Hacking Without Humans

小组成员：

张嘉豪 1801210918　　曹辉 1801210810

王源 1801210670　　龚秋嫦 1801210514

Cyber Grand Challenge

- CGC(Cyber□□ Grand Challenge) 为美国DARPA(国防高等研究计划局)举办的自动网络攻防竞赛。
- 一般CTF（Capture the Flag）竞赛需要大量人力参与分析，而实际网络攻防也面对相同问题，同样需要花费大量人力来应对新的攻击，造成从攻击出现到防御机制中间有段时间，也就是我们常听到的0day所造成的安全问题。
- CGC旨在建立即时自动化的网络防御系统，并且能够快速且大量的应对新的攻击手法，应对大量的网络攻击，并降低攻击从出现到防御之间的时间差。
- Score = Availability × Security × Evaluation

Cyber Grand Challenge

- 现今，机器的运算速度及基本理论已经有长途的进步，许多软体测试及分析的技术与基本理论都日趋完善，虽然许多技术要应用在大型环境中还有一段差距值得研究员们努力，但已经可以一定程度的自动化找出并修补软件程序漏洞。
- 以下是DARPA文件中有提及的技术：
- Dynamic Analysis / Static Analysis
- Symbolic Execution / Constraint Solving
- Data Flow Tracking / Fuzz Testing

The Mayhem Cyber Reasoning System

- Mayhem是一个自动化的推理系统（CRS）
- Mayhem Offense
- Mayhem Defense
- Mayhem Strategy

Mayhem Offense

- Mayhem的攻击能力基于生成测试用例以发现目标服务中的缺陷
- CRS通过每秒生成和分析数千个输入来测试目标服务，每个输入都可能触发一个漏洞

- 漏洞挖掘
  - Symbolic Execution
  - Fuzzing
- 利用检测出的漏洞（生成 Proof of Vulnerability）
  - Automatic Exploit Generation

Mayhem Offense

---Mining Vulnerabilities

- 比赛中服务总数为82（7 CRSs）
- Symbolic Execution 找出崩溃点：62（57 + 5）
- Fuzzing 找出的崩溃点：

  62（57 + 5）
- 两种漏洞发掘方法共发现的漏洞数：67 / 82

漏洞发掘方法	共同发现漏洞数	单独发现漏洞数
Symbolic Execution	57	5
Fuzzing	57	5

Mayhem Offense

---Exploit

- Automatic Exploit Generation（AEG）
- 目的：生成PoVs
  - 黑盒AEG：通过输入突变数据推断输入字符串和崩溃状态之间的关系，然后生成一个导致目标服务崩溃的输入（演示PoV所需的条件）
  - 白盒AEG：符号化地执行崩溃路径并试图满足利用漏洞所需的约束条件，如果约束条件可满足，Mayhem用这种解决方案生成PoV
    - 这些约束可能取决于服务发送的数据。 例如，服务可能希望客户端重放它发送的nonce或解决简单的数学CAPTCHA。 在这些场景中，PoV可以通过查询嵌入式SMT求解器来动态生成输入。

