AFL 生态圈

转载

woswod 于 2019-04-27 20:58:16 发布 1831 收藏 11
分类专栏： 漏洞挖掘 文章标签： AFL 漏洞挖掘

AFL 生态圈

在本文中，我们将讨论的不是经典AFL本身，而是关于为其设计的实用程序及其修改，我们认为，这些实用程序可以显着提高模糊测试的质量。如果你想知道如何提高AFL以及如何更快地找到更多漏洞 - 继续阅读！

什么是AFL，它有什么用？

AFL是一种覆盖引导或基于反馈的模糊器。关于这些概念的更多信息可以在一篇很酷的论文“Fuzzing：Art，Science，and Engineering”中找到。让我们总结一下AFL的一般信息：

它修改可执行文件以了解它如何影响覆盖范围。

改变输入数据以最大化覆盖范围。

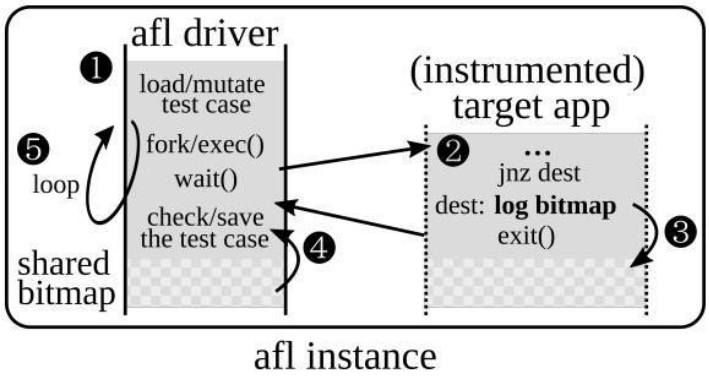
重复上一步以找到程序崩溃的位置。

它非常有效，通过实践证明。

它非常易于使用。

这是一个图形表示：

Repeating
(1) Reading and mutating inputs
(2) Launching the target application
(3) Executing and recording runtime coverage
(4) Bookkeeping results



