物联网安全
Internet of Things Security

第七章 物联网软件安全

冀晓宇
浙江大学

目录

- 物联网软件安全背景
- 物联网软件安全分类
  - **固件逆向**
  - **软件漏洞**
    - 弱口令
    - SQL注入、XSS
    - 缓冲区溢出、格式化字符串漏洞、条件竞争、释放后重用
  - **恶意代码**
- 物联网软件安全防御

软件与固件/软件安全

物联网软件安全背景

“软件正在吞噬世界”

- 物联网、云计算、大数据、移动终端、可穿戴设备和无人驾驶汽车等新兴IT技术借助着**各种软件**改变着人们的生活和工作
- 软件应用范围日益广泛，软件规模也在不断增大

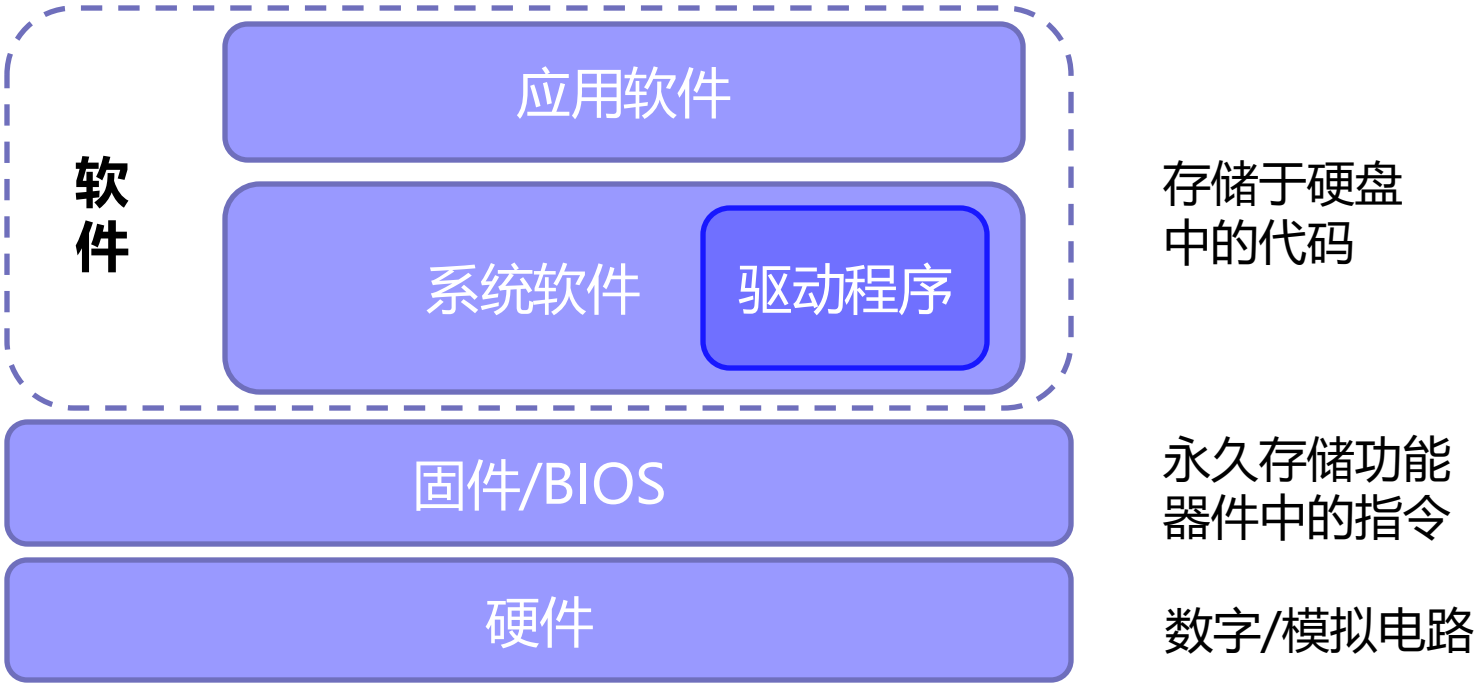
https://informationisbeautiful.net/visualizations/million-lines-of-code/

软件的定义与分类

- 定义
  - 计算机程序、规则和可能相关的文档——《信息技术软件工程术语》
  - Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system. ——《IEEE 软件工程术语标准词汇表》
- 一般来说，计算机软件分为**系统软件**、**应用软件**
  - **系统软件**：主要负责管理计算机系统中各种独立的硬件，支持应用软件开发和运行，例如操作系统、编译程序；
  - **应用软件**：主要包括用各种程序设计语言开发的应用程序，如Microsoft office、Steam、Skype等。

传统计算机软件

- 传统计算机：分为**软件、固件、硬件**三个部分
- 固件直接与硬件进行交互，而软件需要依靠固件才能调用硬件资源



传统计算机结构组成

软件的特殊形态 - 固件

定义：Combination of a hardware device and computer instructions or computer data that reside as read-only software on the hardware device

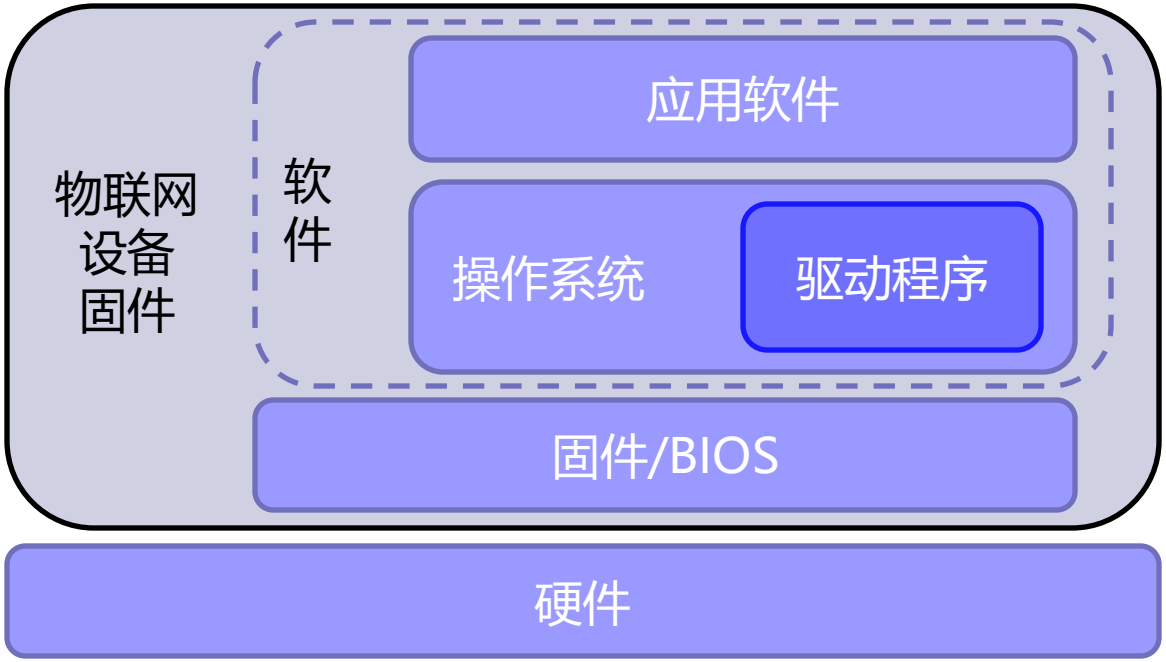
- 发展历史：Ascher Opler在1967年的Datamation文章中第一次提到术语 **firmware** (firm + ware)

- 固件最初指存储在可写入控制存储器的内容，包含定义和实现了计算机指令集的**微代码**(不同于硬件和软件)，这些微代码可以**加载或修改CPU执行的指令**

- “Somewhere between hardware and software”，它**存在于硬件和软件之间的边界上**，因此名称为firmware

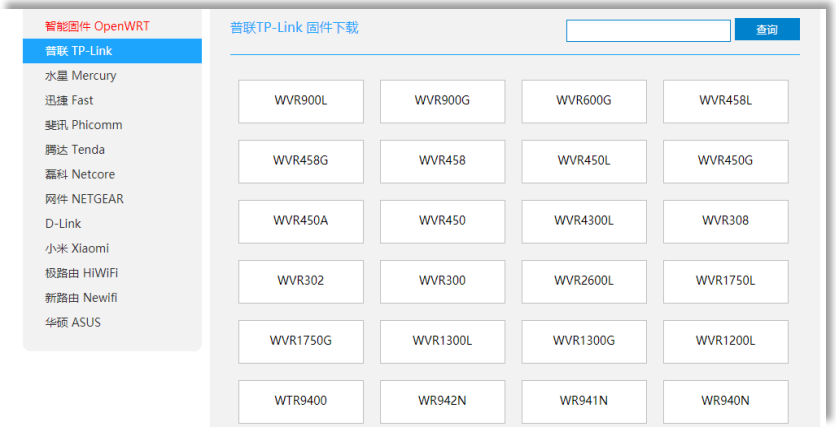
物联网设备软件 - 固件化形态

- 在物联网场景下，软件被嵌入在各类智能设备中，软件主要以固件的形式存在
- 对于物联网设备来说，广义的固件包含**应用软件、操作系统、BIOS等所有二进制代码文件**

