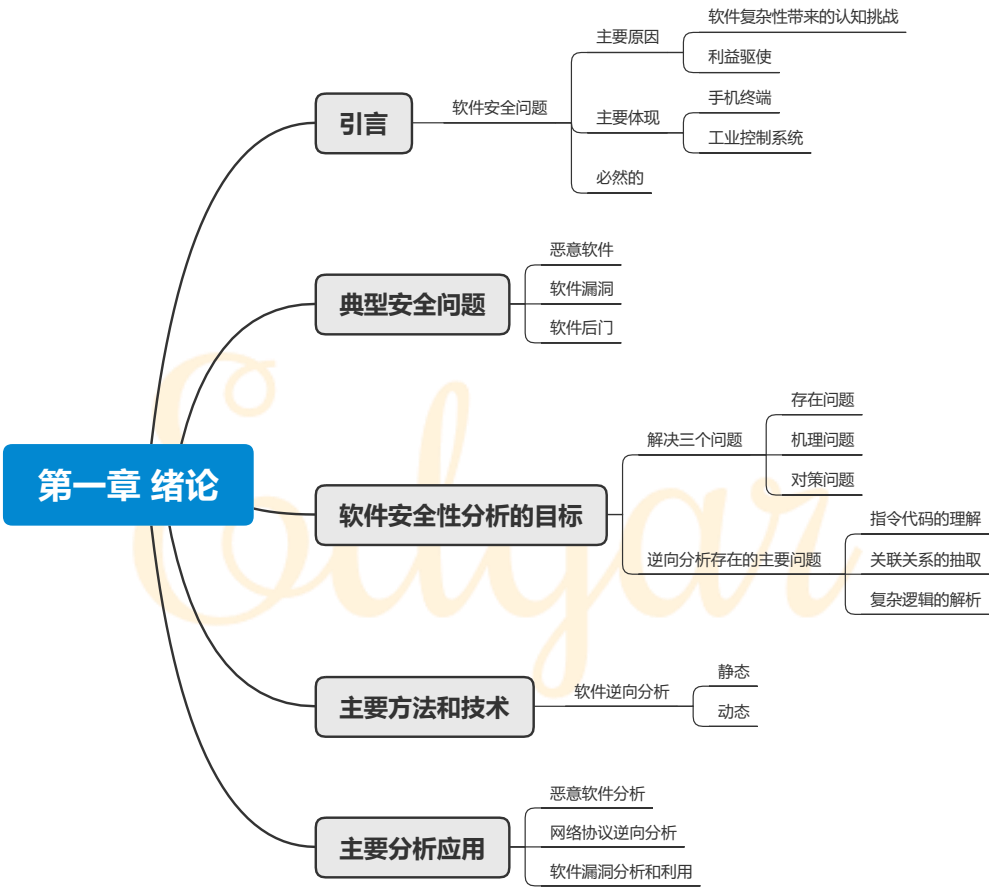
第一章 绪论

一. 章节主要内容



二. 详细内容

1.1 引言

软件是信息系统的“灵魂”。

一方面由于技术和应用的快速发展，**软件自身的复杂性**增加了人类对认知的挑战，不可避免的引入各种程序漏洞；另一方面，当软件产品承载过多利益时，个人或者组织**容易受到利益的驱使**，留下软件后门。软件安全问题虽然是人为的，但是却也是不可避免的。

1.2 典型的安全问题

软件安全问题大致可以分为三类：**恶意软件、软件漏洞、软件后门**

1.2.1 恶意软件

✓ **定义**：包含恶意功能的软件。

✓ 传统类恶意软件：

✧ 主要是指**病毒、木马、蠕虫、僵尸网络、间谍软件**等，其共同特点是在用户不知情的情况下执行一系列的**破坏功能**，或窃取信息，或远程控制、或实施破坏等。

✧ 依靠发展的能力：

1. **渗透和扩散能力**：如何突破相关的防御机制，感染既定的目标系统。
2. **隐蔽能力**：如何有效隐蔽自身的各种特征，避免被用户察觉或被相关的安全检测工具发现，也包括在被发现的情况下，如何防止自身被进一步的分析，保护恶意软件的操控者身份。