如果你不知道AFL是什么，这里有一个有用的资源列表供你开始：

该项目的官方页面。

afl-training - AFL的简短介绍。

afl-demo - 使用AFL模糊C ++程序的简单演示。

afl-cve - AFL发现的漏洞集合（自2017年以来尚未更新）。

在这里,您可以阅读AFL在构建过程中添加到程序中的内容。

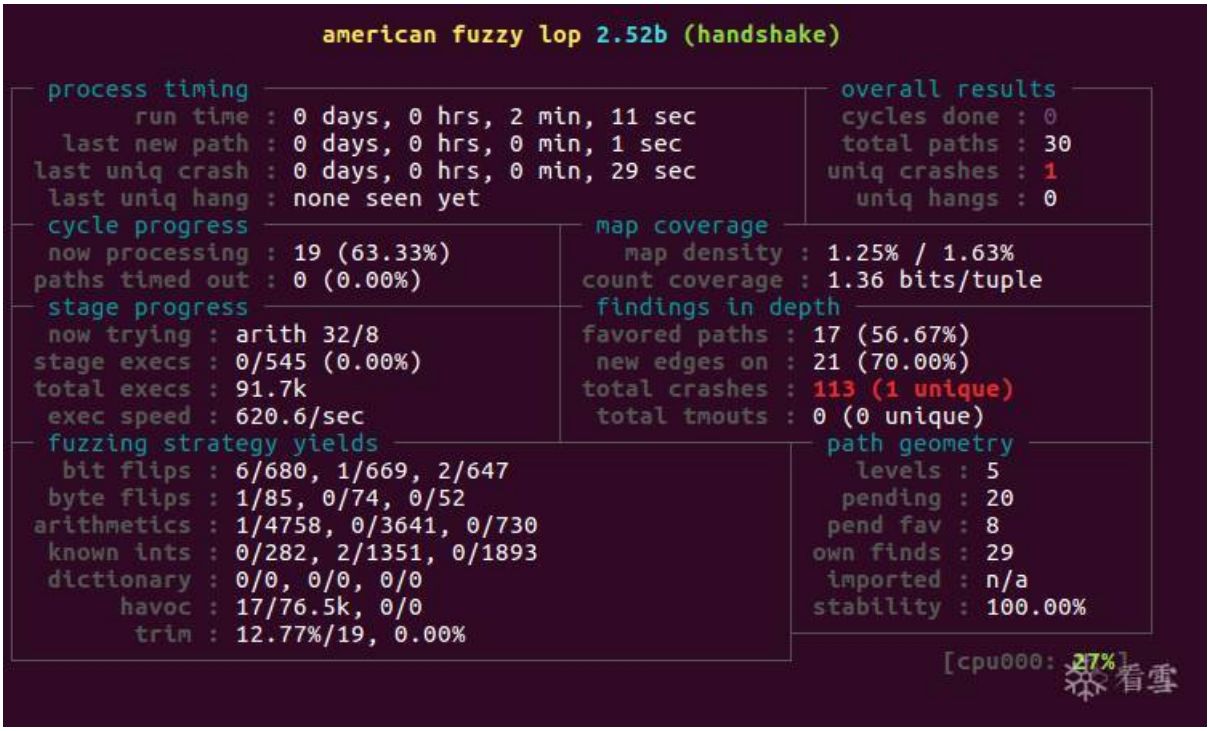
关于模糊网络应用程序的一些有用提示。

在撰写本文的那一刻，AFL的最新版本为2.52b。模糊器正在积极开发中，随着时间的推移，一些侧面开发正在被纳入主AFL分支并变得无关紧要。今天，我们可以列举几个有用的附件工具，这些工具将在下一章中列出。

Rode0day比赛

> 还值得一提的是每月一次的Rode0day竞赛 - 在这里，模糊测试人员试图以更少的时候在预制语料库中比对手找到最大数量的错误，无论是否有源代码。就其本质而言，Rode0day是AFL的不同修改和分叉之间的争斗。

一些AFL用户指出，其作者Michal Zalewski显然已经放弃了自上次修改日期至2017年11月5日的项目。这可能与他离开Google并开展一些新项目有关。因此，用户开始为最新的当前版本2.52b 制作新的补丁。