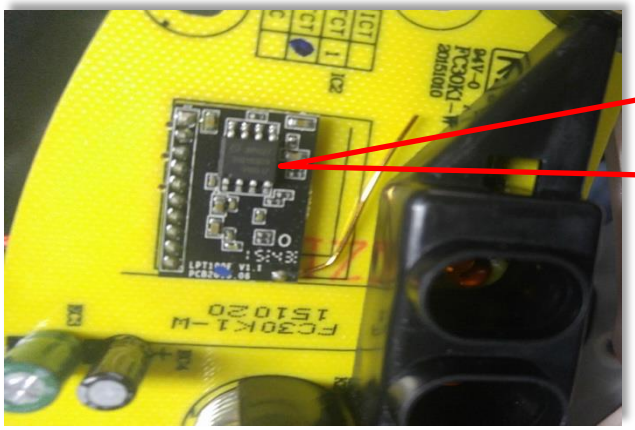


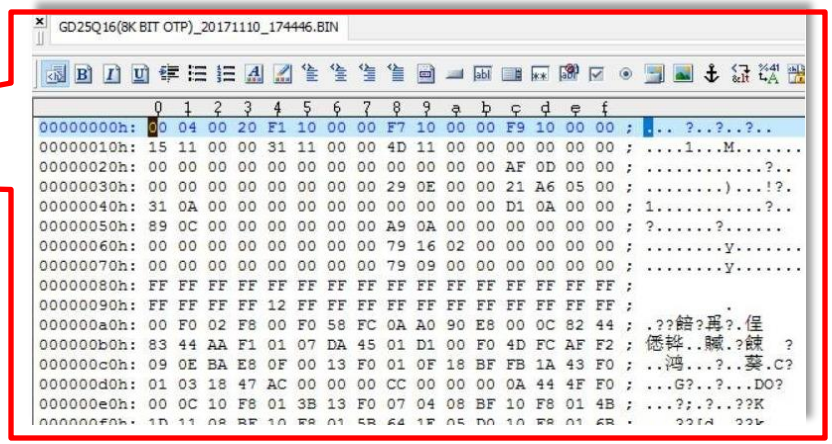
物联网设备固件举例

智能摄像头固件



智能音箱固件





智能电饭锅固件

物联网设备固件的结构

一个完整的固件由以下几部分组成：

- 固件头
- 引导程序部分（BootLoader)
- 可压缩内核部分
- 可压缩根文件系统部分
- 其它应用文件

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             DLOB firmware header, boot partition: "dev=/dev/mtdb
lock/2"
112           0x70            LZMA compressed data, properties: 0x5D, dictionary s
ize: 33554432 bytes, uncompressed size: 4027572 bytes
1376368       0x150070        PackImg section delimiter tag, little endian size: 9
457152 bytes; big endian size: 5148672 bytes
1376400       0x150090        Squashfs filesystem, little endian, version 4.0, com
pression:lzma, size: 5144687 bytes, 1878 inodes, blocksize: 131072 bytes, created:
 2015-11-02 06:12:22
```

软件逆向/软件漏洞

物联网软件安全问题

物联网软件安全事件

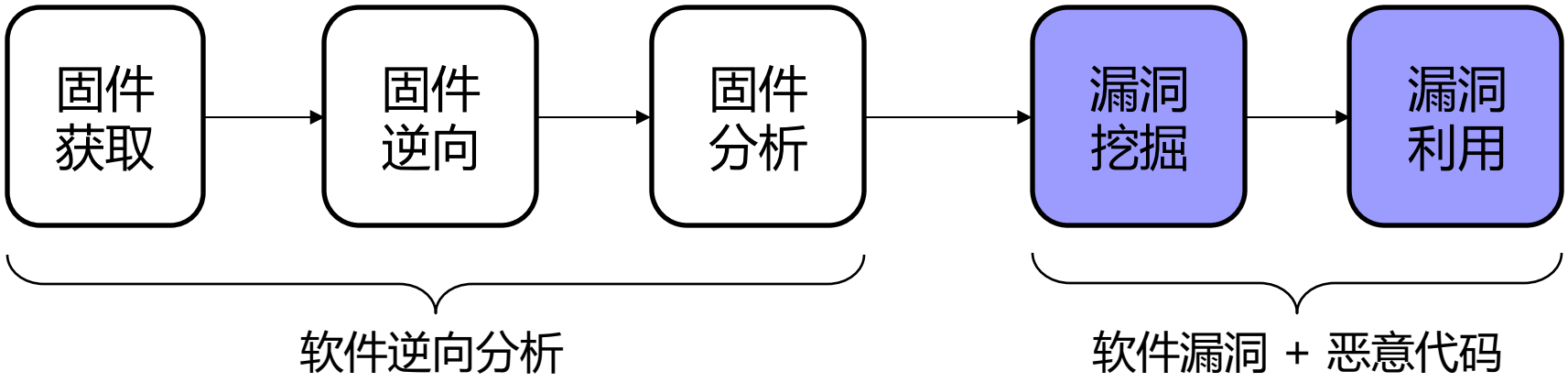
- 2015年12月，由于恶意软件导致乌克兰电网发生突发停电事故，导致约140万人口失去供电3~6小时
- 2018年3月，英国2700万新型智能电表被发现存在软件安全漏洞，可能对数百万居民的物联网设备构成严重风险
- 2018年黑客大会，美国东北大学两位研究员逆向了小米物联网设备的内部固件，发现了整个小米生态存在的漏洞
- ……

物联网软件安全

- **软件安全的定义**
  - 软件工程与软件保障提供一种系统的方法来标识、分析和追踪对危害及具有危害性功能（例如数据和命令）的软件缓解措施与控制。

——《信息技术软件安全保障规范》

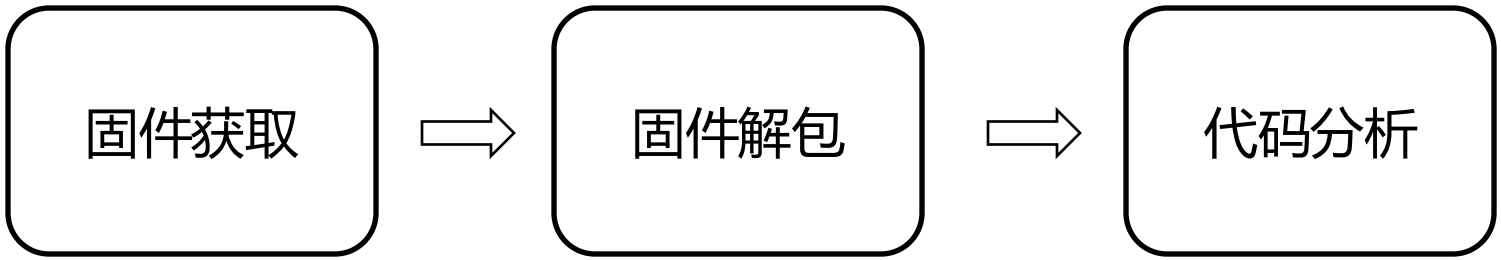
- **分类：软件逆向和软件漏洞**
- 物联网软件安全，通常以针对固件为对象，安全问题包括：



固件逆向

- **固件逆向**：利用逆向工程（Reverse Engineering）方法如解密、反汇编、系统分析、程序理解等技术，对固件及其中软件（通常为二进制代码）的结构、流程、算法、代码等进行逆向拆解和分析，得到软件的**源代码、设计原理、结构、算法、处理过程、运行方法及相关文档**

- 例如，获取到路由器固件后，通过固件逆向工具IDAPro获取固件中应用文件，分析能够利用的软件漏洞，比如实现绕过权限的登录

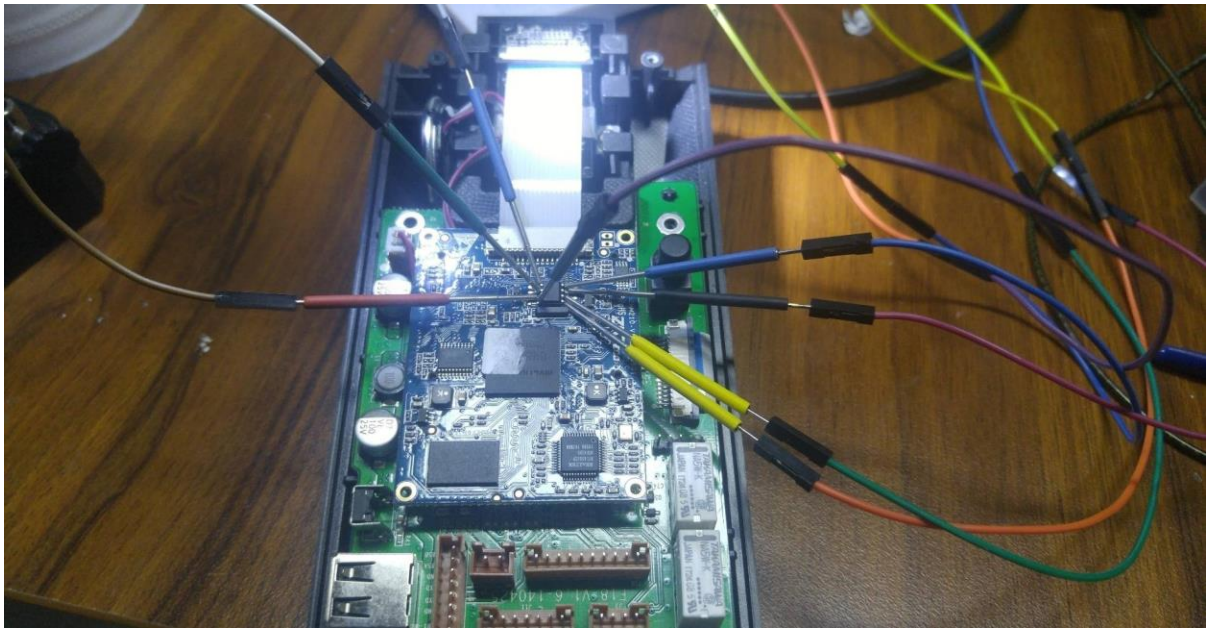


固件逆向过程

固件逆向 – 固件获取

- 方法1：直接获取，以某智能门锁固件为例

拆解设备 ⟹ 寻找存储芯片 ⟹ 读取固件

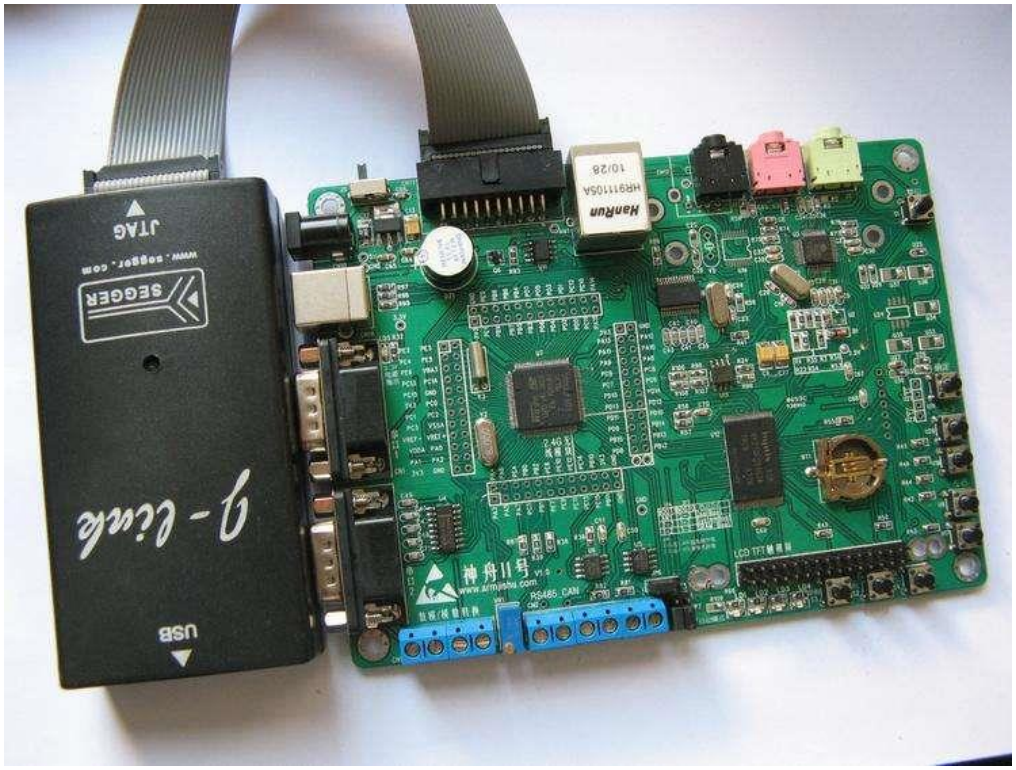


拆掉门禁外壳，通过电路图和芯片印字分析，在主板上有一颗FM25F04A存储芯片，通过夹具连接芯片到编程器，在通过专用编程器软件，对该芯片进行读取

举例：某品牌智能门锁固件逆向

固件逆向 – 固件获取

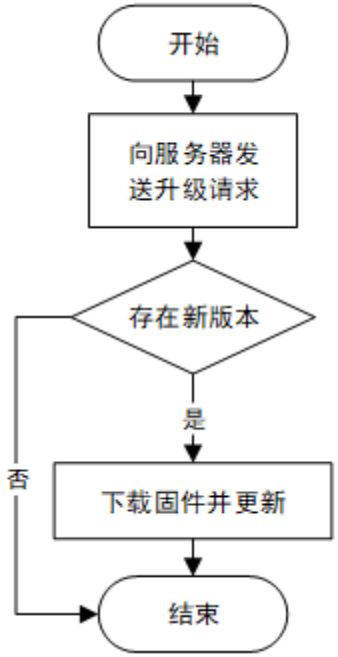
- 方法2：通过硬件接口获取



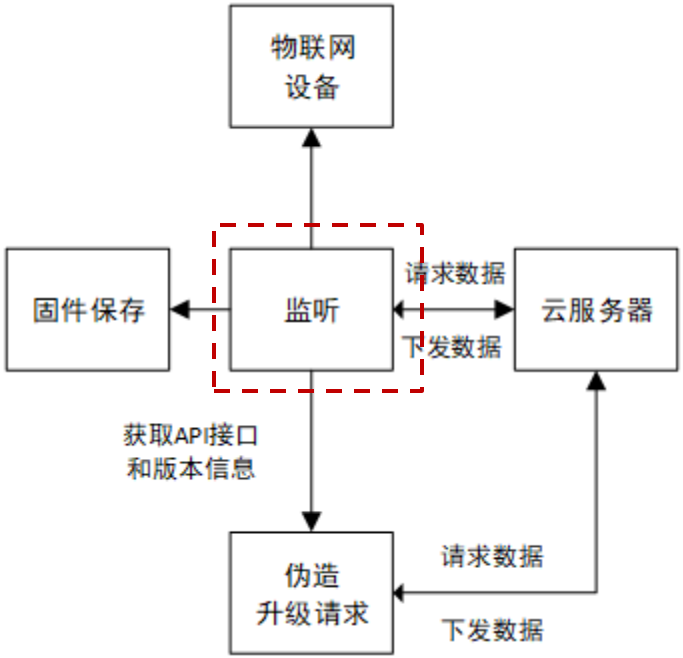
通过J-Link调试器接入JTAG调试结构，可以直接读取内存中的数据

固件逆向 – 固件获取

- 方法3：通过网络更新过程获取



正常的固件更新流程

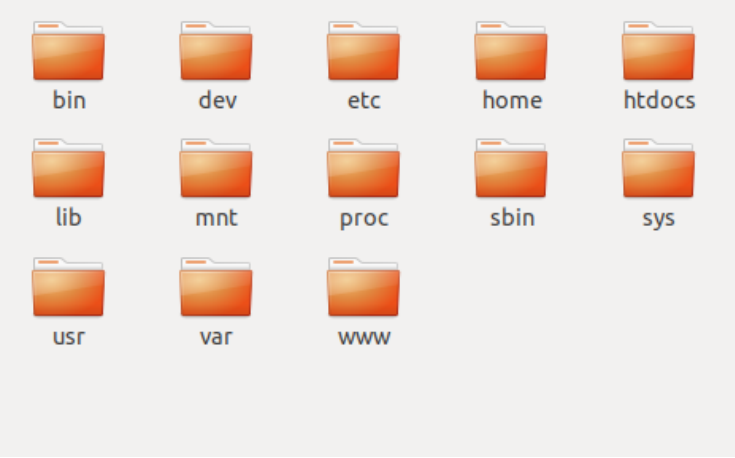


通过劫持网络更新获取固件

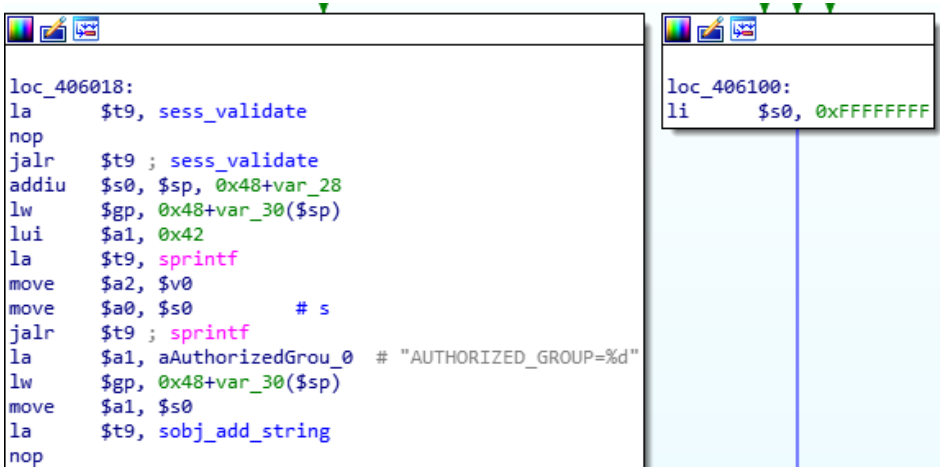
利用Wireshark等抓包工具监听网络环境中的数据包，获取设备固件更新的API接口、目前的版本号等验证信息，再通过伪造升级请求向厂商服务器的固件升级接口索取最新版本的固件数据包。

固件逆向 – 固件解包、代码分析

- 由于物联网设备的固件程序运行在嵌入式设备的特有环境中，所以动态分析受到较大地限制，通常会使用**静态分析技术**来分析
  - 使用Binwalk对固件包进行解压，提取固件的文件系统
  - 使用静态反汇编软件IDAPro对提取的固件进行静态分析



利用Binwalk解压后的固件文件系统



利用IDAPro对代码进行静态分析

固件安全：邪恶女仆攻击

- 邪恶女仆攻击：对无人值守设备的一种攻击方式，具有物理访问权限的攻击者，用某种无法检测的手段对设备进行更改，以便后续访问该设备或设备中的数据。
  - 第一步，攻击者需要接触你的电脑，通过从独立硬盘或分区启动计算机后，攻击者修改bootloader引导程序，关闭电脑；
  - 第二步，你使用修改过的bootloader启动电脑，输入加密钥匙，当加密硬盘解锁后，这个引导程序就可能会安装恶意程序去获得密钥，通过互联网发送到指定地点或储存在非加密分区。
- 之所以叫“邪恶女仆”，因为类似于一种虚构情节：当你将加密笔记本留在饭店内离开去办事之后，女仆可以偷偷的进来修改bootloader，然后第二天再来抹掉痕迹。

固件安全：Tesla Model X无钥匙开门

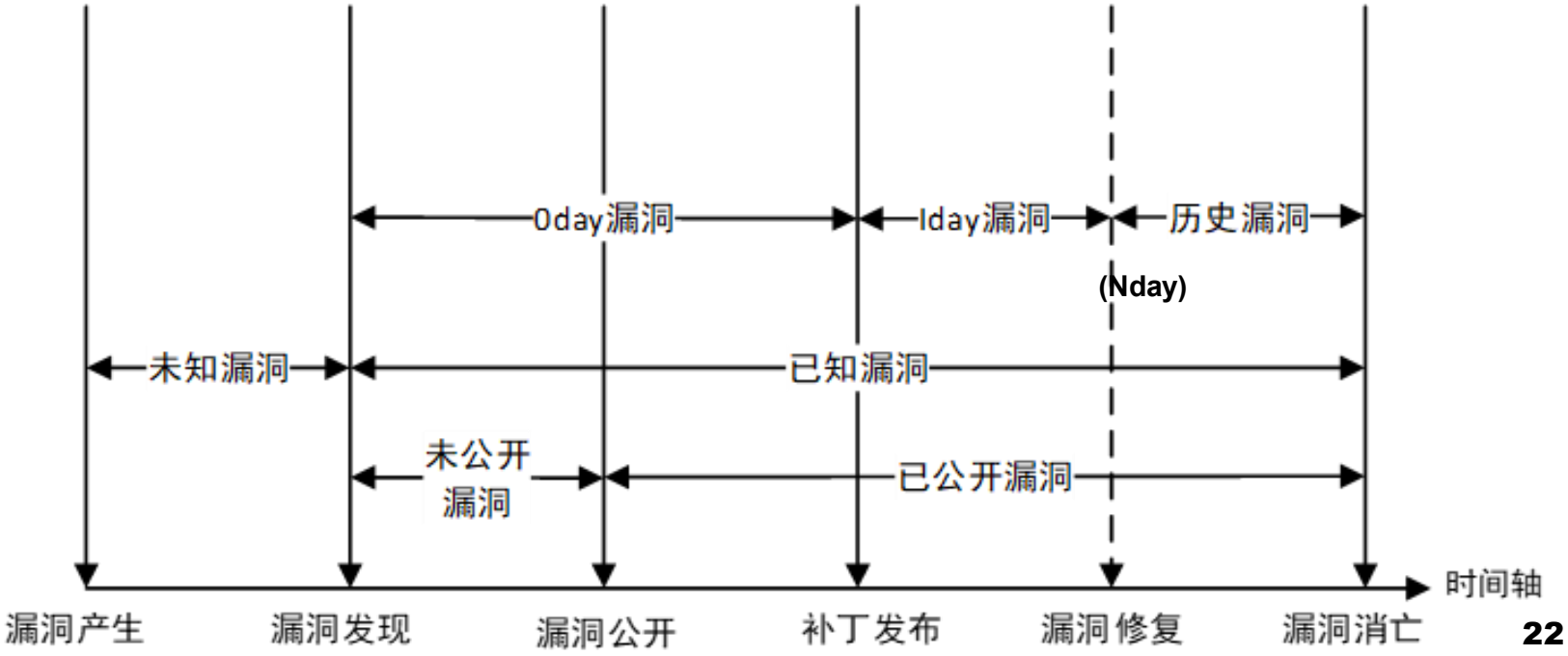
- 特斯拉 Model X无钥匙解锁攻击
  - 问题1：车钥匙固件更新没有验证来源，导致钥匙固件被替换
  - 问题2：车机电脑没有对钥匙内安全证书检查，导致假钥匙可以启动车辆

Ref: https://www.wired.com/story/tesla-model-x-hack-bluetooth/

物联网软件漏洞和恶意代码

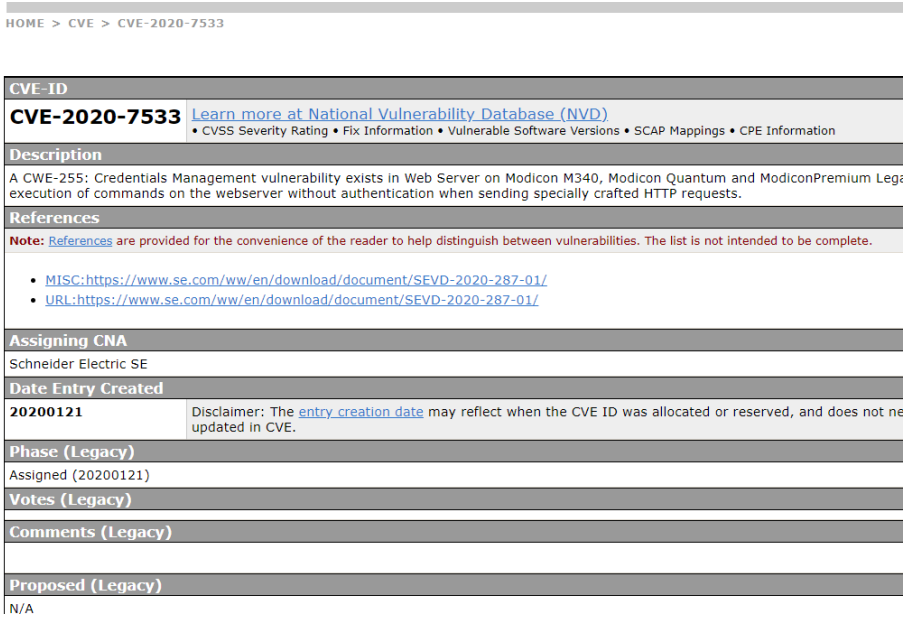
物联网软件安全 - 软件漏洞

- 软件漏洞(Software Vulnerability)：在软件设计、实现、配置或操作过程中存在的缺陷、错误或弱点，可能被攻击者利用以实现未授权的操作，可能导致系统崩溃、数据泄露、服务中断、完全控制目标系统等
- 漏洞贯穿软件生命周期各环节



物联网软件安全 - 软件漏洞

- **漏洞上报/披露平台**
  - **CVE**: Common Vulnerabilities & Exposures
  - **CNVD**: 国家信息安全漏洞库（China National Vulnerability Database of Information Security）



HOME > CVE > CVE-2020-7533

CVE-ID

CVE-2020-7533 Learn more at National Vulnerability Database (NVD)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