Mayhem Defense

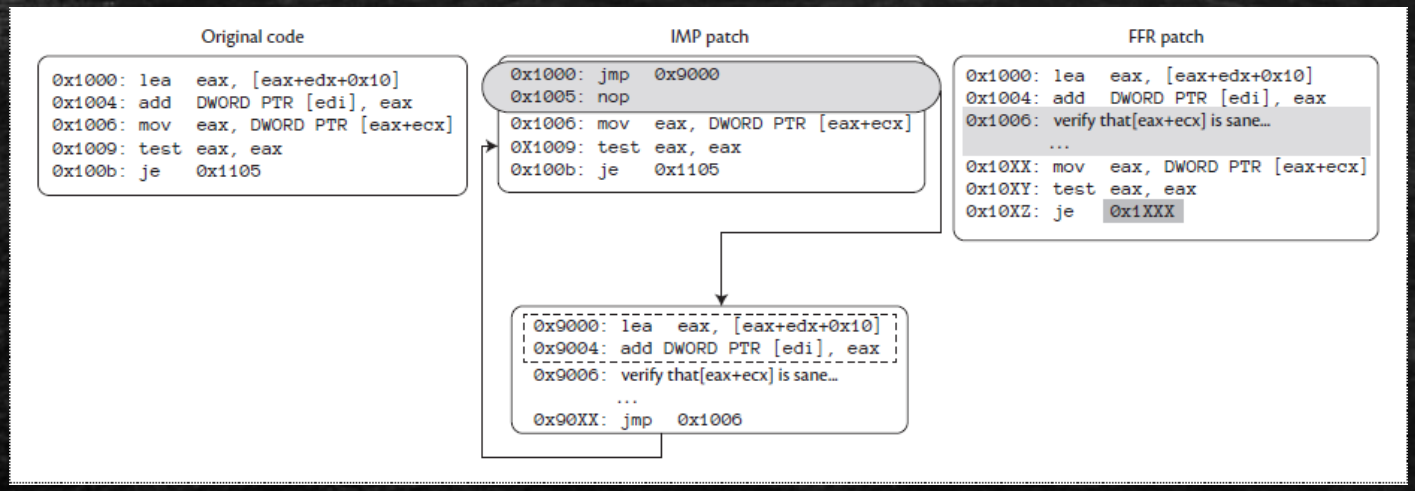
- Mayhem只有在测试用例导致服务崩溃时才能确定一个软件缺陷

- 生成测试用例并进行测试
- 发现软件缺陷
- 插入内省检查找到漏洞（这些检查验证运行时属性，对于正确运行的程序为检查为真，对于正在被利用的CGC服务为假，此时导致服务安全种植，从而阻止攻击成功完成）
- 插入补丁修补漏洞

Mayhem Defense

---Mayhem Binary Patching Techniques

- Full – Function Rewriting (FFR)
- Injection Multipatching (IMP)



Mayhem Strategy

- Patch Scoring and Selection
- Patching Strategies
- Offense Strategy

Mechanical Phish

概述

脆弱性分析过程从人工分析转向自动化方法，DARPA网络攻防竞赛的产物是网络推理系统，如机械网络钓鱼系统，通过分析代码发现漏洞并证明漏洞的存在，然后修复易受攻击的软件。

这篇文章不仅提供了相关的技术细节，而且讨论了网络推理系统的创建以及网络自治的教训。

Mechanical Phish

网络攻防竞赛

可行性研究：符号执行过程中路径的优先次序、静态分析精度的提高、环境建模，DECREE操作系统只包含7个系统调用。

网络攻防资格赛（CQE），团队：Shellphish。CRS雏形，构建了一个结合模糊技术的漏洞检测引擎和一个修复漏洞引擎。

网络攻防竞赛决赛（CFE），CRS相互对峙，需要精心策划攻击（不仅仅是崩溃），生成开销很小的高级补丁，窃取标志，并适应对手的行为。绝对没有人为干预。

Mechanical Phish

网络推理系统的产生

从原型研究到可靠软件，问题是没有足够好的软件开发实践。在CGC的过程中，Shellphish采取了诸如持续的实践整合，问题跟踪，甚至尝试代码冻结。

人员组织：角色齐全，各司其职

利用非时态资源，使用现代集群管理软件设计了一个极其灵活的基础结构，其中根据需要自动分配资源。

难题：封闭的基础设施，对手的不确定性，性能打分不确定，二进制格式的不确定性

Mechanical Phish

机械钓鱼系统

基础设施

every team 64 dedicated servers：

- 1,280 physical cores,
- 16 TB of memory, and
- 64 TB of disk space.

为了充分利用硬件，分离了系统。在独立的组件中运行，并在一个完全隔离的环境中运行每个组件。

Mechanical Phish

通过TI提供的API执行与游戏的每一次交互，Ambassador和Network是负责与游戏API交互的组件，并将收集到的数据存储到中央数据库Farnsworth。

Meistar和Scriba是CRS的大脑。第一个负责阅读从Farnsworth的游戏状态和调度任务。第二个负责根据内部评估和游戏API提供的反馈，决定提交哪些补丁和漏洞。