AFL还有不同的变体和衍生物，允许模糊化Python，Go，Rust，OCaml，GCJ Java，内核系统调用，甚至整个虚拟机。

AFL用于其他编程语言

python-afl - 用于Python。

afl.rs - 用于在Rust上编写的模糊程序。

afl-fuzz-js - 用于javascript的afl-fuzz。

java-afl - 用于Java的AFL模糊测试。

kelinci - Java的另一个fuzzer（关于该主题的文章）。

javan-warty-pig - 用于JVM的AFL式模糊器。

afl-swift - 用于在swift上编写的模糊程序。

ocamlopt-afl - 用于OCaml。

sharpfuzz - 基于afl for .net的模糊器。

附件工具

在本章中，我们收集了AFL的各种脚本和工具，并将它们分为几类：

崩溃处理

afl-utils - 一组用于自动处理/分析崩溃并减少测试用例数量的实用程序。

afl-crash-analyzer - AFL的另一个崩溃分析仪。

fuzzer-utils - 一组用于分析结果的脚本。

atriage - 一种简单的分类工具。

afl-kit - 关于Python的afl-cmin。

AFLize - 一种自动生成适合AFL的debian包构建的工具。

afl-fid - 一组用于处理输入数据的工具。

处理代码覆盖率

afl-cov - 提供有关覆盖范围的人性化数据。

count-afl-calls - 比率评估。脚本计算二进制中的检测块数。

afl-sancov - 就像afl-cov一样，但是使用了一种 clang sanitizer.。

covnavi - Cisco Talos Group负责代码和分析的脚本。

LAF LLVM - 通过类似于AFL的补丁集合，可以修改代码，使模糊器更容易找到分支。

一些用于最小化测试用例的脚本

afl-pytmin - afl-tmin的包装器，它试图通过使用许多CPU内核来加速最小化测试用例的过程。

afl-ddmin-mod - 基于ddmin算法的afl-tmin的变化。

halfempty - 是Tavis Ormandy基于并行化最小化测试用例的快速实用工具。

分布式执行

disfuzz-afl - 为AFL分发模糊测试。

AFLDFF - AFL分布式模糊测试框架。

afl-launch - 用于执行许多AFL实例的工具。

afl-motherhip - 在AWS云上管理和执行许多同步AFL模糊器。

afl-in-the-cloud - 另一个在AWS中运行AFL的脚本。

VU_BSc_project - 使用libFuzzer和AFL对开源库进行模糊测试。

最近，已经发表了一篇非常好的文章，论文名称为“Scaling AFL to a 256 thread machine”。

部署，管理，监控，报告

afl-other-arch - 是一组补丁和脚本，用于轻松添加对AFL的各种非x86体系结构的支持。

afl-trivia - 一些简化AFL管理的小脚本。

afl-monitor - 用于监控AFL的脚本。

afl-manager - 用于管理multi-afl的Python上的Web服务器。

afl-tools - 具有afl-latest，afl-dyninst和Triforce-afl的码头工具的图像。

afl-remote - 用于远程管理AFL实例的Web服务器。

AFL修改

AFL对脆弱性研究人员和模糊本身产生了非常强烈的影响。一段时间后，人们开始根据原始AFL进行修改，这一点都不足为奇。我们来看看它们。在不同的情况下，与原始AFL相比，这些修改中的每一个都有其自身的优缺点。

几乎所有的mod都可以在hub.docker.com找到。

- 提高速度和/或代码覆盖率
  - 算法
  - 环境
    - OS
    - 硬件
- 没有源代码工作
  - 代码仿真
  - 代码检测
    - 静态的
    - 动态的

AFL操作的默认模式

在继续研究AFL的不同修改和分支之前，我们必须讨论两种重要的模式，这些模式在过去也有过修改但最终被合并。他们是Syzygy和Qemu。

Syzygy模式 - 是在instrument.exe中工作的模式

```
instrument.exe --mode=afl --input-image=test.exe --output-image=test.instr.exe
```

Syzygy允许使用AFL静态重写PE32二进制文件，但需要符号和一个额外的开发人员才能识别WinAFL内核。

Qemu模式 - 它在QEMU下的工作方式可以在 “Internals of AFL fuzzer — QEMU Instrumentation”.看到。使用QEMU的二进制文件的支持被添加到版本1.31b中的上游AFL。AFL QEMU模式使用二进制检测功能添加到qemu tcg（一个微小的代码生成器）二进制转换引擎中。为此，AFL有一个构建脚本qemu，它提取某个版本的qemu（2.10.0）的源代码，将它们放到几个小的补丁上，并为定义的架构构建。然后，创建一个名为afl-qemu-trace的文件，该文件实际上是一个用户模式仿真文件（仅可模拟可执行ELF文件）qemu-。因此，对于qemu支持的许多不同体系结构，可以使用对elf二进制文件的反馈进行模糊测试。此外，您还可以获得所有酷炫的AFL工具，从显示器上获取有关当前会话的信息，以及afl-analyze等高级内容。但是你也得到了qemu的局限性。此外，如果使用硬件SoC特性构建工具链文件(启动二进制文件，qemu不支持)，那么一旦使用了特定的指令或特定的MMIO, fuzzing就会中断。

这是qemu模式的 另一个有趣的分支，其中速度增加了3-4倍，使用TCG代码插桩和兑现。

分支

AFL 分支的出现首先与经典AFL算法的变化和改进有关。

pe-afl - 对Windows操作系统中没有源代码的PE文件进行模糊测试的修改。对于其操作，模糊器使用IDA Pro分析目标程序，并生成以下静态检测的信息。然后用AFL插桩版本进行模糊测试。

afl-cygwin - 尝试使用Cygwin将经典AFL移植到Windows。不幸的是，它有很多错误，它很慢，并且已经放弃了开发。

AFLFast（使用Power Schedules扩展AFL） - 首批AFL分支之一。它增加了启发式功能，允许它在短时间内通过更多路径。

FairFuzz - AFL的延伸，针对罕见的分支机构。

AFLGo - 是AFL的扩展，用于获取代码的某些部分而不是完整的程序覆盖。它可用于测试补丁或新添加的代码片段。

PerfFuzz - AFL的扩展，用于查找可能显着减慢程序速度的测试用例。

Pythia - 是AFL的扩展，旨在预测找到新路径的难度。

Angora - 是最新的模糊器之一，用rust写的。它使用新的策略进行变异并增加覆盖范围。

Neuzz - 结合神经网络进行fuzzing。

UnTracer-AFL - 将AFl与UnTracer集成以实现有效跟踪。

Qsym - 为混合模糊测试量身定制的实用复杂执行引擎。从本质上讲，它是一个符号执行引擎（基本组件被实现为intel引脚的插件），它与AFL一起执行混合模糊测试。这是基于反馈的模糊测试演变的一个阶段，需要单独讨论。它的主要优点是可以相对快速地进行执行。这是由于本机执行命令而没有代码，快照和一些启发式的中间表示。它使用旧的Intel引脚（由于libz3和其他DBT之间的支持问题），目前可以使用elf x86和x86_64架构。