A CWE-255: Credentials Management vulnerability exists in Web Server on Modicon M340, Modicon Quantum and ModiconPremium Lega execution of commands on the webserver without authentication when sending specially crafted HTTP requests.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- MISC:https://www.se.com/ww/en/download/document/SEVD-2020-287-01/
- URL:https://www.se.com/ww/en/download/document/SEVD-2020-287-01/

Assigning CNA

Schneider Electric SE

Date Entry Created

20200121 Disclaimer: The entry creation date may reflect when the CVE ID was allocated or reserved, and does not ne updated in CVE.

Phase (Legacy)

Assigned (20200121)

Votes (Legacy)

Comments (Legacy)

Proposed (Legacy)

N/A

漏洞信息详情

Microsoft Internet Explorer 缓冲区错误漏洞

CNNVD编号：CNNVD-202001-876

危害等级：高危

CVE编号：CVE-2020-0674

漏洞类型：缓冲区错误

发布时间：2020-01-17

威胁类型：远程

更新时间：2020-11-30

厂　商：

漏洞来源：Clément Lecigne of...

漏洞简介

物联网软件安全 - 恶意代码

- **恶意代码**（Malicious Software）：在未被授权的情况下，以破坏软硬件设备、窃取用户信息、干扰用户正常使用、扰乱用户心理为目的而编制的软件或代码片段
- **实现方式：**二进制执行文件、脚本语言代码、宏代码或是寄生在其它代码或启动扇区中的一段指令
- **恶意代码种类**：计算机病毒、蠕虫、木马、后门等
- 软件漏洞和恶意代码是软件面临的最主要两大安全威胁
  - **软件漏洞 vs. 恶意代码：**软件漏洞一般不是主观意愿，恶意代码是主观意愿。恶意代码一般利用软件漏洞实施攻击。

物联网软件安全- 恶意代码类型

- **计算机病毒**Virus：一种能够自我复制并附加在其他程序或文件上的恶意代码。当这些被感染的程序或文件被执行或打开时，病毒会被激活并开始复制和传播
- **蠕虫**Worm：一种能够自我复制并且不需要宿主程序或用户启动来传播的恶意代码，通常通过网络自主传播，并在受感染的计算机之间自动传播
- **木马**Trojan：一种看似合法或无害的软件程序，包含一段隐藏的、激活时会运行且具备某种有害功能的代码，使得非法用户达到进入系统、控制系统甚至破坏系统的目的通常伪装成有用的工具或应用程序来诱骗用户进行下载和安装
- **后门**Backdoor：一种故意被留在软件或系统中的隐藏入口，允许攻击者绕过正常认证过程，直接访问系统或应用程序，通常由开发者或攻击者在软件开发或传播过程中插入

物联网软件安全 - 恶意代码

- **不同种类恶意代码之间的关系**
  - 病毒需要活动的宿主程序或已被感染的操作系统才能运行，蠕虫是独立的恶意程序，可以通过计算机网络进行自我复制和传播，不需要人工干预
  - 后门与计算机病毒、蠕虫的区别是后门不会感染其它计算机
  - 木马与后门都隐藏在用户系统中，本身可以提供一定权限，以便远程机器对本机的控制；但是，木马是一个完整的软件，后门不是完整软件，是系统中软件所具有的特定功能的缺陷
- 早期恶意代码的主要形式是计算机病毒

对比

特征/类型	病毒 (Virus)	蠕虫 (Worm)	木马 (Trojan)	后门 (Backdoor)
传播方式	依附宿主文件传播	网络自我传播	伪装正常程序	程序预留隐蔽入口
宿主依赖	✓ 必须	✗ 不需要	✗ 不需要	✗ 不需要
自我复制	✓ 可自我复制	✓ 主动传播	✗ 不能	✗ 不能
主要目的	破坏文件/系统	消耗资源/传播	窃取控制权	隐蔽访问通道
用户交互	✓ 需要执行	✗ 自动传播	✓ 需要安装	✗ 静默运行
典型示例	CIH病毒（1998）	Morris蠕虫（1988）、熊猫烧香	Zeus银行木马	SolarWinds后门

软件木马和后门的区别

- **实现方式**
  - **木马**：通常通过欺骗用户安装，伪装成合法的软件
  - **后门**：通常是由开发者或攻击者故意留在系统中的隐藏入口，可能存在于软件开发阶段或通过漏洞注入到系统中
- **目的**
  - **木马**：主要目的是窃取数据、监视用户或使系统感染更多软件
  - **后门**：主要目的是为攻击者提供长期的隐秘访问和控制权限，绕过正常的认证和安全措施
- **发现难易程度**
  - **木马**：可能被防病毒或反恶意软件工具检测和删除，但在漏洞被利用之前可能会隐藏在系统中
  - **后门**：通常更难被发现，因为它们可能被设计成看似正常的程序或嵌在系统代码中

案例：WannaCry勒索病毒

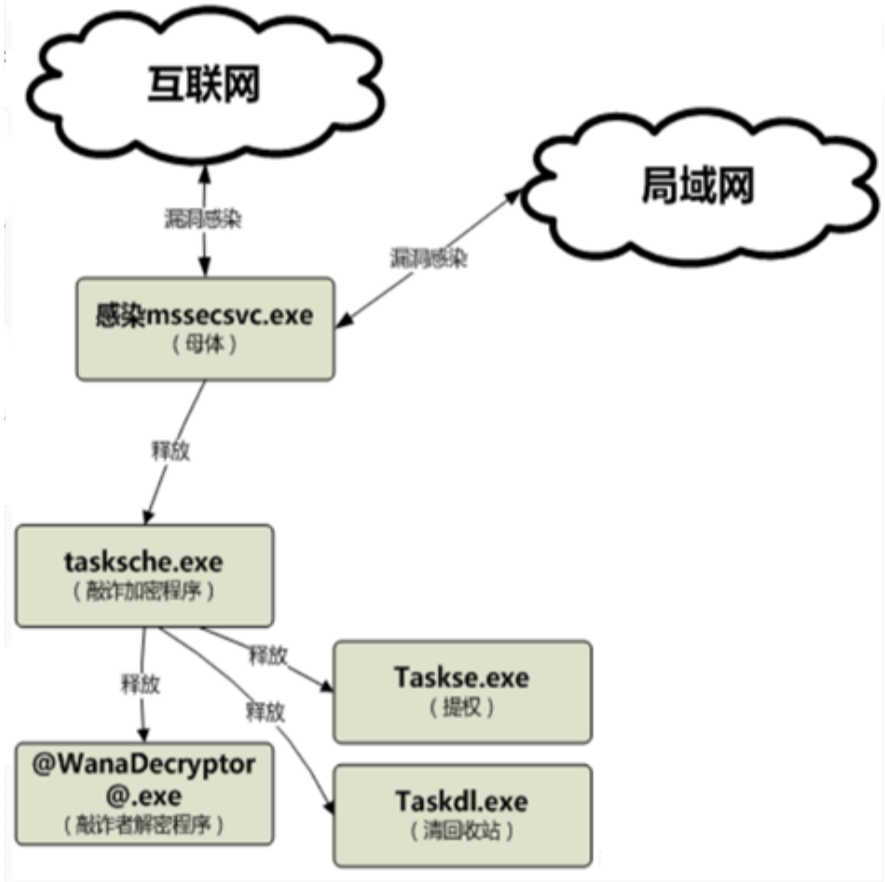
- 2017年5月12日，WannaCry全球爆发，至少150个国家、30万用户中招，造成损失达80亿元。我国部分Windows操作系统用户受感染，其中大部分为校园网用户，大量实验室数据和毕业论文被加密锁定，病毒会提示支付价值300美元的比特币才可解密。

WannaCry勒索病毒背景知识

- **SMB**（Server Message Block）协议：一种客户机/服务器、请求/响应协议，通过SMB协议可以在计算机之间共享文件、打印机、命名管道等资源，网上邻居即是通过SMB协议实现。SMB使用TCP139和445端口来共享文件或资源
- **永恒之蓝**（EternalBlue）：一种漏洞利用工具，该工具利用了微软于17年3月发布的MS17-010补丁中提到的其SMB协议存在的远程代码执行漏洞
- **WannaCry借助永恒之蓝，攻击没有更新MS17-010补丁文件并开放445端口的windows操作系统用户**

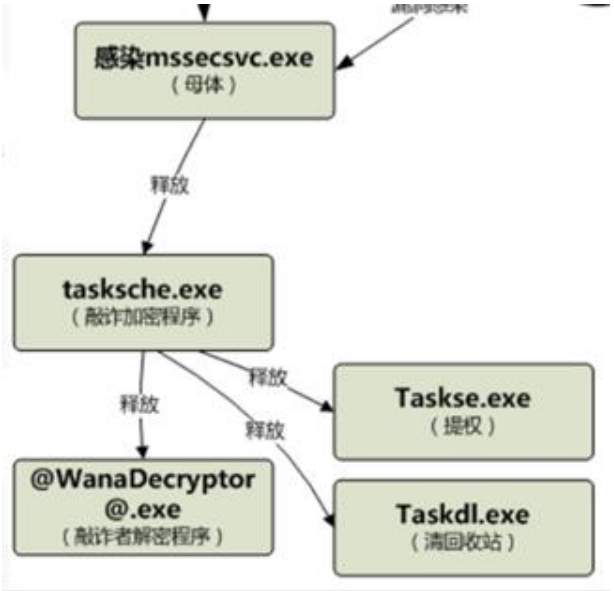
WannaCry传播和工作原理

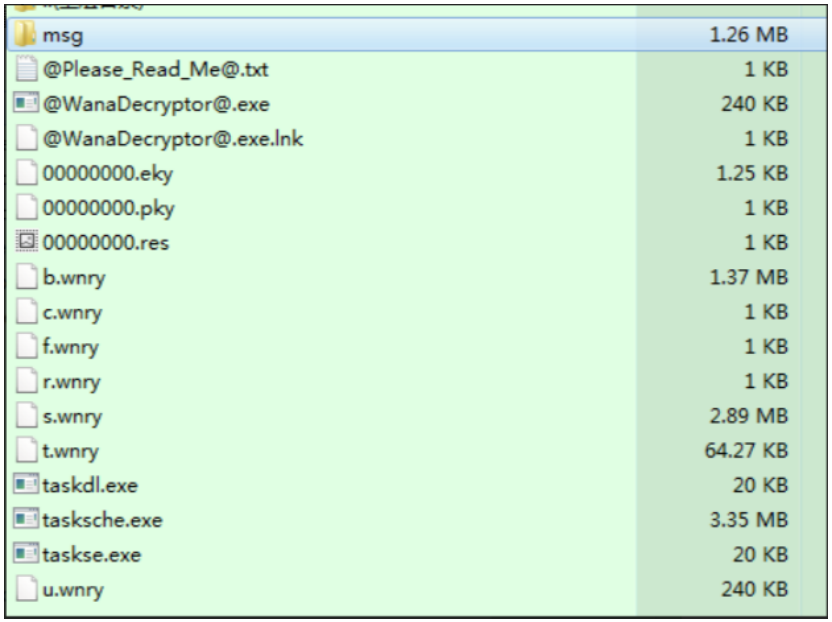
- **病毒传播和感染**
  - 从病毒样本自身读取MS17-010漏洞利用代码
  - 病毒样本创建两个线程，分别扫描内网和外网IP，开始传播感染
  - 对公网随机IP地址445端口进行扫描，若存在MS17-010漏洞则感染
  - 对局域网，直接扫描当前计算机所在网段并尝试连接445端口进行感染



WannaCry传播和工作原理

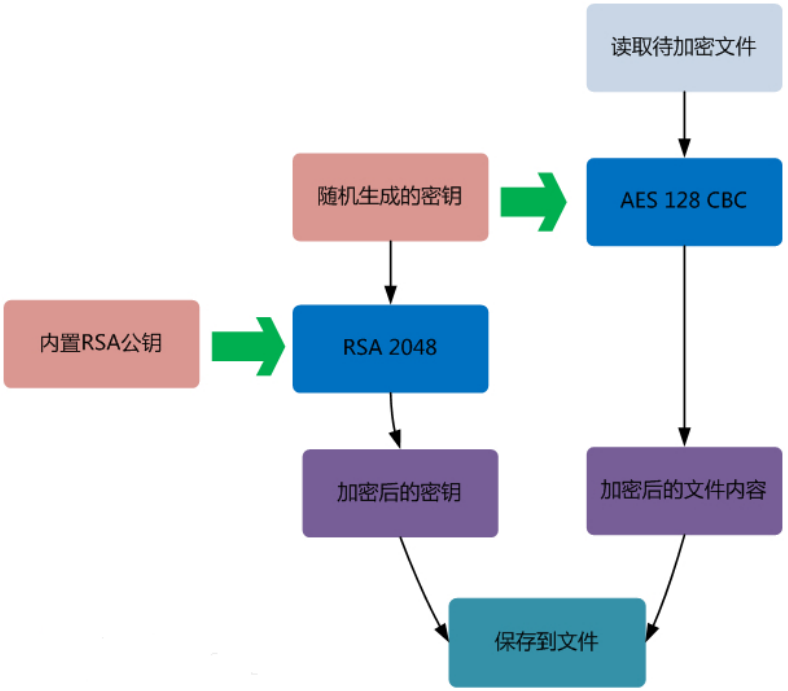
- **释放加密器**
  - 启动tasksche.exe，解压释放出若干文件，包括加/解密程序、提权模块和清空回收站模块，释放的所有文件都被设置为“隐藏”
  - 遍历查找文件，判断是否是要加密的文件类型，加密对象主要包括常用的office文件、压缩文档和媒体文件、程序源代码和项目文件、图片文件等





WannaCry传播和工作原理

- **读取文件并加密**
  - 使用AES-128-CBC模式加密文件内容，保存扩展名为.WNCRY文件。
  - 完成文件加密后，释放说明文档，给出比特币钱包地址和付款金额，3个比特币钱包地址硬编码于程序中，每次随机选取一个显示。



用户被加密文件

物联网软件漏洞

- 国际组织OWASP2018年定义了**物联网十大网络安全问题**

https://owasp.org/www-project-internet-of-things/

