恶意软件动态分析云平台①

徐 欣, 程绍银, 蒋 凡

(中国科学技术大学 计算机科学与技术学院, 合肥 230027)

摘 要: 传统的杀毒软件基于特征码识别的方式有效但具有一定的局限性. 使用沙箱动态分析的方法能通过目标软件的行为特征对其恶意属性进行判断, 可以同时达到检测恶意软件和帮助分析人员快速分析恶意软件的目的. 为了提高沙箱平台分析的易用性和高效性, 本文设计并实现了一个恶意软件动态分析云平台, 通过分布式的沙箱控制机制, 保证沙箱的分析能力以及可扩展性, 并可通过对目标软件的分析结果来判断其是否属于恶意软件. 实验表明, 设计的云沙箱系统能够有效和高效的检测出恶意软件的恶意行为.

关键词: 恶意软件; 动态分析; 沙箱; 云平台

Malware Dynamic Analysis Cloud

XU Xin, CHENG Shao-Yin, JIANG Fan

(School of Computer Science and Technology, University of Science and Techonology of China, Hefei 230027, China)

Abstract: Although traditional anti-virus software is fast, feature code based detection mode will be a restriction. Dynamic analysis based on sandbox is better at finding malware via software behavior. It can not only find malware for client but also help malware analyzer working faster. In purpose of increasing usability and efficiency of analysis based on sandbox, a malware dynamic analysis cloud is designed and implemented. Through distributed control mechanism, we can ensure the system's capability and flexibility. A malicious software can be distinguished based on the result of the analysis. Experiments show the system can effectively detect malware behavior with high efficiency.

Key words: malware; dynamic analysis; sandbox; cloud

在 APT(Advanced Persistent Thread, 即高级持续性威胁)的网络渗透行动中, 木马作为控制被渗透目标系统的工具发挥重要的作用; 例如暴风、冲击波等蠕虫病毒则对全世界大量信息系统造成了极大的破坏; 漏洞利用攻击的载体例如携带有漏洞利用程序的 flash、office 以及 pdf 文档等则对日常工作和生活软件的使用者造成了极大的网络安全威胁. 这些安全威胁都可以被称为恶意软件威胁. 传统的安全软件使用特征匹配的方式发现恶意软件, 在 VirusTotal 网站[1]上, 提供了世界上众多杀毒软件杀毒引擎的查杀结果, 可以看出每个杀毒软件对于一个文件的判断并不一致, 同样一个可执行文件, 一些杀毒软件认为是恶意软件, 另一部分杀毒软件并不报告. 并且即使都报告为是恶意软件的情况, 每款杀毒软件对该软件的恶意归类也不尽相同. 造成这一结果的原因是由于杀毒软件通常使用特征码查杀的方式对特定类型的恶意软件进行扫描和识别, 特征码则是由专门的恶意软件分析专家通过逆向分析或者使用一些特定的工具实现提取. 但是, 一个特征码往往只能识别一种恶意软件的某些副本, 如果恶意软件的制作者通过技术分析手段得知了杀毒软件查杀使用的特征码, 那么其同样可以通过技术手段隐匿该特征, 从而避免杀毒软件的识别. 由于恶意软件的制作者众多, 恶意软件样本不断更新, 也造成了恶意软件特征库中的特征呈现爆炸性的增长, 使得查杀识别率更低. 为了克服静态扫描特征码方法的缺陷, 人们想到了使用动态分析软件行为的方法去查杀

① 收稿时间:2015-06-08;收到修改稿时间:2015-07-30

识别恶意软件.

1 恶意软件及检测研究现状

恶意软件包括不同种类, 依据其目的和实现的差异, 通常分为 4 类.

1.1 特洛伊木马

特洛伊木马是一种通过特殊方法植入目标系统内部, 进行各种非授权操作的攻击工具. 传统特洛伊木马通常由控制端和被控端组成. 被控端又被称为服务端, 源于早期特洛伊木马采用正向连接的客户机服务器模式, 并且由被控端充当服务器, 监听特定端口并等待控制端的连接, 其本质是远程控制软件 RAT, 例如使用广泛的远程控制商业软件 Radmin[2]. 由于 NAT 的广泛应用, 早期的正向连接式木马进化为反向连接式木马, 即被控端充当客户机去反向连接充当服务器的控制端的特定端口, 控制端即通过此连接控制被控端. 为了躲避网络防护系统的监控, 特洛伊木马在反向连接基础上衍生了不同的分支[3].

