软件漏洞自动利用研究进展

和亮，苏璞睿
中科院软件所

关键词：软件漏洞利用，自动化，符号执行，污点分析

1、 引言

软件漏洞发掘是当前的热点问题。尽管模糊测试技术帮助我们解决了程序漏洞的自动发现问题，并行模糊测试平台已经可以高效的发现大量的程序错误，但无论是防御者还是攻击者，都更关心这些程序漏洞或错误是否可能被利用。如何快速分析、评估漏洞的可利用性是当前漏洞发掘与分析的关键问题之一[1-2]。传统软件漏洞利用主要以手工方式构造，该过程不仅需要具备较为全面的系统底层知识（包括文件格式，汇编代码，操作系统内部机理以及处理器架构等），同时还需要对漏洞机理深入、细致的分析，才可能构造成功的利用。在软件功能越来越复杂，漏洞越来越多样化的趋势下，传统利用方式已难以应对上述挑战。

目前，随着程序分析技术的不断发展，尤其是污点分析、符号执行等技术成功运用在软件动态分析以及软件漏洞挖掘等多个领域后，研究者开始尝试利用这些技术来进行高效的软件漏洞利用自动构造。图 1 展示了已有工作及其关键描述，接下来我们将对各项工作展开详细的介绍。

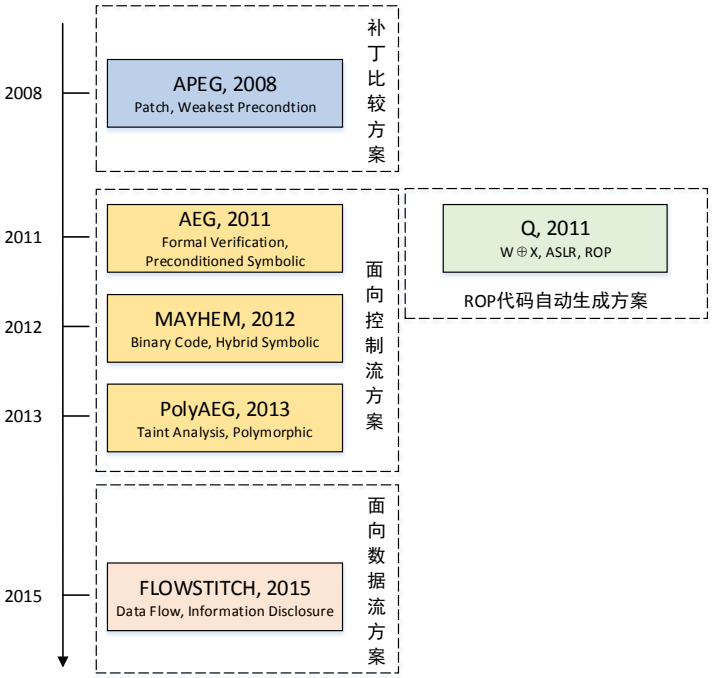

图 1 .软件漏洞自动利用相关工作

2、 基于补丁比较的自动利用方案

在 2008 年的 IEEE S&P 会议上，D.Brumley 等人首次提出了基于二进制补丁比较的漏洞利用自动生成方法 APEG[3]。其核心思路是基于以下的假设条件，即补丁程序中增加了对触发原程序崩溃的过滤条件。因此，只要能够找到补丁程序中添加过滤条件的位置，同时构造

不满足过滤条件的“违规”输入，即可认为是原始程序的一个可利用的输入候选项。根据其具体介绍内容可知，该工作主要分为三个步骤：首先，利用二进制差异比较工具（例如 BinDiff 与 EBDS 等）找到补丁存在的位置，即补丁程序的检测点；其次，找出不满足补丁程序检测点的输入数据作为原始程序的利用候选项；最后，利用污点传播等监控方法筛选所有能够对原始程序造成溢出或者控制流劫持等崩溃发生的有效利用。根据对微软所发布的多个补丁程序的实验结果表明，该方法具有较强的可靠性和实用性。