物联网软件漏洞

- **不安全的认证和授权**
  - 弱口令登录
- **不安全的Web接口**
  - SQL注入
  - XSS攻击
  - 信息泄露
- **不安全的系统和程序**
  - 缓冲区溢出漏洞
  - 格式化字符串漏洞
  - 释放后重用漏洞
  - 条件竞争漏洞

https://owasp.org/www-project-internet-of-things/

不安全的认证和授权 - 弱口令

- 弱口令(weak password)：通常认为容易被别人（他们有可能对你很了解）猜测到或被破解工具破解的口令均为弱口令
- 弱口令举例：仅包含简单数字和字母的口令，如“123”、“abc”等，这样的口令很容易被别人破解
- **物联网弱口令客观原因**：由于物联网设备数量多、种类众多，人们对密码记忆能力有限；用户缺乏安全意识，导致弱口令在物联网设备中依旧十分普遍
- **撞库攻击**：Credential Stuffing，攻击者利用来自数据泄露或其他非法途径获取的大量用户名和密码（或者弱口令）组合，尝试在多个网站或服务上自动登录。这类攻击的成功率利用了用户在多个平台上重复使用相同的密码这一常见现象

英文弱口令排名

Top 10 Worst Passwords - Historic Analysis

	2023	2015	2010	2005	2000
#1	123456	123456	123456	password	password
#2	123456789	password	password	123456	123456
#3	qwerty	12345	12345678	12345678	12345678
#4	password	12345678	qwerty	abc123	qwerty
#5	1234567	qwerty	abc123	qwerty	abc123
#6	12345678	1234567890	123456789	monkey	monkey
#7	12345	1234	111111	letmein	1234567
#8	iloveyou	baseball	1234567	dragon	letmein
#9	111111	dragon	iloveyou	111111	trustno1
#10	Covid	football	adobe123	baseball	dragon

英文常见弱口令种类


dashlane
A Decade of Passwords Trends
Champions League Teams
liverpool
chelsea
arsenal
barcelona
manchester
Most Popular Brands
myspace
mustang
linkedin
ferrari
playboy
mercedes
cocacola
snickers
corvette
skittles
Movies & Music
superman
pokemon
slipknot
starwars
metallica
nirvana
blink182
spiderman
greenday
rockstar
Love & Hate
iloveyou
f***you
a**hole
f***off
iloveme
trustno1
beautiful
ihateyou
bulls***
lovelove

中文常见（弱）口令

- 针对天涯、7k7k、CSDN等主流网站的用户口令进行统计分析
- 常见弱口令：123456、password、5201314等

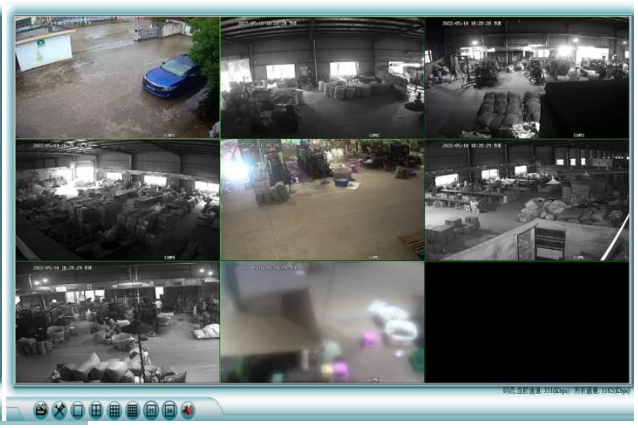
Rank	Tianya	7k7k	Dodonew	178	CSDN	Duowan	Rockyou	Yahoo	Phpbb
1	123456	123456	123456	123456	123456789	123456	123456	123456	123456
2	111111	0	a123456	111111	12345678	111111	12345	password	password
3	000000	111111	123456789	zz12369	11111111	123456789	123456789	welcome	phpbb
4	123456789	123456789	111111	qiulaobai	dearbook	123123	password	ninja	qwerty
5	123123	123123	**5201314**	123456aa	00000000	000000	**iloveyou**	abc123	12345
6	123321	**5201314**	123123	wmsxie123	123123123	**5201314**	princess	123456789	12345678
7	**5201314**	123	a321654	123123	1234567890	123321	123321	12345678	letmein
8	12345678	12345678	12345	000000	88888888	a123456	rockyou	sunshine	111111
9	666666	12345678	000000	qq66666	111111111	suibian	12345678	princess	1234
10	111222tianya	wangyut2	123456a	w2w2w2	147258369	12345678	abc123	qwerty	123456789
Sum of top-10	2,297,505	440,300	533,285	793,132	670,881	338,012	669,126	4,476	7,135
Total accounts	30,901,241	5,423,287	16,258,891	9,072,965	6,428,277	4,982,730	32,581,870	442,834	255,373
% of top-10	7.43%	8.12%	3.28%	8.74%	10.44%	**6.78%**	2.05%	1.01%	**2.79%**

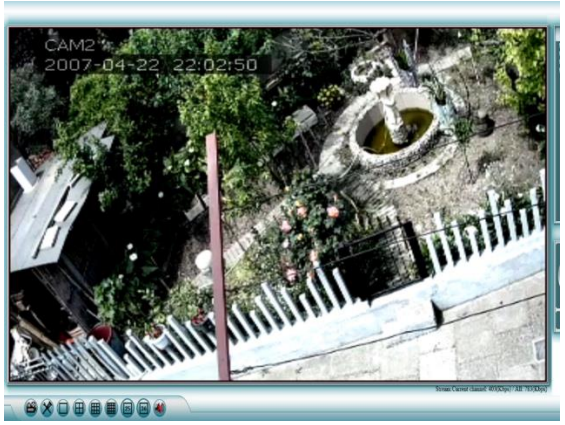
- 想一想：不同长度的中英文密码破解难度有何差异?

Ding Wang, etc., “Birthday, Name and Bifacial-security: Understanding of Chinese Web Users”, in Usenix Security 2019

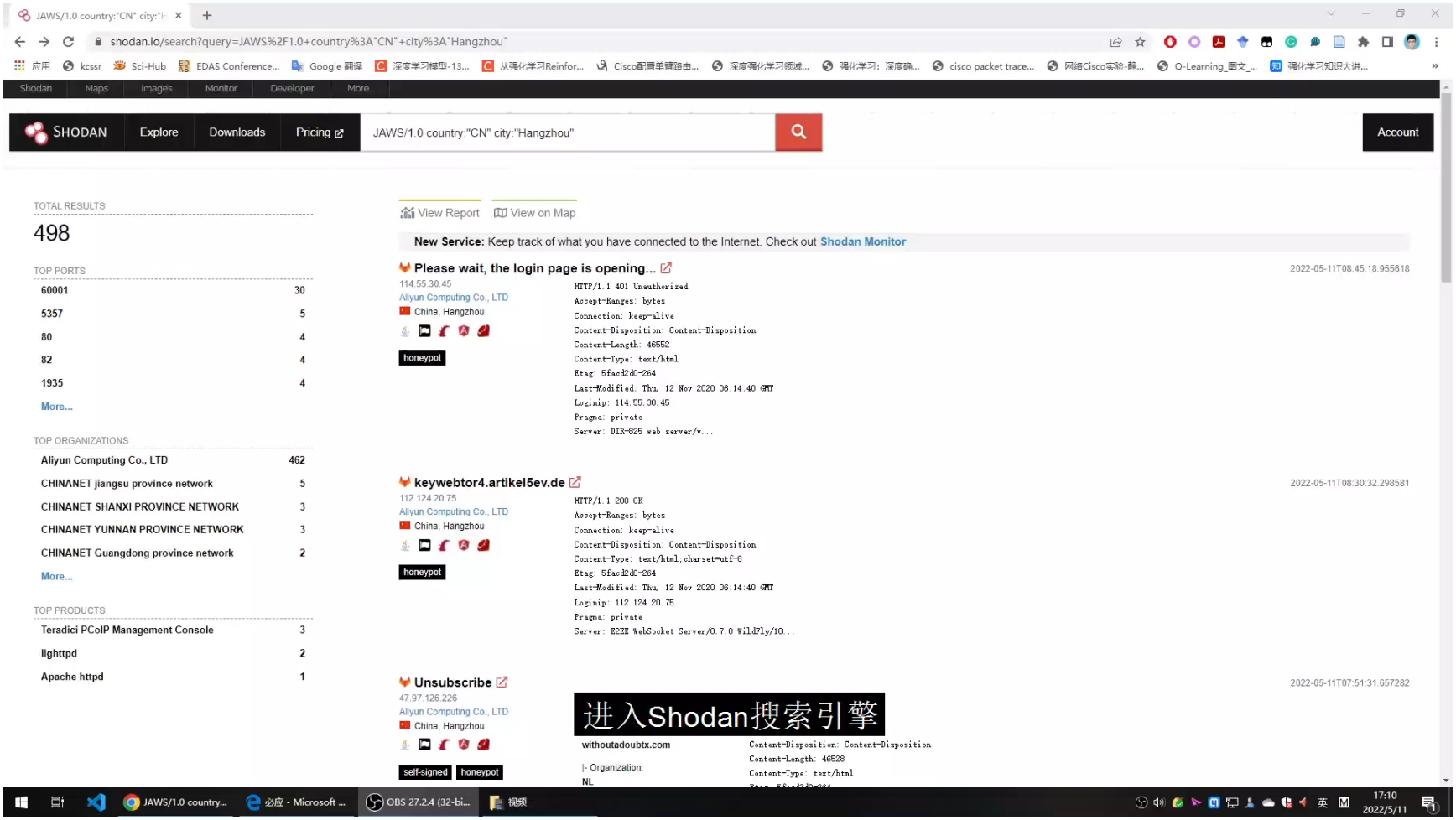
案例：利用弱口令漏洞入侵监控摄像头

- 使用Shodan工具扫描物联网设备：https://www.shodan.io/
  - 自定义：端口、国家、协议类型等，如60001、CN、JAWS
- 发现目标设备，并尝试使用弱口令进行登录
  - http://113.64.36.xxx:60001/ （国内）
  - http://117.79.xxx.52:60001/ （国内）
  - http://151.26.5.xxx:60001/ （国外）





案例：利用弱口令漏洞入侵监控摄像头



不安全的Web接口 - SQL注入攻击

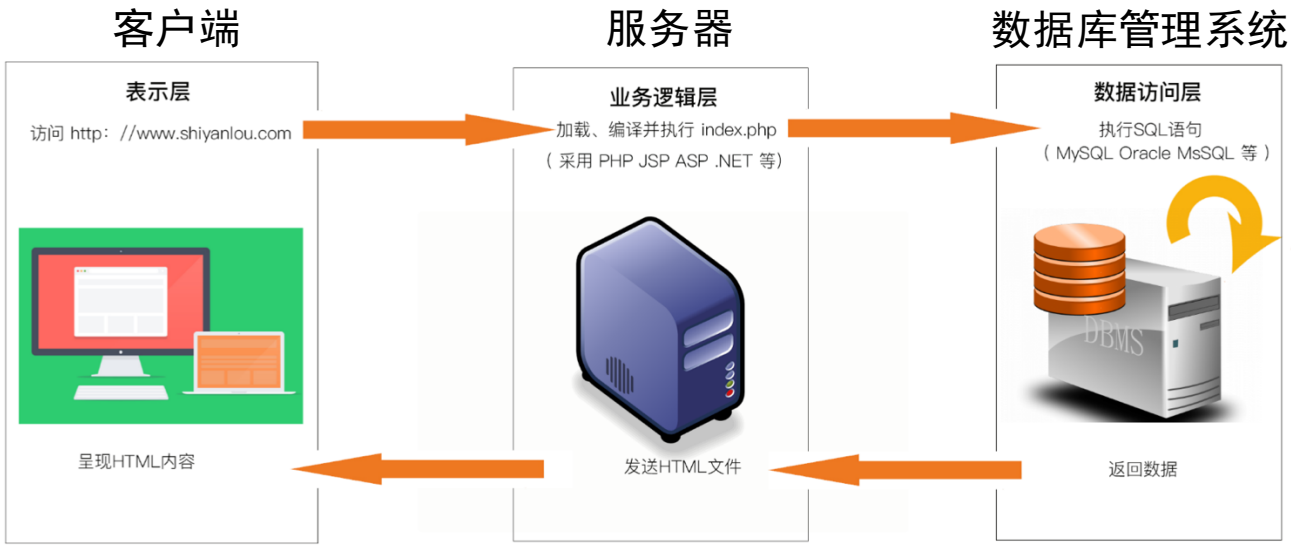
- **定义**：攻击者通过将恶意的SQL代码片段注入到应用程序输入字段，诱导后台数据库执行未预期的SQL命令，从而实现获取、篡改和删除数据库中数据的目的
- **原理**：SQL注入通常在输入的字符串之中注入精心设计的恶意SQL指令，当程序设计存在缺陷，注入的恶意指令就会被数据库服务器误认为是正常的SQL指令而运行，因此遭到破坏或是入侵
- **危害**：数据泄露、数据篡改、数据删除、权限提升、拒绝服务
- **举例**：在登录表单中的用户名字段输入 ' OR '1'='1，导致查询变为：

