**计算机系统中的安全**

# 什么是安全

时至今日，计算机安全在经济运行、国防以及个人隐私中的核心地位已经得到了广泛的共识。计算机安全的重要性与互联网的发展密不可分，自20世纪90年代开始，随着互联网日益深入到人类活动的各个角落，计算机安全的重要性也随之日益增长，也越来越被一般人所关注。

在计算机诞生之时，有关的安全问题就开始被人所关注。在个人电脑普及前，计算机主要以大型机的形式应用于军事领域、银行系统以及大型企业的业务中。此时计算机中就已经保存了大量的机密数据，然而计算机安全并不像今天这样是一个重要的研究领域。这是因为这时安全还可以用老派的手段来保证，例如将机柜锁在安全的房间中保证物理隔离，只允许被认证的人员操作，并通过背景调查等筛选程序来保证操作人员的可靠性。即使是到了个人电脑刚刚出现时，类似的方法仍然能有效地保护计算机的安全。

然而互联网的出现极大地改变了这一情况。从1972年因特网的雏形ARPANET出现开始，计算机越来越多地连接在了一起。今天，高度敏感的经济、金融、军事和个人信息在跨越国家、政府、企业、组织和个人的全球网络中进行存储和处理。在很多领域中，计算机与互联网的连接已经不可避免，数据的存储也倾向于分布式地存放于多个数据中心之中。在这种环境下，通过物理隔离来维护计算机安全的方式已经不再适用，需要开发更加适应时代的方法来保证计算机以及信息的安全。

## 计算机安全概念

NIST的《计算机安全手册》[1]针对计算机安全这一术语的定义如下：对于一个自动化的信息系统，采取保护措施确保信息系统资源（包括硬件、软件、固件、信息/数据和通信）的完整性、可用性和保密性。

这个定义引入了处于计算机安全核心地位的三个关键目标：

* **保密性（Confidentiality）：**保密性包括两个相关的概念：数据保密性和隐私性。数据保密性确保隐私或者私密信息不向非授权者泄露，也不被非授权者所使用；隐私性确保个人能够控制或确定与其自身相关的哪些信息是可以被收集、被保存的，这些信息可以由谁来公开以及向谁公开。
* **完整性（Integrity）：**完整性包括两个相关的概念：数据完整性和系统完整性。数据完整性确保信息和程序只能以特定和授权的方式进行改变；系统完整性确保系统能以一种正常的方式来执行预定的功能，免于有意或者无意的未授权操纵。
* **可用性（Avalability）：**确保系统能够工作迅速，对授权用户不能拒绝服务。

这三个概念共同构成了被称为“CIA”的三元组。这三个概念体现了数据、信息和计算服务的基本安全目标。例如，NIST标准FIPS 199（《联邦信息和信息系统安全分类标准》）将保密性、完整性和可用性作为信息和信息系统的三个安全目标。FIPS 199从安全需求和安全确实的角度对三个目标进行刻画。

* **保密性：**对信息的访问和公开进行授权限制，包括保护个人隐私和秘密信息。保密性缺失的定义是信息的非授权泄露。
* **完整性：**防止对信息的不恰当修改或破坏，包括确保信息的不可否认性和真实性。完整性缺失的定义是对信息的非授权修改和毁坏。
* **可用性：**确保对信息的及时和可靠的访问与使用。可用性的缺失是对信息和信息系统访问与使用的中断。

CIA三元组在大多数情况下足以定义安全目标，但是许多从事安全领域研究的人认为还需要其他概念来定义才更全面。下面是其中两个被提及较多的概念。

* **真实性（Authenticity）：**真实性指一个实体是真实的、可被验证的和可被信任的；对传输信息来说，真实性指信息和信息的来源是正确地。也就是说，能够验证那个用户是否是他所声称的那个人，以及系统的每个输入是否均来自可信任的信源。
* **可追溯性（Accountability）：**这一安全目标要求实体的行为可以追溯到该实体。这一属性支持不可否认性、阻止、故障隔离、入侵检测和预防、事后恢复，以及法律诉讼等方面。因为无法得到绝对安全的系统，我们必须把安全泄露追查到负有责任的一方。系统必须保留实体的活动记录，以便事后的审计分析来追踪安全事件或者解决争端。

## 安全的作用领域

从1946年ENIAC诞生到今天，计算机已经发展了超过半个世纪。从最初的大型机到今天移动设备的普及，计算机之间的相互联系越来越多。在这种背景下，考察计算机安全不能将计算机从互联网络中孤立开来，不仅需要考虑单个计算机如何维持自身安全，同时也需要考虑计算机与计算机交互的过程中如何达到安全，以及多个计算机构成的整体要如何达到安全。在考虑了所有这些因素后，才能准确地为计算机安全划定作用领域。

下图是对计算机安全作用领域的一个简单划分[2]：

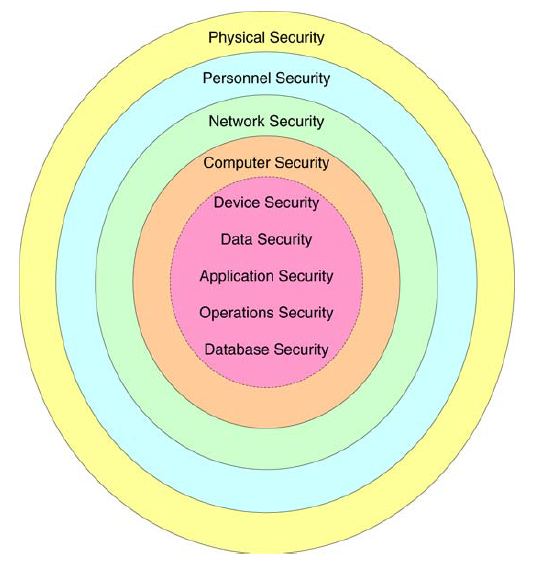


图 1 安全的作用领域

上图中，每一个环表示需要在对应的安全区域上提供特定的安全措施。需要注意的是不同环之间并不是完全的层次化关系，在构建一个安全的信息系统时，不同的安全区域之间存在大量的相互依赖关系。每一个安全区域要针对一种特定的需求，采取相应的措施来防止对重要资产的不当访问或使用。

物理安全（Physical Security）控制对信息系统及其组件的物理接触，需要管理的对象包括存储、传输、处理数据的设备，存放这些设备的房间、办公室等地点，以及设备间通信时使用的电缆等传输介质。物理安全领域中所采取的措施与传统的安全措施很接近，保证信息系统能够运行在一个安全的物理环境中。

人员安全（Personnel Security）首先要保证的是接触信息系统的人员是可以信任的，其次需要制定安全相关的规范、流程以及录用标准等规章制度。

网络安全（Network Security）关注的是互联以及数据传输过程中的安全问题。网络安全要求数据在传输过程中不会被篡改、挪用或销毁，同时还要求联网的设备被正确地管理、监控以及仅应用于许可的活动中。

计算机安全（Computer Security）在这里指单个计算机的安全，包括了硬件、软件、使用、管理等各方面的内容。计算机安全需要保证这些内容不会被篡改、误用、破坏，且仅应用于许可的活动之中。

在这些作用领域中，物理安全和人员安全与计算机安全研究的关系并不密切。计算机安全研究主要关注的是网络安全与计算机体系结构、计算机应用中的安全。接下来将分别介绍这两方面内容，并着重介绍计算机安全体系结构。

# 网络安全——OSI安全框架

在网络安全方面，国际电信联盟（ITU）电信标准化组（ITU-T）提出了推荐方案X.800，即OSI安全框架。ITU-T是一个联合国资助的机构，主要职责是开发与长途通信、开放系统互连（OSI）有关的各种标准。OSI安全框架是在OSI协议框架范围内开发的，它给出了一种系统化定义安全的方法。对安全人员来说，OSI安全框架是提供安全的一种组织方法。同时，因为这个框架是作为国际标准而开发的，所以许多计算机和通信的服务商已经开发了与OSI框架的安全特性相适应的产品和服务。

OSI安全框架主要关注安全攻击、安全机制和安全服务。这三点可以简短地定义如下：

* **安全攻击：**任何危及信息系统安全的行为
* **安全机制：**用来检测、阻止攻击或者从攻击状态恢复到正常状态的过程（或实现该过程的设备）
* **安全服务：**加强数据处理系统和信息传输安全性的一种处理过程或通信服务。其目的在于利用一种或多种安全机制进行反攻击。

## 安全攻击

X.800将安全攻击分为被动攻击与主动攻击两类。被动攻击试图了解或利用系统的信息但不影响系统资源；主动攻击试图改变系统资源或影响系统运作。

### 被动攻击

网络安全中被动攻击的形式是对传输进行窃听和监测。攻击者的目标是获得传输的信息。信息内容的泄露和流量分析就是两种典型的被动攻击。

信息内容泄露攻击很容易理解。电话、电子邮件信息和传输的文件都有可能含有敏感或秘密的信息。我们希望能阻止攻击者获得传输的内容。

另一种攻击是流量分析。假设我们已有一种方法来隐藏消息内容或其他信息流量，使得攻击者即使捕获了消息也无法从消息中获得传输的内容。在这种情况下，攻击者虽然无法直接获得传输的消息内容，但是可以通过捕获消息来获得消息传输过程中呈现的模式。利用这些模式信息，攻击者可以确定通信主机的身份和位置，可以观察传输消息的频率和长度。这些信息可以用于判断通信的性质。

被动攻击由于不涉及对数据的修改，所以很难察觉。典型的情况是，信息流表面上以一种常规的方式在收发，但是收发双方都不知道有第三方已经读了信息或者观察了流量模式。阻止这种攻击的通常手段是对传输进行加密，包括对内容的加密以及对流量模式的加密。处理被动攻击的重点是预防，而不是检测。

### 主动攻击

主动攻击包括对数据流进行修改或伪造数据流，可以分为四类：伪装、重播、消息修改和拒绝服务。

伪装是指某实体假装自己是其他实体。伪装攻击通常还包含其他形式的主动攻击，例如：捕获认证消息，并在真的认证消息之后进行重播，使得没有权限的实体通过冒充有权限的实体获得了额外的权限。

重播是指将获得的消息再次发送以产生非授权的效果。

消息修改指修改合法消息的一部分、延迟消息的传输或改变消息的顺序以获得非授权的效果。例如，将消息“Allow John Smith to read confidential file accounts”修改为“Allow Fred Brown to read confidential accounts”。