特洛伊木马在系统中能够长期存在, 能够随着操作系统的启动而启动, 一般情况下, 其很少会破坏目标系统的正常运行以期不被信息系统的使用者察觉. 特洛伊木马会有很多异于正常软件的行为来帮助其隐藏自身以及达到其盗取信息的目的. 例如, 特洛伊木马执行过程中会在注册表的特定项目中写入信息以使其可以随操作系统启动, 文献[4]中总结了多种常见的 Windows 下木马跟随系统的启动方式, 包括改写注册表键值, 改写文件执行关联选项, 和使用系统服务启动木马等. 特洛伊木马具有显然是为了隐藏自身的行为特征, 如将自身的代码注入合法进程的内存空间中, 并使用远程过程调用的方式将功能代码“寄生”其中, 使自身伪装成合法进程以增加隐蔽性. 为了盗取信息系统的隐私信息, 木马往往会使用挂钩事件处理函数的方法设置键盘记录, 并通过读取关键系统进程的内存空间以及存有用户隐私数据的文件来获取关键的认证以及口令信息.

1.2 蠕虫病毒

可以自我复制和传播的蠕虫病毒主要目的是破坏系统. 在一定程度上, 蠕虫病毒和特洛伊木马程序有着一些共同的特性, 例如随系统启动以及隐藏自身的文件和进程等. 文献[5]详细分析了蠕虫病毒使用的各种传播和攻击手段, 将蠕虫分为只传播没有功能、网络远程控制功能、转发恶意邮件功能、发动 DOS 攻击功能、收集数据功能、广告功能、数据破坏功能等类型. 可以看出, 蠕虫病毒也有修改关键文件、批量修改被感染文件、发起扫描等在内的很多种行为.

1.3 恶意文档

漏洞利用在 APT 攻击的过程中常被用于攻击关键目标系统. 在文档中嵌入漏洞触发和利用代码是常见的形式. 在漏洞利用的实现过程中, 会有一些与正常操作行为不同的程序执行特点. 漏洞利用代码的最终目的通常是在系统中植入木马程序. 无论是否启动一个新的目标程序进程, 漏洞攻击代码的目的都是为了控制目标系统, 那么其一定会执行例如系统命令行程序或者将木马文件写入系统目录并执行的操作, 所以, 使用动态行为分析的方法也能够有效的检测出含有漏洞攻击代码的文档文件. 文献[6]中研究人员提出了使用动态污点传播的方式检测攻击代码的方法.

1.4 恶意软件的检测

对于恶意软件的检测作为对恶意软件防护中的重要一环, 一直以来, 研究人员对于恶意软件及其恶意行为的检测提出了很多技术和方法. 文献[7]中研究人员基于 PE 文件的静态结构特征提出了一种检测恶意软件的方法, 文[8]的方法则是通过敏感 API 的调用来检测恶意软件的恶意行为, 进一步的, 文献[9]通过监控被感染主机的通信行为与用户或者操作系统的哪些行为相关来判断发生通信的软件是否为恶意软件. 文献[10]则将二进制的软件转换为中间代码, 并通过中间代码的语义信息来判断目标软件是否有恶意行为.

2 沙箱技术

沙箱是一个在程序执行过后能够完全抹去程序的执行痕迹并恢复到原始状态的执行容器. 沙箱实现方式有多种, 各有优缺点. 基于虚拟机的沙箱仿真程度高, 能够构建出基本上真实的运行环境, 安全性最高, 只要虚拟机控制系统不存在漏洞就很难实现虚拟机逃逸, 其缺点就是在沙箱中要运行整个操作系统才能够执行恶意软件的程序, 造成系统开销较大, 沙箱的还原速度较慢, 同一台物理设备上能够同时运行的虚拟机数量有限. 基于进程行为监控和访问控制的沙箱[11]则速度较快, 由于其在运行被测程序的过程中将其运行的根目录做了重定向并限制了控制权限, 并且在系统中监控被测试程序的访问行为, 缺点是这种沙箱只