3. **破坏能力**：具体如何搜集信息或实施破坏等。

✓ 恶意软件发展的三个阶段：

1. 单机传播阶段
2. 网络传播阶段
3. 协同攻击阶段

✓ 恶意软件用途：

1. 被黑客及部分民间技术爱好者用于窃取虚拟财产、个人隐私信息以牟利。
2. 某些国家和组织用来实施破坏或获取情报。

✓ 恶意软件是实施**主动攻击、情报搜集**的重要手段。

✓ 高可持续性威胁(Advanced Persistent Thread)：简称 APT 攻击，针对特定目标，采取高技术手段。

1.2.2 软件漏洞

✓ 定义：由于程序设置实现错误造成的软件问题。

✓ 彻底消除的困难性：

1. 软件自身越来越复杂
2. 软件漏洞越来越多样化
3. 软件开发周期越来越短，造成现实中的许多漏洞

✓ **JPEG 图像格式**：

✧ 子结构：“2B 的标志码+2B 的长度域+信息数据”

1. 起始标志： FFD8 SOI(Start of Image)
2. 终止标志： FFD9 EOI(End of Image)
3. 更多结构如下图：

Common JPEG markers[51]

Short name	Bytes	Payload	Name	Comments
SOI	0xFF, 0xD8	*none*	Start Of Image	
SOF0	0xFF, 0xC0	*variable size*	Start Of Frame (baseline DCT)	Indicates that this is a baseline DCT-based JPEG, and specifies the width, height, number of components, and component subsampling (e.g., 4:2:0).
SOF2	0xFF, 0xC2	*variable size*	Start Of Frame (progressive DCT)	Indicates that this is a progressive DCT-based JPEG, and specifies the width, height, number of components, and component subsampling (e.g., 4:2:0).
DHT	0xFF, 0xC4	*variable size*	Define Huffman Table(s)	Specifies one or more Huffman tables.
DQT	0xFF, 0xDB	*variable size*	Define Quantization Table(s)	Specifies one or more quantization tables.
DRI	0xFF, 0xDD	4 bytes	Define Restart Interval	Specifies the interval between RST*n* markers, in Minimum Coded Units (MCUs). This marker is followed by two bytes indicating the fixed size so it can be treated like any other variable size segment.
SOS	0xFF, 0xDA	*variable size*	Start Of Scan	Begins a top-to-bottom scan of the image. In baseline DCT JPEG images, there is generally a single scan. Progressive DCT JPEG images usually contain multiple scans. This marker specifies which slice of data it will contain, and is immediately followed by entropy-coded data.
RST*n*	0xFF, 0xD*n* (*n*=0..7)	*none*	Restart	Inserted every *r* macroblocks, where *r* is the restart interval set by a DRI marker. Not used if there was no DRI marker. The low three bits of the marker code cycle in value from 0 to 7.
APP*n*	0xFF, 0xE*n*	*variable size*	Application-specific	For example, an Exif JPEG file uses an APP1 marker to store metadata, laid out in a structure based closely on TIFF.
COM	0xFF, 0xFE	*variable size*	Comment	Contains a text comment.
EOI	0xFF, 0xD9	*none*	End Of Image	

From wiki

✧ 通过静心构造的 shellcode，栈溢出的时候便可以执行 shellcode 中的内容。

1.2.3 软件后门

- ✓ 定义：软件开发人员有意设计，刻意对用户隐瞒的一些功能，往往这些功能用于产品应用之后的一些特殊目的。
- ✓ 以软件后门设计为软件漏洞：
  1. 难发现
  2. 易利用
  3. 难取证

1.3 软件安全性分析

- ✓ 三大问题：
  1. 存在问题：目标软件中是否存在恶意功能，是否存在漏洞或者后门。
  2. 机理问题：确定问题存在之后，进一步分析具体是如何实现的或者是什么原因造成的。
  3. 对策问题：根据其相关机理分析结果，提出相应的防御对策。
- ✓ 三大挑战：
  1. 指令代码的理解
  2. 关联关系的抽取
  3. 复杂逻辑的解析

1.4 主要方法和技术

- 按照分析方式：

✓ **静态分析**

✧ 定义：直接对软件的可执行代码进行分析，一般是在对代码反汇编或反编译的基础上，对汇编代码或其他高级语言代码进行进一步的分析

✧ 优势：可以对软件代码进行较为全面的整体性分析

✧ 缺点：难以分析较大规模，复杂的软件代码；无法处理无法反汇编或反编译的软件

✓ **动态分析**

✧ 定义：通过直接运行软件，然后检测软件运行过程，执行后的结果数据，减少分析中的推理分析过程。

✧ 优势：分析过程中可以根据软件的运行直接获得在各个指令执行后的结果数据，减少分析中的推理分析过程，分析过程的复杂度更低，准确性更高。

✧ 缺点：每次分析只能针对动态执行额一条路径进行，分析的全面性较差。

✓ 目前的**主流思路**以动态分析为主，一方面利用模糊测试等技术构造执行的不同路径，另一方面也利用静态分析手段弥补动态分析过程中的不足，优化和提升动态分析的能力。

