



鲁宏伟/luhw@hust.edu.cn

逆向工程分析技术

教学安排

- 本课程课堂讲授24学时，实验内容8学时。
- 成绩计算方法
平时成绩（40%）+ 分析报告（60%）

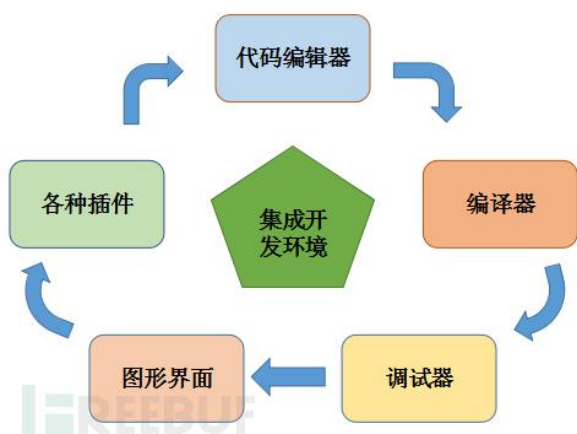
Ada • Lovelace



- 阿达·奥古斯塔，1815年生于伦敦，19世纪诗人拜伦的女儿，数学家。穿孔机程序创始人，建立了循环和子程序概念。为计算程序拟定“算法”，写作的第一份“程序设计流程图”，被珍视为“第一个给计算机写程序的人”。
- 阿达早逝，年仅36岁。在1843年发表的一篇论文里，阿达认为机器今后有可能被用来创作复杂的音乐、制图和在科学研究中运用，这在当时确是十分大胆的预见。以现在的观点看，阿达首先为计算拟定了“算法”，然后写作了一份“程序设计流程图”。

正向 · 逆向

传统的软件工程是从计算机的功能需求出发，将高层抽象的逻辑结构和设计思想通过计划和开发，生产出可实际运行的计算机软件，这个过程称为软件的“正向工程”。



从可运行的程序系统出发，运用解密、反汇编、系统分析以及程序理解等多种计算机技术，对软件的结构、流程、算法和代码等进行逆向拆解和分析，推导出软件产品的源代码、设计原理、结构、算法、处理过程、运行方法及相关文档等的过程，称为软件的“逆向工程” (Software Reverse Engineering)，又称软件“反向工程”。



中国实战化白帽人才能力 白皮书

中国实战化白帽人才能力白皮书

2021.1

基础安全工具

Web漏洞利用

社工钓鱼

Web漏洞挖掘

命令执行

开源情报收集

SQL注入

Web开发与编程

PHP

编写PoC或EXP等利用

智能硬件/IoT漏洞

高级安全工具

掌握CPU指令集

系统漏洞挖掘

内网渗透

团队协作 系统层漏洞利用与防御

编写PoC或EXP等高级利用

补天漏洞响应平台 · 奇安信安服团队 · 奇安信行业安全研究中心

实战化白帽人才概况

- 从行业分布来看，36.3%的白帽子来自于安全企业，34.9%的白帽子仍是学生，7.1%的白帽子来自政府机构事业单位。
- 从年龄分布来看，近半数的白帽子年龄在22岁及以下；35.7%的白帽子年龄在23-27岁。从学历来看，本科及以下学历超过9成。其中，本科学历占比36.3%，本科在读占比21.9%，还有24.5%的白帽子为大专学历。
- 从从业时间来看，进入安全行业1-3年的白帽子最多，占比51.2%；其次为拥有4-6年安全从业经验的白帽子，占比21.2%。
- 2020年实战化白帽人才基础能力中，平均每个白帽子掌握4个Web漏洞利用方式；会使用6个安全工具。

实战化白帽人才概况

- 2020年实战化白帽人才进阶能力中，平均每个白帽子掌握3个Web漏洞的挖掘能力，能够使用两种编程语言对Web开发和编程，更擅长使用社工库与鱼叉邮件进行社工钓鱼。
- 2020年实战化白帽人才高阶能力较弱，系统层漏洞利用与防护、系统层漏洞挖掘以及对不同系统编写PoC或EXP等高级利用能力掌握不够，相比之下，多数白帽子身份隐藏与内网渗透能力掌握较好。约74.0%的白帽子具有组队参加有关部门组织的实战攻防演习活动的经验，19.4%的白帽子表示自己能够胜任团队协作中的任意角色

实战化能力与传统能力的区别

- 实战化白帽人才能力，是指在政企机构实际运行的业务系统、生产系统上进行的实战攻防演习过程中，作为攻击方的白帽子所需要具备各种攻防能力的集合。
- 由于实战攻防演习是对真实黑客攻防过程进行模拟和再现，因此也要求白帽子在攻击过程中所使用的战术手法能够达到、甚至超过黑产组织或APT组织的攻击水平。
- 与传统的挖洞型白帽人才能力要求不同，实战化能力要求白帽子具备在真实的业务系统上，综合利用各种技术和非技术手段，进行动态实战攻防的能力。

实战化能力与传统能力的区别

- 针对业务系统，而非IT系统

- ✓ 传统的或一般的白帽子挖洞工作，主要都是针对各类IT信息系统本身或系统中的设备、协议等，如各类Web系统、操作系统、PC终端、IoT设备、工控设备协议、区块链协议等等。而实战攻防演习工作的核心目标，是发现和解决由网络安全问题引发的业务安全及生产安全问题，攻击过程也是针对实际运行中的业务系统或生产设备。
- ✓ 此外，传统的挖洞工作主要关注的是对单一系统的单点突破，实战攻防演习更多关注的则是多个系统并存的复杂体系，关注的是复杂体系在运行、管理过程中存在的安全隐患。对于多数大中型政企机构来说，内部存在几十个，甚至上百个不同的信息化系统的情况是非常普遍的。

