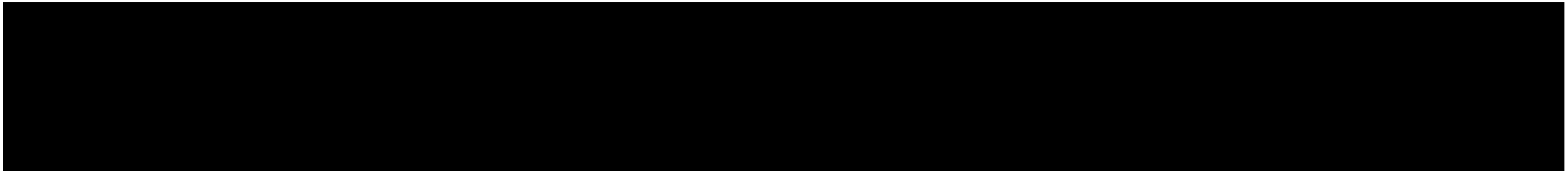
计算机系统工程导论 2024

实验七 计算机系统安全



背景介绍

对课程中学习的数据的机密性、完整性保护和身份认证相关的内容中实践中进一步加深理解与学会应用。

本次实验分为三个部分：

1. 使用python的cryptography库完成基本的数据安全加密/解密、签名/验证、完整性保护/鉴别等操作。
2. 学习基于缓冲区溢出攻击的进程控制流劫持，练习使用gcc内置的缓冲区溢出防护技术。
3. 了解海大研究团队设计的基于程序动态加载的进程安全防护技术。

背景介绍

预习要求：

- 学习python cryptography库的加解密及签名相关的API
  - https://cryptography.io/en/latest/development/
- 学习栈溢出漏洞和ROP攻击的基本原理
  - https://ctf-wiki.org/pwn/linux/user-mode/stackoverflow/x86/stackoverflow-basic/
  - https://ctf-wiki.org/pwn/linux/user-mode/stackoverflow/x86/basic-rop/
- 阅读海大研究团队公开发表的论文
  - https://arxiv.org/abs/2303.12612

实验环境

1. 软件环境：

- 本地Linux操作系统，安装python、gcc
- 安装python cryptography包
- （选做）安装IDA等反汇编工具

2. 要求：

- 禁止进行攻击他人计算机或数据的行为，遵守国家法律法规

实验目标

1. 掌握数据安全和防御的基本操作；

2. 掌握缓冲区溢出攻击和防护的基本技术；

3. 理解如何利用安全技术，设计实现更安全的系统。

实验安排

知识介绍与环境配置		10分钟
实验内容1：分组密码		20分钟
实验内容2：序列密码		20分钟
实验内容3：公钥密码		20分钟
实验内容4：缓冲区溢出防护技术		20分钟
实验内容5：进程安全防护技术		10分钟

实验内容1：分组密码算法（1）

操作：

1. 产生32字节的密钥key：os.urandom(32)
2. 产生16字节的初始向量iv
3. 使用cryptography.hazmat.primitives.ciphers模块中的：
   - Cipher, algorithms, modes
4. 产生密码机c：Cipher(algorithms.AES(key), modes.CBC(iv))
5. 加密器e和解码器d：cipher.encryptor()　cipher.decryptor()

问题：

- iv有什么用处?
- 使用了什么加密算法?
- 使用了什么工作模式？这种工作模式有什么优缺点?