APEG 是对漏洞利用自动化构建的首次尝试，虽然核心思想较为简单，但由于其具有很强的可操作性，因此也得到了其他研究人员的普遍认可。然而 APEG 的局限性主要体现在两个方面：首先，该方法无法处理补丁程序中不添加过滤判断的情况，例如，为了修复缓冲区溢出而增加缓冲区长度的补丁程序；其次，从实际利用效果来看，所构造的利用类型主要属于拒绝服务，即只能造成原程序的崩溃，而无法造成直接的控制流劫持。

3、 面向控制流的自动利用方案

3. 1、基于源码的自动利用方案

为了克服 APEG 对于补丁依赖以及无法构造控制流劫持的缺陷，在 2011 年的 NDSS 会议上，T.Avgerinos 等人首次提出了一种有效的漏洞自动挖掘和利用方法 AEG[4]。该方法的核心思想是借助程序验证技术找出能够满足使得程序进入非安全状态且可被利用的输入，其中非安全状态包括内存越界写、恶意的格式化字符串等，可被利用主要是指程序的 EIP 被任意操纵。其具体流程为：首先，在预处理阶段，利用 GNU C 编译器构建二进制程序以及通过 LLVM 生成所需的字节码信息；其次，在实际分析的过程中，AEG 首先通过源码分析以及符号执行找出存在错误的位置，并通过路径约束条件生成相应的输入；之后，AEG 利用动态分析方法提取程序运行时的各类信息，例如栈上脆弱缓冲区的地址、脆弱函数的返回地址以及在漏洞触发之前的其他环境数据等；随后，综合漏洞利用约束条件以及动态运行时环境信息，最终构建可利用样本。通过对 14 组真实程序漏洞的自动利用实验，证明了该方法的可靠性和有效性。

AEG 集成了优化后的符号执行和动态指令插装技术，实现了从软件漏洞自动挖掘到软件漏洞自动利用的整个过程，并且生成的利用样本直接具备控制流劫持能力，是第一个真正意义上的面向控制流漏洞利用的自动化构建方案。该方案的局限性主要体现在：首先，该方案需要依赖源代码进行程序错误搜索；其次，所构造的利用样本主要是面向栈溢出或者字符串格式化漏洞，并且利用样本受限于编译器和动态运行环境等因素。

3. 2、基于二进制的自动利用方案

为了摆脱对源代码的依赖以及保证系统适用场景的广泛性，S.K.Cha 等人在 2012 年的 IEEE S&P 会议上提出了基于二进制程序的漏洞利用自动生成方法 Mayhem[5]。该方法通过综合利用在线式符号执行的速度优势和离线式符号执行的内存低消耗特点，并通过基于索引的内存模型构建，进而实现较为实用化的漏洞挖掘与利用自动生成方法。其具体流程如下：首先，通过构建两个并行的符号执行子系统，具体执行和符号执行子系统；其次，对于具体执行子系统，通过引入污点传播技术，寻找程序执行过程中，由用户输入所能控制的所有 jmp 指令或者 call 指令，并将其作为 bug 候选项交给符号执行子系统；之后，符号执行系统将所

有接收到的污点指令转化为中间指令，并进行执行路径约束构建和可利用约束构建；最后，符号执行系统通过约束求解器来寻找满足路径可达条件和漏洞可利用条件的利用样本。

在实际进行符号执行的过程中，为了保证效率问题，Mayhem 系统使用了一种基于索引的内存模型用来优化处理符号化内存的加载问题，进而使其成为一种高使用性的漏洞自动利用方案。目前 Mayhem 的局限性主要集中在以下三个方面：首先，系统只能建模部分系统或者库函数，因此无法高效处理大型程序；其次，系统无法处理多线程交互问题，例如消息传递和共享内存问题；最后，由于使用了污点传播方法，同样具有漏传和误传等典型问题。