实战化能力与传统能力的区别

- 挖洞只是辅助，攻击必须有效

- ✓ 单纯的挖洞工作，一般只需证明漏洞的存在，提交漏洞报告即可。但在实战化的业务环境中，存在漏洞不等于能够实现有效的攻击。
- ✓ 一方面，这是因为漏洞的实际触发可能依赖于诸多条件，这些条件在实际的业务环境中未必存在；
- ✓ 另一方面，即便漏洞是有效的，但如果攻击者只能实现单点突破，而无法达到预设的最终目标，同样不能完成有效的攻击

实战化能力与传统能力的区别

- 攻击是个过程，允许社工方法

- ✓ 对单一漏洞进行挖掘和利用，往往只能实现某个局部的技术目标。但事实上，在绝大多数的实战攻防演习过程中，攻击方需要连续突破多个外围系统或关联系统，才能最终达到计划中的攻击目标。也就是说，攻击者需要掌握一系列的漏洞，并能够对机构内部的IT架构、运行管理机制进行有效分析，才有可能找到有效的攻击路径，实现实战攻防演习环境下的有效攻击。事实上，在实战攻防演习过程中，攻击方可能需要连续数日，多人协作才能完成一整套攻击。
- ✓ 此外，一般的漏洞挖掘或渗透测试，是不允许使用社会工程学方法的。但在实战化环境下，社会工程学是必不可少的攻击手法，因为真实的攻击者一定会使用这项技能。事实上，以人为突破口，往往是实战攻防演习中攻击队的优先选择