艰苦的程序分析工作由Workers完成，它是一组执行不同任务的组件，如bug查找、修补、开发和结果评估。

Mechanical Phish

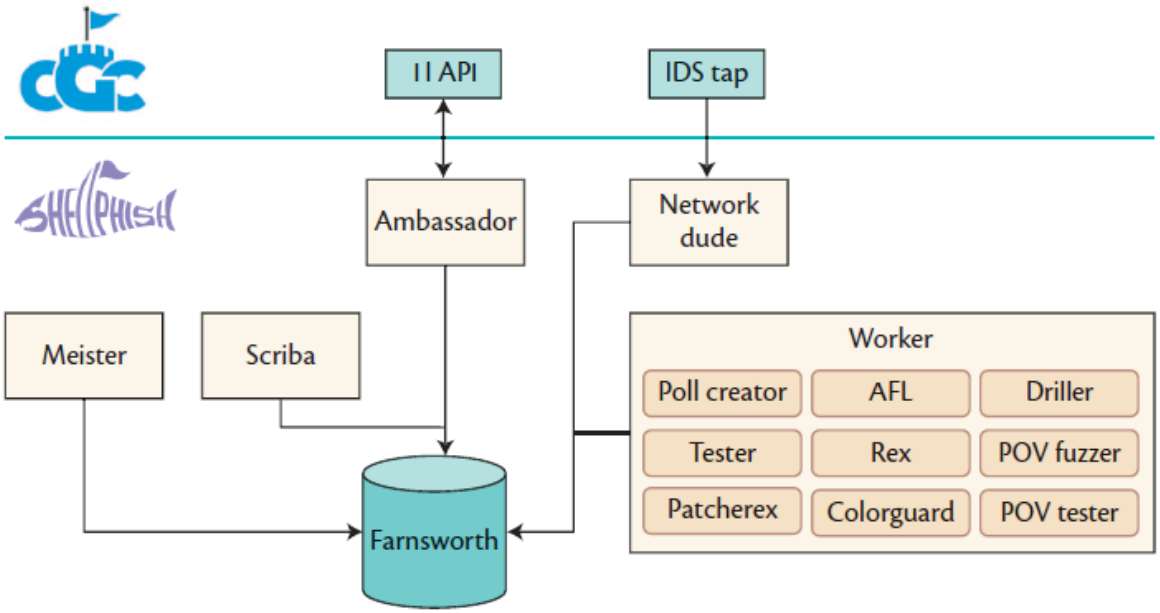


Figure 2. The architecture of Mechanical Phish.

Mechanical Phish

发现错误

第一个发现目标程序中的崩溃。第二步处理这些崩溃，并试图找出如何修改它们，以产生控制程序的漏洞。

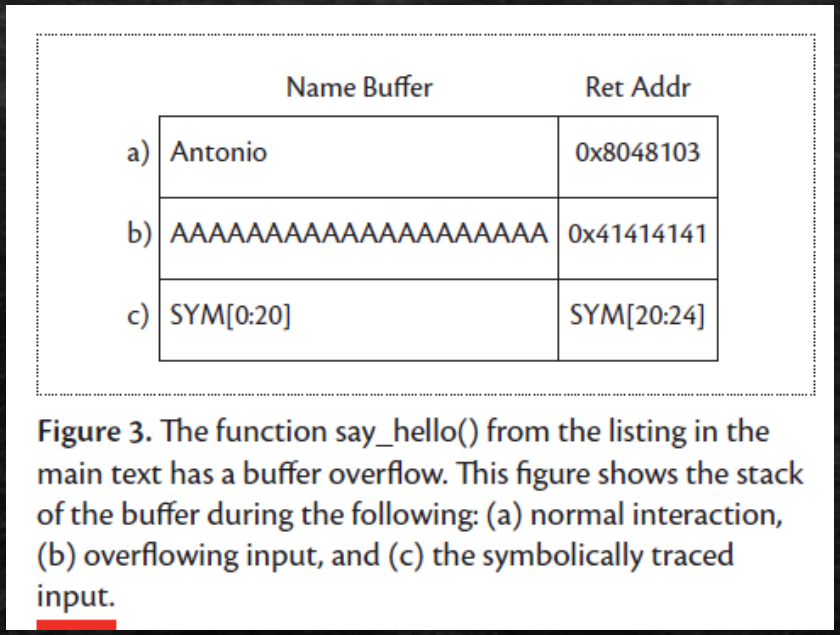
使用AFL，一个众所周知的和高度成功的进化模糊体，作为CRS的bug发现组件的核心。作者发现它在样本程序中难以满足特定和困难的检查，为了处理这些问题，开发了一种将模糊和符号执行相结合的工具。

开发利用

在崩溃输入之后，我们象征性地跟踪程序，并且当我们到达崩溃时，根据需要修改输入以进行利用。

我们想出了一个崩溃类型的列表，我们可以利用和方法化来利用这些特定的崩溃。

Mechanical Phish



	Name Buffer	Ret Addr
a)	Antonio	0x8048103
b)	AAAAAAAAAAAAAAAAAAAA	0x41414141
c)	SYM[0:20]	SYM[20:24]

Figure 3. The function say_hello() from the listing in the main text has a buffer overflow. This figure shows the stack of the buffer during the following: (a) normal interaction, (b) overflowing input, and (c) the symbolically traced input.