3.3、多样化自动利用方案

由于高质量、多样性的漏洞利用样本对漏洞危害评估具有重要的意义，因此在 2013 年 SecureComm 会议上，M.H.Wang 等人针对控制流劫持类漏洞提出了一套多样性利用样本自动生成方法 PolyAEG[6]。该方法的核心思想是通过动态污点分析找出程序所有控制流劫持点，通过构建不同的控制流转移模式完成漏洞利用样本的多样性构造。其具体流程为：首先，通过扩展硬件虚拟化平台 QEMU 实现程序动态监控并提取程序执行的相关信息；其次，在动态获取信息的基础上构建指令级污点传播流图 iTPG 以及全局污点状态记录 GTSR，并以此为基础获取程序中所有可能的控制流劫持点、可利用的跳板指令以及可存放攻击代码的污点内存区域等内容；最后，通过构建不同的跳转指令链以及不同污点内存区域的攻击代码，并通过路径约束条件求解生成具有多样性的利用样本集合。通过对 8 个实际漏洞样本的实验结果来看，该方案针对单个控制流劫持漏洞最多生成了 4724 个利用样本。

PolyAEG 针对控制流劫持类漏洞实现了一套完整的自动化漏洞利用的多样性构造，对漏洞危害评估提供了有效的支持。但是，该方案的局限性主要体现在以下两个方面：一是该方案在面对数据执行保护机制时存在一定的局限性；另一个是本方案在利用构造过程中只考虑了依靠程序自身或其他类库中已有的指令，并没有考虑借助动态生成代码。

3.4、ROP 代码自动生成方案

为了解决数据执行保护和地址随机化给控制流劫持类漏洞利用带来的困扰，在 2011 年的 USENIX Security 会议上，E.J.Schwartz 等人实现了一套面向高可靠性漏洞利用的 ROP 代码自动生成方法 Q[7]。其核心思想是收集目标程序中的 Gadget 并通过面向 Gadget 的编程语言自动构建 ROP。具体的流程主要如下：首先，向 Q 提供未随机化的脆弱程序或者其他二进制库，并由 Q 找出具备特定功能的 Gadget 集合；其次，利用 Q 提供的编程语言 QooL 实现满足特定语义功能的目标代码，并通过 Q 将目标代码编译为面向 Gadget 的指令序列；随后，通过利用已获取的 Gadget 集合填充上一步得到的指令序列，从而形成最终的 ROP 代码。通过对 9 个真实软件漏洞的实验，可以看到在开启数据执行保护和地址随机化功能后，通过 Q 仍然可以保证这些漏洞利用代码稳定执行。

Q 方案证明了在含有少量未随机化代码的系统中仍可以有效自动构建 ROP 代码，进而强化了面向控制流劫持类漏洞利用在真实环境下的攻击效果。Q 方案本身的局限性主要体现在：首先，Q 方案未考虑自动构建不含 ret 指令的 ROP；其次，Q 方案仅从实际应用效果出发，没有考虑满足图灵完备性。