拒绝服务阻止或禁止对通信设施的正常使用或者管理。这种攻击可能有具体的目标。例如，某实体可能会阻止所有发向某目的地（如安全审计服务）的消息。拒绝服务的另一种形式是破坏整个网络，它或者是使网络失效，或者是使其过载以降低其性能。

由于物理通信设施、软件和网络本身潜在弱点的多样性，主动攻击难以绝对预防，但相对被动攻击更容易检测。因此对于主动攻击，重点在于检测并从破坏或延迟中恢复过来。另一方面，由于检测主动攻击有一定的威慑效果，所以检测机制的存在也可以在某种程度上阻止主动攻击。

## 安全服务

X.800框架将安全服务定义为通信开放系统的协议层提供的服务，其目的是保证系统或者数据传输有足够的安全性。RFC 2828（互联网安全术语表）[3]对此的定义为：安全服务是一种由系统提供的对系统资源进行特殊保护的处理或通信服务；安全服务通过安全机制来实现安全策略。

X.800将安全服务分为5类共14个特定服务。下面将分类讨论。

### 认证

认证服务与保证通信的真实性有关。在单条消息的情况下，如一条警告或报警信号，认证服务是向接收方保证消息来自所声称的发送方。对于正在进行的交互，如终端和主机的连接，就涉及两个方面。首先，在连接建立时，认证服务保证两个实体都是他们所声称的实体；其次，认证服务必须保证该连接不受第三方干扰，即第三方不能伪装成两个合法实体中的一个进行非授权传输或接收。

X.800在认证服务中定义了两种特定服务：

* **同等实体认证：**为连接中的同等实体提供身份确认。如果处于不同系统中的两个实体实行相同的协议，则考虑他们为对等的，例如位于两个通信系统中的两个TCP模块。对等实体认证用于连接的建立或数据传输阶段。该服务想提供的保证是：一个实体没有试图进行伪装，或对以前的连接进行非授权重播。
* **数据源认证：**为数据的来源提供确认，但对数据的复制或修改并不提供保护。这种服务支持电子邮件这样的应用，在这种应用的背景下，通信实体之间并没有预先的交互。

### 访问控制

在网络安全中，访问控制是一种限制和控制那些通过通信连接对主机和应用进行访问的能力。访问控制阻止对资源的非授权使用，即这项服务控制谁能访问资源，在什么条件下可以访问，以及这些被访问的资源可用于做什么。为此，每个试图获得访问控制的实体必须被识别或认证后才能获得其相应的访问权限。

### 数据保密性

保密性的目的是防止传输的数据遭到被动攻击。关于数据，可以有几层保护。最广泛的服务在一段时间内为两个用户间所传输的所有用户数据提供保护。例如，如果两个系统间建立了TCP连接，则这种广泛的保护将防止在TCP连接上传输的任何用户数据的泄露。也可以定义一种较窄的保密性服务，可以使对单条消息或对单条消息内某个特定的范围提供保护。

保密性的另一个方面是防止流量分析。这要求攻击者不能观察到消息的发送者和接受者、频率、长度，或通信设施上的其他流量特征。

X.800为数据保密性定义了4种特定服务：

* **连接保密性：**保护一次连接中所有的用户数据
* **无连接保密性：**保护单个数据块中的所有用户数据
* **选择域保密性：**对一次连接或单个数据块中指定的数据部分提供保密
* **流量保密性：**保护可以通过观察流量而获得的信息

### 数据完整性

与保密性类似，完整性可以应用于消息流、单条消息或消息的指定部分。同样，最有效和直接的方法是为整个数据流提供完整性保护。

用于处理信息流、面向连接的完整性服务保证收到的消息和发出的消息一直，没有复制、插入、更改顺序或重播。该服务也涉及对数据的破坏。因此，面向连接的完整性服务处理消息流的修改和拒绝服务两个问题。另一方面，无连接的完整性服务，仅仅处理单条消息，而不管大量的上下文信息，其通常仅仅防止对单条消息的修改。

除去面向有连接和无连接的区别，我们还可以区分有恢复和无恢复的服务。因为完整性服务和主动攻击有关，所以在完整性服务中我们更关心检测而不是阻止攻击。如果检测到完整性被破坏，那么服务可以简单地报告这种破坏，并通过软件的其他部分或人工干预来恢复被破坏的部分。

X.800定义了5种特定的数据完整性服务：

* **具有恢复功能的连接完整性：**保护一次连接中所有用户数据的完整性。检测整个数据序列内存在的修改、插入、删除或重播，且试图恢复。
* **无恢复的连接完整性：**保护一次连接中所有用户数据的完整性。检测整个数据序列内存在的修改、插入、删除或重播，但仅提供检测，无恢复。
* **选择域连接完整性：**保护一次连接中传输的单个数据块内用户数据指定部分的完整性，并判断指定部分是否有修改、插入、删除或重播。
* **无连接完整性：**保护单个无连接数据块的完整性，并检测是否有数据修改，另外提供有限的重播检测。
* **选择域无连接完整性：**保护单个无连接数据块内指定域的完整性，判断指定域是否被修改。

### 不可否认性

不可否认性防止发送方或接收方否认传输或接收过的某条消息。在保证不可否认性的情况下，当消息发出后，接收方能够证明消息是由声称的发送方发出的；同样，当消息接收后，发送方能证明消息确实由声称的接收方收到。

X.800定义了两种特定的不可否认性服务：

* **源不可否认：**证明消息是由特定方发出。
* **宿不可否认：**证明消息是被特定方收到。

### 可用性服务

X.800和RFC 2828都定义可用性为：根据系统的性能说明，能够按被授权系统实体的要求访问或使用系统和系统资源的性质，即当用户请求服务时，如果系统能够提供符合系统设计的这些服务，那么系统是可用的。许多攻击可导致可用性的损失。一些自动防御措施，如认证、加密，可以对付某些攻击。而其他的一些攻击需要用物理措施来阻止或恢复系统中可用性的损失。

X.800将可用性视为和各种安全服务相关的性质。但是，单独说明可用性服务是有意义的。可用性服务处理由拒绝服务攻击所引发的安全问题，它依赖于对系统资源的恰当管理和控制，因此依赖于访问控制服务和其他安全服务。

## 安全机制

X.800中将安全机制分为两大类：一类在特定的协议层实现，如TCP或应用层协议，称为特定安全机制；另一类不属于任何的协议层或安全服务，称为普遍安全机制。

X.800中特定安全机制包括：

* **加密：**运用数学算法将数据转换成不可知的形式。数据的变换和还原依赖于算法和零个或多个加密密钥。X.800区分可逆和不可逆加密机制。可逆加密机制是加密算法，数据可以加密和解密；不可逆加密机制包括Hash算法和消息认证码，用于数字签名和消息认证应用。
* **数字签名：**附加于数据单元后的一种数据，它是对数据单元的密码变换。利用数字签名可以证明数据源和完整性，防止伪造。
* **访问控制：**对资源行使访问控制的各种机制。
* 数据完整性：用于保证数据单元或数据单元流完整性的各种机制。
* **认证交换：**通过信息交换来保证实体身份的各种机制。
* **流量填充：**在数据流空隙中插入若干位以阻止流量分析。
* **路由控制：**能够为某些数据选择特殊的物理上安全的路线，并允许路由变化（尤其是在怀疑有侵犯安全的行为时）。
* **公证：**利用可信的第三方来保证数据交换的某些性质。

X.800中普遍安全机制包括：

* **可信功能：**据某些标准被认为是正确的功能。例如，根据安全策略所建立的标准。
* **安全标签：**资源（可能是数据单元）的标志，命名或指定该资源的安全属性。
* **事件检测：**检测与安全相关的事件。
* **安全审计跟踪：**收集可用于安全审计的数据，它是对系统记录和行为的独立回顾和核查。
* **安全恢复：**处理来自安全机制的请求，如事件处理、管理功能和采取恢复行为。

基于X.800给出的定义，下表列出了安全服务与安全机制之间的关系。

表 1 安全服务与安全机制间的关系

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **服务** | **加密** | **数字签名** | **访问控制** | **数据完整性** | **认证交换** | **流量填充** | **路由控制** | **公证** |
| 同等实体认证 | Y | Y |  |  | Y |  |  |  |
| 数据源认证 | Y | Y |  |  |  |  |  |  |
| 访问控制 |  |  | Y |  |  |  |  |  |
| 保密性 | Y |  |  |  |  |  | Y |  |
| 流量保密性 | Y |  |  |  |  | Y | Y |  |
| 数据完整性 | Y | Y |  | Y |  |  |  |  |
| 不可否认性 |  | Y |  | Y |  |  |  | Y |
| 可用性 |  |  |  | Y | Y |  |  |  |

# 本地计算机中的安全

X.800等网络安全框架所中，攻击主要发生在传输过程中，对计算机本地发生的攻击关注较少。即使将计算机从网络中独立出来，也会有各种各样的攻击威胁到计算机安全，这些攻击可能发生在从底层硬件到操作系统，再到用户应用中的多个方面。与此同时，攻击的应对措施根据软硬件协同情况也有多种不同的实现，但是也存在着维护本地计算机安全的基本原则。这一部分将介绍本地计算机中安全的一些基本原则，在后面的章节中将根据不同的攻击类型来总结对应的防御对策。

## 硬件保护软件

为了保护计算机的安全，软件与硬件需要协同工作。在一项安全机制中，软件与硬件各自占的比例有多种选择，但是在这些选择之中，硬件所承担的任务具有一些基本的属性。除去自身正常工作的基本要求之外，硬件需要保证软件之间的相互隔离，并且承担一部分保证操作系统及系统应用完整性的任务。通过保护软件的完整性，硬件可以间接地保证系统中存储的信息和处理中信息的安全。在一些特殊的环境中，硬件上需要提供特殊的支持，来保证系统组件不受环境干扰。硬件上提供的容错以及错误检测功能也能够有效地提升系统组件的可用性。硬件加密机制也可以用来支持多种多样的安全服务。

硬件上与安全相关的组件主要可以分为以下几类：

* 处理器状态寄存器
* 内存管理
* 特殊的加密硬件

下面将分别介绍这三类组件以及他们在维护计算机系统安全中的作用。

### 处理器状态寄存器

现代的微处理器中都包含有处理器状态寄存器。这些寄存器既包括特权等级、中断屏蔽位等直接控制处理器运行行为的控制状态寄存器，也包括记录了处理器执行的指令数、当前经过时间以及流水线中发生的各种事件数量的性能计数器。性能计数器中保存了一段时间内处理器的行为记录，通过这些记录可以探测出程序执行的行为模式。这些行为模式不仅可以用来探测系统中是否存在恶意软件[4]，也可以用来探测系统中是否存在可疑的程序执行行为（例如利用反复读写内存某一行来改写临近的内存行中数据内容的Rowhammer攻击），从而采取针对性的防御措施[5]。