实验内容1：分组密码算法（2）

操作:

- 使用加密器e对消息：b"hello ouc cser!"进行加密
  - ctxt = e.update (b"hello ouc cser!")+e.finalize()

问题:

- 出现了什么错误？为什么？

操作:

- 用填充的方法弥补以上错误，重新生成e并获得新消息的ctxt，打印出来
- 修改消息中1个字符，再次加密并打印密文，对比两次的密文

问题:

- 从两次的密文对比来看，有多少个Base64字符出现了变化？这说明加密算法应该有什么样的性质？

实验内容1：分组密码算法（3）

操作:

- 使用解密器d对密文进行解密
  - msg = d.update (ctxt)+d.finalize()
- 修改ctxt中的1个base64字符，重新解密

问题:

- 对比两次解密结果，思考其体现出的算法性质

附加操作（有时间可进行）

- 尝试不同的工作模式进行加密
- 尝试不同的密码算法进行加密

实验内容2：序列加密算法

操作：

1. 随机生成32字节key，16字节nonce
2. 用ChaCha序列密码算法生成密码机c，以及加密器e、解密器d：
   - Cipher(algorithms.ChaCha20(key,iv),mode = None)
3. 对b"hello ouc cser!"进行加解密（不要填充）

问题

- ChaCha是个什么样的算法，其安全性如何？
- 为什么本次操作不需要进行填充？

实验内容3：公钥密码算法（1）

操作：

- rsa模块
  - from cryptography.hazmat.primitives.asymmetric import rsa
- 生成私钥priv：
  - rsa.generate_private_key(key_size=2048,public_exponent=65537)
- 生成公钥pub：
  - priv.public_key()

问题：

- 2048的含义是什么？如果换成其他数字会怎么样？
- 为什么第二级的模块名叫做hazmat?

实验内容3：公钥密码算法（2）

操作：

- 导入单向散列（hash）和填充模块
  - from cryptography.hazmat.primitives import hashes
  - from cryptography.hazmat.primitives.asymmetric import padding
- 产生填充
  - pad=padding.OAEP(mgf=padding.MGF1(algorithm=hashes.SHA256()),algorithm=hashes.SHA256(),label=None)
- 加密b"OUC CSE GRADE 99"，再次加密b"OUC CSE GRADE 99"
  - pub.encrypt(b"OUC CSE GRADE 99",pad)

问题

- 两次加密的结果是否相同？为什么？
- 如果设计的加密算法未考虑填充，会有什么问题？

实验内容3：公钥密码算法（3）

操作：

- 选择sha-256算法
  - sha=hashes.SHA256()
- 生成hash器h
  - hashes.Hash(sha)
- 对消息求散列值d
  - h.update(b"OUC CSE")
  - d=h. finalize()

问题

- 密码学中的hash函数应具有什么性质?

实验内容3：公钥密码算法（4）

操作：

- 导入util模块
  - from cryptography.hazmat.primitives.asymmetric import utils
- 生成填充pad
  - padding.PSS(mgf=padding.MGF1(sha),salt_length=padding.PSS.MAX_LENGTH)
- 对消息签名得到sig
  - priv.sign(d,pad,utils.Prehashed(sha))
- 验证签名
  - pub.verify(sig,消息,pad,sha)

问题

- 验证签名的函数，是如何返回验证成功或失败的消息给调用者的？为什么？

实验内容4：缓冲区溢出防护技术

4-1 阅读CTF-WIKI上的ROP示例：

- 栈溢出原理：

https://ctf-wiki.org/pwn/linux/user-mode/stackoverflow/x86/stackoverflow-basic/

- 基本 ROP

https://ctf-wiki.org/pwn/linux/user-mode/stackoverflow/x86/basic-rop/

问题

- 用5-10句话简略说明ROP的基本原理

实验内容4：缓冲区溢出防护技术

4-2 GCC中的安全防护机制

- 自行编写测试程序，使用GCC提供保护机制进行编译，比较保护前后的二进制代码的特征。
- 分析二进制代码的工具有：
  - objdump 命令可以输出可执行文件的汇编代码
    - 常用指令：objdump -d test > test.obj
  - gdb 可以动态调试程序，查看运行时的数据和地址等各种数据。
  - checksec 工具可用于查看二进制文件采取了哪些安全措施。
    - 安装方法：sudo apt-get install checksec
  - IDA Pro 或者 Ghigra 等专业逆向分析工具，可以提供更强大的反汇编功能。