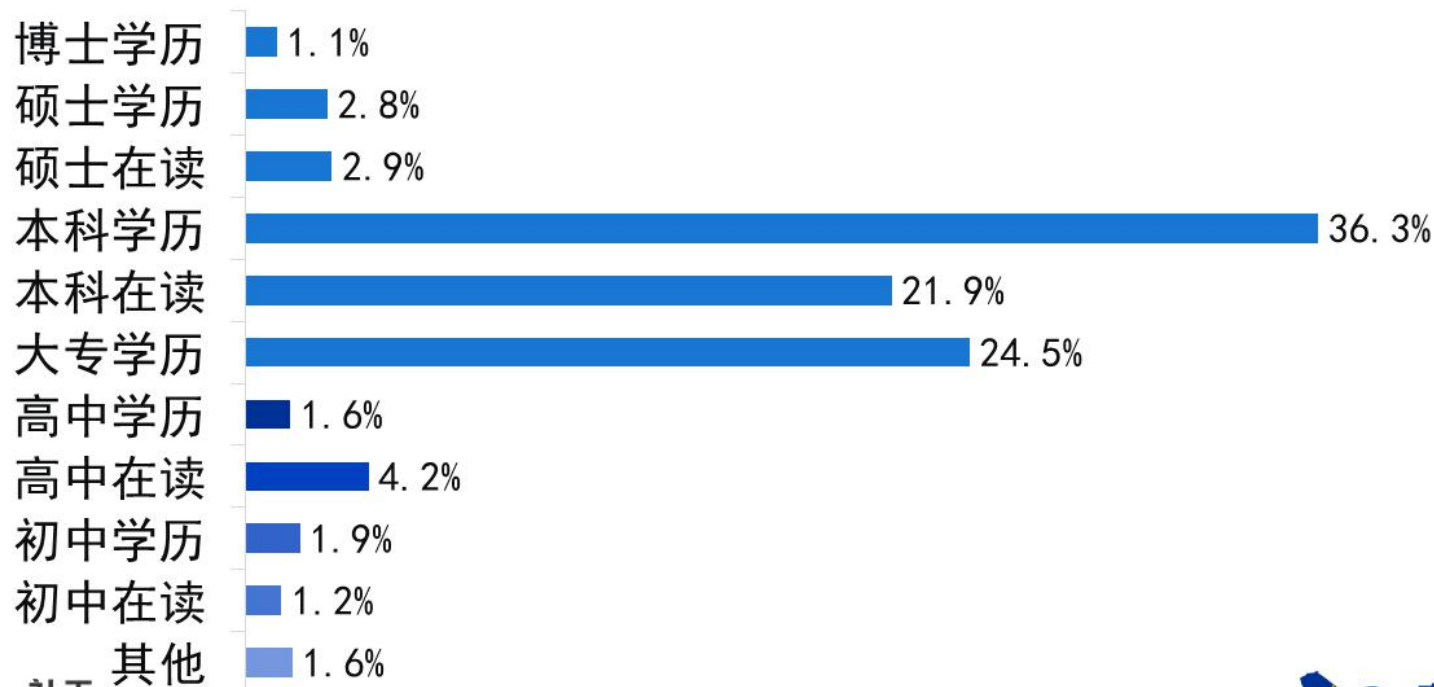
实战化能力与传统能力的区别

- 动态攻防环境，有人运行值守

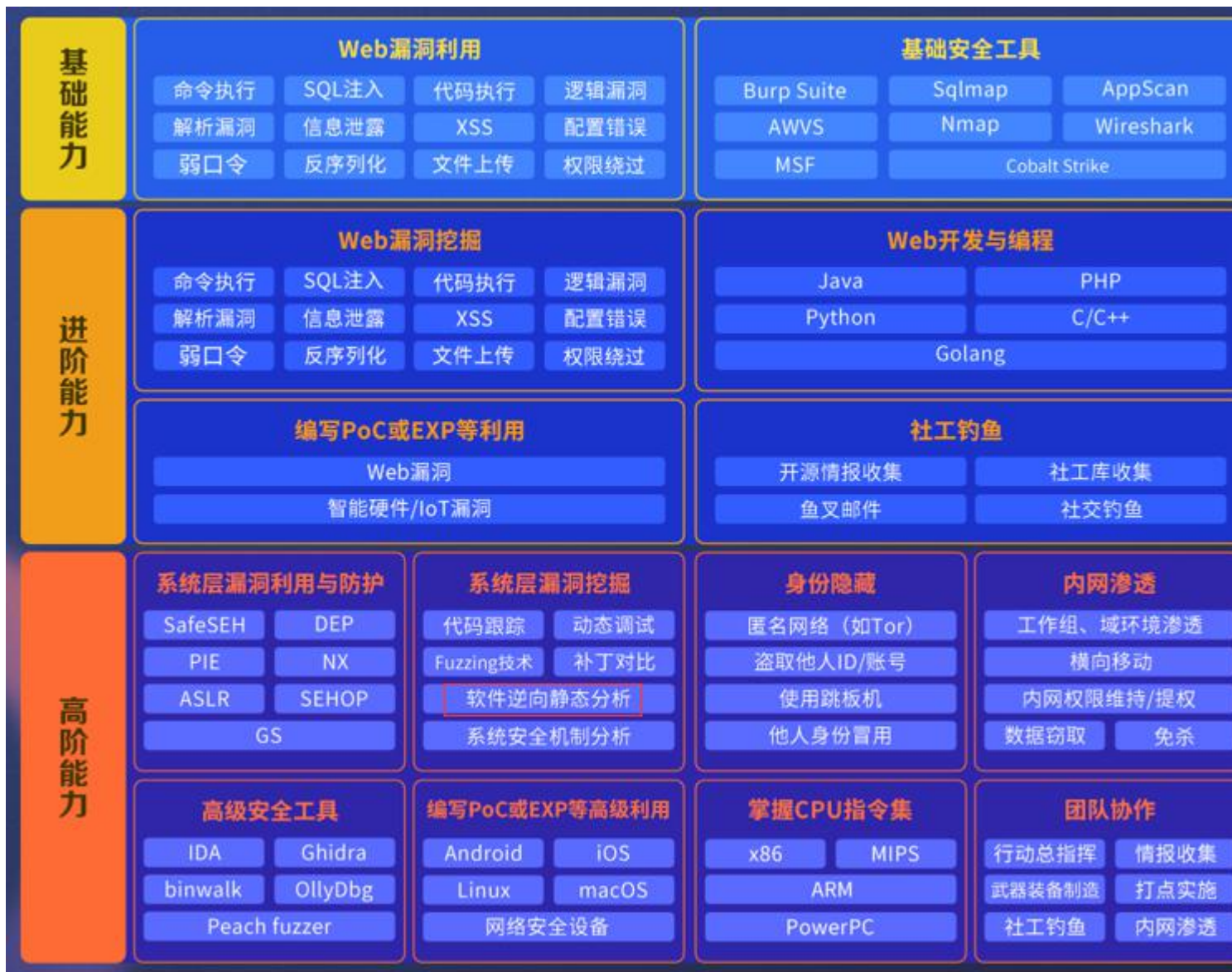
- ✓ 单纯的漏洞挖掘工作一般不需要考虑攻防过程，也就是不需要考虑防守方的参与。但在实战攻防演习过程，防守方实际上是有专业团队在进行安全运行维护和24小时值守工作的。攻击方一旦开始行动，就有可能被防守方发觉。而防守方一旦发现入侵行为，也会采取各种反制措施、诱捕行动，以及攻击溯源。
- ✓ 所以，实战化能力就要求白帽子必须掌握一定的身份隐藏技能，诸如匿名网络、免杀技术、权限维持等各种安全对抗技术


实战化白帽人才学历分布

实战化白帽人才学历分布



实战化白帽人才能力需求图谱





如何成为一个白帽

如何成为一个白帽--找好方向

- 方向有什么用？决定了你在碰到一门新知识时，是好好研究它还是随意了解下甚至不管它。
- 从能力本源讲，漏洞挖掘是白帽的基础能力。
- 衍生出来，白帽一般可以选择三个技术方向成长：
 - ✓ 漏洞挖掘专业方向的深度能力
 - ✓ 漏洞挖掘跨越领域的横向能力
 - ✓ 从攻击方转向防御

如何成为一个白帽--找好方向

- **漏洞挖掘专业方向的深度能力**
- 不管是web还是IoT，不管是java还是php，你若能研究透，对相关内容滚瓜烂熟，成为大牛
- 其中需要有对代码能力的熟练掌握，也需要对官方文档的详细解读，大多数漏洞不是靠fuzzy出来的，而是懂安全的人在阅读文档过程中发现了设计的缺陷，多掌握几个你就有核武器了

如何成为一个白帽--找好方向

- **挖掘跨越领域的横向能力**
- web做多了总会碰到一些App项目，有时也会碰到IoT项目，神挡杀神，见招拆招，都研究一遍之后你发现没有攻不破的系统，只有你还没开始研究的漏洞
- 跨领域往往会有普遍浅尝则止的问题，建议无论挖什么方向，都往那些能评成高危级别的漏洞看

如何成为一个白帽--找好方向

- **从攻击方转向防御**
- 有时候你会发现有些人根本不懂漏洞，你让他修个XSS，他屏蔽了alert参数。
- 如果作为防御方，有时你需要把队友的能力尽量低估，而对自己的要求无限放高，这个漏洞若你来修补应该是怎样，安全开发生命周期又该学些什么做些什么，研究多了你就是企业安全防御专家，大多数企业非常需要这样的角色

如何成为一个白帽--找对方法

- 方法是路径，决定了你用什么方式成长。技术成长的方式很多，但总的来说都是在积累新的方法、经验，总不能挖洞半年还是只会用那几个XSS的POC试来试去，
- 成长三要素：学习，实践，思考。
- 成长的几个路径：
 - ✓ 实战成长
 - ✓ 潜心学习
 - ✓ 经验的工具化