是基于访问行为的监控和还原, 并没有实现完全的虚拟化设计, 造成有一些程序的行为可能与在真实系统中运行过程有所偏差, 并且更容易出现造成沙箱逃逸的漏洞. 为了能够最大程度的保证程序动态分析的完整性, 我们在本文的系统设计中采用了基于虚拟化的沙箱作为基础分析容器. 在虚拟化沙箱的基础上, 本文中采用了分布式的方法实现沙箱的部署和控制, 设计并实现了一个基于虚拟化沙箱检测的恶意软件分析云环境. 采用云的形式设计该系统有以下优点, 首先, 通过用户友好的Web界面以及Web Service接口, 可以为用户提供方便的服务, 并对用户隐藏系统实现的具体细节, 其次, 云系统的实现基于对沙箱系统的分布式管理, 与单一沙箱的动态分析相比, 其具有更高的并发分析效率. 在针对一些长期执行的程序需要较长时间监控分析的过程中, 会占用一个沙箱容器, 对于单一沙箱来说, 之后的任务需要顺序等待, 极大影响的分析效率, 但是对于分布式管理的沙箱来说, 可以利用其他空闲沙箱资源, 动态的分配给后续分析任务, 保证分析任务高效的进行. 图 1 说明了采用分布式控制的沙箱具有更好的可扩展性, 并且在一个时间段内具有动态分析更多目标程序的能力.

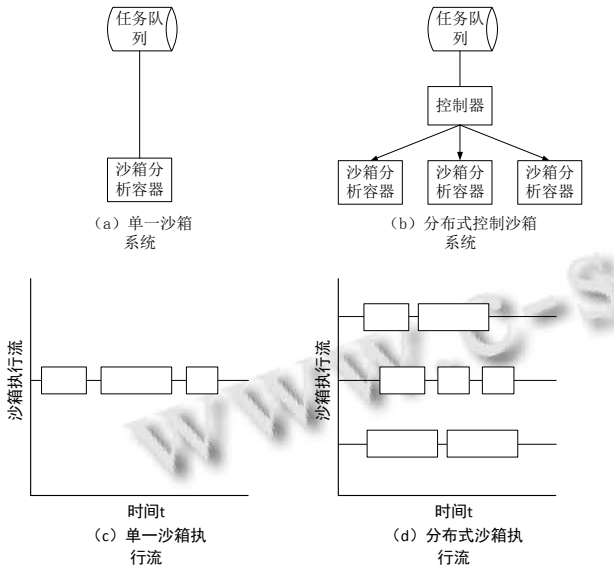

图 1　单一沙箱系统与分布式沙箱系统的比较

3　软件行为监视模块的设计和实现

针对各种恶意软件的原理以及行为特点可以设计出有效的监视机制. 在虚拟化沙箱中, 在系统运行时加载行为监视模块, 对目标程序的行为进行全面的监视, 并依据其行为特征判别其是否为恶意软件. 图 2 是软件行为监视模块的设计.

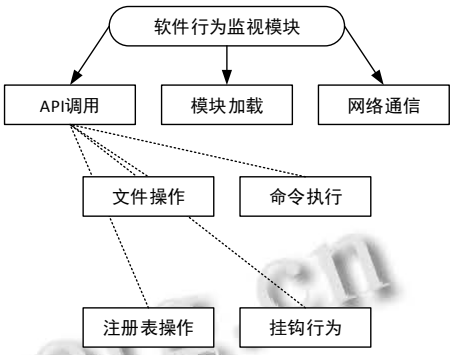

图 2　软件行为监视模块

(1) API 调用监视模块, 即监视在程序的执行过程中调用了哪些 API 函数. 在文献[12]中, 作者提出使用 API 函数监视技术实现恶意软件取证的方法. 这一方法也是对其他例如文件与注册表行为监视的基础, 由于一个软件有关于操作系统的行为依赖调用系统 API 函数实现, 所以依据一个软件对系统 API 函数的调用序列, 可以形成一个能够区别于其他软件的行为特征. 也就是说可以使用 API 序列识别的方法来检测恶意软件[13]. 不同于之前的方法, 本文中使用 Pin[14]插桩工具在程序执行中对所有常用的系统库函数进行动态插桩, 当发生对应的函数调用时, 程序就会执行到插桩函数, 可监视程序调用了哪些系统 API 以及调用发生的时序. 并且能够避免挂钩方法容易被恶意软件检测和对抗的问题.