与性能计数器不同，控制状态寄存器（Control State Register, CSR）会直接影响计算机的安全。控制状态寄存器中最为关键的便是与处理器特权状态有关的寄存器。计算机系统中，软件层次可以大体分为两层：操作系统和用户程序。操作系统需要管理用户程序所使用的资源，因而需要比用户程序更高的特权级别。用户程序在使用系统资源时，需要通过系统调用将处理器的特权等级切换至操作系统的特权等级，然后才能对需要使用的系统资源执行各类操作。特权等级的划分为软件栈中的不同组件提供保护，尝试跨越特权级执行指令将会产生异常。硬件主要通过控制状态寄存器来对不同的特权等级划分提供支持。

虚拟机也可以为计算机安全提供支持。Popek与Goldberg最早将虚拟机定义为“真实计算机的一个高效且隔离的副本”[6]。虚拟化可以分为全虚拟化与半虚拟化，两者之间的区别在于虚拟机中运行的用户操作系统是否知道自己运行在虚拟机中而不是运行在真实的物理硬件上。虚拟机在特权等级上高于操作系统，一般来说，将操作系统称为监督者（supervisor），将虚拟机管理器称为超级监督者（hypervisor）。近年来，一些安全研究假定操作系统是不安全的。为了在不安全的操作系统中支持用户程序的安全执行，一部分研究提出了利用超级监督者来支持安全用户程序的做法[7, 8]。在硬件层面上，处理器需要提供额外的特权等级来支持虚拟机管理器对用户操作系统的管理，这些额外的特权等级需要反映在控制状态寄存器之中。除此之外，在指令系统体系结构中也要为虚拟化提供必要的支持。

### 内存管理

过去的50多年时间里出现了很多不同的内存管理方式，从简单地在操作系统内核与用户程序之间划定一条界线，到段式管理、页式管理，以及现在最为常用的段页式虚拟内存。内存管理的职责是防止一个进程影响到其他进程的内存，这不仅包括用户进程之间的相互影响，也包括了用户进程与操作系统之间的互相影响。内存管理机制需要硬件上提供一定的支持，在不同的内存管理机制中，硬件上需要提供的支持复杂程度也不尽相同。根据内存管理机制的不同，以及操作系统与硬件的协同机制不同，处理器为内存管理提供的支持部件可以从单独一个寄存器变化到一个专门的部件。

最简单的内存管理机制可以仅用一个寄存器来实现。这个寄存器中保存了操作系统内核的边界。这种内存管理机制适用于只有一个用户程序的简单操作系统，其主要目标是防止用户程序破坏内存中的操作系统代码。当用户程序试图向内存中写入数据时，处理器将会把目标地址与寄存器中保存的系统内核边界进行比较。如果目标地址越过了边界，那么这个操作将被阻止，并产生一个异常。

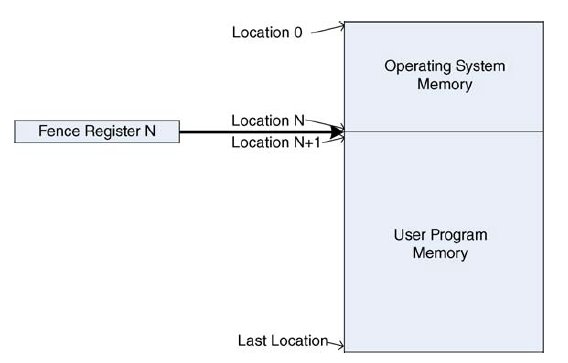


图 2 使用边界寄存器的内存管理

这种最简单的内存管理机制只适用于所有内存访问都使用物理地址的情况。如果计算机系统中只使用物理地址对内存进行访问，那么程序员在编写程序时就必须考虑内存的空间的分配方式，这对于程序的可移植性非常不利。为了解决这一问题，曼彻斯特大学的Atlas小组在1959年时开发出了第一个虚拟内存的原型[9, 10]。1961年Burroughs公司推出的B5000计算机被认为是第一个使用了虚拟内存的商用计算机[11]。在现代处理器和操作系统中，虚拟内存已经成为了必不可少的组成部分。虚拟内存有多种实现方式，可以使用完全的段式管理，也可以使用完全的页式管理，还有一些处理器结构支持段式和页式结合的管理方式，下面将分别介绍这几种虚拟内存的实现方式。

1. **段式虚拟内存管理**

在完全的段式虚拟内存实现中，程序由相互独立的“段”组成。一般来说，一个程序可以分为过程（如子程序、函数等）、全局数据以及过程中的局部数据。程序的这些组成部分每个都可以构成一个段，每个段包括一个基址和一个边界，并且具有独一无二的编号。程序使用<段号，偏移量>的形式来访问内存数据，偏移量与段基址共同构成访存时使用的物理地址，偏移量超出段的边界时访存会失败，并产生一个异常。为了支持这一访存机制，处理器需要提供段基址寄存器和边界寄存器。另一方面，在段式虚拟内存管理中，系统中需要维护一个段表，表中保存了段号与这个段当前在内存中位置的对应关系。当程序产生了一个<段号，偏移量>形式的地址时，操作系统在表中查找这个段的基地址，然后利用基地址与偏移量的和去访问真实的物理地址。利用段号在段表中查找基地址的过程既可以由操作系统来完成，也可以交给处理器来完成，如果使用硬件实现的话，那么处理器中需要使用一个专门的寄存器来保存段表在物理内存中的位置。为了加速整个地址翻译过程，处理器可以使用类似Cache的结构保存近期翻译过的虚拟地址，利用程序中的局部性来快速找到物理地址。

段式的虚拟内存管理机制允许操作系统在物理内存中任意放置程序段，并且在运行时也可以对段进行重新定位，而且每次地址翻译的过程都需要查找段表，在操作系统是可信的情况下，这可以有效地确保内存访问的合法性。但是完全的段式管理也存在缺点：由于段大小可能增长，在程序运行是需要时刻追踪段的当前大小；在段表中查找段号对应的基地址是按内容查找，这是一个很耗时间的查找方式；另外，可变的段大小不利于段在内存中的安放，容易产生外部碎片。

1. **页式虚拟内存管理**

页式内存管理是虚拟内存的另一种实现方式，也是现代处理器中最常见的虚拟内存管理方式。与段式管理按照程序逻辑切分段的方式不同，页式管理将整个内存空间划分为大小相同的若干个内存页（page）。相对的，用户程序也按页大小分割，存放在不同的内存页之中。在页式虚拟内存中，系统中维护一个追踪内存页面使用情况的页表，页表中保存了页号与实际物理地址的对应关系。程序使用<页号，偏移量>形式的虚拟地址来访问内存，系统根据页号在页表中查找到该页面对应的物理地址起始位置，与偏移量组合后得到实际的物理地址。

页式的虚拟内存管理在实现上比段式的管理方式更为简便。同时，由于页式管理使用固定大小的页面，能够有效地减少段式管理中由于段长度可变而导致的外部碎片问题。但是另一方面，按照固定大小切分的页面并没有考虑到程序本身的逻辑特点。在段式管理中，可以按照程序逻辑为程序分段，并为每个段针对性地设置读、写、执行等权限保护；而页式管理中，保护只能以页面为单位来提供，无法兼顾程序的逻辑特点。

与段式管理类似，页式管理也需要相应的地址翻译机制来进行支持。地址翻译可以完全由操作系统来完成，也可以由硬件来完成。如果由硬件来完成，那么处理器中需要设置一个页表寄存器来保存页表的起始地址，并提供相应的硬件单元来支持在页表中查找特定页面，以及遍历页表分配空闲页等操作。

1. **段式与页式相结合**

从前文可以看出，分页的方式提供了简便的实现，而分段的方式提供了基于程序逻辑的保护手段。这两种虚拟内存管理方式并不是互斥的，可以将两种方式结合在一起来共同实现虚拟内存管理。有几个著名的系统实现了对段进行分页的支持，早期的例子是MULTICS，Intel的Pentium处理器也同时支持了分段与分页。

MULTICS运行在Honeywell 6000计算机和它的一些后继机型上。它为每个程序提供了最多个段，每个段的虚拟地址空间最长为65536个字。为了实现它，MULTICS的设计者把每个段看成是一个虚拟内存，并对段进行分页，以结合分页的优点（统一的页面大小和按需要调度段的一部分进入内存）和分段的优点（易于模块化、保护和共享）。

Pentium的虚拟内存与MULTICS类似，不过段的数量更少，单个段的长度更大。Pentium处理器有16K个独立的段，每个段最多可以容纳10亿个32位字。Pentium认为较大的段比更多的段个数更为重要，这是因为几乎没有程序需要超过1000个段，但是很多程序需要大段。Pentium处理器中虚拟内存的核心是两张表：LDT（Local Descriptor Table，局部描述符表）和GDT（Global Descriptor Table，全局描述符表）。每个程序有自己的LDT，用于描述各个程序自己的代码、数据、堆栈等部分；同一台计算机上的所有程序共用一个GDT，描述操作系统本身的数据。

为了访问一个段，一个Pentium程序需要把这个段的选择子装入机器的6个段寄存器中的某一个里面。选择子是一个16位数，其中的一位指出这个段是局部的还是全局的，两位指出该段的特权等级，其他的13位是LDT或GDT的表项编号。在运行的过程中，CS寄存器保存代码段的选择子，DS寄存器保存数据段的选择子，其他的段寄存器相对而言不太重要。如果某些应用程序不需要分段，只需要一个单独的、分页的32位地址空间，那么可以将所有的段寄存器用同一个选择子设置，段长度被设置为最大，此时效果等同于正常的分页，这也是当前所有的Pentium操作系统的工作方式。

以上介绍了虚拟内存的三种管理方式。虚拟内存允许操作系统能够方便地为不同软件分配不同的内存空间，从而保证软件之间的相互隔离，起到保护软件的效果。另一方面，虚拟内存需要通过一个集中式的翻译过程来变化成供内存控制器使用的物理内存地址，这种翻译过程允许对内存的访问权限进行集中式的管理，既可以提供读、写、执行等权限的管理，也可以提供处理器特权等级的管理。虚拟内存管理机制的实现依赖于软硬件之间的协同工作，软硬件参与的成分多少决定了相应硬件的复杂程度。现代处理器中为了提高执行效率，都使用专用的硬件部件来完成地址翻译的工作，同时使用TLB来加快地址翻译的速度。