如何成为一个白帽--找对方法

- **实战成长**

- 找有授权的项目做，SRC（安全响应中心）的漏洞悬赏、乙方公司的服务项目、众测平台的项目、以及海外平台的项目
- 不要做私人的项目防止未授权，做更多新的授权项目或者挖SRC新的域名，碰到无法解决的问题要想办法解决掉，卡住了寻找探索新的思路去使用扩展，有一个**非常有效的办法就是去看那些公开的漏洞**，如何挖的，背后的漏洞原理是什么，寻找的过程是怎样

如何成为一个白帽--找对方法

- **潜心学习**

- 网络书籍和文章很多，各种大会的PPT更是满天飞，就像是找一本5000页的书花一段时间消化它，再做个实验实现它
- 若有一个老师设立一个一个问题去让你研究解决，提高得更快一些，现在的很多CTF也是这样的路线，只不过有些低质量的CTF题目纯粹是脑筋急转弯的体力活
- 很多人用闭关一段时间再出来实战再闭关再实战的方式，很不错

如何成为一个白帽--找对方法

- **经验的工具化**
- 让自己的字典更加强大，让POC集更多，让信息收集和利用的过程更加简易，让原来10分钟的事情现在10秒完成
- 网络安全发展这么多年，真正能称之为新的突破的知识，更新频次是非常低的，但是把所有现有的经验沉淀下来非常不容易，成型的Metasploit神器对你来说已经够用，但有些人还会在上面封装一层，对它进行优化，让工具更便捷、让渗透更高效
- 不管你擅长C、PHP、Python或者是Java，都需要思考工具化

逆向合法吗

逆向合法吗？

- 围绕着逆向工程合法性的争论已经有很多年了。这场争论通常是围绕着逆向工程对整个社会造成了怎样的社会影响和经济影响展开的。当然，估算这种影响主要取决于利用逆向工程做什么。
- 值得注意的是，我们绝不可能事先准确地预测出某个特定的逆向情景是否合法——这取决于多种因素。通常在将自己投入高风险的逆向项目之前，需要先寻求法律援助。

逆向合法吗？-互操作性

- 让两个程序通信并互操作绝不是一项简单的任务。即便是由同一组人开发的单个产品，在尝试实现与独立的组件互操作时，也常常会引发接口问题。软件接口是如此复杂，程序更是如此敏感，以至于这些程序在一开始很难正常运行。
- 当软件开发人员想要开发能与另外一个公司开发的组件通信的软件时，另一方必须暴露足够多的接口信息。
- 暴露软件的接口意味着其他软件开发人员能够开发出运行于这个平台之上的软件。这能增加平台的销售量，但是供应商也要提供他们自己的运行于此平台上的软件。
- 公开软件接口也会对供应商自己的应用程序造成新的竞争。与这种类型的逆向工程合法性的有关的内容包括版权法、商业机密保护和专利权等。

逆向合法吗？-竞争

- 当用于实现互操作性时，逆向工程简化了新技术的开发，显然，这有利于社会的进步。当利用逆向工程开发竞争产品时，情况就变得有些复杂
- 逆向工程的反对者通常认为逆向阻碍了创新，因为如果这些技术可以通过逆向工程轻易地从竞争对手那里"偷来"的话，那么新技术的开发人员在研究和开发新技术时只会投入极少的激情。
- 这就给我们带来了一个问题：**什么样的逆向才能构成以开发竞争产品为目的逆向工程？**
- 最极端的例子就是直接从竞争对手那里偷取代码片断，再把这些代码片段嵌入自己的代码中。这明显违反了版权法，而且很容易找到证据。
- 更为复杂的例子是对程序实施某种反编译过程，并以某种方式重新编译反编译结果，最终生成具有相同的功能但是从表面上看却是不同的二进制代码。

逆向合法吗？-竞争

- 当用于实现互操作性时，逆向工程简化了新技术的开发，显然，这有利于社会的进步。当利用逆向工程开发竞争产品时，情况就变得有些复杂
- 逆向工程的反对者通常认为逆向阻碍了创新，因为如果这些技术可以通过逆向工程轻易地从竞争对手那里“偷来”的话，那么新技术的开发人员在研究和开发新技术时只会投入极少的激情。
- 这就给我们带来了一个问题：什么样的逆向才能构成以开发竞争产品为目的逆向工程？
- 最极端的例子就是直接从竞争对手那里偷取代码片断，再把这些代码片段嵌入自己的代码中。这明显违反了版权法，而且很容易找到证据。
- 更为复杂的例子是对程序实施某种反编译过程，并以某种方式重新编译反编译结果，最终生成具有相同的功能但是从表面上看却是不同的二进制代码。

逆向合法吗？-漏洞利用

- 代码在开发过程中可能会存在一些安全漏洞，特别是涉及到一些商业利益的安全漏洞。
- 如果借助于逆向工程技术发现了这些漏洞，同时利用这些漏洞获取不当利益，这就很可能涉及到了法律问题。
- 利用游戏漏洞赚钱是属于盗窃罪，触犯《中华人民共和国刑法》第二百六十四条和第二百八十四条。非法所得一万元处三年以下有期徒刑、拘役或者管制，并处或者单处罚金。非法所得十万元处三年以上十年以下有期徒刑，并处罚金。
- 2009年，一名玩家曾通过编写外挂程序，篡改《梦幻西游》客户端软件，从而利用游戏漏洞刷取大量游戏币，并将刷取的游戏币置于第三方交易平台出售牟利，非法获利320万元，最后被判刑10年。