Superion - Greybox模糊器，其显而易见的优点是，除了插桩程序外，它还使用ANTLR语法获取输入数据的规范，然后在此语法的帮助下执行突变。

AFLSmart - 另一个Graybox模糊器。作为输入，它以Peach模糊器使用的格式获得输入数据的规范。

有许多研究论文致力于实现AFL被修改的新方法和模糊测试技术。只有白皮书，所以我们甚至没有提到这些。如果你愿意，你可以谷歌他们。例如，最新的一些是CollAFL: Path Sensitive Fuzzing, EnFuzz, «Efficient approach to fuzzing interpreters», ML 用于AFL。

基于Qemu的修改

TriforceAFL - AFL / QEMU模糊化系统的完全模拟。由nccgroup提供的一个分支。允许在qemu模式下对整个操作系统进行模糊测试。它是通过一个特殊指令（aflCall（0f 24））实现的，它是在QEMU x64 CPU中添加的。不幸的是，它不再受支持; AFL的最后一个版本是2.06b。

TriforceLinuxSyscallFuzzer - Linux系统调用的模糊测试。

afl-qai - QEMU增强插桩（qai）的小型演示项目。

基于KLEE的修改

kleefl - 用于通过符号执行生成测试用例（在大程序上非常慢）。

基于Unicorn 的修改

afl-unicorn - 允许通过在Unicorn引擎上模拟代码片段进行模糊测试。我们在实践中成功地使用了AFL的这种变体，在SOC上执行的某个RTOS的代码区域，因此我们无法使用QEMU模式。在我们没有源代码的情况下（我们无法构建用于分析解析器的独立二进制文件）和程序不直接获的输入数据（例如加密数据或者像CGC二进制文件的信号样本），然后我们可以逆向并找到所谓的place-function，其中数据的格式能被模糊器方便的进行处理。这是AFL最普遍/通用的修改，即它允许任何模糊测试。它独立于架构，源，输入数据格式和二进制格式（bare-metal的最引人注目的例子 - 只是来自控制器内存的代码片段）。研究人员首先检查这个二进制文件并编写一个模糊器，它在解析器过程的输入端模拟状态。显然，与AFL不同，这需要对二进制进行一定的检查。对于bare-mete固件，如Wi-FI或基带，您需要记住一些缺点：

1. 我们必须本地化控制和的检查。
2. 请记住，模糊器的状态是保存在内存转储中的内存状态，这可以防止模糊器进入某些路径。
3. 没有动态内存调用的sanitation，但它可以手动实现，它将取决于RTOS（必须进行研究）。
4. 未模拟Intertask RTOS交互，这也可能阻止查找某些路径。

使用这种修改的例子 “afl-unicorn: Fuzzing Arbitrary Binary Code” 和 afl-unicorn: Part 2 — Fuzzing the ‘Unfuzzable。

在我们继续基于动态二进制检测（DBI）框架进行修改之前，让我们不要忘记这些框架的最高速度由DynamoRIO，DynInst以及最终的PIN显示。

基于PIN的修改

aflpin - 采用英特尔PIN工具的AFL。

afl_pin_mode - 通过英特尔PIN实现的另一种AFL插桩。

afl-pin - 带PINtool的AFL。

NaF1 - AFL 模糊器的克隆（基本核心）。

PinAFL - 这个工具的作者试图将AFL移植到Windows，以便对已编译的二进制文件进行模糊测试。好像是为了好玩而一夜之间完成的; 该项目从未进一步发展。存储库没有源代码，只有已编译的二进制文件和启动指令。我们不知道它基于哪个版本的AFL，它只支持32位应用程序。

正如您所看到的，有许多不同的修改，但它们在现实生活中并不是非常有用。

基于Dyninst的修改

afl-dyninst- American Fuzzy Lop + Dyninst == AFL balckbox模糊。这个版本的特点是首先使用Duninst对一个研究过的程序（没有源代码）进行静态检测（静态二进制插桩，静态二进制重写），然后对经典的AFL进行模糊测试，认为程序是用afl-gcc / afl-g ++ / afl-as）因此，它允许在没有源代码的情况下以非常好的生产率工作 - 与本机编译相比，它曾经是0.25倍速。与QEMU相比，它具有显着优势：它允许动态链接库的检测，而QEMU只能检测与库静态链接的基本可执行文件。不幸的是，现在它只与Linux有关。支持Windows，更改Dyninst本身需要，这正在做。