### 特殊的加密硬件

由于对称加密与非对称加密日益广泛的使用，制造商开始在硬件系统中提供基于硬件的加解密支持。可信计算组织（Trusted Computing Group，TCG）提出的可信计算标准中，可信计算基（Trusted Computing Base，TCB）是整个可信计算标准的基础，而加密硬件是TCB的一个重要组成部分。加密硬件提供的功能包括：

* 密钥的安全生成
* 密钥的安全存储
* 加解密算法
* 对敏感数据进行保护，防止未授权的使用和潜在的攻击

加密硬件可以大体分为两类，一类是集成于系统芯片之中，另一类在系统芯片之外，以外部设备的形式存在。系统芯片之外的加密硬件包括了总线上的加速卡、USB设备等多种形式。

## 软件保护信息

软件在维护计算机安全中的作用主要体现于用户认证、访问控制以及维护存储和处理中信息的完整性等方面。在此之上，需要特定软件的正确运作来保证系统的可用性，以及为安全服务中特定的安全策略提供必要的支持。由于用户软件种类繁多，这里主要考虑操作系统应该如何保护本地计算机系统中的信息。

首先需要考虑的问题是，系统的访问控制服务应该位于什么位置。有两点是我们必须要注意的：其一，如果攻击者获取了更底层软件的控制权，那么位于这一软件层次上方的任何安全机制都是可以被绕开的；其二，我们需要确保安全机制无法被绕开。

由此得出的答案是基础的安全机制应该位于软件层次的核心位置。核心中的基础安全机制需要简单，通用，并且是大多数用户所确实需要的机制。与之相对的，位于更高层的安全机制倾向于应用程序专用。位于核心位置的软件机制需要满足几点要求：能够确保更高软件层次中的安全机制的正确性；保证更高层次中的安全机制无法被规避；减少大多数用户或进程需要的安全服务的开销。

那么，假设一个操作系统能够保证所有的访问控制策略，而且只要操作系统还在如设计预期一样工作就能阻止所有的对资源的未授权访问。这样的操作系统毫无疑问可以保证运行在操作系统中用户程序的信息安全。但是如果攻击者成功地修改了操作系统的话，那么已有的安全措施很可能会失效。因此，操作系统并不能仅仅作为一个服务访问请求的仲裁者，操作系统本身也是访问控制服务的管辖对象之一。因此，为了有效地维护信息安全，必须要确保用户不能够修改操作系统而且不能够直接使用内核功能。

现代操作系统中，多任务和多用户已经成为了基本特征。为了保护信息安全，操作系统有必要在用户程序之间提供必要的保护。这方面的具体措施可以有以下几种：

* 隔离：进程不知道其他进程的存在情况，每个进程有独立的地址空间、文件和其他数据。进程之间相互隔离，不共享数据。
* 全部共享或不共享：一个对象的所有者可以声明这个对象的属性是公有的还是私有的。公有的对象可以被所有的用户访问，同时操作系统保护私有对象不被任何其他用户访问。
* 通过访问限制共享：用户对于特定的对象有着有限的访问权限，操作系统会检查用户持有的权限是否能够访问这一对象。用户的访问权限可以预先设置好并固定，也可以动态地创建。

# 软件攻击及其应对

计算机安全的发展过程是一场防御措施与攻击手段的军备竞赛。每当新的攻击手段攻破了已有的防御，就会有新的防御措施出现来填补漏洞。因此，为了更好地理解计算机安全，有必要从攻击形式入手。这一节介绍软件攻击的形式与应对措施。软件攻击以恶意软件的形式出现，利用系统的漏洞，或者通过诱骗用户执行等方式来达到传播与破坏的目的。

## 恶意软件类型

恶意软件，可以说是对计算机系统威胁最大的种类之一。恶意软件可以定义为这样的程序：“偷偷地嵌入到其他程序中，意图摧毁数据，运行破坏性或入侵性程序，或者破坏受害者数据、应用程序或操作系统的保密性、完整性和可用性。”[12]因此，我们关心恶意软件对应用程序，以及像编辑器、编译器等实用程序和内核层程序所造成的威胁。我们也关心恶意软件在被感染的或恶意的网站或服务器中的使用，以及恶意代码在意图诱使用户泄露个人信息的垃圾邮件或其他消息中的使用。

这一领域的术语还存在一些争议，因为对这些术语还没有达成完全一致的看法，而且一些分类中还存在交叉的地方。下表是一些常见的恶意软件，在之后的部分我们会按照恶意软件的传播方式与有效负载对其进行一个粗略的分类。

表 2 常见的恶意软件

|  |  |
| --- | --- |
| **名称** | **描述** |
| 病毒 | 一种恶意软件。当被执行时，试图复制自己到其他可执行代码中；复制成功后，则称可执行代码被感染了。当执行被感染代码时，病毒也将被执行 |
| 蠕虫 | 一种可以独立运行并可以将自己的完整版本传播到网络中其他主机上的恶意软件。 |
| 逻辑炸弹 | 由攻击者插入到软件中的一段程序。在预定义的条件得到满足之前，逻辑炸弹处于休眠状态；而条件满足后，程序将触发一些未经授权的行为 |
| 特洛伊木马 | 一种看起来有着有用功能，但同时也存在隐藏的恶意功能的计算机程序。这种程序有时候可以通过利用系统实体的合法授权来调用恶意程序，从而绕开安全机制 |
| 后门 | 可以绕开正常的安全检查，并能够提供不经授权直接访问相关功能的机制（或程序） |
| 下载器 | 在被攻击的机器上下载安装其他程序的程序 |
| Auto-Rooter | 用来进行远程攻击的黑客工具 |
| 病毒生成器 | 一组用来自动生成新病毒的工具集 |
| 垃圾邮件发送程序 | 用来发送大量垃圾邮件 |
| 洪泛攻击器 | 在攻击网络计算机系统时，发送大量信息，实施拒绝服务攻击（DoS） |
| 键盘记录器 | 在受攻击的系统中捕获用户敲击键盘的记录 |
| Rootkit | 在入侵计算机并获取root访问权限后使用的一套黑客工具 |
| 僵尸程序 | 在被感染的计算机上对其他计算机实施攻击的程序 |
| 间谍软件 | 用来在计算机上收集信息并将其传送到其他系统的软件 |
| 广告软件 | 植入到软件中的广告程序。它可以弹出广告窗口或者将浏览器重定向到一些商业网站 |

## 恶意软件的传播方式

恶意软件的传播机制包括三种：

* 感染已存在的可执行程序或需要解释执行的内容，这些被病毒所感染的内容随后会传播到其他系统中
* 在本地或通过网络利用软件漏洞进行攻击，蠕虫或路过式下载程序（drive-by-download）通过这种方式使恶意软件得以复制
* 利用社会工程实施攻击，诱导用户绕过安全机制来安装木马或响应网络钓鱼攻击

在这方面，早期的恶意软件分类方法将恶意软件分为两种：一种是需要宿主程序的寄生代码，如病毒；另一种是运行在系统中的独立程序，如木马、蠕虫和僵尸程序。另一种分类方法则根据恶意软件是否进行自身的复制来分类：一种是不进行复制的，如木马和垃圾邮件；另一种进行复制，包括蠕虫和病毒。

### 感染内容——病毒

恶意软件的第一种传播方式是将自身附加到可执行内容中，以寄生软件片段的形式进行传播。该片段可能会感染一些现有的用户应用或系统应用程序，甚至会感染计算机系统的引导代码。

计算机病毒是这种传播方式的代表。计算机病毒可以通过修改其他程序或任何形式的可执行内容来实现感染。这种修改通常是将病毒程序自身的副本注入其他目标程序，从而使得注入后的程序同样具有感染其他内容的功能。当被感染的程序通过磁盘或USB驱动器交换，或通过网络传送时，病毒就可以实现从一台计算机到另一台计算机的感染扩散。网络环境下，能够访问其他计算机上的文件、应用程序或系统服务的功能为病毒代码的传播提供了完美的条件。

当被感染的宿主程序执行时，病毒代码会被秘密执行。病毒代码能够执行当前宿主程序用户权限所允许的任何操作，例如擦除文件和程序。早年病毒在恶意软件中占主导地位的一个原因是当时的个人计算机系统缺乏用户身份验证和访问控制机制。现代操作系统采用了更加严格的访问控制，极大地增加了传统的机器可执行代码类病毒的感染难度，于是新种类的病毒转而利用即时编译器（Just-in-Time Compiler）来绕开系统的访问控制。

对于病毒的分类，目前还没有一个简单的普遍认可的方案。Aycock的著作中以病毒感染目标类型和病毒用以逃避用户和杀毒软件检测的自我隐藏手段作为两个不同的维度对病毒进行分类[13]。

若以感染目标类型为分类依据，可以分为以下几类：

* **引导区感染病毒：**感染主引导记录（或引导记录），当系统从包含病毒的磁盘上启动后再进行传播。
* **文件感染病毒：**感染那些可以被操作系统或者shell执行的文件。
* **宏病毒：**感染包括宏代码的文件，这些宏代码将被解释器解释执行。
* **多重感染病毒：**以多种方式感染文件。通常，多重感染病毒能够感染多种类型的文件，所以清除病毒时必须处理所有可能被感染的地方。

若以隐藏手段为分类依据，可以分为以下几类：

* **加密型病毒：**通过对病毒体加密进行隐藏。一种典型的方法是，病毒的一部分创建一个随机加密密钥来加密其余部分，并将密钥保存在病毒中。当调用被感染的程序时，病毒将使用存储的随机密钥解密病毒；当病毒复制时，将会随机选择一个不同的密钥。这样对于每个病毒示例而言，病毒体都由不同的密钥加密，因此没有固定的位组合模式特征来对其进行检测。
* **隐蔽型病毒：**与加密型病毒不同，隐蔽型病毒隐藏整个病毒，而不仅仅是有效负载部分。为了躲避杀毒软件的检测，这类病毒可能使用代码变形技术（如压缩）和Rootkit技术。
* **多态性病毒：**这种病毒在进行感染时，产生功能相似但字节排列方式截然不同的多个样本，从而改变每个副本的特征码，以对抗杀毒软件的检测。
* **变形病毒：**与多态性病毒类似，它也在每次感染时发生变异。不同之处在于变形病毒每次感染时会重写一遍自己的代码，而且行为可能像外形一样发生变化。