逆向合法吗？-规避的例外情况

- **互操作性**：在为了与有疑问的软件产品进行互操作的情况下，逆向技术是允许使用的。例如，如果某个程序出于拷贝保护的目而加了密，如果这个程序有问题，而逆向或破解是仅有的与其进行互操作的方式，软件开发人员就可以破解这个程序。
- **加密研究**：它只允许研究加密技术的人员规避加密产品中的版权保护技术。只有当保护技术妨碍了对加密技术的评估时，才允许使用规避手段
- **安全测试**：为了评估或提高计算机系统的安全性，人们可以逆向和破解版权保护软件。
- **教育机构和公共图书馆**：在购买之前出于对受版权保护作品评估的目的，这些机构可以规避版权保护技术。

逆向合法吗？-规避的例外情况

- **政府调查**：毫不意外，政府机构进行调查研究并不受版权保护的影响。
- **规章制度**：为了规范因特网上未成年人可访问的资料，可以对版权保护技术实施规避。所以，为了控制未成年人对因特网的使用，可以逆向一个允许不受任何监督和控制的因特网浏览的产品。
- **保护隐私**：收集和传递个人信息的产品可以被逆向，这类产品中所包含的任何保护技术可以规避。

逆向合法吗？-案例

- 2001年7月，FBI逮捕了一名俄罗斯的程序员Sklyarov，被控告违反了DMCA（美国数字千禧版权法）。
- 当 Sklyarov在墨西哥的一家称为 ElcomSoft 的软件公司工作时，曾对 Adobe eBook 文件格式做了逆向工程。他利用逆向工程收集的信息创建了一个称为 AdvancedeBook Processor 的程序，这个程序能破解这种 eBook文件（这些文件实际上是加密的.pdf文件，用来发布受版权保护的资料），这样这些文件就可以用任意的PDF阅读器阅读了。这种破解意味着原先关于浏览、打印或拷贝eBook 文件的限制都被绕过去了，这些文件也就变成了不受保护的了的。Adobe提出了一份控诉，声称开发和散播Advanced eBook Processor违反了DMCA，ElcomSoft和Sklyarov都应由政府提出诉讼。最终，ElcomSoft和 Sklyarov 都被宣告无罪，因为陪审团认为开发人员最初就没有意识到自己行为实际上是非法的。

逆向合法吗？-案例

8.2.2 软件使用

除非法律法规允许或腾讯书面许可，您不得从事下列行为：

- (1) 删除本软件及其副本上关于著作权的信息。
- (2) 对本软件进行反向工程、反向汇编、反向编译，或者以其他方式尝试发现本软件的源代码。
- (3) 对腾讯拥有知识产权的内容进行使用、出租、出借、复制、修改、链接、转载、汇编、发表、出版、建立镜像站点等。
- (4) 对本软件或者本软件运行过程中释放到任何终端内存中的数据、软件运行过程中客户端与服务器的交互数据，以及本软件运行所必需的系统数据，进行复制、修改、增加、删除、挂接运行或创作任何衍生作品，形式包括但不限于使用插件、外挂或非腾讯经授权的第三方工具/服务接入本软件和相关系统。
- (5) 通过修改或伪造软件运行中的指令、数据，增加、删减、变动软件的功能或运行效果，或者将用于上述用途的软件、方法进行运营或向公众传播，无论这些行为是否为商业目的。
- (6) 通过非腾讯开发、授权的第三方软件、插件、外挂、系统，登录或使用本软件和/或服务，或制作、发布、传播上述工具。
- (7) 自行、授权他人或利用第三方软件对本软件和/或服务及其组件、模块、数据等进行干扰。
- (8) 其他可能影响、干扰本软件和/或服务正常运行的行为。

目录 CONTENTS



1

目的

2

示例

3

基础

4

工具

目的

逆向工程可以让人们了解程序的结构以及程序的逻辑，因此，利用逆向工程可以深入洞察程序的运行过程

可以了解目标程序正在使用的系统函数的类型，也可以了解目标程序访问的文件

可以了解目标软件使用的协议以及目标软件是如何与网络的其他部分进行通信

目的

与安全相关的逆向分析

- 恶意软件的分析
- 逆向分析加密算法
- 数字版权管理
- 二进制代码审核

针对软件开发的逆向分析

- 与私有软件之间的交互
- 开发竞争产品
- 评估软件质量和鲁棒性

目的

- **逆向是一门艺术也是一种研究方法，能够让你弄清楚程序运行的本来面目**
 - ✓ 看别人的实现也可以用来提升自己的架构能力
 - ✓ 逆向能够找到一些诡异问题的根本原因
 - ✓ 最重要的是逆向能力加强后，对程序，代码，算法，数据结构，计算机体系的认识会深刻很多，做项目也会从容很多，不太再会遇到什么BUG搞不定
- **逆向是进行漏洞挖掘的重要途径**
- **不会逆向的程序员不是好程序员**

示例1-一段简单的程序

```
#include <stdio.h>
int main()
{
    int a;
    int n=32;
    int x=0;

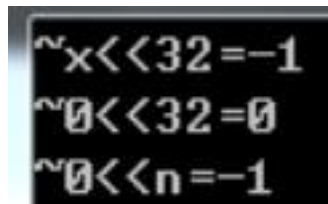
    a=~x<<32;
    printf("~x<<32=%d\n",a);

    a=~0<<32;
    printf("~0<<32=%d\n",a);

    a=~0<<n;
    printf("~0<<n=%d\n",a);

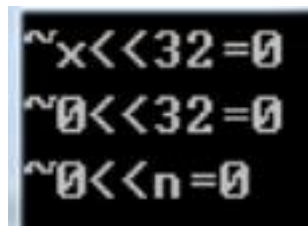
    return 0;
}
```

Win32 Debug模式下的结果:



~x<<32=-1
~0<<32=0
~0<<n=-1

Win32 Release模式下的结果:



~x<<32=0
~0<<32=0
~0<<n=0


```
a=~x<<32;
printf("~x<<32=%d\n",a);
```

```
a=~0<<32;
printf("~0<<32=%d\n",a);
```

```
a=~0<<n;
printf("~0<<n=%d\n",a);
```

```
_main_0      proc near      ; CODE:_main_0

var_4C      = byte ptr -4Ch
var_C       = dword ptr -0Ch
var_8       = dword ptr -8
var_4       = dword ptr -4

push        ebp
mov         ebp, esp
sub         esp, 4Ch
push        ebx
push        esi
push        edi
lea         edi, [ebp+var_4C]
mov         ecx, 13h
mov         eax, 0CCCCCCCCh
rep stosd

mov         [ebp+var_8], 20h
mov         [ebp+var_C], 0
mov         eax, [ebp+var_C]
not         eax
shl         eax, 20h
mov         [ebp+var_4], eax
mov         ecx, [ebp+var_4]
push        ecx
push        offset aX32D ; "~x<
call        _printf
add         esp, 8
xor         eax, eax
pop         edi
pop         esi
pop         ebx
add         esp, 4Ch
cmp         ebp, esp
call        __chkesp
mov         esp, ebp
pop         ebp
retn

_main_0      endp
```

初始化栈空间

var_C即x

```
proc near      ;_main_0

var_4C      = byte ptr -4Ch
var_C       = dword ptr -0Ch
var_8       = dword ptr -8
var_4       = dword ptr -4

push        ebp
mov         ebp, esp
sub         esp, 4Ch
push        ebx
push        esi
push        edi
lea         edi, [ebp+var_4C]
mov         ecx, 13h
mov         eax, 0CCCCCCCCh
rep stosd

mov         [ebp+var_8], 20h
mov         [ebp+var_C], 0
mov         [ebp+var_4], 0
mov         eax, [ebp+var_4]
push        eax
push        offset a032D ;
call        _printf
add         esp, 8
xor         eax, eax
pop         edi
pop         esi
pop         ebx
add         esp, 4Ch
cmp         ebp, esp
call        __chkesp
mov         esp, ebp
pop         ebp
retn

_main_0      endp
```

```
proc near      ;

var_4C      = byte ptr -4Ch
var_C       = dword ptr -0Ch
var_8       = dword ptr -8
var_4       = dword ptr -4

push        ebp
mov         ebp, esp
sub         esp, 4Ch
push        ebx
push        esi
push        edi
lea         edi, [ebp+var_4C]
mov         ecx, 13h
mov         eax, 0CCCCCCCCh
rep stosd

mov         [ebp+var_8], 20h
mov         [ebp+var_C], 0
or          eax, 0FFFFFFFFh
mov         ecx, [ebp+var_8]
shl         eax, cl
mov         [ebp+var_4], eax
mov         ecx, [ebp+var_4]
push        ecx
push        offset a0ND ;
call        _printf
add         esp, 8
xor         eax, eax
pop         edi
pop         esi
pop         ebx
add         esp, 4Ch
cmp         ebp, esp
call        __chkesp
mov         esp, ebp
pop         ebp
retn

_main_0      endp
```



```
a=~x<<32;
printf("~x<<32=%d\n",a);
```

```
_main      proc near
argc        = dword ptr  4
argv        = dword ptr  8
envp        = dword ptr  0Ch

    push    0
    push    offset aX32D
    call    _printf
    add     esp, 8
    xor     eax, eax
    retn
_main      endp
```

```
a=~0<<32;
printf("~0<<32=%d\n",a);
```

```
_main      proc near
argc        = dword ptr  4
argv        = dword ptr  8
envp        = dword ptr  0Ch

    push    0
    push    offset a032D
    call    _printf
    add     esp, 8
    xor     eax, eax
    retn
_main      endp
```

```
a=~0<<n;
printf("~0<<n=%d\n",a);
```

```
_main      proc near
argc        = dword ptr  4
argv        = dword ptr  8
envp        = dword ptr  0Ch

    push    0
    push    offset a0ND
    call    _printf
    add     esp, 8
    xor     eax, eax
    retn
_main      endp
```

示例2-一段简单的程序

```
#include <stdio.h>

int sum(int a[], unsigned len);

void main() {
    int a[3]={1,2,3};
    int rtn = 0;

    rtn = sum(a, 0);

    printf("rtn = %d\n", rtn);
}

int sum(int a[], unsigned len)
{
    int i, sum = 0;
    for(i=0; i<len-1; i++)
        sum += a[i];
    return sum;
}
```

Microsoft Visual Studio

example.exe 中的 0x003014cb 处有未经处理的异常: 0xC0000005: 读取位置 0x00290000 时发生访问冲突

```
text:0041149E      mov     [ebp+var_14], 0
text:004114A5      mov     [ebp+var_8], 0
text:004114AC      jmp     short loc_4114B7
text:004114AE      ; -----
text:004114AE      loc_4114AE:                                ; COD
text:004114AE      mov     eax, [ebp+var_8]
text:004114B1      add     eax, 1
text:004114B4      mov     [ebp+var_8], eax
text:004114B7      loc_4114B7:                                ; COD
text:004114B7      mov     eax, [ebp+arg_4]
text:004114BA      sub     eax, 1
text:004114BD      cmp     [ebp+var_8], eax
text:004114C0      ja      short loc_4114D3
text:004114C2      mov     eax, [ebp+var_8]
text:004114C5      mov     ecx, [ebp+arg_0]
text:004114C8      mov     edx, [ebp+var_14]
text:004114CB      add     edx, [ecx+eax*4]
text:004114CE      mov     [ebp+var_14], edx
text:004114D1      jmp     short loc_4114AE
```