(2) 文件操作监视模块, 即监视目标软件对文件的新建、修改、复制、删除等操作, 通过监视目标软件的文件操作行为, 得到目标软件是否对重要系统文件进行了读写的行为, 或是否释放了其他可执行文件以及是否有盗取用户隐私的行为. 文件操作监控的实现依赖 API 监视进行, 在 API 监视的过程中, 通过对于 CreateFile, CloseFile, ReadFile, WriteFile 等 20 多个函数的调用进行监视, 能够了解目标程序对文件对象的各种操作. 本文总结了常用的文件读写过程中的 API 调用序列, 实现对文件操作的监控.

(3) 命令执行监视模块, 即监视程序是否执行了某些系统应用程序或者结合文件监视判断其是否运行了其释放至文件系统中的其他应用程序. 通过判断目标软件执行的命令, 能够判别其是否是恶意软件以及

运行了哪些命令对系统信息进行探测. 命令执行的监视主要通过 Pin 插装相应的调用, 其中包括 WinExec、CreateProcess、system 等函数都是命令执行相关的 API 函数.

(4) 注册表操作监视模块, 为了实现随系统的启动, 以及篡改其他软件配置信息, 甚至制作克隆账号, 注册表操作是重要的实现途径. 采用 API 调用监控的方式实现注册表操作的监视是十分有效的. 在注册表操作监视模块中, 针对 40 个 Windows 注册表操作 API 进行插桩, 实现注册表操作的监视.

(5) 挂钩行为监视模块, 恶意软件为了实现监控系统的一些行为, 例如键盘输入以及鼠标点击, 需要使用钩子技术挂钩重要的消息和事件处理函数, 将自身线程注入到系统线程中也会用到钩子技术, 检测挂钩行为对于监视这些行为提供了有效的依据. 检测钩子通过插桩 SetWindowsHookEx 函数实现, 另外真对 CreateRomoteThread 族函数以及 APC 调用族函数也实现了监视.

(6) 模块加载监视模块, 对于程序加载了哪些模块的监视, 在一定程度上也体现了程序实现了哪些功能, 如程序加载了操作系统的哪些动态连接库, 尤其是在目标程序不仅是一个单独可执行程序的情况下, 可能带有多个可加载模块, 那么通过加载模块的监视可以得知程序的全貌. 程序加载的监控通过 CreateToolHelp32Snapshot 函数、Module32First 函数以及 Module32Next 函数进行枚举.

(7) 网络通信监视模块, 通过对目标程序所在系统网卡中通过流量的监视, 能够监视其是否对特定网络域名或者 IP 地址发起了特定协议的连接. 文[15]中提出, 可以使用协议特征结合机器学习判别的方式进行恶意软件协议行为的识别. 本文利用了 libpcap 的网络过滤驱动, 提取其与虚拟网络控制器之间虚拟网卡的数据并通过特征匹配的方法进行分析, 实现网络数据的获取和分析, 获取其发送了哪些常用协议的数据.

4 云沙箱检测系统设计

本文设计的恶意软件动态分析云平台主要由三部分组成: Web 服务器、样本分析控制器以及分布式的虚拟化沙箱服务器. 用户通过浏览器可以通过 Web 服务器提交未知软件的分析请求, 另外通过服务器上设计的 Web Service 接口, 也可以通过脚本上传恶意软件并读取分析报告.

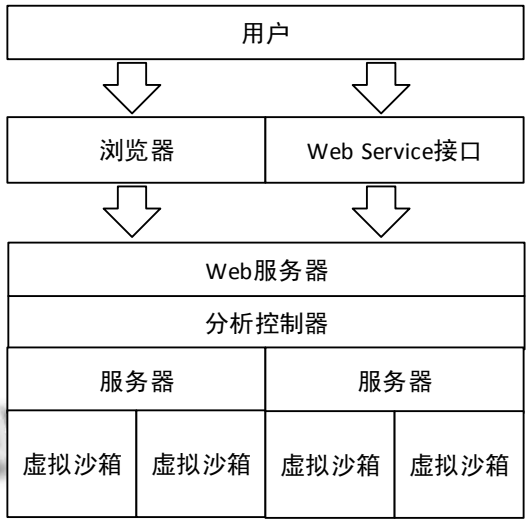

图 3 云平台框架设计

用来作为目标文件测试容器的虚拟沙箱内可安装 WindowsXP, Windows7 以及 Linux 操作系统, 在指定沙箱中安装文档阅读软件以测试恶意文档文件, 并关闭操作系统中诸如自动更新、防火墙、NetBIOS 等服务和固定 IP 地址以避免产生额外的网络数据以及妨碍目标软件的正常运行和向外通信连接的建立.

