file)=500 \\ f(interpreter_2, file)=200 \end{cases} \tag{14}$$

不一致的结果无法通过表决器, 错误提示页面返回至用户。

由于请求通过分发表决模块进行复制转发, 攻击者无法向冗余的服务器软件中注入不一致的恶意代码, 因此恶意代码无法在随机化变换方式不同的冗余解释器中同时执行, 有效避免了攻击者在白盒环境下绕过随机化防御机制的可能。该改进方法在实现中需要着重考虑如何保证程序冗余执行的正确性, 例如数据库的冗余操作是否会影响数据的正确性等问题还需要深入研究。

7 总结

本文提出了一种基于指令集随机化的抗代码注入方法, 详细论述了对程序源代码和程序运行环境的随机化变换过程, 为应用程序构建具有随机化特征的内生安全构造, 在不依赖于攻击者采用何种攻击方式的情况下, 使得直接或间接注入的恶意代码无法执行, 从而有效抵御代码注入攻击。基于该方法并引入动态和冗余因素, 设计实现了原型系统 PHPRAND, 并通过 GSPN 模型计算出该系统的安全理论值, 计算结果表明攻击者即使获得了随机化方法先验知识, 突破系统防御的概率仍小于 6%。性能测试结果显示, 该方法在对源程序进行随机化处理时会带来时间和计算资源开销, 如何降低性能开销, 提高随机化处理速率将是未来工作的重点之一。通过应用程序随机化前后的对比实验和现网测试证明, 该方法可有效抵御大部分代码注入攻击, 解决被动防御存在被绕过风险的难题, 具有良好的实用性。

致 谢 本文工作受到国家重点研发计划网络空间安全专项(No.2018YFB0804003)资助, 以及工信部网安局关于现网试点工作的指导, 感谢在本文涉及的现网测试工作中,郑州景安网络科技股份有限公司的苑立涛、王子、李华普做出的贡献。

参考文献

[1] Prokhorenko V, Choo K K R, Ashman H. Web Application Protection Techniques: A Taxonomy[J]. *Journal of Network and Computer Applications*, 2016, 60: 95-112.

[2] Gurina A, Eliseev V. Anomaly-Based Method for Detecting Multi-

ple Classes of Network Attacks[J]. *Information*, 2019, 10(3): 84.

[3] Medeiros I, Neves N, Correia M. Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining[J]. *IEEE Transactions on Reliability*, 2016, 65(1): 54-69.

[4] Sahingoz O K, Buber E, Demir O, et al. Machine Learning Based Phishing Detection from URLs[J]. *Expert Systems With Applications*, 2019, 117: 345-357.

[5] Abaimov S, Bianchi G. CODDLE: Code-Injection Detection with Deep Learning[J]. *IEEE Access*, 2019, 7: 128617-128627.

[6] M.Y. Ma, L.W. Chen, D. Meng. A Survey of Memory Corruption Attack and Defense[J]. *Journal of Cyber Security*, 2017, 2(4):. 82-98.(马梦雨，陈李维，孟丹. 内存数据污染攻击和防御综述[J]. *信息安全学报*, 2017, 2(4): 82-98.)

[7] F.F. Wang, T. Zhang, W.G. Xu, et al. Overview of control-flow hijacking attack and defense techniques for process[J]. *Chinese Journal of Network and Information Security*, 2019, 5(6): 10-20.(王丰峰，张涛，徐伟光, 等. 进程控制流劫持攻击与防御技术综述[J]. *网络与信息安全学报*，2019, 5(6): 10-20.)

[8] Kc G S, Keromytis A D, Prevelakis V. Countering Code-injection Attacks with Instruction-set Randomization[C]. *The 10th ACM conference on Computer and communication security*, 2003: 272-280.

[9] Keromytis A D. Randomized Instruction Sets and Runtime Environments Past Research and Future Directions[J]. *IEEE Security & Privacy Magazine*, 2009, 7(1): 18-25.

[10] S. Du, H. Shu, F. Kang. Design and implementation of hardware-based dynamic instruction set randomization framework[J]. *Chinese Journal of Network and Information Security*, 2017, 3(11): 29-39. (杜三，舒辉，康绯. 基于硬件的动态指令集随机化框架的设计与实现[J]. *网络与信息安全学报*，2017, 3(11): 29-39.)

[11] Barrantes E G, Ackley D H, Palmer T S, et al. Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks[C]. *The 10th ACM conference on Computer and communication security*, 2003: 281-289.

[12] T. Liu, G. Shi, D. Meng. A Survey of Code Reuse Attack and Defense Mechanisms[J]. *Journal of Cyber Security*, 2016, 1(2): 15-27.(柳童，史岗，孟丹. 代码重用攻击与防御机制综述[J]. *信息安全学报*，2016, 1(2): 15-27.)

[13] X.D. Qiao, R.X. Guo, Y. Zhao. Research progress in code reuse attacking and defending[J]. *Chinese Journal of Network and Information Security*, 2018,4(3): 1-12. (乔向东，郭戎潇，赵勇.代码复用对抗技术研究进展[J]. *网络与信息安全学报*，2018, 4(3): 1-12.)

[14] W. Wu, W. Huo, W. Zou. Survey on Attacking and Defending Technologies of Dynamic Code Generation[J]. *Journal of Cyber Security*, 2016, 1(4): 52-64.(吴炜，霍玮，邹维. 面向动态生成代码的攻防技术综述[J]. *信息安全学报*，2016, 1(4): 52-64.)