```
SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '';
```

注意：这个查询总是返回真，如果没有其他安全措施，那么攻击者将绕过登录验证

Web程序基本原理

- Web 浏览器中输入域名并连接到服务器
- Web 服务器从本地存储中加载 index.php 脚本并解析
- 脚本连接DBMS（数据库管理系统），并执行 SQL 语句
- 数据库管理系统返回 SQL语句执行结果给 Web 服务器
- Web 服务器将 Web 页面封装成 HTML 格式发送给 Web 浏览器
- Web 浏览器解析 HTML 文件，将内容展示给用户。



SQL注入攻击案例：验证绕过

- 针对一个Web网页登陆界面，账号密码通常都是存储在数据库中的，正常登录过程的SQL查询语句为如下形式：

```
$sql = "select * from users where username='$name' and password='$pwd'"
```

- 用户名和密码都输入“admin”时，真正执行的SQL语句为：

```
select * from users where username='admin' and password='admin'
```

SQL注入攻击案例：验证绕过

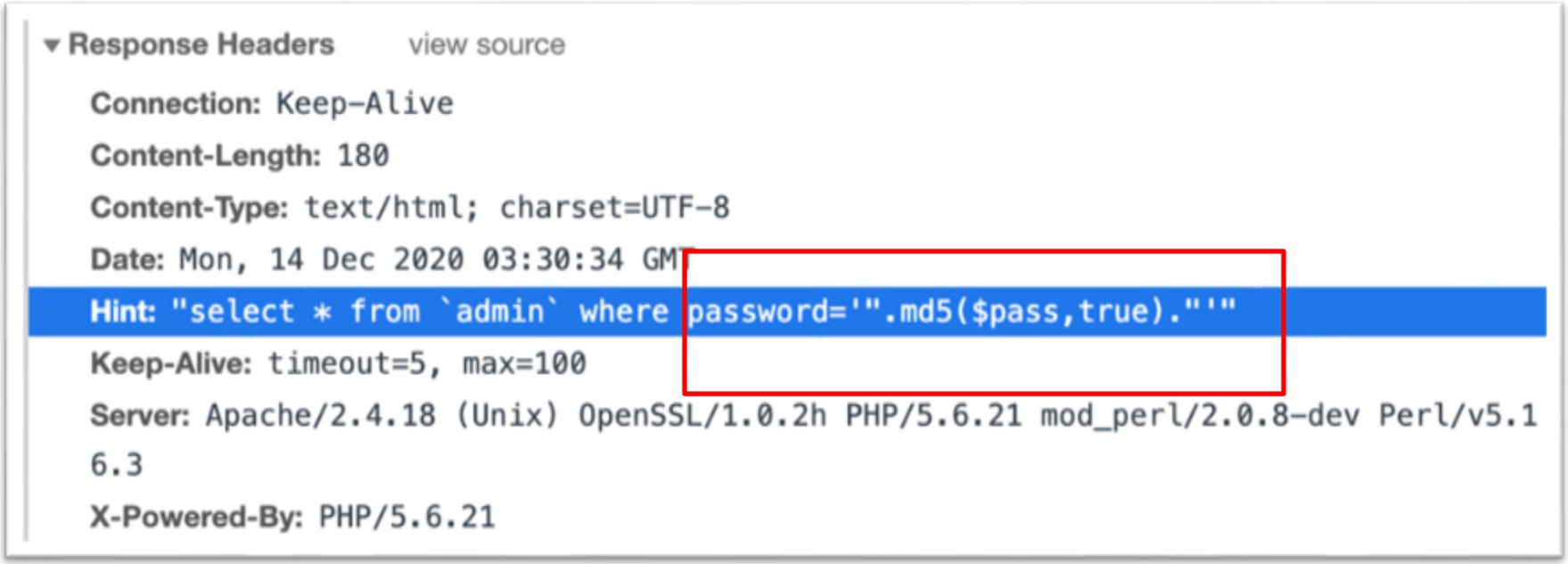
- 如果不知道登录密码，可以利用SQL注入攻击绕过验证。具体输入用户名为“admin’ or 1=1#”，密码“admin”，此时被执行的SQL语句为:

```
select * from users where username='admin' or 1=1 #' and password='admin'
```

- 在注入的SQL语句中：**#后被语句被注释，所以对password的判断语句不会执行**
- 而用户名判断语句 or 1=1 恒成立!
- 因此，结果返回“1”，在不知道密码的情况下绕过验证登录成功

SQL注入攻击案例：验证绕过

- 通常，密码检查不会直接用字符串匹配，而是利用admin表中查询用户输入密码对应的md5值和数据库中的是否相等，从而判断用户是否是认证的用户。
- 如何实现SQL注入攻击实现验证绕过？



SQL注入攻击案例：验证绕过

- **方法**：利用md5生成SQL注入攻击语句