vs2010-debug模式编译运行

示例-一段简单的程序

```
视图(V) 项目(P) 生成(B) 调试(D) 团队(M) 数据(D)
Release Win32
mple_sum.cpp x
全局范围)
#include <stdio.h>

int sum(int a[], unsigned len):
void main() {
    int a[3]={1,2,3};
    int rtn = 0;

    rtn = sum(a, 0);

    printf("rtn = %d\n", rtn);
}

int sum(int a[], unsigned len)
{
    int i, sum = 0;
    for(i=0; i<=len-1; i++)
        sum += a[i];
    return sum;
}
```

vs2010-release模式编译运行: rtn=1

```
; int __cdecl main(int argc, const char **argv, const cl
_main proc near ; CODE XREF: __
var_4 = dword ptr -4
argc = dword ptr 8
argv = dword ptr 0Ch
envp = dword ptr 10h

push ebp
mov ebp, esp
push ecx
mov eax, __security_cookie
xor eax, ebp
mov [ebp+var_4], eax
push 1
push offset Format ; "rtn = %d\n"
call ds:printf
mov ecx, [ebp+var_4]
xor ecx, ebp
add esp, 8
xor eax, eax
call @@_security_check_cookie@4 ; __
mov esp, ebp
pop ebp
retn
_main endp
```

编译器直接把
sum结果算出
了来！！



示例-一段简单的程序

```
视图(V) 项目(P) 生成(B) 调试(D) 团队(M) 数据
Release Win32
mple_sum.cpp x
全局范围)
#include <stdio.h>

int sum(int a[], unsigned len);
void main() {
    int a[3]={1, 2, 3};
    int rtn = 0;

    rtn = sum(a, 0);

    printf("rtn = %d\n", rtn);
}

int sum(int a[], unsigned len)
{
    int i, sum = 0;
    for(i=0; i<=len-1; i++)
        sum += a[i];
    return sum;
}
```

vc6-release模式编译运行: 程序出错!

```
.text:00401040 sub_401040 proc near ;
.text:00401040
.text:00401040 arg_0 = dword ptr 4
.text:00401040 arg_4 = dword ptr 8
.text:00401040
.text:00401040 arg_4对应参数len mov ecx, [esp+arg_0]
.text:00401044 mov edx, [esp+arg_4]
.text:00401048 push esi
.text:00401049 xor eax, eax
.text:00401048 loc_401048: ;
.text:00401048 mov esi, [ecx]
.text:0040104D add ecx, 4
.text:00401050 add eax, esi
.text:00401052 dec edx
.text:00401053 jnz short loc_401048
.text:00401055 pop esi
.text:00401056 retn
.text:00401056 sub_401040 endp
```

示例-一段简单的程序

```
Debug Win32
mple_sum.cpp x
(全局范围)
#include <stdio.h>

int sum(int a[], int len);
void main() {
    int a[3]={1,2,3};
    int rtn = 0;

    rtn = sum(a, 0);

    printf("rtn = %d\n", rtn);
}

int sum(int a[], int len)
{
    int i, sum = 0;
    for(i=0; i<=len-1; i++)
        sum += a[i];
    return sum;
}
```

vs2010-debug模式编译运行: rtn=0

```
text:0041149E      mov     [ebp+var_14], 0
text:004114A5      mov     [ebp+var_8], 0
text:004114AC      jmp     short loc_4114B7
text:004114AE ; -----
text:004114AE      loc_4114AE:                                     ; CO
text:004114AE      mov     eax, [ebp+var_8]
text:004114B1      add     eax, 1
text:004114B4      mov     [ebp+var_8], eax
text:004114B7      loc_4114B7:                                     ; CO
text:004114B7      mov     eax, [ebp+arg_4]
text:004114BA      sub     eax, 1
text:004114BD      cmp     [ebp+var_8], eax
text:004114C0      jg      short loc_4114D3
text:004114C2      mov     eax, [ebp+var_8]
text:004114C5      mov     ecx, [ebp+arg_0]
text:004114C8      mov     edx, [ebp+var_14]
text:004114CB      add     edx, [ecx+eax*4]
text:004114CE      mov     [ebp+var_14], edx
text:004114D1      jmp     short loc_4114AE
```


示例-一段简单的程序

```
.text:0041149E      mov     [ebp+var_14], 0
.text:004114A5      mov     [ebp+var_8], 0
.text:004114AC      jmp     short loc_4114B7
.text:004114AE      ; -----
.text:004114AE loc_4114AE:      ; COI
.text:004114AE      mov     eax, [ebp+var_8]
.text:004114B1      add     eax, 1
.text:004114B4      mov     [ebp+var_8], eax
.text:004114B7 loc_4114B7:      ; COI
.text:004114B7      mov     eax, [ebp+arg_4]
.text:004114BA      sub     eax, 1
.text:004114BD      cmp     [ebp+var_8], eax
.text:004114C0      ja      short loc_4114D3
.text:004114C2      mov     eax, [ebp+var_8]
.text:004114C5      mov     ecx, [ebp+arg_0]
.text:004114C8      mov     edx, [ebp+var_14]
.text:004114CB      add     edx, [ecx+eax*4]
.text:004114CE      mov     [ebp+var_14], edx
.text:004114D1      jmp     short loc_4114AE
```