### 利用漏洞——蠕虫

蠕虫程序是这样一种程序：首先主动寻求更多的计算机来感染，然后再将每个被感染的计算机作为平台向其他计算机发起攻击。蠕虫程序利用客户端或服务器程序中的漏洞，来获得每个新系统的访问权限。蠕虫一旦被激活，可能再次复制和传播。除了传播，蠕虫通常携带有某种有效负载，这留在有效负载一节再进行讨论。

为复制自身，蠕虫用来访问远程系统的方式包括[14]：

* **电子邮件或即时通信工具：**蠕虫将自身副本以附件的形式发送到其他系统，使得当电子邮件或附件被接收或查看时，能够运行蠕虫的代码。
* **文件共享：**蠕虫创建自身的副本，或像病毒一样感染可移动媒介上的特定文件。当可移动驱动器连接到另一个系统时，借助特定的软件漏洞使蠕虫自动运行，或者在用户打开了受感染的文件时使蠕虫运行。
* **远程执行功能：**直接利用远程执行工具或利用网络服务中存在的程序缺陷，获取远程计算机的控制权，从而在其他系统中执行蠕虫副本。
* **远程文件访问或传输功能：**蠕虫利用远程文件访问或传输服务将自身的副本传播到远程系统。目标系统中的用户稍后可能会执行蠕虫程序。
* **远程登录功能：**蠕虫以用户身份进入远程系统，然后将自身的副本复制到远程系统中并执行。

蠕虫的传播过程往往需要借助系统或用户程序之中的漏洞。最早出现也是最著名的蠕虫是由Robert Morris于1988年发布到互联网上的的Morris蠕虫[15]。Morris蠕虫利用了UNIX finger协议中的一个漏洞，通过该协议获得远程用户的位置，还利用了一个用来发送与接收邮件的远程进程，这个远程进程的调试选项存在后门。最近几年常见的方法是利用浏览器的漏洞，当用户浏览由攻击者控制的网页时，网页可以通过浏览器的漏洞在用户未知或未经用户许可的情况下向系统中下载并安装恶意软件的代码。

### 社会工程——垃圾邮件和特洛伊木马

最后一种恶意软件的传播类型是利用社会工程，“欺骗”用户来协助恶意软件攻击其系统或窃取其个人信息。这种方式在用户浏览或回复垃圾邮件，或是允许木马程序或脚本代码的执行时会发生。利用社会工程的恶意软件主要有垃圾邮件和特洛伊木马。

未经请求的批量电子邮件，通常被称为垃圾邮件。它们无论是对于转播此通信所需要的网络基础设施，还是对于需要从大量邮件中筛选出合法邮件的用户，都产生了巨大的开销。针对垃圾邮件的爆炸式增长，反垃圾邮件行业也同样迅猛地发展，以提供检测和筛选垃圾邮件的产品。垃圾邮件中有很大一部分仅仅是劝说收信人购买某一产品，或是某种诈骗信息。但是垃圾邮件也是恶意软件的一个重要载体——电子邮件的附件可能会利用软件漏洞在用户系统中安装恶意软件，或者其本身就是木马程序或恶意脚本代码。最后，垃圾邮件可能用于网络钓鱼攻击，它们通常会指示用户访问一个虚假网站，来尝试获取用户的身份验证信息，使得攻击者能够盗取用户的身份。

特洛伊木马是一种有用的，或表面上看起来有用的程序，但其内部藏有恶意代码，当其被调用时，会执行非预期的或有害的功能。特洛伊木马可以间接地完成攻击者不能直接完成的攻击。例如，为了获取存储在某个用户文件中的敏感个人信息，攻击者会创建一个木马程序。当木马程序执行后，它会扫描用户文件来查找期望的敏感信息，并将搜索到的敏感信息副本通过电子邮件或文本信息等网络形式发送给攻击者。为了诱使用户运行木马程序，攻击者通常会用游戏软件等实用程序进行伪装。

## 恶意软件的有效负载

恶意软件一旦在目标系统中被激活，下一步所关注的就是它在目标系统中将采取什么行动，也就是它携带了何种有效负载。对于一些恶意软件，它们的有效负载不存在或者无功能。不管是刻意为之还是因为意外情况而提前释放，这类恶意软件唯一的目的就是传播。更常见的情况是，恶意软件携带一个或多个为攻击者执行隐蔽行动的有效负载。恶意软件一旦进入目标系统后，所实施的有效负载行为包括以下几种：

* 破坏系统或数据文件
* 窃取服务，以把目标系统变成一个用来进行攻击的僵尸代理，成为僵尸网络的一部分
* 窃取系统中的信息，例如利用键盘记录或间谍程序获取系统中的登录名、口令或其他个人详细信息
* 隐藏留在系统中的痕迹，使得恶意软件不能被检测或阻止

早期的恶意软件使用单一的传播方式来传送单一的有效载荷。随着恶意软件的演变，混合型的恶意软件也逐渐增多。混合型恶意软件融入了多种传播机制和有效载荷来提高其传播、隐藏和在目标主机上执行恶意行为的能力。混合型攻击利用多种感染或传播方式，最大限度地提高蔓延速度和攻击的严重程度。一些恶意软件甚至配置了更新机制，允许恶意软件部署完成后变更传播范围和使用的有效负载。

### 系统破坏

系统破坏是很多病毒和蠕虫早期使用的一种有效负载。当特定的触发条件被满足时，它们会破坏受感染系统的数据。另一种有效负载触发时会在用户系统中显示不想要的信息或内容。更严重的变种则尝试对系统进行物理破坏。所有这类行为都以破坏计算机系统软硬件或用户数据的完整性为目标。这些破坏不会直接发生，而是需要等待特定触发条件被满足才会发生。

系统破坏型负载的一个变种是勒索软件。这类软件并不销毁数据，而是对用户的数据进行加密，并要求用户付款以获得恢复信息所需要的密钥。1989年出现的PC Cyborg Trojan就是勒索软件的一个早期的例子，2017年5月在全球范围爆发的比特币勒索软件WannaCry则是最近的一个典型例子。

这类负载的另一个变种旨在对物理设备造成损害。著名的Chernobyl病毒（CIH病毒）不仅仅破坏数据，还尝试重写用于引导计算机启动的BIOS代码。一旦重写成功，计算机的启动过程就会失败，而且在BIOS芯片被重新编程或替换之前系统无法使用。另一个著名的此类恶意软件是Stuxnet蠕虫，该蠕虫以特定的工业控制系统软件为目标[16]。如果使用特定设备配置的西门子工业控制软件被感染，Stuxnet蠕虫就会用故意迫使控制设备执行其正常范围外操作的代码来替换原始控制代码。在伊朗铀浓缩计划中使用的离心机曾被严重怀疑为该蠕虫的攻击目标，因为在其活跃期间的报告显示离心机的故障率要比平时观测到的高很多。

数据损坏类恶意软件的一个关键组成部分是逻辑炸弹。逻辑炸弹是恶意软件中嵌入的代码，被设定为当满足指定的条件时就会“爆炸”。用来触发的条件可以是系统中特定文件或设备的存在或缺少、日期、某些软件的特定版本或配置，或者运行应用程序的特定用户。逻辑炸弹一旦被触发，可能会更改或删除数据或所有的文件、引发机器停止工作或其他的破坏活动。

### 攻击代理

攻击代理型的有效负载被恶意软件利用来控制受感染的计算资源和网络资源，使这些资源为攻击者所用。这类被感染的系统称为机器人（rot/robot）、僵尸（zombie）或无人机（drone）。这类恶意软件会秘密地接管其他与因特网相连的计算机，然后使用这些计算机发起或管理攻击，从而使真正的攻击者被隐藏起来。僵尸程序通常被植入到成百或成千的不知情的第三方计算机中，这些被感染的僵尸主机有时会以协同的方式工作，组成一个僵尸网络。这一类有效负载攻击被感染系统的完整性和可用性。

僵尸程序有以下主要用途[17]：

* **分布式拒绝服务攻击：**DDoS攻击是针对计算机系统或网络的一种攻击，目的是令用户无法使用服务。
* **垃圾邮件：**借助僵尸网络或大量的僵尸主机，攻击者能够发送大量的垃圾邮件。
* **网络嗅探：**僵尸主机也会使用数据包嗅探器（packet sniffer）来监视途经受感染主机的明文信息。嗅探器大多用于获取用户名和密码等敏感信息。
* **键盘记录：**如果受感染主机使用加密的通信信道，那么仅仅嗅探网络数据包是没有用的，因为缺少解密数据包的密钥。但是借助记录键盘敲击信息，攻击者就能够获取敏感信息。
* **传播新的恶意软件：**僵尸程序都实现了通过HTTP或FTP下载并执行文件的机制，所以很容易成为邮件病毒或蠕虫传播的起点。
* **攻击实时聊天网络：**此类攻击常用克隆攻击的方式。这种攻击方式中，僵尸网络的控制者命令僵尸主机将大量的克隆主机连接到目标IRC网络中，使得目标网络被来自上千台僵尸主机的服务请求或与这些主机的信道连接所淹没，从而瘫痪掉目标网络。这种攻击方式与DDoS攻击类似。

远程控制功能是区别僵尸程序与蠕虫的重要特征。蠕虫能够自我传播和自激活，而僵尸程序至少在最初阶段需要由中心设备来控制。

### 信息窃取

信息窃取类的有效负载收集存储于受感染系统中的数据，以供攻击者使用。这类有效负载的常见目标是用户在银行、游戏以及各类网站中的登录名、口令和凭证等信息，攻击者可以利用这些数据假冒用户访问来谋取私利。不太常见的目标是文件或系统配置详细信息，这类信息会被用于侦察或间谍活动。信息窃取型的攻击破坏了信息的秘密性。

在僵尸网络的应用中我们提到过，键盘记录可以绕开加密机制来获取受感染主机上的敏感信息。键盘记录是一种常见的信息窃取型有效负载，它通过捕获受感染主机的击键信息来监控敏感信息。键盘记录器通常会实现某种形式的过滤机制，仅返回接近于期望关键字的信息。为了对付键盘记录器，一些网上银行或站点使用图形化的虚拟键盘来输入重要信息，从而避免从键盘输入敏感信息。作为回应，攻击者开发了更为普遍的间谍软件。这些间谍软件可以监视受感染系统中的各类行为，包括监视浏览活动的历史记录和当前内容、将某些网页请求重定向到攻击者控制的虚假站点、动态修改浏览器和站点之间的交换数据等。