```php
<?php
$password='ffifdyop';
echo(md5($password,true));
?>
```

276f722736c95d99e
921722cf9ed621c

'or'6É]é!r,ùíb

- **原理**：**ffifdyop**这个字符串经过md5加密之后得到的字符串是一个可以用来触发SQL注入攻击的语句，ffifdyop经过md5加密之后得到：

276f722736c95d99e921722cf9ed621c

再将上述16进制数据转成字符串：

‘or’6É]é!r,ùíb

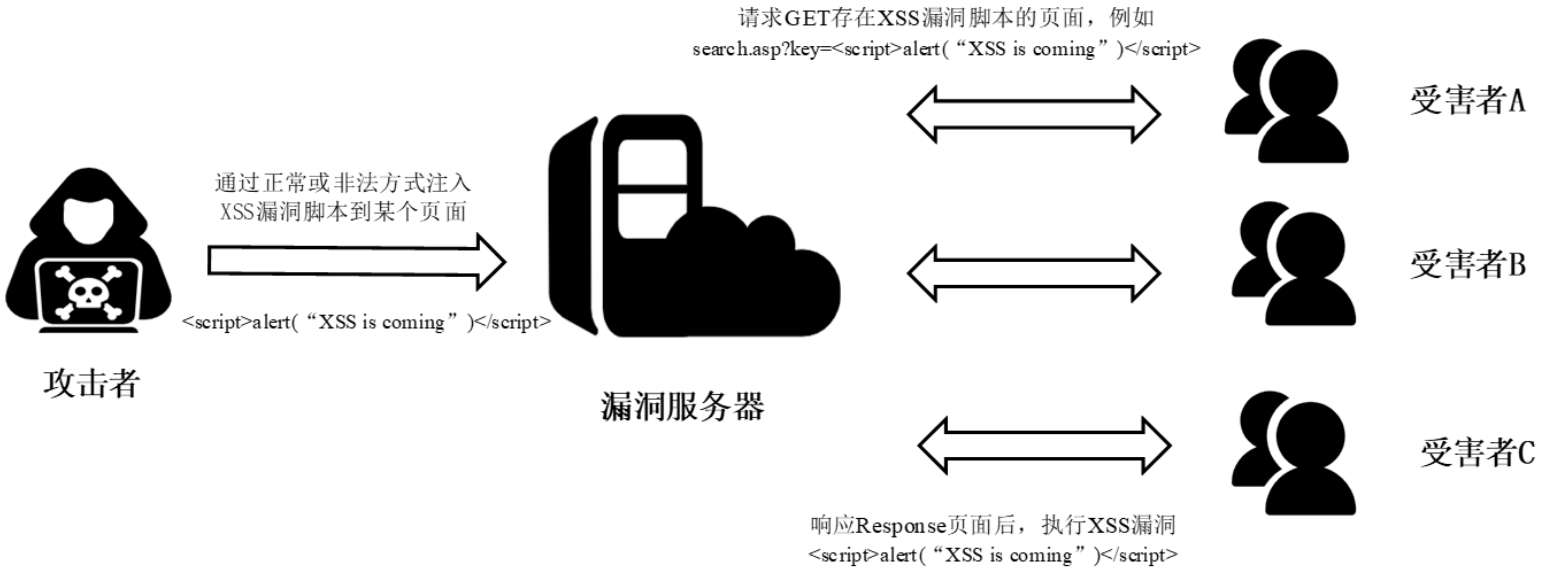
上述语句包含‘or’ 字符，因此被执行的SQL语句是：

SELECT * FROM admin WHERE pass= ‘’**or‘6É]é!r,ùíb**’

等效成 pass = ‘’ or 1，最后返回true

不安全的Web接口 - XSS跨站脚本攻击

- **定义**：XSS（cross site scripting）是一种Web安全漏洞，攻击者通过向受害者的浏览器注入恶意脚本（通常是JavaScript），从而窃取数据、劫持会话、篡改内容或者进行钓鱼攻击
- **特点**：XSS漏洞的攻击具有隐蔽性高、攻击容易发起等特点，同时还具有快速扩散能力
- **后果**：窃取信息、会话劫持、内容篡改、钓鱼攻击、散布蠕虫病毒等

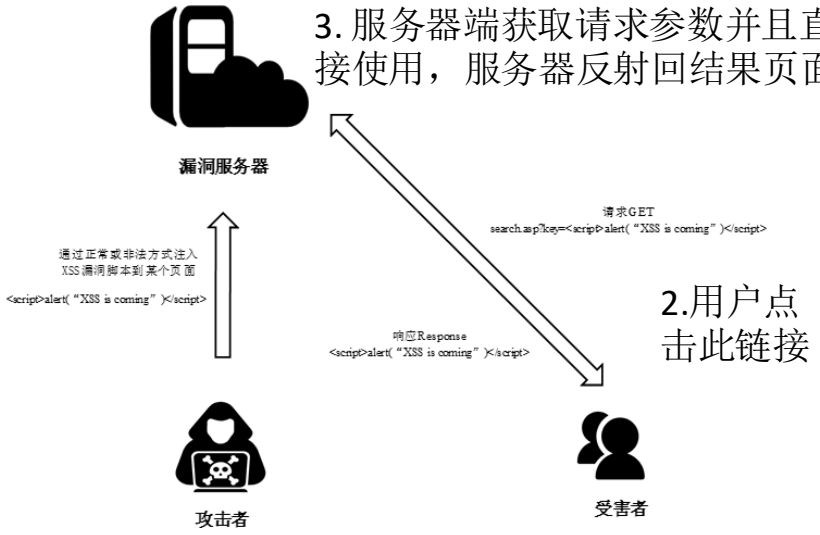


XSS跨站脚本攻击

- 根据特性和利用手法，跨站脚本攻击分为反射型和持久型跨站脚本

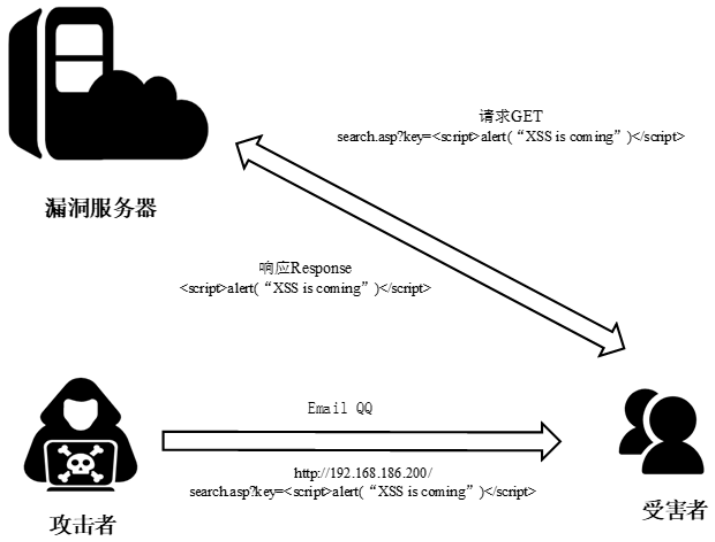
反射型XSS

- 通过诱导用户点击恶意链接来造成一次性攻击
- 攻击者注入的攻击数据反映在请求响应中



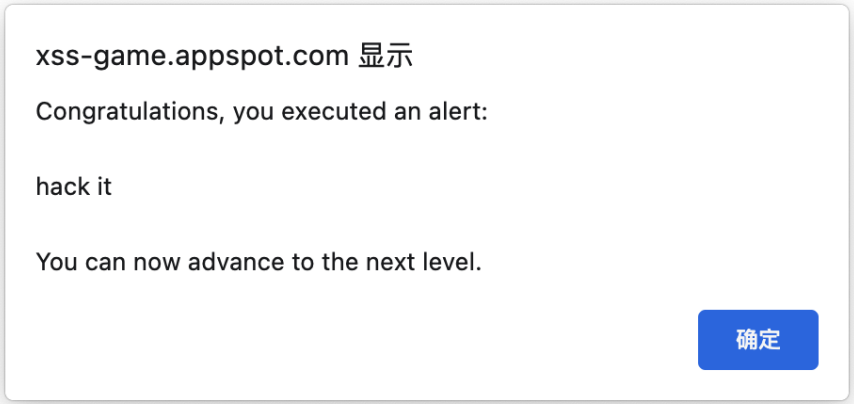
持久型XSS

- 通常攻击者将代码存储到漏洞服务器中，用户浏览相关页面时，攻击便会发生



XSS跨站脚本攻击 - Online demo

- 假设在某个网站(https://xss-game.appspot.com/)有一个查询或留言功能且存在XSS漏洞，攻击者可以在输入内容嵌入恶意Javascript代码



- Try it yourself.

- <h1 style="color: #2ecc71">You are fooled!</h1>
- <script>alert('hack it')</script>
- <script>alert(document.domain)</script>
- <style> * { background-color: #FFFF00 } </style>

Google online XSS gaming: https://xss-game.appspot.com/

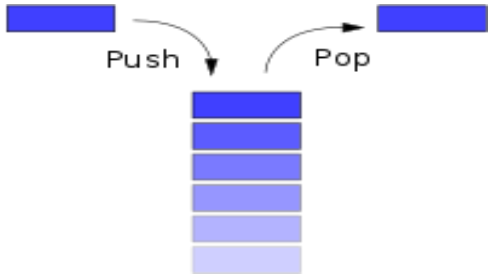
https://www.acunetix.com/blog/web-security-zone/test-xss-skills-vulnerable-sites/

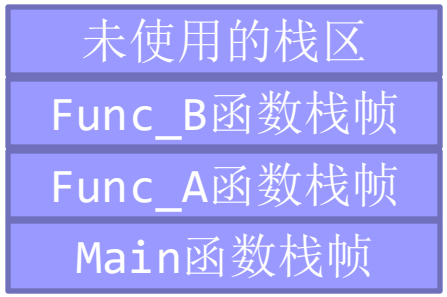
不安全的系统和程序 - 缓冲区溢出漏洞

- **定义**：利用程序设计缺陷，程序向缓冲区写入数据时超过了缓冲区的容量，导致数据覆盖了相邻内存区域。
- **结果**：破坏栈上其他变量、破坏程序运行、覆盖返回地址劫持程序控制流（将程序跳转到指定内存地址），甚至获取系统的**控制权**
- **本质原因**
  - **输入验证不足**：程序没有正确地验证或限制输入数据的大小
  - **边界检查缺失**：程序没有检查数组或缓冲区的边界
  - **不安全的函数调用**：使用了不安全的字符串和内存操作函数，如 `strcpy()`、`gets()` 等
- 一般常见的缓冲区包括**栈缓冲区**和**堆缓冲区**，本课程以栈缓存区溢出为主讲解
- 存在于各种操作系统：Windows、Linux、Free-BSD等

预备知识：堆和栈

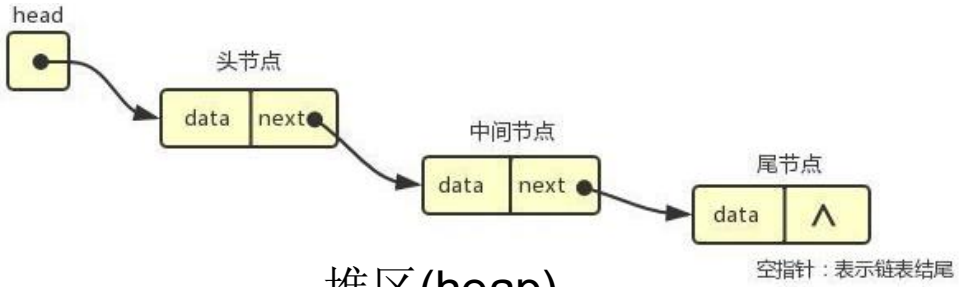
- **栈区(stack)**：由编译器自动分配和释放，存放函数的参数值、局部变量值等，用于实现高级语言函数的调用。
  - 举例：函数被调用时，系统为其分配新栈，并压入栈区，所以正在运行的函数总是在系统栈区的栈顶。
  - 特点：先入后出、后入先出。





栈区(stack)

- **堆区(heap)**：一般由程序员分配和释放，若程序员不释放，程序结束时可能由OS回收。它与数据结构中的堆不同，分配方式类似链表。



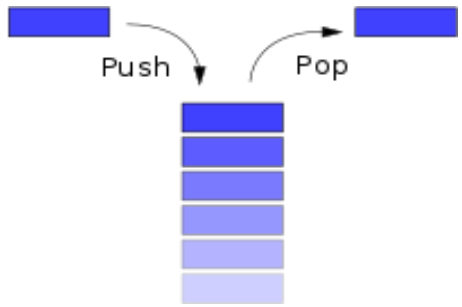
堆区(heap)

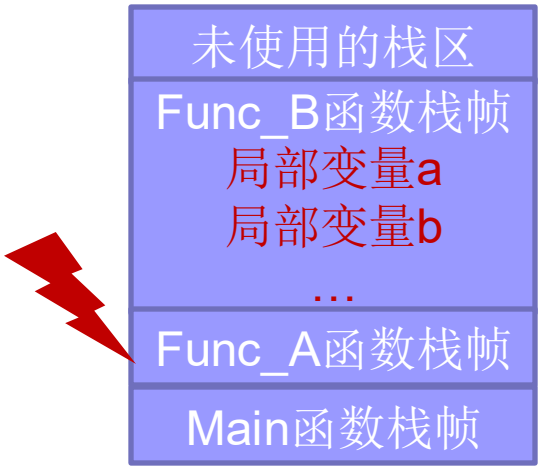
缓冲区溢出漏洞

- 函数栈中，局部变量是顺序排列的，局部变量下紧跟着前栈（如Func_A）的地址和函数返回地址

- 局部变量长度越界之后，将会覆盖相邻的局部变量、前栈（如Func_A）的地址甚至函数返回地址，造成程序异常

- 直接后果：**修改函数变量**和**修改返回地址**





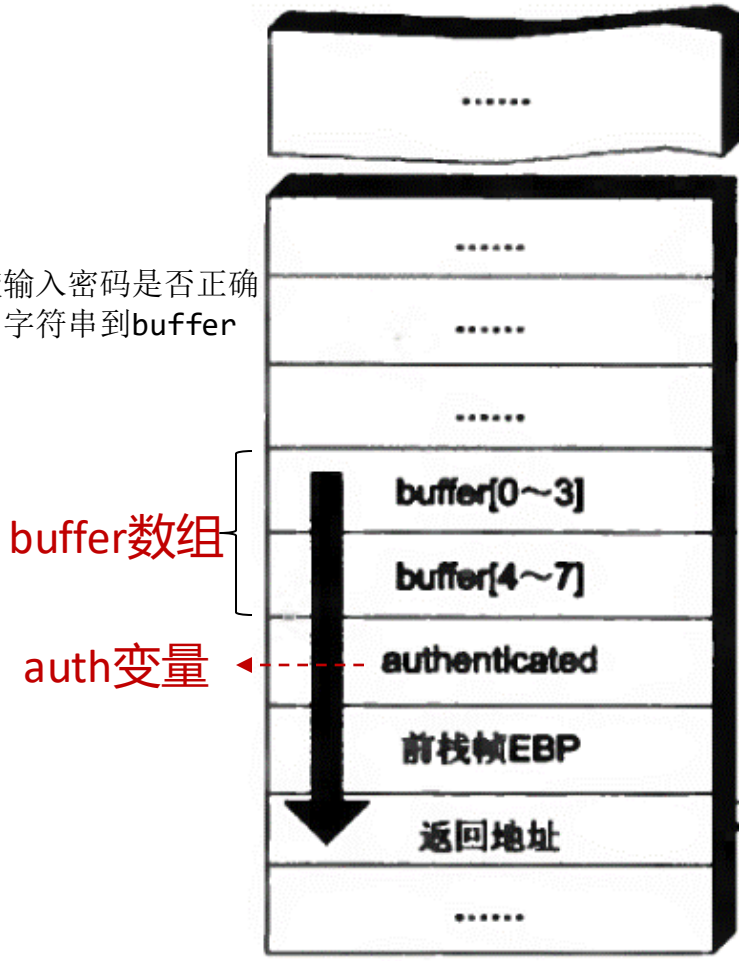
栈区(stack)

缓冲区溢出漏洞案例

分析：buffer是长度为8的char类型数组，scanf()函数不限制输入长度，strcpy()将用户输入的pass拷贝到buffer导致缓冲区溢出，与buffer相邻的auth变量值被改变为“0”

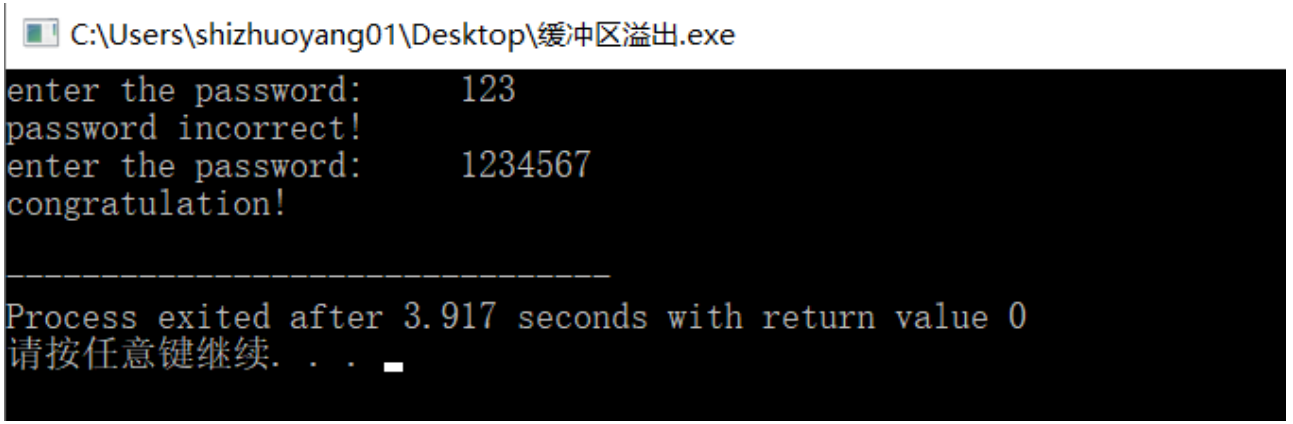
```
1  #include <stdio.h>
2  #include<string.h>
3  #define PASSWORD "1234567"
4  int verify(char *password)
5  {
6      int auth;
7      char buffer[8];
8      auth = strcmp(password, PASSWORD);
9      strcpy(buffer, password);
10     return auth;
11 }
12
13 int main(void){
14     int flag = 0;
15     char pass[1024];
16     while(1){
17         printf("enter the password:\t");
18         scanf("%s", pass);
19         flag = verify(pass);
20         if(flag)
21             printf("password incorrect!\n");
22         else{
23             printf("congratulation!\n");
24         }
25     }
26 }
```

//strcmp()比较输入密码是否正确
//strcpy()拷贝字符串到buffer



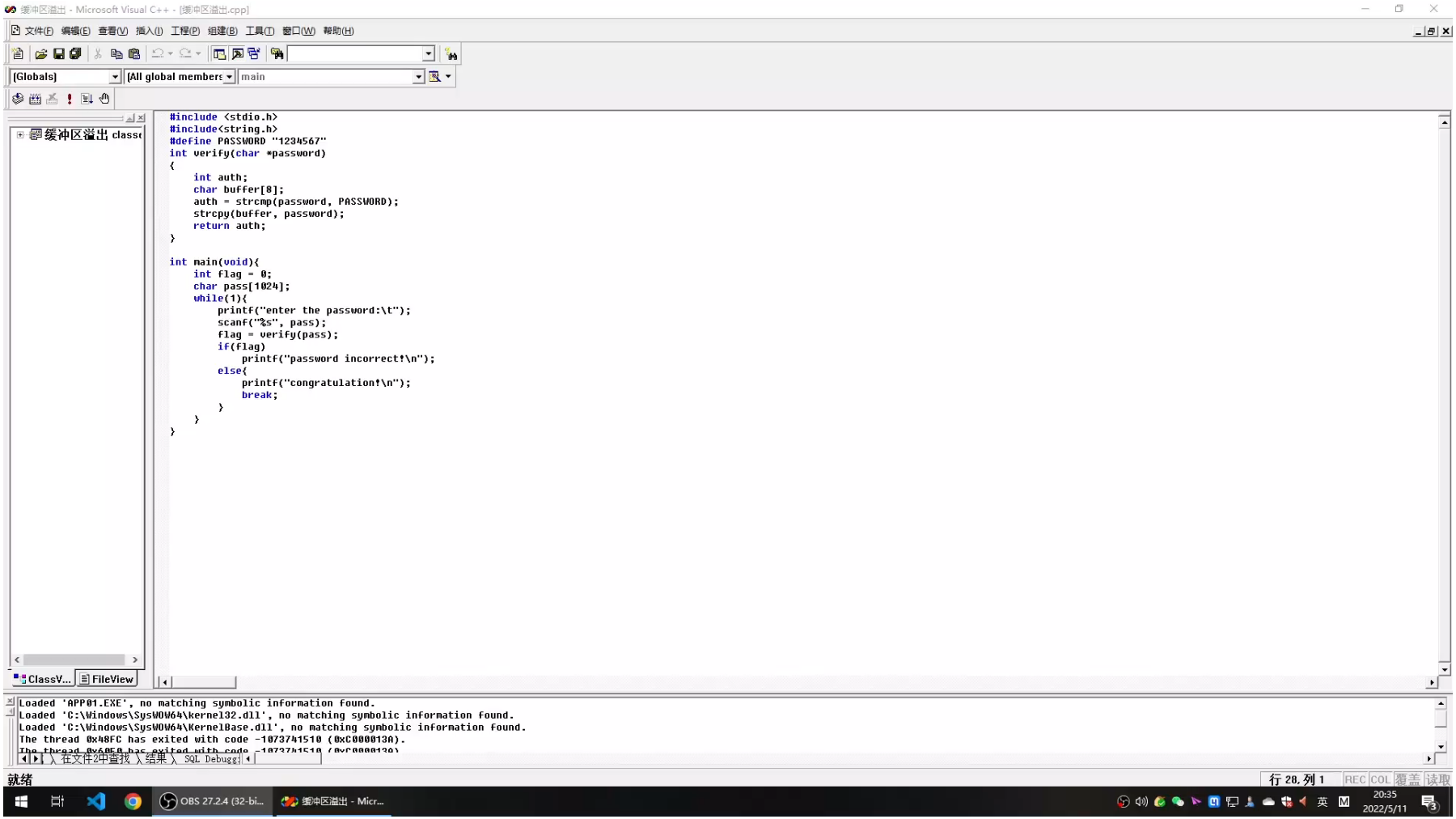
缓冲区溢出漏洞案例

- 正常情况下，输入正确password “1234567”：



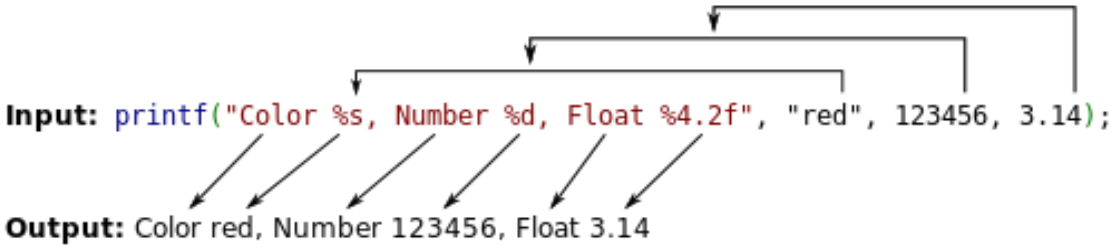
- 利用缓冲区溢出漏洞，输入错误password “qqqqqqqq”：

缓冲区溢出漏洞案例


缓冲区溢出 - Microsoft Visual C++ - [缓冲区溢出.cpp]
文件(F) 编辑(E) 查看(V) 插入(I) 工程(P) 组建(B) 工具(T) 窗口(W) 帮助(H)
(Globals)
(All global members
main
缓冲区溢出 classe
#include <stdio.h>
#include<string.h>
#define PASSWORD "1234567"
int verify(char *password)
{
int auth;
char buffer[8];
auth = strcmp(password, PASSWORD);
strcpy(buffer, password);
return auth;
}
int main(void){
int flag = 0;
char pass[1024];
while(1){
printf("enter the password:\t");
scanf("%s", pass);
flag = verify(pass);
if(flag)
printf("password incorrect!\n");
else{
printf("congratulation!\n");
break;
}
}
}
ClassV...
FileView
Loaded 'APP01.EXE', no matching symbolic information found.
Loaded 'C:\Windows\SysWOW64\kernel32.dll', no matching symbolic information found.
Loaded 'C:\Windows\SysWOW64\KernelBase.dll', no matching symbolic information found.
The thread 0x48FC has exited with code -1073741510 (0xC000013A).
在文件2中查找 结果 SQL Debugg
就绪
行 20, 列 1
REC COL 覆盖 读取
OBS 27.2.4 (32-bi...
缓冲区溢出 - Micr...
英
20:35
2022/5/11

不安全的系统和程序 - 格式化字符串漏洞

- **定义**：发生在程序在处理格式化字符串时没有对输入进行正确的验证和处理，导致攻击者可以通过精心构造的格式化字符串插入恶意指令，从而实现未授权的代码执行或其他恶意操作
- 例如，常见的printf()函数：
  - printf("<格式化字符串>", <参量表>);