当len为unsigned时

```
.text:0041149E      mov     [ebp+var_14], 0
.text:004114A5      mov     [ebp+var_8], 0
.text:004114AC      jmp     short loc_4114B7
.text:004114AE      ; -----
.text:004114AE loc_4114AE:      ; CC
.text:004114AE      mov     eax, [ebp+var_8]
.text:004114B1      add     eax, 1
.text:004114B4      mov     [ebp+var_8], eax
.text:004114B7 loc_4114B7:      ; CC
.text:004114B7      mov     eax, [ebp+arg_4]
.text:004114BA      sub     eax, 1
.text:004114BD      cmp     [ebp+var_8], eax
.text:004114C0      jg      short loc_4114D3
.text:004114C2      mov     eax, [ebp+var_8]
.text:004114C5      mov     ecx, [ebp+arg_0]
.text:004114C8      mov     edx, [ebp+var_14]
.text:004114CB      add     edx, [ecx+eax*4]
.text:004114CE      mov     [ebp+var_14], edx
.text:004114D1      jmp     short loc_4114AE
.text:004114D3      ; -----
```

当len为int时

JA ... -----CF=0 且 ZF=0, 不满足条件, 不转移

JG ... -----SF=OF且 ZF=0, 满足条件, 转移

JA ... -----大于转移, 是针对无符号数的

JG ... -----大于转移, 是针对有符号数的

基础

- PE文件格式
- ELF文件格式
- Dex文件格式
- 密码学知识

- Android
- Windows
- Linux

- Java
- C/C++
- Python
- 汇编

工具

Java代码逆向工具

- Android Killer
- Jeb
- Xposed框架

C/C++逆向工具

- IDA Pro
- OllyDbg

其他工具

- Wireshark
- Fiddler
- DDMS

Android逆向工程师技能表

密码学基础

- 1.1 古典密码学基础
- 1.2. 代密码学基础
 - 1.2.1. 哈希算法
 - 1.2.2. 非对称加密与解密
 - 1.2.3. 椭圆曲线
 - 1.2.4. 数字签名
 - 1.2.5. 数据分组与校验
- 1.3. 密码攻击理论
- 1.4. 密码库的识别

网络基础

- 2.1. HTTP/HTTP2协议
 - 2.1.1. 网络数据包分析
 - 2.1.2. 数据包回放与编码实现
- 2.2. SSL/TLS协议
 - 2.2.1. 协议规范
 - 2.2.2. 协议握手过程
 - 2.2.3. 协议攻击方法
 - 2.2.4. 安全部署
- 2.3. 数据传输协议框架
 - 2.3.1. Protocol buffer
 - 2.3.2. Flat Buffer

逆向工程师技能表

网络抓包

- 3.1. 抓包工具
 - 3.1.1. Wireshark
 - 3.1.2. tcpdum
 - 3.1.3. Fiddler
 - 3.1.4. Charle
 - 3.1.5. BurpSuite
- 3.2. 数据包过滤与分析
- 3.3. 数据包重放与分析
- 3.4. WEBAPI安全审计
- 3.5. 数据包格式解析插件编写

Android逆向工程师技能表

文件格式相关

- 4.1. class字节码
 - 4.1.1. Java语言基础
 - 4.1.2. 字节码分析工具 010 Editor
 - 4.1.3. AOP编程基础
 - 4.1.4. AOP工具与框架
- 4.2. DEX/ODEX
 - 4.2.1. smali汇编语音
 - 4.2.2. DEX/ODEX文件格式
 - 4.2.3. DEX加载过程
 - 4.2.4. DEX重组与修复
- 4.3. OAT文件格式
 - 4.3.1. OAT文件布局
 - 4.3.2. DEX文件提取

- 4.4. ELF
 - 4.4.1. ELF文件格式、ABI规范
 - 4.4.2. ELF文件的动态加载
 - 4.4.3. ELF文件的篡改与修复
 - 4.4.4. Android Linker源码分析
 - 4.4.5. ARM/Aarch64汇编语音
 - 4.4.6. LLVM
 - 4.4.6.1. IR文档
- 4.5. AXML文件格式
- 4.6. ARSC文件格式
- 4.7. 文件格式分析工具
 - 4.7.1. 010 Editor
 - 4.7.2. gnu binutil

Android逆向工程师技能表

分析工具

- 5.1. IDAPro
 - 5.1.1. 静态分析
 - idc脚本、IDA python脚本开发
 - 5.1.2. 动态调试
 - 动态调试dex、动态调试so、调试器脚本与插件
- 5.2. Hopper
 - 5.2.1. 静态分析
- 5.3. Radare2

- 5.4. 动态调试器
 - 5.4.1. GDB
 - 5.4.2. LLDB
 - 5.4.3. JDB
 - 5.4.4. Small IDEA + AndroidStudio
- 5.5. 编译与反编译
 - 5.5.1. apktoo
 - 5.5.2. JEB
 - 5.5.3. Smali/BakSmali
- 5.6. 反编译查看
 - 5.6.1. JD-GUI
 - 5.6.2. JADX
 - 5.6.3. ByteCode Viewer

Android逆向工程师技能表

Hook与注入


- 6.1. hook类型
 - 6.1.1. dalvik hook (AOP hook, runtime method replacement)
 - 6.1.2. ART hook
 - 6.1.3. so hook (LD_PRELOAD hook, inline hook, got hook)
- 6.2. 注入类型
 - 6.2.1. dex注入
 - 6.2.2. so注入
- 6.3. hook注入框架
 - 6.3.1. xposed
 - 6.3.2. Frida

Android逆向工程师技能表

反破解

- 7.1. 资源加密
- 7.2. dex混淆
 - 7.2.1. proguard
 - 7.2.2. dexguard
- 7.3. so混淆
 - 7.3.1. ollvm
- 7.4. 反调试
 - 7.4.1. 阻止附加
 - 7.4.2. 多进程保护
 - 7.4.3. hook检测
 - 7.4.4. root检测

- 7.5. 模拟器检测
 - 7.5.1. 调试器状态
 - 7.5.2. 调试器端口
 - 7.5.3. 进程状态
 - 7.5.4. 运行时差

A decorative graphic consisting of two vertical bars, one gray and one yellow, positioned to the left of the email address.

luhw@hust.edu.cn