获取敏感信息的另一种方法是在垃圾邮件中包含一个指向攻击者控制的虚假网站，或者引导用户填写附件中包含敏感信息内容的表格，从而使攻击者获得用户的一系列私密信息。利用这些私密信息，攻击者能够顶替用户的身份，从而直接获取经济利益，或者利用用户身份访问其他敏感资源。这被称为网络钓鱼攻击，它利用社会工程伪装成来自受信任来源的通信，从而获取用户的信任。钓鱼攻击的一种更危险的变体是鱼叉式钓鱼攻击（spear-phishing attack）。这类变体也是把自己伪装成来自可信来源的邮件，但邮件的接收者被认真调研过，每封邮件都根据接收者定制，从而使接收者更有可能按照攻击者预期的方式来回应。

### 痕迹隐藏

最后一类有效负载的目的是在受感染系统上隐藏恶意软件的存在，以及对受感染系统的隐蔽访问技术，这种类型的有效负载攻击系统的完整性。

最常见的痕迹隐藏是后门（backdoor），又称为陷门（trapdoor），它是一个程序的秘密入口点，允许知情人不经过通常的安全访问程序而获取访问权限。后门能够在识别某些特定的输入序列时，或者被某个用户ID所运行时，或发生某个不可能的事件序列时所触发。后门通常作为运行在受感染系统中的网络服务来实现。这种网络服务监听某个攻击者能够连接和发布命令的非标准端口。由于很难通过操作系统对应用程序中的后门进行控制，因此针对后门的安全措施必须重点关注程序的开发、软件的更新活动和提供网络服务的程序。

另一种常见的痕迹隐藏手段是Rootkit。Rootkit是一组程序集合，用来维持对系统以管理员权限的隐蔽访问，同时尽可能隐藏其存在痕迹。Rootkit提供了对操作系统中所有功能和服务的访问，能够以恶意和隐蔽的方式更改主机的标准功能。具备root权限的攻击者可以拥有对系统的完全控制，能够添加和更改程序和文件、监测进程、发送和接收网络通信以及在需要时访问后门。由于Rootkit能够对系统做出很多更改来隐藏其存在痕迹，所以用户很难确认Rootkit是否存在，也很难识别其做出的更改。本质上来说，Rootkit通过破坏计算机中对进程、文件和注册表的监测和报告机制来实现隐藏。

## 防范措施

恶意软件防范主要包括四个要素：策略、意识、漏洞缓解、威胁缓解[12]。

合适的恶意软件防范策略为实施相应的防范措施提供了基础。防范策略有一些总体性的原则：首先，防范措施应该保证所有的系统尽可能地保持实时的更新，所有的软件补丁都已安装，从而减少系统中可能被利用的漏洞数量；其次，对系统中的应用程序和数据设置适当的访问控制，减少任何用户都可以访问的文件数量和恶意代码执行造成文件感染或破坏的可能性。这类措施直接针对蠕虫、病毒和一些特洛伊木马的关键传播机制。

在传播机制中我们提到，还有一种传播是利用社会工程来攻击用户。应对这类传播需要提高用户的警惕意识并开展相应的培训，这样可以让用户更加了解这类攻击，从而减少由于用户操作而被恶意软件感染的几率。

在恶意软件防范机制失效时，可以采取下列技术手段缓解威胁。

* **检测：**感染一旦发生，就立即判定恶意软件是否存在并对其进行定位。
* **鉴别：**对恶意软件进行检测后，辨别该恶意软件的类型。
* **清除：**一旦确定恶意软件的类型，就立即从所有被感染系统中删除所有的恶意软件痕迹，阻止其进一步传播。

如果对恶意软件的检测成功，但不能鉴别或清除，那么可行的方案就是放弃所有被感染的文件，并重新装入干净的备份。特别严重的情况下，可能需要完全擦除所有存储内容，并从已知的安全媒介中重建系统。

对恶意软件是否存在的检测可以在很多地方进行。比如在被感染的系统上，会有一些基于主机的反病毒软件在运行，这些软件监视导入系统的数据、可执行文件以及运行在系统中的程序行为。或者检测可以作为边界安全机制的一部分，应用在机构的防火墙和入侵检测系统中。检测也可以使用分布式机制，从分布于大量网络和机构中的基于主机的传感器和外围传感器中获取数据，来最大范围地观察恶意软件的活动。

### 基于主机的防护

终端系统是反病毒软件部署的主要位置。在终端上部署允许反病毒软件最大限度地获得恶意软件与目标系统交互的行为信息，同时也使得反病毒软件可以最大限度地获得恶意软件活动的最小全局视图。反病毒软件对恶意软件的检测主要基于恶意软件自身的特征和原始文件的特征。

早期的扫描软件要求知道恶意软件的结构特征，或者简单地通过文件长度是否变化来判断文件完整性是否遭到破坏。更新一点的扫描软件利用启发式规则应对恶意软件对自身的加密，例如寻找多态性恶意软件中加密循环代码的起点，并寻找加密密钥来对恶意软件进行解密。在完整性方面，可以利用校验和或者用加密密钥对文件进行哈希，通过校验和或者哈希值的变化来判断文件是否受到感染。

近些年来，新的技术通过检测软件的行为特征来发现系统是否被感染。例如我们可以利用处理器中内置的性能计数器来判断系统中是否存在恶意软件[4]，其原理在于，无论恶意软件如何重排代码来避开探测，其整体语义不会改变，实现整体语义而需要的子任务语义也不会改变，这就在性能计数器中形成了独特的数值模式。在一些运行的应用较为固定的系统中（如IoT），类似的技术可以应用在可信的应用上，例如我们可以预先学习合法程序运行时发出的电磁波模式，判断当前系统发出的电磁波模式是否符合预期，从而在不了解恶意软件行为特征的情况下发现入侵[18]。

利用软件行为特征来的检测与防范方法主要从恶意软件在攻击漏洞时的特征入手。恶意软件的传播过程和有效负载的工作过程往往需要依靠特定的软硬件漏洞，其中使用最为广泛的一类漏洞是内存破坏型（Memory Corruption）的漏洞。由于C语言等类型不安全语言的广泛使用，内存破坏型漏洞在软件中十分常见。通过这类漏洞，攻击者能够读写存放了敏感数据的内存区域，还可以执行其他攻击代码来扩大攻击成果。由于这些原因，内存破坏攻击在恶意软件中相当常见，根据微软公司的报告[19]，基于内存破坏的攻击方式在远程代码执行类的CVE（Common Vulnerabilities and Exposures）中占据了主导地位，可见其使用的广泛性。

内存破坏攻击可以通过三种类型的内存错误来完成：访问未初始化的内存，访问越界的内存，和访问已经被释放的内存。有许多种软件上的漏洞可以用来触发这三种内存错误。例如，为了触发一次内存的越界访问，攻击者可以通过整型数据的溢出或下溢出来绕开边界检查；通过在引用计数上制造一次溢出，攻击者可以访问本应失效的指针，从而访问已经被释放的内存。通过内存破坏，攻击者能够达成的攻击形式有以下几种：

* **代码注入：**在代码注入式的攻击中，攻击者利用内存破坏漏洞向现有代码中添加新的代码，或者直接修改程序原始代码。一旦被注入的代码被执行，那么攻击者就可以由此来控制系统。
* **控制流劫持：**控制流劫持型的攻击中，攻击者通过修改代码指针来将程序的控制流指向自己想要的方向，可以选用的攻击对象包括函数返回地址和函数指针等。早期的控制流劫持攻击将控制流转向一段shellcode，现在的控制流劫持则往往利用glibc等库代码中的片段来完成攻击，这被称为“返回导向编程”（Return Oriented Programming，ROP）。
* **数据导向攻击：**除了直接修改代码指针，还可以通过修改关键数据的方式来干扰程序执行[20, 21]。数据导向攻击的攻击能力非常强大，它可以用来攻击浏览器[22]，攻击操作系统内核[23, 24]，有研究表明数据导向攻击有能力进行图灵完备的计算[21]。
* **信息泄露攻击：**利用内存破坏漏洞，攻击者可以从内存区域中进行非法的读取从而获取重要的信息。在现代操作系统中，为了防范控制流劫持操作系统会使用地址随机化技术（Address Space Layout Randomization，ASLR）。在这种情况下，为了达成对系统的攻击首先需要将代码的地址暴露出来，研究表明信息泄露攻击能够绕开近期提出的一些基于随机化的细粒度防御机制[25]。除此之外，信息泄露攻击也可以用于直接获取敏感数据，例如在Heartbleed攻击中[26]，攻击者试图获取的信息是网站签名的私钥。
* **数据覆盖：**数据覆盖式的攻击利用的是时间维度上的内存错误，也就是访问未初始化的内存以及访问已释放的内存。数据覆盖攻击将一块攻击者控制的内存区域中的数据覆盖为其他位置的敏感数据，从而将系统中的重要信息泄露给攻击者。例如，通过访问未初始化的内存区域，攻击者可以获取操作系统内核的信息[27]。