```
%d - 十进制：输出十进制整数
%s - 字符串：从内存中读取字符串
%x - 十六进制：输出十六进制数
%c - 字符：输出字符
%p - 指针：指针地址
%n - 到目前为止所写的字符数
```

- 函数一般带有可变数量的参数，并将第一个参数作为格式化字符串，根据该参数来解析之后的参数。当格式化串可控时，就产生了格式化字符串漏洞
- 漏洞产生原因：**程序员偷懒**！

不安全的系统和程序 - 格式化字符串

- 格式化字符串的利用
  - **泄露内存信息：**通过 %x、%s 等格式化字符串，读取内存中的数据，包括程序的栈数据、敏感信息等
  - **覆写内存数据**：通过 %n 格式符，向特定的内存地址写入数据，从而修改程序的控制流程或数据内容。
  - **代码注入**：结合上述两种方式，攻击者可能会覆写函数返回地址或其他关键数据结构，引导程序执行恶意代码

程序展示

```
#include <stdio.h>
#include<string.h>

int main(void){
    char a[100];
    scanf("%s",a);
    printf(a);
    return 0;
}
~
~
```

程序(string.cpp)

```
xji@XiaoyudeAir:
→  Code ./string
123456
123456%
→  Code ./string
123456%d%s
123456186680595612345 6%d%s%
→  Code
```

利用格式化字符串漏洞的执行结果

不安全的系统和程序 - 格式化字符串

- 示例程序string.cpp

```c
#include <stdio.h>

int main(void)
{
        char a[100];
        scanf("%s",a);
        printf(a);
        return 0;
}
```

- 程序输入

```
AAAA%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x
```

- 程序输出

```
AAAA61fe4c,61ffcc,76e4d250,70734fbf,fffffffe,76e473da,41414141,252c7825,78252c78,2c78252c,252c7825
```

栈状态：

```
0061FE30   0061FE4C  |format = "AAAA%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x"
0061FE34   0061FE4C  |<%x> = 0x61FE4C
0061FE38   0061FFCC  |<%x> = 0x61FFCC
0061FE3C   76E4D250  |<%x> = 0x76E4D250
0061FE40   FF12BE58  |<%x> = 0xFF12BE58
0061FE44   FFFFFFFE  |<%x> = 0xFFFFFFFE
0061FE48   76E473DA  |<%x> = 0x76E473DA
0061FE4C   41414141  |<%x> = 0x41414141
0061FE50   252C7825  |<%x> = 0x252C7825
0061FE54   78252C78  |<%x> = 0x78252C78
0061FE58   2C78252C  |<%x> = 0x2C78252C
0061FE5C   252C7825  \<%x> = 0x252C7825
0061FE60   78252C78
0061FE64   2C78252C
0061FE68   252C7825
0061FE6C   78252C78
0061FE70   00000000
0061FE74   00000000
0061FE78   00000000
```

不安全的系统和程序 - 释放后重用

- **定义**：Use-After-Free，UAF漏洞是一种内存管理错误，发生在程序在释放内存(free)之后，其他程序继续使用这块内存的情形。由于这块内存可能已经被重新分配给其他程序，继续使用它可能导致不可预知的行为，如程序崩溃、数据泄露，甚至被攻击者利用来执行任意代码

- **原因**：由于在释放内存后没有将指向该内存区域的指针清空，这类释放后没有被置为null的内存指针为悬垂指针（dangling pointer）

不安全的系统和程序 - 释放后重用

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char *p1;
    p1 = (char *) malloc(sizeof(char)*10);
    memcpy(p1,"hello",10);
    printf("p1 addr: %p, content: %s\n",p1,p1);
    free(p1);

    char *p2;
    p2 = (char *)malloc(sizeof(char)*10);
    memcpy(p1,"world",10);
    printf("use-after-free.\n");
    printf("p2 addr: %p, content: %s\n",p2,p2);
    printf("p1 addr: %p, content: %s\n",p1,p1);
    return 0;
}
```

```
➜ ./uaf
p1 addr: 0x600002bd0040, content: hello
use-after-free.
p2 addr: 0x600002bd0040, content:world
p1 addr: 0x600002bd0040, content:world
```

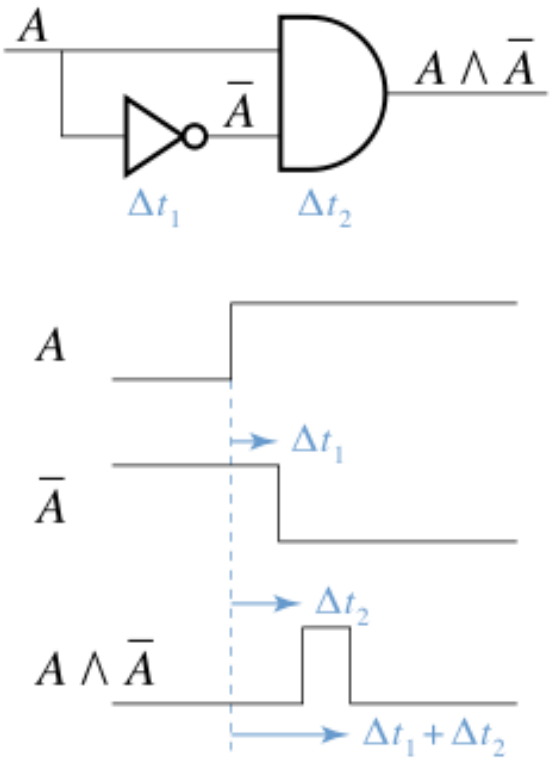
程序(uaf.cpp)及输出结果

1. 指针p1申请内存，并写入“hello”
2. 释放p1，但是没有将指针p1置为null
3. 指针p2申请同样大小内存，并写入“world”
4. p1指针释放后，p2申请相同的大小的内存，操作系统会将之前给p1的地址分配给p2，p1与p2地址相同，修改p1的值，p2也被修改
5. 后果：虽然p1已经释放，但是可以被用来修改p2的值

程序解读

不安全的系统和程序 - 条件竞争漏洞

- **定义**：是指程序的执行结果依赖于多个操作的执行顺序，而如果这些操作的执行顺序无法保证，可能导致不期望的行为
- **表现**：一个程序的运行结果依赖于不受控制的事件的先后顺序
- **原因**：操作系统大量采用并发编程，经常对资源进行共享。在多线程环境下，当一个软件运行结果依赖于进程或者线程的顺序时，就可能会出现条件竞争
- **后果**：程序异常执行乃至崩溃、获得系统特权等



逻辑电路中两个电信号的互相竞争

条件竞争漏洞案例

```c
int i = 1;
void *mythread1()
{
  if(i == 1){
    sleep(3);
    if(i == 2)
      printf("hack it!\n");
    else
      printf("you can try again!\n");
  }
}

void *mythread2()
{
    sleep(1);
    i=2;
}

int main(int argc, const char *argv[])
{
    pthread_t id1,id2;
    pthread_create(&id1, NULL, (void *)mythread1,NULL);
    pthread_create(&id2, NULL, (void *)mythread2,NULL);
    pthread_join(id1,NULL);
    pthread_join(id2,NULL);
    return 0;
}
```

```
1. 物联网安全/2024/Code via base
→ ./race
you can try again!

1. 物联网安全/2024/Code via base
→ ./race
hack it!
```

程序(race.c)及结果展示

1. 程序首先执行mythread1()，然后在判断 i 等于1后执行sleep(3)函数，也即进入了睡眠状态，而这时因为效率原因，程序不会等待mythread1()，转而去执行mythread2();
2. 在mythread2中，也执行sleep(1)，但由于时间较短，mythread2()可能比mythread1()先“醒”过来，然后对 i 进行了修改;
3. 之后mythread1()继续执行时，i 的值已经被修改为2了，于是输出”hack it!”。

程序解读

软件反逆向/软件漏洞防护/漏洞挖掘

物联网软件安全防御技术

软件安全防御

- **固件安全防护**
  - 固件防窃取
  - 固件反逆向
- **软件漏洞防护**
  - 代码规范
  - 程序保护机制
  - 漏洞挖掘

固件安全防护

- 固件安全防护包括：**固件防窃取**和**固件反逆向**

- 固件防窃取：
  - **配置字加密**：通过配置字加密方式(CRP，Code Read Protection)，将配置字(CRP Key)写入特定的Flash地址中，芯片上电启动后根据配置选择禁用固件读出功能
  - **硬件接口隐藏**：烧录接口的管脚如JTAG的相应管脚可以通过复用的方式令其被配置为普通的IO管脚使用
  - **固件加密**：通过私有密钥对整个程序进行加密计算**生成密文**，即使攻击者能获得固件程序，也无法破解出固件源代码
  - **固件更新验证**：在产品的固件升级过程中进行**加密传输和身份认证**

固件反逆向

- **防静态反汇编：**通过代码重叠、伪造虚假分支和返回地址、添加大量虚假的子函数模块等方式产生大量的虚假或无用信息，从而混淆反汇编器
  - 代码重叠技术
  - 分支跳转地址多重定向技术
  - 程序控制流混淆技术
- **防动态反汇编：**采用多种方法来增加固件逆向工程的难度
  - 代码混淆和加密
  - **反调试和反仿真技术**：如果检测到调试器，终止程序或触发错误
  - **时间和环境检查**：判断是运行在真实硬件还是仿真器
  - 硬件加密
  - ……

软件漏洞防护 - 代码规范

- 计算机安全应急响应组织CERT，提出10条安全代码规范建议：

  1. **输入验证**：验证来自所有不受信任的数据源的输入，正确的输入验证能减少大量软件漏洞

  2. **注意编译器警告**：启用编译器的警告和错误提示功能，确保不将任何一个警告带入到程序的最终编译版本中

  3. **安全策略的架构和设计**：创建软件架构来实现和增强安全策略

  4. **代码简单化**：程序应尽量短小精悍，每个函数应该具有明确功能

  5. **默认拒绝**：用户访问的默认权限应该是拒绝的

  6. **坚持最小权限原则**：每个程序都应该仅拥有完成工作所需的最小权限，并且权限的拥有时间要尽可能短

软件漏洞防护 - 代码规范

- 计算机安全应急响应组织CERT，提出10条安全编码建议：

  7. **清除发送给其它系统的数据**：清除所有发送给子系统的数据从而避免攻击者实施注入类攻击

  8. **纵深防御**：使用多种防御策略来管理风险，例如，将安全的编程技术与安全的运行环境相结合，可以减少在操作环境中利用部署时代码中残留的漏洞的可能性

  9. **使用有效的质量保证技术**：好的质量保证技术可以有效发现和清除漏洞，模糊测试、源代码审计可以作为有效的质量保证

  10. **采用安全编码标准**：为目标开发语言和平台开发或应用安全的编码标准，例如对代码进行规范缩进显示，可以有效避免出现遗漏错误分支处理的情况

软件漏洞防护 - 程序保护机制

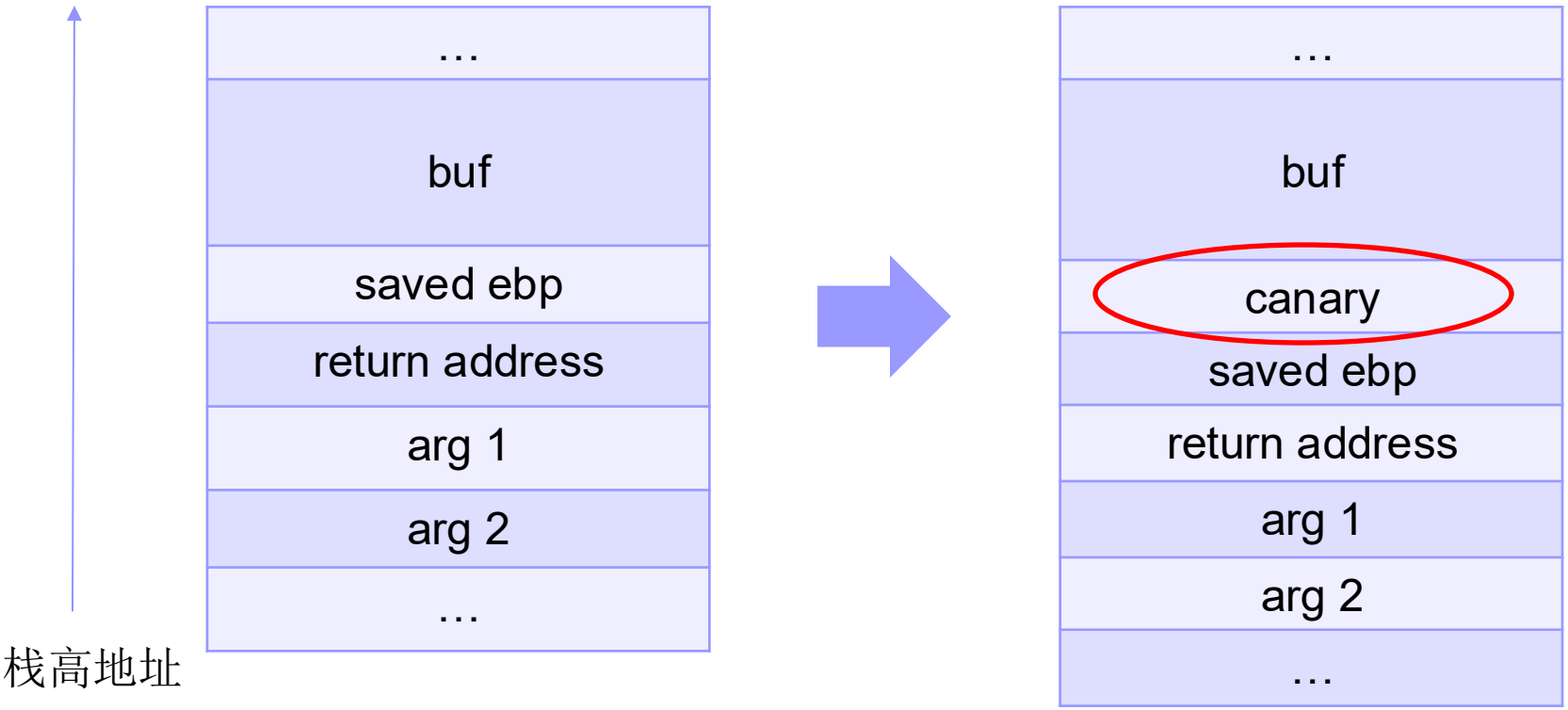
- **定义**
  - 从程序（也即软件本身）角度加入的一些特定的保护机制，有效增加攻击者漏洞利用的难度，阻止利用漏洞带来攻击
- **特点**
  - 仅针对程序本身的保护
  - 可在程序编译阶段加入不同的编译选项开启相应的保护