[15] G.M. Zhang, Q.B. Li, G.Y. Zeng, et al. Defensing Code Reuse Attacks Using Live Code Randomization[J]. *Journal of Software*, 2019, 30(9): 2772-2790.(张贵民，李清宝，曾光裕, 等. 运行时代码随机化防御代码复用攻击[J]. *软件学报*，2019, 30(9): 2772-2790.)

[16] Nystrom N, Clarkson M R, Myers A C. Polyglot: An Extensible Compiler Framework for Java[M]. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003: 138-152.

[17] Guanciale R. Protecting Instruction Set Randomization from Code Reuse Attacks[J]. *Secure IT Systems*, 2018: 421-436.

[18] M. Werner, T. Unterluggauer, L. Giner, et al. ScatterCache: thwarting cache attacks via cache set randomization[J]. *28th USENIX Security Symposium (USENIX Security 19)*, 2019: 675-692.

[19] Portokalidis G, Keromytis A D. Global ISR: Toward a Comprehensive Defense Against Unauthorized Code Execution[M]. Advances in Information Security. New York, NY: Springer New York, 2011: 49-76.

[20] Boyd S W, Keromytis A D. SQLrand: Preventing SQL Injection Attacks[M]. Applied Cryptography and Network Security. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004: 292-302.

[21] Wang Y, Li Z J, Guo T. Literal Tainting Method for Preventing Code Injection Attack in Web Application[J]. *Journal of Computer Research and Development*, 2012, 49(11): 2414-2423. (王溢, 李舟军, 郭涛. 防御代码注入式攻击的字面值污染方法[J]. 计算机研究与发展, 2012, 49(11): 2414-2423.)[维普]

[22] Hranický R, Zobal L, Ryšavý O, et al. Distributed Password Cracking with BOINC and Hashcat[J]. *Digital Investigation*, 2019, 30: 161-172.

[23] Z. Zhang, B.L. Ma, J.X. Wu. The Test and Analysis of Prototype of Mimic Defense in Web Servers[J]. *Journal of Cyber Security*, 2017, 2(1): 13-28.(张铮，马博林，邬江兴. web 服务器拟态防御原理验证系统测试与分析[J]. *信息安全学报*，2017, 2(1): 13-28.)

[24] B.L. Ma, Z. Zhang, J.X. Liu. A Similarity Calculation Method applied to Dynamic Heterogeneous Web Server System[J]. *Computer Engineering and Design*, 2018, 1(2): 282-287.
(马博林，张铮，刘健雄. 应用于动态异构 web 服务器的相似度求解方法[J]. *计算机工程与设计*, 2018, 1(2): 282- 287.)

[25] 林闯. 计算机网络和计算机系统的性能评价[M]. 北京: 清华大学出版社, 2001.

[26] Q. Ren, J.X. Wu, L. He. Research on Mimic DNS Architectural Strategy Based on Generalized Stochastic Petri Net[J]. *Journal of Cyber Security*, 2019, 4(2): 37-52.(任权，邬江兴，贺磊. 基于 GSPN 的拟态 DNS 构造策略研究[J]. *信息安全学报*，2019, 4(2): 37-52.)

[27] https://sourceforge.net/projects/bwapp.

[28] https://github.com/ethicalhack3r/DVWA.

[29] https://github.com/SpiderLabs/MCIR.

[30] J.X. Wu. Research on Cyber Mimic Defense[J]. *Journal of Cyber Security*, 2016, 1(4): 1-10.(邬江兴. 网络空间拟态防御研究[J]. *信息安全学报*，2016, 1(4): 1-10.)

[31] Q. Tong, Z. Zhang, J.X. Wu. The Active Defense Technology Based on the Software/Hardware Diversity[J]. *Journal of Cyber Security*, 2017, 2(1): 1-12.(仝青，张铮，邬江兴. 基于软硬件多样性的主动防御技术[J]. *信息安全学报*，2017, 2(1): 1-12.)

[32] W.C. Li, Z. Zhang, L.Q. W, et al. The Modeling and Risk Assessment on Redundancy Adjudication of Mimic Defense[J]. *Journal of Cyber Security*, 2018, 3(5): 64-74.(李卫超，张铮，王立群, 等. 基于拟态防御架构的多余度裁决建模与风险分析[J]. *信息安全学报*，2018, 3(5): 64-74.)

[33] W. Peng, F. Li, C.T. Huang, et al. A moving-target defense strategy for cloud-based services with heterogeneous and dynamic attack surfaces[J]. *2014 IEEE International Conference on Communications*, 2014: 804-809.

马博林 于 2015 年在哈尔滨工业大学信息安全专业获得学士学位。现在战略支援部队信息工程大学网络空间安全专业攻读博士学位。研究领域为网络空间安全。研究兴趣包括: 主动防御技术、多变体执行技术、拟态防御。Email: msd_mbl@qq.com

张铮 于 2006 年在解放军信息工程大学计算机科学与技术专业获得博士学位。现任数学工程与先进计算国家重点实验室副教授。研究领域为网络空间安全、先进计算。研究兴趣包括: 主动防御技术、高性能计算。Email: ponyzhang@126.com

陈源 现任江南计算技术研究所工程师。研究领域为信息安全。研究兴趣包括: 云安全、拟态防御。Email: chen13426241793@163.com

邬江兴 现任国家数字交换系统工程技术研究中心主任, 教授, 中国工程院院士。研究领域为信息通信网络、网络空间安全。Email: ndscwjx@126.com