针对这些利用内存破坏漏洞的攻击方式，现有的检测与防御措施有以下几种：

* **W⊕X：**W⊕X机制要求内存区域的写权限与可执行权限是互斥的，可写的内存区域不允许执行，可执行的内存区域不允许写入。这一机制可以有效地防范代码注入攻击，而且只需要在硬件的内存管理机制中设置一位，实现简单且对性能几乎无影响，因此得到了广泛的使用。但是由于动态代码生成技术（Dynamic Code Generation，DCG）的出现，代码注入式的攻击得到了复兴。这是由于DCG技术中需要动态地生成可执行的代码，生成代码的内存区域存在一个既允许写入又允许执行的时间窗口。
* **随机化：**随机化方法主要针对控制流劫持攻击。目前最为广泛使用的随机化技术是ASLR。在内存地址空间被随机化之后，攻击者难以简单地构造出攻击用的代码指针。随机化技术存在几个弱点：首先，如果地址空间不够大，那么无法引入足够的熵，导致随机化可能被暴力搜索攻破[28]；其次，可以通过信息泄露攻击获得随机化方法中的关键信息，从而绕开整个防御机制[25]。
* **控制流完整性：**控制流劫持类攻击是一类非常普遍的攻击方式，如果一种机制能够阻止非法的控制转移，那么这种机制就可以有效地阻止控制流劫持攻击。“金丝雀”[29]是第一个这类机制，它通过在栈中放置一个随机值来阻止缓冲区溢出攻击修改返回地址。但是金丝雀只保护了控制流图中的后向边，而且无法防止直接修改返回地址的攻击。为了更全面地保护控制流，Abadi等人提出了控制流完整性的概念[30]，其基本思想是通过静态分析建立程序的控制流图，在运行时确保控制流不会偏离这张图。但是需要注意的是，控制流检测的方法在基础理论上存在两个问题：首先，存在不偏离静态控制流图的控制流劫持攻击方式[31]；其次，无法应对数据导向的攻击方式。
* **基于指针的检查：**内存破坏型漏洞所利用的是对指针的错误使用，这包括了空间上的错误和时间上的错误。如果可以从系统设计上杜绝这两类错误的话，就可以避免恶意软件进行内存破坏。指针在空间上的错误使用是指针的越界访问，完全避免此类错误需要对所有指针的边界进行检查。CCured[32]和Cyclone[33]将传统的指针扩展为一个结构体来加入额外的边界信息，这样的指针被称为“胖指针”（Fat Pointer）。指针在时间上的错误使用包括访问未初始化的内存空间和访问已释放的内存空间。为了消除此类错误需要追踪指针引用对象的状态，并将状态更新到所有引用该对象的指针中。CETS[34]利用一个全局的对象字典来确保这一点，所有被引用的对象具有一个独特的标识符，每个指向该对象的指针根据这一标识符在字典中查找对象的引用情况。不过基于指针检查的各类方案都会带来巨大的性能损失，而且常常需要修改源代码或编译器，导致其兼容性不够好，难以适配现有的应用。

### 边界扫描方法

部署反病毒软件的另一个位置是机构的防火墙或入侵检测系统（Intrusion Detection Systems，IDS），反病毒软件通常位于这些系统上运行的电子邮件和网站代理服务中，也可能会在IDS流量分析组件中。这使得反病毒软件能够接触通过网络连接传输到组织各系统中的恶意软件，为观察其活动提供了更广阔的的视野。反病毒软件也可以包括入侵防御措施，以阻止任何可疑流量的流动，防止可疑流量到达并攻击机构内外的目标系统。

边界扫描可以检测与僵尸网络活动有关的非正常流量模式，对僵尸网络活动的检测和响应很有帮助。一旦僵尸程序被激活，攻击开始进行，边界监视就可以用来检测攻击。然而应对攻击的首要目标仍然应该是各类扫描技术，这样可以在僵尸网络的构造阶段检测并摧毁它。

边界扫描方式也可以对蠕虫程序起到一定的防范作用。蠕虫程序传播的过程会产生相当活跃的网络活动，因此边界网络活动监视器可以为蠕虫防御奠定基础。边界监视器可以基于蠕虫程序的特征码来防止蠕虫进入或离开网络或主机，或是检查数据包内容来判断数据包中是否含有蠕虫代码或是蠕虫程序负载的攻击代码。除此之外，监视器还可以对可疑流量进行限制或停止，以阻止蠕虫程序的扩散。

### 分布式的信息收集

部署反病毒软件的最后一个位置是分布式配置环境。它从许多基于主机的传感器和边界传感器中收集数据，并将这些情报转发给中央分析系统。中央分析系统能够关联和分析数据，然后返回更新后的特征码和行为模式，从而使得所有的协同系统响应和防御恶意软件的攻击。

分布式地部署恶意软件监测机制的一个好处是可以使信息采集装置与管理用的分析系统相分离。2013年Liu等人提出的一个在内存上做扩展的安全方案一定程度上利用了这一点[35]。该方案的威胁模型中假定攻击者已经绕开了所有的软件防御措施，并取得了系统最高权限。采取的方法是在内存控制器上增加一个轻量的RISC处理器核来采集信息，再把数据传输到独立的管理用计算机上进行处理。

2014年Radu Bananic等人提出的Achilles利用分布式信息收集在分布式系统中查找木马程序发出的消息[36]。Achilles将木马消息定义为：木马消息可以作为合法的消息被接收，但这一消息本身不可能由正确运行的程序合法产生。Achilles动态地分析服务器端与客户端的实现，通过对语法的符号分析来产生两个集合：客户端能够产生的消息集合，以及被服务器认为合法的消息集合。这两个集合的差即为木马消息的集合。

# 硬件攻击及其应对

除了从软件层面进行攻击之外，还有一些攻击是从计算机系统的硬件入手来进行攻击。这类攻击往往需要与被攻击的计算机设备进行物理接触，或者对硬件的结构信息有一定的了解，这使得硬件攻击的应用范围比软件攻击要小。但是针对某一种类硬件的攻击一旦成立，市面上所有利用该硬件的计算机系统都会暴露在攻击下。而且由于硬件的补丁与升级相比软件来说更为繁琐，硬件漏洞的弥补比软件漏洞更加困难。如果某些漏洞无法通过升级微码的形式来解决，只能修改硬件结构的话，那么这种漏洞造成的影响将更加恶劣，2017年谷歌Project Zero小组发现的“熔断”（Meltdown）和“幽灵”（Spectre）便是两个影响广泛的硬件漏洞例子。

硬件攻击可以大致分为两类，侧信道攻击和隐藏信道攻击，下面将分别介绍这两类攻击以及相应的应对手段。

## 侧信道攻击

在计算机安全中，侧信道攻击指的是不从算法来寻找弱点，而是从计算机系统的实现本身来获取信息进行攻击。时间信息、系统的功率消耗、泄露的电磁辐射，甚至声音都有泄露计算机系统信息的可能。

一些侧信道攻击需要对系统内部运转情况的知识，但也存在将整个系统作为一个黑盒子的攻击手段，例如差分功耗分析（Differential Power Analysis）可以通过分析加密系统的能耗情况来获取加密密钥。

侧信道攻击在网络攻击中十分常见。根据微软公司与印第安纳大学的研究[37]，即使浏览器与服务器之间的数据交换经过了加密（例如使用HTTPS传输或者Wi-Fi加密），侧信道攻击仍然起到攻击效果。Paul Kocher是侧信道攻击研究的重要先驱，他在1996年提出了针对Diffie-Hellman、RSA、DSS等多种加密方式的时间信道攻击方式[38]，此后出现的很多强力的攻击方式也大多利用了Paul Kocher所提出的统计分析方式。

在这里我们主要关心的是侧信道攻击在硬件攻击中的使用。在针对硬件的侧信道攻击中，常见的攻击方式有基于高速缓存的攻击和基于访存模式分析的攻击。另外还有一种较为少见的攻击方式是利用了软件引发的内存故障。

## 高速缓存侧信道攻击

基于高速缓存的侧信道攻击利用缓存命中与未命中之间访问时间的差异来发掘系统的额外信息。由于这一时间差异是所有高速缓存的固有性质，所以缓存侧信道攻击很难彻底消除。而且高速缓存已经是现代计算机中不可缺少的一部分，移除高速缓存将会导致严重的性能损失，因此彻底去除高速缓存也是不可取的。总的来说，高速缓存的侧信道攻击无法根除，但是仍有多种方式可以减少被成功攻击的可能性。

高速缓存侧信道攻击可以出现在各级缓存之中，例如Joseph等人成功地在L1缓存之中实施攻击，恢复了AES加密的密钥[39]。Fangfei Liu等人证明了侧信道攻击在末级高速缓存中同样可行[40]。Yunjing Xu等则证明了在虚拟机环境中L2缓存上可以进行侧信道攻击[41]。Yinqian Zhang等则成功地在云服务器环境中实施了侧信道攻击，并跨虚拟机获得了加密算法的私钥[42]。

研究者提出了许多种方法来挫败高速缓存侧信道攻击，这些方法可以分为两类：基于软件的方法和基于硬件的方法。软件方法只能减少对特定应用的攻击。例如在AES加密算法方面，[43]提出了针对AES的特殊指令；[44]提出将AES加密中使用的变换表设置为不可缓存并设置不可访问的高精确度计时器来避免侧信道攻击，但是不可缓存的变换表会导致加密过程用时增加，而且计时器不可访问也会导致一些良性程序无法正常工作；[45]提出了一种防止末级高速缓存中侧信道攻击的方法，这种方法通过特殊的分配策略将安全相关的重要数据固定在缓存中，从而保证所有与安全相关的数据访问都是命中的，但是这种将数据固定在缓存中的方法会导致缓存的碎片化，而且需要特殊硬件的支持。

基于硬件的方法主要从改进高速缓存的结构入手，总体上可以分为两种：缓存划分和随机化。基于划分的方法包括Static Partitioned (SP)缓存[46]，Partition Locked (PL)缓存[47]等。基于随机化的缓存结构包括Random Permutation (RP)缓存[47]，Newcache[48]，Random Fill (RF)缓存[49]和Random Eviction (RE)缓存[50]等。还有一些方法从缓存替换算法入手，例如[51]提出将存储层次信息引入到替换算法中来防止驱逐-重载（evict-reload）式的侧信道攻击。

还有一些研究关注如何评估高速缓存的安全性。例如[52]利用相互信息来评价高速缓存的安全性。[50]提出了侧信道脆弱因子（Side-channel Vulnerability Factor，SVF），利用模式间的关联来评价缓存。[53]则系统地分析了受攻击的程序、攻击者以及缓存之间的数据流动特征，据此提出了一个根据攻击成功可能性这一参数衡量缓存安全性的模型。

### 内存侧信道攻击

内存侧信道攻击主要利用处理器访问主存时发送的访存地址，以及整体呈现出的访存模式来获取额外信息。传统上安全处理器通过加密数据内容来保证数据的安全性，但是研究表明即使数据已经被加密，数据访问的模式仍然能够泄露可观的敏感信息{Zhuang, 2004 #146}{Rane, 2015 #147}，例如获知安全机制中使用了何种加密算法，或者更进一步获取加密密钥。

为了克服这一问题，通常使用的手段是使用Oblivious RAM，即ORAM。ORAM可以隐藏内存访问的模式，从而消除程序的访存模式导致的信息泄露。ORAM的基本思想是为内存中存储的数据维护一个加密且打乱的形式，在每次发生内存访问时，数据会被重新加密并重新打乱。在ORAM中，相同长度的内存访问模式在计算上是不可区分的。自从1987年ORAM被初次提出以来{Goldreich, 1987 #148}{Goldreich, 1996 #149}，ORAM在安全领域中获得了越来越多的关注。