➢ 按照获取信息方式：

✓ **反汇编与反编译**

✧ 定义：

反汇编是汇编的逆过程，即将可执行代码转化成可读的汇编代码；反编译就是将机器码、汇编代码转换成高级语言的逆过程

✧ 存在的问题：

逆向分析的时候并不能完全恢复出设计阶段的高级语言代码。

✓ **程序调试**

✧ 定义：通过实际运行软件，利用断点、单步执行等方式对软件执行过程进行细粒度分析的过程。

✧ 一般性的应用软件调试可借助 WinDbg 等工具，其基本原理是利用 CPU 的调试功能，将应用软件在调试模式下运行。

✧ 对于操作系统驱动程序等内核代码的调试工具，需要借助 SoftICE 或者 Vmware+WinDbg 等调试工具，调试过程中会阻塞整个操作系统的继续运行。

✧ 优点：可以避免用户对指令的复杂推理分析过程，可以直接对每条指令的执行结果进行分析。

✧ 缺点：每次分析只能分析一条执行路径；部分软件可能会阻碍调试；调试只能够提供动态的细节信息，其分析理解难度仍然很大

✓ **程序切片**

✧ 定义：主要通过分析程序代码之中的依赖关系来分析指令的相关性，从而帮助用户提取其所“感兴趣”的代码片段。

✧ 能够减少其他无关代码的影响，是解决超大规模软件理解困境的一种思路

✧ 主要用于代码的静态分析，也可以适用于动态分析场景

✧ 适用于局部代码片段，不太适用于大规模代码分析

✓ **污点传播分析**

✧ 基本思想：将所感兴趣的数据做标记(如同放射性标记),即将数据标记成污点数据，然后通过分析对该污点数据的处理过程，根据每条指令的污点传播规则，分析数据的传递关系。数据传递、扩散的过程就是污点传播过程。

✧ 污点传播分析一般使用动态分析方式。

✧ 获取动态执行过程中的每一条指令和指令执行前后状态的方式：基于插桩，基于硬件，基于编译器扩展和基于硬件模拟器等

✧ 代表性系统：TEMU、DECAF、AOTA 等

✧ 存在的问题：在实际分析过程中面临的隐式污点传播问题，由于控制依赖、查表操作等引入的隐式污点传播无法简单地引入或去除，直接影响到了分析的准确性。

✓ **符号执行**

✧ 基本思想：将目标程序代码中部分变量和运算符号化，通过对各种条件分支的符号化表达来形成路径的约束条件。

✧ 分析程序内部逻辑的基础方法，常用于路径约束的分析

- ✧ 符号执行也是一种数据流分析方法，其基本思想是：用符号变量作为输入参数，对程序进行模拟执行，然后对程序的执行路径进行分析。
- ✧ 传统的符号执行使用静态分析方法，精度高，但测试效率低。后来采用动态符号执行方式，并针对符号执行中路径爆炸等问题，采用符号执行、选择符号执行等方式。

✓ **模糊测试**

- ✧ 基本思想：通过构造不同的输入数据，尽可能地触发执行软件的各种路径，通过对执行结果的检测来实现相关的分析或者检测目标。
- ✧ 主要用途：尽可能多的触发软件的各种执行路径
- ✧ 代表性工具：Peach，Sulley 等
- ✧ 当前研究重点：如何提供测试数据的针对性，提高模糊测试的效率。

1.5 主要分析和应用

1.5.1 恶意软件分析

- ✧ 实现目标：分析其主要功能，评估危害；分析关键代码和关键行为，提取代码或行为特征，以更新杀毒软件、入侵检测系统等防御系统配置，实现对该恶意软件的防御能力；分析实现机理，研发恶意软件清除手段。

1.5.2 网络协议逆向分析

- ✧ 分析方式：基于网络流量统计特征；通过逆向进行分析

1.5.3 软件漏洞分析与利用

- ✧ 内容：对软件漏洞形成机理的分析，对其可利用性的评估，以及潜在利用路径的分析和设计
- ✧ 提高自动化分析的能力