当 Web 服务器接收到用户的目标软件分析请求之后, 则调用分析控制器的相关功能模块, 并将待分析的目标文件提交给分析控制器处理, 分析控制器程序记录了当前正在进行分析的所有任务, 并且管理和调度所有空闲资源. 当接收到一个分析请求之后, 其工作流程如下:

1) 判断文件的 MD5 值与数据库中有无匹配项目, 若有则直接返回结果, 若没有则转入步骤 2;

2) 判断是否有空闲的测试沙箱资源, 若没有空闲资源则将任务放入等待队列;

3) 若有空闲资源则在列表中查找第一个拥有空闲资源的服务器以及在服务器上的空闲虚拟沙箱;

4) 向对应服务器发出命令启动选中的虚拟沙箱;

5) 连接沙箱中运行的服务端程序, 上传分析工具以及队列中第一个待分析的目的软件;

6) 启动接收程序并开放端口等待沙箱中的分析软件回传分析结果;

7) 将分析结果返回给 Web 服务器.

服务器端程序伪代码如图 4 所示.

```
while(!IsEmpty(taskQueue))
{
    // 获取队列中的任务
    curTask = taskQueue.pop_front();
    hash = MD5Hash(curTask.malwarebin);
    // 判断 hash 是否在数据库中
    if (hash not in database)
    {
        workContainer = selectEmptyContainer();
        analyzer = selectAnalyzer(curTask);
        uploadAnalyzer(analyzer,
workContainer);
        StartAnalysisThread(curTask,
workContainer)
    }
    else
    {
```

图 4　服务器端程序伪代码

沙箱启动后, 控制器连接沙箱中的服务程序, 并由服务程序完成如下工作:

1) 接收控制器传来的本文第 3 部分设计实现的分析程序以及待分析目标软件;

2) 运行分析程序和目标软件, 并由分析程序记录目标软件执行过程中的各种行为;

3) 将分析程序生成的结果以 XML 格式返回给控制器.

对于特定格式文件, 例如 office 文档或者 pdf 文档, 则被放入安装有相应软件的虚拟机中进行测试. 基于控制器的分布式动态分析沙箱的设计, 保证了沙箱分析环境的可扩展性, 如果计算资源不能满足要求, 需要增加新的计算机设备并在其上部署其能够承载的沙箱即可实现沙箱规模的扩展. 云沙箱的设计则为用户提供了更友好的访问接口, 通过 Web 方式或者编写脚本提交给 Web 服务器即可等待沙箱分析完毕返回结果.

5　沙箱系统实现与测试

为了检测云沙箱分析系统的分析能力, 需要选取可控环境进行测试, 我们选取了带有线程注入功能的 Poison Ivy Rat 木马、Bifrost 木马以及 Gh0st 木马样本进行分析, 由于实验中主要针对常见的 Windows 平台下恶意软件样本, 我们部署了 Windows 测试沙箱容器. 并通过配置木马样本的功能, 使其带有各种恶意行为. 首先采用手工分析的方法获取其恶意行为的种类, 并采用沙箱分析进行分析效果验证, 分析结果如图 5.

恶意行为	手工检测			沙箱分析检测		
	PIRat	Bifrost	Gh0st	PIRat	Bifrost	Gh0st
生成可执行文件	是	是	否	是	是	否
启动系统进程	是	是	否	是	是	否
注入系统进程	是	是	否	是	是	否
设置 Run 键自启动	是	是	否	是	是	否
设置组件自启动	是	否	否	是	否	否
伪造系统服务	否	否	是	否	否	是
设置互斥量	是	是	是	是	是	是
DNS 查询	是	是	是	是	是	是
通信行为	是	是	是	是	是	是

图 5　恶意软件沙箱分析结果

在沙箱的执行效率上, 由于采用多个沙箱并行执行分析的运行方式, 当发生需要长时间占用沙箱的任务时, 分布式的沙箱控制系统不需要等待特定沙箱返回再部署下一个任务, 而是直接选取空闲的沙箱资源分配给需要执行的任务. 图 6 比较了在执行 60 个分析任务的情况下单一沙箱分析与多个沙箱并行分析消耗时间的情况, 途中虚线和实现分别表示了多沙箱并行分析与单一沙箱分析情况下的任务完成数量与实践的对应情况.