举例：CANARY

- **定义**：CANARY，中文“金丝雀”，又称**栈溢出保护**，是一种栈溢出攻击的缓解手段
- **原理**
  - 函数开始时先往栈里插入canary值，函数返回前验证canary是否被篡改，如果发现被篡改就报错并退出程序
  - 当函数存在栈溢出漏洞时，攻击者可以覆盖栈上的返回地址来劫持控制流。然而，插入canary后，攻击者在覆盖返回地址时也会覆盖canary，导致检查失败而阻止栈溢出攻击
- **优点**
  - 针对栈溢出攻击的有效保护手段，可以阻止栈溢出漏洞的利用

CANARY

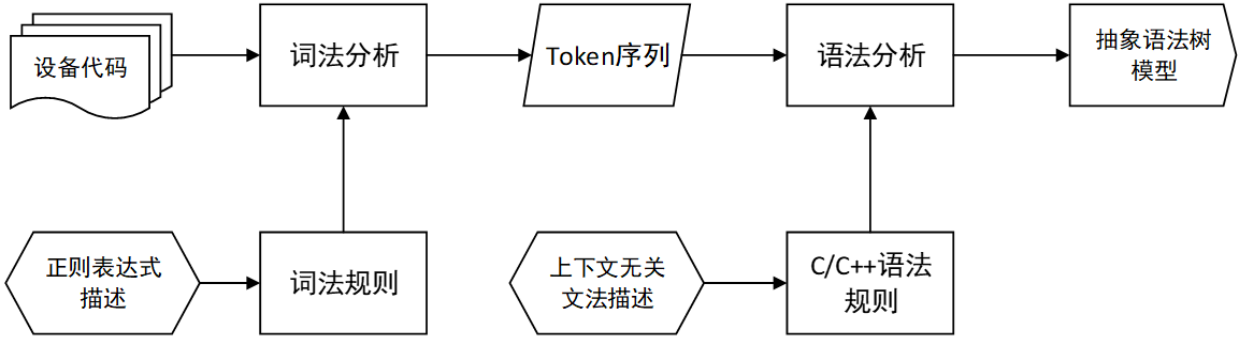
```
1.  gcc -o test test.c                          // 默认情况下，不开启 Canary 保护
2.  gcc -fno-stack-protector -o test test.c  //禁用栈保护
3.  //启用堆栈保护，不过只为局部变量中含有 char 数组的函数插入保护代码
4.  gcc -fstack-protector -o test test.c
5.  gcc -fstack-protector-all -o test test.c  //启用堆栈保护，为所有函数插入保护
    代码
```



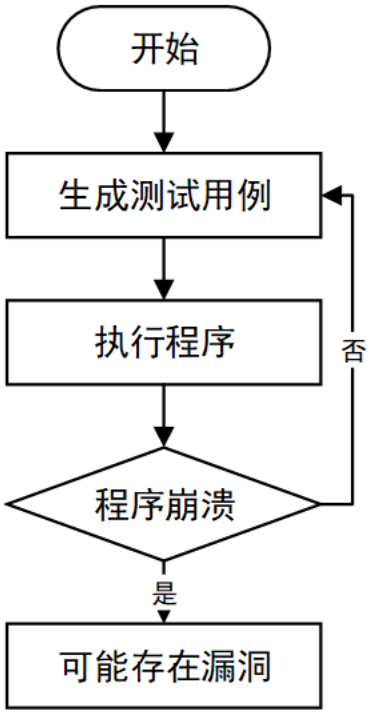
栈上没有canary时　　栈上插入canary时

软件漏洞防护 - 漏洞挖掘

- **静态代码审计**：在不执行程序的情况下对程序代码通过人工或者运用半自动化、自动化工具进行阅读分析，检测程序中可能存在安全漏洞
- **动态漏洞挖掘**：基于程序执行的漏洞挖掘方法



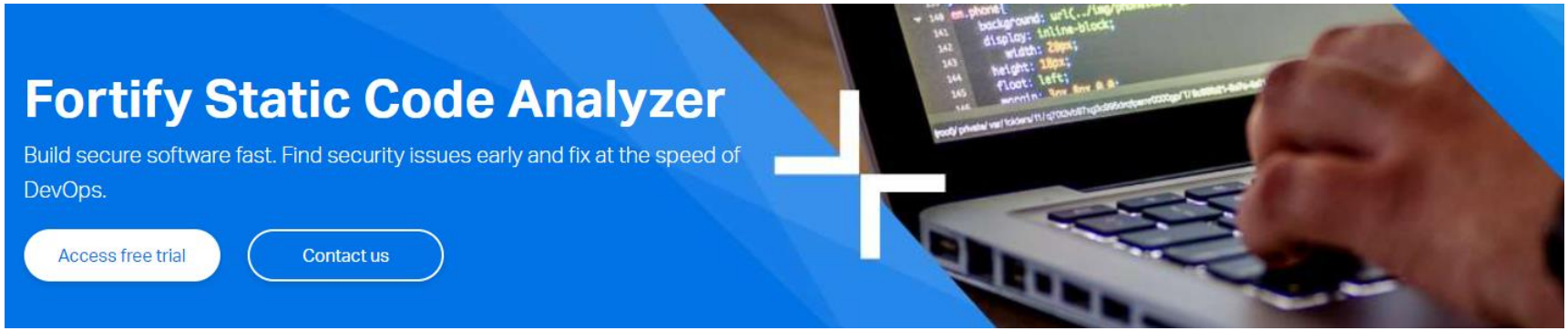
静态分析-词法分析



动态分析-模糊测试

软件漏洞挖掘工具 - FORTIFY

- FORTIFY：一种自动化代码静态分析的工具
- **作用**：用于检查是否存在缓冲区溢出等软件漏洞，可以分析程序采用的字符串或者内存操作函数，如memcpy，memset，stpcpy，strcpy，strncpy，strcat，strncat，sprintf，snprintf，vsprintf，vsnprintf，gets等
- 趋势：人工智能自动测试



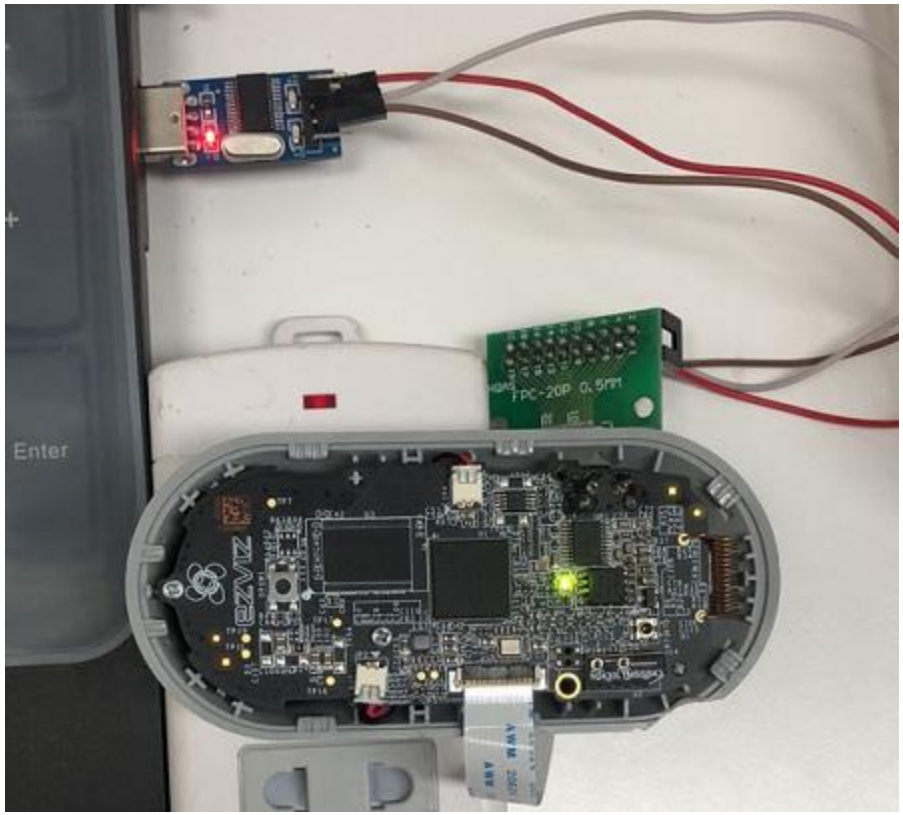
https://www.microfocus.com/en-us/products/static-code-analysis-sast/overview

Cyber Grand Challenge

- CGC(Cyber Grand Challenge)：安全大会Defcon引入的一种新的竞赛，美国DARPA(国防高等研究计划署)举办的**自动网络攻防竞赛**
- CGC旨在建立实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，来应对频发的网络攻击，并降低攻击出现到防御生效之间的时间差。
- ForAllSecure的机器人Mayhem获得了胜利，指导教授David Brumley from CMU CyLab)

https://en.wikipedia.org/wiki/2016_Cyber_Grand_Challenge

推荐阅读和操作—针对某品牌智能门锁分析



https://bbs.pediy.com/thread-261679.htm

本章小结

- 软件和固件的概念
- 物联网软件安全的内涵，包括固件逆向和软件漏洞
- 常见的软件漏洞
  - 不安全的认证和授权：弱口令
  - 不安全的Web接口：SQL注入、XSS
  - 不安全的系统和程序：缓冲区溢出、格式化字符串漏洞、条件竞争、释放后重用
- 常见的软件安全防护技术

BACKUP

传统意义上的固件

- 固件定义： Combination of a hardware device and computer instructions or computer data that reside as read-only software on the hardware device
- 固件是一类为设备提供**底层控制**的软件，包括控制、监控和数据操作功能。它是一个系统最为基础和底层的工作软件，包括BIOS、BootLoader等。
  - BIOS：basic input/ output system，基本输入输出系统，目的是初始化和测试硬件组件，从硬盘等加载BootLoader。是系统启动后加载的第一个软件
  - BootLoader：即引导程序，是计算机开机自检完成后装载操作系统或者其他系统软件的计算机程序。
- Q：你刷过机吗？

软件、固件和驱动

- **相同点**：软件、固件和驱动本质上都是二进制代码，实际由指令和数据组成，都是一种广义的“软件”或者“程序”。

- **不同点**：
  - **软件**：不同厂商开发的具有特定功能的程序，运行必须依赖于特定的计算机系统环境，对**硬件的依赖小**。
  - **固件**（为硬件提供服务）：是担任着一个系统最基础最底层工作的软件，是**固化在设备内部的软件或者驱动程序**。
  - **驱动**（为软件提供服务）：允许**上层软件和底层硬件进行交互**的程序。驱动创造硬件与硬件、硬件与软件沟通的接口，经由主板上的总线（bus）或其它沟通子系统（subsystem）与硬件形成连接的机制。
    驱动存在的原因是因为操作系统多样，使得固件无法做得通用。

- 软件的运行依赖驱动访问硬件，硬件的运行需要固件进行控制。

软件和固件的区别

- **用途**
  - 软件是为用户交互设计的程序或代码段，它是使用户能够实现所需功能的顶部代码。
  - 固件的设计目的不是让用户直接与之交互，相反，它是在设备上运行的隐藏的“最低级别”代码。
- **编程模式**
  - 软件通常用Java、Python之类的高级语言编写，有许多库函数和预制的功能以简化开发。
  - 固件通常以低级语言（如C语言）编写，几乎没有库支持，因为代码是针对单个设备量身定制的。

软件和固件的区别

- **保存位置**
  - 软件一般运行在CPU和其他主处理器上，并利用RAM和闪存存储来保存和加载数据。
  - 固件通常不是在主CPU上运行，而是在专用于各个硬件的较小处理器上运行，例如用于闪存驱动器的存储器控制器。
- **更新难易程度**
  - 软件的更新无需更改任何硬件即可进行。
  - 固件的更新比较麻烦，例如需要有线连接更新，通常用户需要在更新应固件后重启设备。

物联网设备固件的结构

- **固件头：固件头的数据信息始终是处在二进制数据流的初始部分**，表示设备固件的相关特征信息并占据一定的起始位。

- **意义：**特征字段组成不同的组合表示成该固件设备类型的**处理器平台架构、内核版本、根文件系统格式**等特征字段信息。

```
struct trx_header{
        uint32_t magic;
        uint32_t len;
        uint32_t crc32;
        uint32_t flag_version;
        uint32_t offsets[3];
}
```

trx_header的文件格式

物联网设备固件的结构

■ **引导程序部分**：BootLoader是在操作系统内核运行之前运行，负责整个系统的加载启动任务，作用是**初始化硬件设备、建立内存空间映射图，从而将系统的软硬件环境带到一个合适状态，以便为最终调用操作系统内核准备好正确的环境。**

■ **内核**是设备操作系统的**核心部分**，负责将应用程序的请求传递给硬件，并充当底层驱动程序，同时负责将各种底层硬件资源分配到各个进程，并对各个进程间的切换、调度进行管理。

物联网设备固件的结构

■ **根文件系统**是内核启动时所挂载的第一个文件系统，内核代码镜像文件就保存在根文件系统中，所以根文件系统不仅包含了普通文件系统的存储文件的功能，同时会把一些初始化脚本和服务加载到内存中去运行。

/bin	存放root权限和一般用户可以使用的命令
/sbin	存放系统命令，只有系统管理员可以使用
/dev	存放设备与设备接口的文件
/etc	存放系统主要的配置文件
/lib	存放共享库和可加载的驱动程序
/home	系统默认的用户文件夹
/root	系统管理员的主文件夹
/usr	存放共享、只读的程序和数据
/var	存放可变的数据，例如log文件、临时文件等
/proc	空目录，常作为proc文件系统的挂接点
/mnt	临时挂载某个文件系统的挂接点，通常是空目录
/tmp	用于存放临时文件，通常是空目录

根文件系统目录