① canary保护

操作

- gcc -fno-stack-protector -o test test.c
  - 禁用栈保护
- gcc -fstack-protector -o test test.c
  - 启用栈保护，只为局部变量中含 char 数组的函数插入保护代码
- gcc -fstack-protector-all -o test test.c
  - 启用栈保护，为所有函数插入保护代码

问题

- Canary一词的来源是什么？借此解释Canary保护的原理

②Fortify保护

检测和防止缓冲区溢出、格式化字符串漏洞等与内存操作相关的潜在安全问题。

- 缓冲区溢出检查：在进行字符串复制或连接操作时，会检查源字符串的长度是否超过了目标缓冲区的大小。
- 格式化字符串检查：在使用格式化字符串函数（如printf，sprintf）时，会检查格式字符串的参数是否与格式化字符串中的占位符匹配。
- 内存操作检查：在进行内存操作时（如memcpy，memset），会检查源和目标内存块的大小是否匹配。

②Fortify保护

操作

- gcc -o test test.c
  - 默认情况下，不会开这个检查
- gcc -D_FORTIFY_SOURCE=1 -o test test.c
  - 仅在编译时进行检查
- gcc -D_FORTIFY_SOURCE=2 -o test test.c
  - 程序执行时也会有检查
- gcc -D_FORTIFY_SOURCE=3 -o test test.c
  - 进一步增强安全性

问题

- 当检测到溢出时，程序的行为有何变化?

③NX（DEP）

基本原理是将数据所在内存页标识为不可执行，当程序溢出成功执行shellcode时，CPU就会抛出异常。

操作

- gcc -o test test.c
  - 默认情况下，开启NX保护
- gcc -z execstack -o test test.c
  - 禁用NX保护
- gcc -z noexecstack -o test test.c
  - 开启NX保护

问题

- 如何查看内存页的各项权限？
- 攻击者如何绕过NX防御？给出简单的步骤说明

④PIE(ASLR)

内存地址随机化机制有以下三种设置：

0 - 关闭进程地址空间随机化。

1 - 将mmap的基址，stack和vdso页面随机化。

2 - 在1的基础上增加堆(heap)的随机化。

可通过 cat /proc/sys/kernel/randomize_va_space 查看当前操作系统采取的保护机制

④PIE(ASLR)

操作

- gcc -o test test.c // 默认情况下，关闭PIE
- gcc –fpie -pie -o test test.c // 开启PIE 强度为1
- gcc –fPIE -pie -o test test.c // 开启PIE 强度为2（堆随机化）

问题

- PIE和PIC的区别?
- 如何查看程序的虚拟地址，如何体现地址随机化?

⑤RELRO保护

表示Relocation Read-Only，链接器在执行开始时解析所有动态链接的函数，然后使GOT只读，从而减少对GOT表的攻击。

操作

- gcc -z norelro -o test test.c // 关闭，即No RELRO
- gcc -z lazy -o test test.c // 部分开启，即Partial RELRO
- gcc -z now -o test test.c // 全部开启，即 Full RELRO

问题

- 什么是GOT表？如何查看GOT表权限？

实验内容5：进程安全防护技术

基于动态加载的进程安全防护（中国海洋大学研究团队成果）

操作

- 阅读论文： LoadLord: Loading on the Fly to Defend Against Code-Reuse Attacks，地址： https://arxiv.org/abs/2303.12612
- 下载代码 ropkill-main
- 阅读源码，理解工作过程
- 配置IDA环境，执行build.sh

问题：

- 描述该技术的基本原理和简要流程，你认为可以如何改进它？

课后实验内容及思考

在密码学实验中，加长密文长度，你是否观察到了对称密码和公钥密码的速度差距？如何设计加密体制，使得保持对称密码的速度的同时获得公钥密码的安全性？

- 思路：用公钥密码传输对称密码的密钥
- 尝试使用python cryptography包进行实现