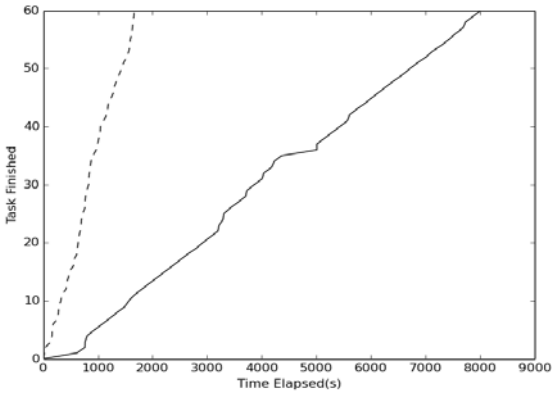

图 6　单一沙箱和多沙箱执行效率对比

可见, 基于云的分布式沙箱系统能够极大的提高恶意软件的分析效率.

通过分析实验样本可知, 本文的沙箱分析工具能

够分析恶意软件的大部分恶意行为, 并且由于采用了云沙箱的设计, 能够针对目标恶意软件反馈给用户更加直观的数据, 并且由于后端采用了多服务器承载虚拟沙箱的设计, 能够提高恶意软件的分析效率, 有效加快分析进程和缩短用户的等待时间.

6 结语

本文设计并实现了一个恶意软件动态分析沙箱云平台, 对比面向专业分析人员的单个沙箱具有更好的分析效率和用户友好性. 沙箱分析在针对各种恶意软件的分析工作中具有极其重要作用, 通过沙箱动态分析, 不仅可以得出本文中所检测的在 API 层面的行为特征, 结合 Pin 等动态插桩分析工具的特性, 还能够自动分析程序运行过程中的指令行为特征, 从而辅助恶意软件分析人员的工作. 但是, 当前的沙箱机制从本质上只是实现了恶意软件的分析, 识别恶意行为. 但是例如性能分析等正常软件也会存在类似恶意软件的行为特点, 不能排除误报的情况. 如何确定目标软件是恶意软件还要结合具体场景进行分析. 考虑更多情况, 具有更智能判断方法的恶意软件分析专家系统是下一步的研究重点.

参考文献

1 VirusTotal, https://www.virustotal.com/.

2 Radmin, http://www.radmin.cn/.

3 方滨兴,崔翔,王威.僵尸网络综述.计算机研究与发展, 2011,8:1315–1331.

4 曹光辉,鄂旭,杜颖,马雨时.一种全新的木马自启动方案.渤海大学学报(自然科学版),2008,4:390–393.

5 Weaver N, Paxson V, Staniford S, et al. A taxonomy of computer worms. Proc. of the 2003 ACM workshop on Rapid malcode. ACM, 2003: 11–18.

6 Newsome J, Song D. Dynamic taint analysis: Automatic detection and generation of software exploit attacks. NDSS. 2005.

7 白金荣,王俊峰,赵宗渠.基于 PE 静态结构特征的恶意软件检测方法.计算机科学,2013,1:122–126.

8 白金荣,王俊峰,赵宗渠,刘达富.基于敏感 Native API 的恶意软件检测方法.计算机工程,2012,13:9–12.

9 张永斌,张艳宁.基于主机行为特征的恶意软件检测方法.计算机应用研究,2014,2:547–550,554.

10 杨洪深,赵宗渠,王俊峰.基于中间代码的恶意软件检测技术研究.四川大学学报(自然科学版),2013,6:1216–1222.

11 Berman A, Bourassa V, Selberg E. TRON: Process-specific file protection for the UNIX operating system. USENIX. 1995: 165–175.

12 Peisert S, Bishop M, Karin S, et al. Analysis of computer intrusions using sequences of function calls. IEEE Trans. on Dependable and Secure Computing, 2007, 4(2): 137–150.

13 Shankarapani MK, Ramamoorthy S, Movva RS, et al. Malware detection using assembly and API call sequences. Journal in Computer Virology, 2011, 7(2): 107–119.

14 Luk CK, Cohn R, Muth R, et al. Pin: building customized program analysis tools with dynamic instrumentation. Acm Sigplan Notices. ACM, 2005, 40(6): 190–200.

15 Tegeler F, Fu X, Vigna G, et al. Botfinder: Finding bots in network traffic without deep packet inspection. Proc. of the 8th International Conference on Emerging Networking Experiments and Technologies. ACM. 2012. 349–360.