Mechanical Phish

机械钓鱼系统中的模糊组件可以容易地生成太长并溢出返回地址的输入，从而导致程序崩溃（图3b）。

它增加了符号跟踪过程中收集的方程的约束条件。

修补：

Patcherex遵循一种无目标的方法。

实现了二进制的修改，从而阻碍了已修补的二进制文件的静态和动态分析，使得自动分析非常困难（如果不是不可能的话）。

在开发Patcherex时，主要关心的不是降低原始二进制文件的功能和性能。

策略：二进制硬件化，反分析，二进制优化

Mechanical Phish

经验教训

团队合作

战胜其他团队的动力比研究截止日期或达到某种抽象结果的需要更强烈。

理解规则：总是修补和产生则修补的取舍

策略：产生修补，永不修补，No-op strategy

开发

二进制修补

基础设施

Mechanical Phish

总结

通过在大规模复杂漏洞分析过程中对人类进行编排，可以识别纯自动化手段无法识别的漏洞，从而对程序分析中最具挑战性的问题之一进行新的解释。

A Honeybug for Automated Cyber Reasoning Systems

- A Honeybug for Automated Cyber Reasoning Systems 与 Effects of a Honeypot on the Cyber Grand Challenge Final Event两篇文章对网络大挑战赛决赛中采用的一种特定策略有不同的看法
- A Honeybug for Automated Cyber Reasoning Systems中，作者（Rubeus系统的创建者）详细介绍了一种挫败对手预期分析技术的方法
- Effects of a Honeypot on the Cyber Grand Challenge Final Event从比赛裁判的角度提供了对这种方法的分析

A Honeybug for Automated Cyber Reasoning Systems

- 在网络挑战赛中，我们的网络推理系统用一个Honeybug修补二进制文件，诱使竞争对手的系统追逐Honeybug，而不是利用实际的漏洞。
- 现实世界中检测和抵消自动化的方法，可以用于阻止恶意行为者发现漏洞。
- Honeybug是一种以二进制文件形式存在的伪漏洞，目的是将CRS的注意力转移到合法的现有漏洞之上

A Honeybug for Automated Cyber Reasoning Systems

- Enabling Shenanigans——Honeybug策略的可能性

- 虽然参赛者二进制文件将完全暴露给对手，但其本身几乎无法了解评分系统的执行环境
- DECREE（DARPA实验网络安全研究评估环境）操作系统具有精心限制的七个系统调用
- DECREE禁止高保真定时信息，可用于检测不同的执行环境。
- 竞争对手是全自动机器， 这意味着没有人会仔细检查CPUID并识别其目的，并从我们的二进制文件中删除检查。

A Honeybug for Automated Cyber Reasoning Systems

- Fingerprinting the Scoring System

- 评分系统声称它使用的是英特尔i5-4258U处理器 - 笔记本电脑的通用处理器，报告的值与常用的仿真器和虚拟机管理程序不同
- 结论——评分系统要么经过大量修改，要么完全是自定义模拟器或管理程序。
- 办法——确定了12个CPUID值，这些值最能将评分系统与竞争对手可能使用的其他仿真器和虚拟机管理程序区分开来。

A Honeybug for Automated Cyber Reasoning Systems

- Choosing the Effect

- Honeybug是一种刻意种植的漏洞，实现了一个简单的堆栈缓冲区溢出，只需要12个字节的输入来覆盖返回地址，从而获得代码执行。
- 实现了让CRS可以轻松控制程序计数器和通用寄存器（类型1 PoV需要控制两者）。
- CRS还确保蜜罐补丁始终在我们的其他防御补丁之前执行，例如地址随机化（ASLR）或不可执行堆栈（NX）。

A Honeybug for Automated Cyber Reasoning Systems

- Results

- 在不了解竞争对手CRS的内部情况的条件下，仅能推测可能对其整体性能产生的不利影响。
- Honeybug改变整体评分的潜力，部分取决于CRS找到漏洞证据的能力。
- Honeybug与CGC语料库中的真实漏洞有很大不同，因此我们测试中成功的PoV意味着Honeybug正是对手的目标。