还有另一个分支 ，具有改进的速度和某些功能（AARCH64和PPC架构的支持）。

基于DynamoRIO的修改

drAFL - AFl + DynamoRIO - 在Linux上没有源代码的模糊测试。

afl-dr - 另一种基于DynamoRIO的实现，它在Habr上有很好的描述。

afl-dynamorio - vanhauser-thc的修改。以下是他所说的：«run AFL with DynamoRIO when normal afl-dyninst is crashing the binary and qemu mode -Q is not an option»。它支持ARM和AARCH64。关于生产力：DynamoRIO比Qemu慢大约10倍，比dyninst慢25倍，但比Pintool快10倍。

WinAFL - 最着名的在Windows上面的AFL分支。（DynamoRIO，也是syzygy模式）。这个mod出现只是时间问题，因为许多人想在Windows上尝试AFL并将其应用于没有源代码的应用程序。目前，这个工具正在积极改进，无论相对过时的AFL代码库（撰写本文时为2.43b），它都有助于发现多个漏洞（CVE-2016-7212，CVE-2017-0073，CVE- 2017-0190，CVE-2017-11816）。Google Zero Project团队和MSRC漏洞与缓解团队的专家正在参与此项目，因此我们希望进一步发展。开发人员使用动态检测（基于DynamoRIO）而不是编译时间检测，这显着减慢了分析软件的执行速度，但是产生的开销（加倍）与二进制模式下的经典AFL相当。他们还解决了快速启动过程的问题，称其为持续模糊测试模式; 他们选择模糊函数（通过文件内部的偏移量或导出表中存在的函数名称）并对其进行检测，以便可以在循环中调用它，从而启动多个输入数据样本而无需重新启动过程。一个文章最近出版，描述了作者如何使用WinAFL在大约50天内发现大约50个漏洞。在发布之前很短暂，英特尔PT模式已被添加到WinAFL; 在这里可以找到detalis 。

高级读者可以注意到除了Frida之外，所有流行的插桩框架都有修改。唯一提到Frida与AFL的使用是在«Chizpurfle: A Gray-Box Android Fuzzer for Vendor Service Customizations»中找到的。带Frida的AFL版本非常有用，因为Frida支持多种RISC架构。

许多研究也期待由Capstone，Unicorn和Keystone的创建者发布DBI Scopio框架。基于这个框架，作者已经创建了一个模糊器（Darko），据他们说，成功地使用它来模糊嵌入式设备。有关这方面的更多信息，请参阅«Digging Deep: Finding 0days in Embedded Systems with Code Coverage Guided Fuzzing»。

基于处理器硬件功能的修改

当涉及到处理器硬件功能支持的AFL修改时，首先，它允许模糊内核代码，其次 - 它允许在没有源代码的情况下更快速地对应用程序进行模糊测试。

当然，谈到处理器硬件功能，我们最感兴趣的是Intel PT （处理器跟踪）。它可从第6代处理器开始（大约自2015年起）。因此，为了能够使用下面列出的模糊器，您需要一个支持Intel PT的处理器。

WinAFL-IntelPT - 第三方WinAFL修改，使用Intel PT而不是DynamoRIO。

kAFL - 是一个学术项目，旨在解决内核的独立于操作系统的模糊测试的覆盖引导问题。问题是通过使用管理程序和英特尔PT来解决。有关它的更多信息可以在白皮书[«kAFL: Hardware-Assisted Feedback Fuzzing for OS Kernels»。

ptfuzzer- 它采用了Intel Processor Trace硬件部件来收集程序执行的路径信息，改进了原来AFL通过编译插桩方式获取程序执行路径信息的方法。和AFL相比，硬件收集的路径信息更加丰富，同时可以直接对目标程序进行fuzz，无需源码支持。

结论

如您所见，AFL修改的领域正在积极发展。不过，还有实验和创意解决方案的空间; 您可以创建一个有用且有趣的新修改。

感谢您阅读我们，并祝您好运！

原文链接： https://habr.com/en/company/dsec/blog/449134

翻译原文转自看雪论坛翻译板块 https://bbs.pediy.com/thread-251051.htm