4、面向数据流的自动化利用方案

在数据执行保护、地址随机化以及控制流完整性防护手段大范围部署的情况下，大多数

攻击者已经从面向控制流劫持的漏洞利用攻击转向面向数据流利用的攻击。正是在这样的背景下，H.Hu 等人在 2015 年 USENIX Security 会议上首次提出了一种面向数据流利用的自动化构造方法 FlowStitch[8]。该方法的核心思想是在不改变程序控制流的前提下，利用已知的内存错误直接或者间接篡改程序原有数据流中关键位置上的变量，进而完成利用的自动化构造。根据文章介绍，其具体过程分为以下几个步骤：首先，以含有内存错误的程序、触发内存错误的输入以及特殊的正常输入作为整个自动利用系统的三个前提条件，其中“特殊的正常输入”是指在程序错误发生之前，其执行路径必须和触发内存错误的执行路径相同；其次，分别以错误输入和正常输入获取其对应的错误执行记录和正常执行记录，并以此为基础分别进一步提取内存错误影响的范围以及正常数据流中的敏感数据；最后，通过比对错误执行记录和正常执行记录的方式确定内存错误影响范围中可能涉及到的敏感数据，最终筛选所有可能被篡改的敏感数据并完成面向数据流的利用自动化构建过程。通过在 8 个真实漏洞样本上的实验结果可以看出，FlowStitch 自动构建的 19 个利用样本不仅可以绕过数据执行以及细粒度控制流完整性等防护手段，并且其中 10 个利用样本还可以在开启地址随机化的环境下成功执行。

FlowStich 是第一个面向数据流的漏洞自动利用方案，尽管其构造的利用样本无法直接运行任意的恶意代码，但由于可以泄露目标主机上的敏感数据，因此仍然具有很强的实用价值。从文章的描述细节来看，该方案的局限性主要体现在：首先，面向数据流的利用是以程序中存在已知的内存错误为前提；其次，在构建利用过程中，不仅需要错误输入对应的执行记录，同时还需要构造相应的正常输入以及正常执行路径。

5、 总结与展望

为了快速分析和判定由模糊测试技术生成的大量软件漏洞的可利用性问题，研究人员相继提出了一系列高效的漏洞利用自动化构建方案，包括补丁比较方案、面向控制流方案以及面向数据流方案等。这些方案的实施不仅能够帮助我们从大量程序漏洞中快速甄别出具有高危险性的漏洞，同时也一定程度上帮助我们降低遭受高危漏洞攻击的可能。

尽管目前软件漏洞自动利用工作已经取得了初步成果，但随着软件的复杂性增加、控制流完整性检测等防御手段的部署以及软件漏洞类型的发展与变化，都对漏洞的可利用性评估带来了挑战。因此，对于软件漏洞利用工作，我们还需要进一步去探索和研究，提出更加高效和可靠的自动化方案。

作者

和　亮，中国科学院软件研究所 助理研究员，主要研究领域：程序动态分析，软件漏洞分析与评估等。

苏璞睿，中国科学院软件研究所 研究员、博士生导师，主要研究领域：恶意代码动态分析，智能移动终端应用安全，软件漏洞分析与评估等。

参考文献

[1.] C.Miller, J.Caballero, N.M.Johnson, M.G.Kang, S.McCamant, P.Poosankam and D.Song. Crash Analysis with BitBlaze. *BlackHat*, 2010.

[2.] S.Heelan and D.Kroening. Automatic Generation of Control Flow Hijacking Exploits for Software Vulnerabilities. *MSc Computer Science Dissertation, University of Oxford*, 2009

[3.] D.Brumley, P.Poosankam, D.song and J.Zheng. Automatic Patch-Based Exploit Generation is Possible: Techniques and Implications. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2008

[4.] T.Avgerinos, S.K.Cha, B.L.T.Tao and D.Brumley. AEG: Automatic Exploit Generation. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2011

[5.] S.K.Cha, T.Avgerinos, A.Rebert and D.Brumley. Unleashing MAYHEM on Binary Code. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2012

[6.] M.H.Wang, P.R.Su, Q.Li, L.Y.Ying, Y.Yang and D.G.Feng. Automatic Polymorphic Exploit Generation for Software Vulnerabilities. In *Proceedings of International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2013

[7.] E.J.Schwartz, T.Avgerinos and D.Brumley. Q: Exploit hardening made easy. In *Proceedings of the USENIX Security Symposium*, 2011

[8.] H.Hu, Z.L.Chua, S.Adrian, P.Saxena and Z.K.Liang. Automatic Generation of Data-Oriented Exploits. In *Proceedings of the USENIX Security Symposium*, 2015