Effects of a Honeypot on the Cyber Grand Challenge Final Event

- CGC决赛包括“共识评估”，在共识评估中，每当CRS提交替换服务时，对手在提供服务之前提供替换服务的副本。
- 共识评估的一个目的是允许竞争对手确定是否减轻了漏洞或者仅通过重新配置服务来掩盖漏洞。 例如，移动堆栈地址可能会破坏为原始版本的服务制作的漏洞证明（PoV），但可以通过分析修订后的服务来补偿它。
- 共识评估还为竞争对手提供了一个机会，来确定补丁是否可能无意中引入了服务的新缺陷。

Effects of a Honeypot on the Cyber Grand Challenge Final Event

- Mechanics of the Honeypot

- 蜜罐使用IA32 cpuid指令来获取有关执行二进制文件的主机处理器的信息
- CGC DECREE 执行环境在CFE期间使用了一个管理程序，它捕获了cpuid指令并返回了一个不变的结果
- FE之前，每个CRS可以与模拟对手竞争，测试他们的API，并利用这个机会摸索底层硬件平台。
- 记录cpuid指令的结果，使己方的RCB可以将CGC赛事平台与其他平台区分开来。

Effects of a Honeypot on the Cyber Grand Challenge Final Event

- Rubeus’ s Deployment of RCBs

- Rubeus在不同轮次中部署了一些服务的多个版本。 在Rubeus取代的32项服务中，28项至少在一轮中包含Honeypot
- 最重要的是使用随机基址分配不可执行的新堆栈（NX）——为了使静态分析复杂化，并在启动函数中使用跳转而不是调用/返回。
- 对于多项服务，Rubeus部署的初始RCB后来又对服务进行了另一次更改——这些额外的RCB通常不包括任何防御性添加或补丁。相反，它们包括对初始数据的更改，这些更改控制了执行路径是继续执行Honeypot还是实际的服务。

Effects of a Honeypot on the Cyber Grand Challenge Final Event

- Effectiveness

- 只有一个对手完全避开了Honeypot。
- 两位对手为追求Honeypot而放弃了完美的PoV。
- CFE冠军多次攻击Honeypot，从未对含有Honeypot的RCB得到过分数。

Effects of a Honeypot on the Cyber Grand Challenge Final Event

- Methodology

- 每个Rubeus RCB都使用IDA Pro进行分析，以识别哪些包含了Honeypot，哪些没有被混淆，并且是二进制调用的第一个函数。
- 使用PoV识别以Honeypot为目标的团队，不能直接从比赛日志中获取。 这是使用分析重现工具发现的，该工具用于审查CFE中竞争对手部署软件的系统。
- 在这些试验中成功对抗Honeypot的每个PoV被认为在其他会话中也取得了成功。

The Past, Present, and Future of Cyberdyne
——关于Cyberdyne

- Cyberdyne是一个分布式系统
- 其可以发现第三方现成的二进制程序中的漏洞，在CGC中就各个方面展开竞争
- 从CGC起，Cyberdyne已成功应用于商业代码审核

The Past, Present, and Future of Cyberdyne
——构建Cyberdyne

- Cyberdyne集群中的每个节点由四个主要服务组成。
- 模糊器
- 符号执行器
- 混音器
- 思维模式。

- 自动化Cyberdyne的配置和部署，使得其能够更灵活地被应用

The Past, Present, and Future of Cyberdyne
——使用Cyberdyne

- 在CGC挑战中Cyberdyne依靠其准确高效表现优异

- 其未来：重新定位技术
- 在Linux应用程序上使用Cyberdyne
- 首次付费的自动安全审查

The Past, Present, and Future of Cyberdyne
——使用Cyberdyne

- 自动安全审查的困难与挑战包含：
- 构建目标软件
- 将软件移植到DECREE的最大挑战是识别所有构建依赖关系并修改所有构建和配置系统，以便为32位x86处理器发出代码
- 执行程序功能
- 执行程序功能是自动审计的第二大挑战。 最终用户应用程序只有一个入口点（即main），但会处理影响程序的多个命令行标志

The Past, Present, and Future of Cyberdyne
——关于未来的展望

- 自动代码审查是安全测试的未来
- 人工审核较为粗糙并且费用高昂。自动化则覆盖率高，费用低。
- 在安全测试上的变革将大大改变互联网核心基础设施的态势
- 自动审核改革已经成为新趋势，其终将能够提供可比、客观的安全指标。

谢谢！

张嘉豪、曹辉、王源、龚秋嫦