在计算机系统中使用ORAM的最简单方式是用单独一个巨大的ORAM体来容纳所有的代码与数据，这样就可以保证所有的内存访问都不会因为呈现出自己所特有的访存模式而泄露额外信息。这种简单方法的最大问题是其性能非常差。由于所有的代码与数据都存放在ORAM体中，任何一次访存都会导致ORAM所要求的数据重新加密和重排，这个开销与ORAM的大小呈对数关系{Goldreich, 1996 #149}。在实践中，这种变化将会带来约100倍的额外带宽，以及每个内存块上约10倍的额外延迟{Liu, 2015 #69}。另一个问题是，如果不加入任何的填充，那么一个程序的访存操作总数仍然能够被暴露出来，这也可能导致信息的泄露。

为了解决这一问题，研究者提出了新的ORAM模型，Path ORAM[54]。在Path ORAM中，主存被组织为二叉树的形式，处理器用一个从叶子节点到二叉树根部的随机路径来访问存储器。Path ORAM的控制器中包括两个主要结构，位置映射表（position map）和数据存放处（stash），位置映射表将一个内存块与二叉树上的一条路径关联起来，实际的数据要么在这条路径上，要么在数据存放处里。每次对内存块进行访问后，Path ORAM在二叉树中为该内存块分配一个新的叶节点。

Path ORAM的效率相比于简单的ORAM实现要高，而且结构简单，但是对于很多访存密集型的应用，Path ORAM带来的额外开销仍然很高，使得其难以实际应用。Path ORAM在性能上的额外开销主要来自于掩盖访存模式时需要加入的冗余访存操作，一些研究尝试在不损害安全性的前提下减少冗余的访存，例如Fork Path ORAM[55]和GhostRider[56]；另一些研究则尝试在ORAM中引入预取机制，从而加快整体的访存速度[57]。

Path ORAM面临的另一个问题是其映射表的大小。Path ORAM需要为内存中存储的数据建立映射表，这个映射表需要存放在可以信任的处理器中，但是映射表常常过于庞大。递归的ORAM（Recursive ORAM）[58]可以解决这一问题。递归ORAM参考了多级页表的组织方式，以一个2级的递归ORAM为例，可以将原始的位置映射表存储在第二个ORAM中，第二个ORAM的体积将变得很小，这样第二个ORAM的映射表就可以存放在处理器中。这种方法可以通过进一步加深层数来扩展ORAM的大小，代价是访问时间会变长。

### 软件诱发内存故障——Rowhammer

Rowhammer攻击最早在一篇卡内基梅隆大学与英特尔实验合作的论文中被提出[59]。这种攻击针对特定的DRAM设备，主要表现是：当攻击者反复访问内存中特定的几个内存行时，这些行邻接的内存单元中存储的数据会被翻转。

Rowhammer攻击源于DRAM部件在物理尺寸上的不断缩小。现代DRAM为了提高整体的容量而将更多的内存单元集成在一块芯片上，更多的内存单元使得各单元之间的电磁干扰情况变得更严重。由于自然泄露的存在，DRAM需要定时刷新内存行来确保数据不会丢失。现有的DRAM结构要求每7.8us发射一次刷新指令，每个内存行的刷新周期是64ms。Rowhammer攻击利用了密集相连的内存单元之间相互的电磁影响，通过在一个刷新周期内多次重复访问同一个内存行来加速相邻内存单元的漏电，从而最终影响到内存单元中存储的数据。

Rowhammer攻击可以在很多实际系统中成立，例如在x86指令系统体系结构中，可以利用CLFLUSH指令来直接刷新特定的缓存行，从而允许对DRAM某一区域的高频访问。谷歌公司的Project Zero小组发现了两种利用Rowhammer攻击计算机系统的实际例子[60]：第一个例子中，攻击者利用Rowhammer来向谷歌公司的一个沙盒执行环境中注入恶意代码；第二个例子中，攻击者利用Rowhammer来提高进程的特权等级，这使得该进程可以获得对自身页表的写权限，并进而获得对全部物理内存的读写权限。

针对现有的系统，硬件厂商提出的解决方案包括两类。一类是加快DRAM的刷新频率，从每64ms刷新一次加快到每32ms刷新一次，从而减少每个刷新周期中被攻击的可能性。另一类是限制CLFLUSH指令的使用，从而阻止程序对DRAM的直接访问。但是这两种解决方案仍然存在漏洞，[5]指出，在使用CLFLUSH指令时，最快可以在15ms的时间内完成内存单元的翻转，而且存在不使用CLFLUSH指令的Rowhammer攻击方式。

另一些解决方案则试图侦测Rowhammer攻击，并对相邻的行进行针对性的刷新来阻止攻击者成功翻转内存单元内容。PARA[59]在内存行被访问时以一个低概率同时打开相邻的内存行进行刷新，由于Rowhammer攻击会反复访问同一个内存行，其相邻行在多次访问之中有较大的概率被同时刷新。ANVIL[5]则利用处理器中的性能计数器来追踪是否有内存行在短时间内被多次访问，如果有的话，则针对性地读取其相邻的内存行以进行刷新操作。

## 隐藏信道攻击

隐藏信道（Covert Channel）是一种允许进程以违背系统安全策略的形式传输信息的通信通道。隐藏信道本身不是传输信息的通道，但是信息的发送者可以通过对一些条件进行编码来达成传递信息的目的。用来编码的条件需要能够被接收者检测到，例如高速缓存中的冲突，某些服务的可用性或是事件发生的时间。特洛伊木马程序可以利用隐藏信道绕开系统的安全机制，将信息传递给攻击者。隐藏信道攻击利用系统中合法的共享资源来进行信息的传递，例如使用文件系统[61]、输入设备[62]、高速缓存和其他微体系结构部件[63]，近期的一些研究表明GPGPU也可以用来构造隐藏信道[64]。

隐藏信道攻击的一种常见方式是利用共享高速缓存。这种方式中特洛伊木马程序将会制造出一系列的缓存冲突，间谍进程可以通过这些冲突呈现出的模式来获得木马程序传递的信息。一般而言，间谍进程遵循“填充+探测”的方式获取隐藏信道传递的信息。在填充过程中，间谍程序将选定的缓存位置上填满自己的数据，然后将自己挂起。接下来木马程序根据需要传递的信息，将缓存中的数据驱逐或者保持原样。最后间谍程序执行探测过程，用与填充过程同样的地址访问内存，根据缓存未命中的数量来获取木马程序传递的消息。

为了防范基于缓存的隐藏信道攻击，可以在缓存中将进程隔离开[45]，或者在系统中加入噪音，使得间谍程序无法准确地得知缓存未命中的数量[65]。但是这些措施都存在缺陷，或者是无法干扰所有种类的隐藏信道攻击，或者是会导致巨大的性能损失。例如缓存分区的方式只能支持有限数量的分区，而且性能损失很大。

另一种解决方法是探测隐藏信道的通信，然后阻止进一步的信息泄露。由于隐藏信道攻击需要长时间的信息交换，所以如果探测机制能够及时发现正在进行的隐藏信道通信，那么即使有少部分信息已经泄露，也可以有效降低隐藏信道攻击的危害。CC-Hunter[66]就是这类机制的一种，它追踪进程间的缓存冲突模式，并且能够准确地发现木马程序和间谍程序之间的信息交换过程。但是CC-Hunter能够探测到的通信遵循一个严格的交互模式，违背这个模式的通信则难以被CC-Hunter所检测到。

ReplayConfusion[67]是另一种探测隐藏信道通信的机制。ReplayConfusion基于这样一个发现：隐藏信道攻击中，不仅发生缓存冲突的地址是经过精确调节的，木马与间谍之间的数据传输也遵循一个独特的节奏。即使将冲突失效的访存地址更改为其他地址，数据传输的节奏也会被保持下来，但是这种现象不会出现在良性程序中。于是ReplayConfusion利用了重放的办法来检测隐藏信道通信，它记录程序执行的过程，然后将访存地址到缓存的映射关系更改后进行重放，分析记录过程与重放过程中缓存失效的模式是否存在相似性，从而判断隐藏信道攻击是否存在。

# 结语

本文总结了计算机系统中安全的概念，安全所涉及的各个层面，以及不同层面中安全所需要的各种性质，并从软、硬件两方面介绍了常见的攻击手段和应对方法。

本文首先通过NIST对计算机安全的定义介绍了计算机安全的基本概念，并进一步将计算机中的安全分为几个层级。接下来通过OSI安全框架对网络安全中所需要的各类服务以及安全机制进行了总体上的概括。除了网络安全，我们同样关心本地计算机中如何维护信息安全。在这里本文首先从总体上阐述了本地计算机中维护信息安全的两个基本原则：硬件保护软件和软件保护信息。之后我们从硬件和软件两个角度对常见的攻击形式和应对手段进行了简要的描述。受篇幅所限，本文并没有覆盖到所有的攻击种类，而是将注意力集中在了本地计算机中软硬件攻击方式以及相应的防范措施，对网络传输中的攻击手段没有进行深入的论述。除了文中所论述的软硬件攻击及其应对手段，安全领域中还有大量其他的研究方向。例如对物理不可复制器件的制造及其在安全领域中应用的研究，一些研究还尝试通过控制系统中器件的使用寿命来阻止攻击者的暴力破解手段。

近几年计算机安全领域中出现的新攻击手段里仍不乏利用内存破坏漏洞的攻击方式，这在通过浏览器用户脚本的攻击中尤为常见。于此同时，新的攻击手段也呈现出向计算机体系结构底层延伸的趋势，例如利用DRAM结构特性的Rowhammer攻击，2017年出现的“熔断”与“幽灵”两个影响广泛的漏洞中甚至利用了现代处理器中广泛使用的转移预测机制。这证明了计算机安全需要软硬件共同参与才能够确保，同时也为计算机体系结构研究者提出了问题：现代处理器中广泛使用的各类提高性能的机制是否会对计算机安全造成影响？

# 参考文献

[1] CLARK D D, BOEBERT W E, GERHART S, et al. Computers at risk: safe computing in the information age [J]. Final report of the System Security Study Committee, 1990,

[2] JACOBS S. Engineering Information Security: The Application of Systems Engineering Concepts to Achieve Information Assurance [M]. Wiley-IEEE Press, 2015.

[3] SHIREY R. RFC 2828 [J]. Internet Security Glossary, 2000,

[4] DEMME J, MAYCOCK M, SCHMITZ J, et al. On the feasibility of online malware detection with performance counters [M]. Proceedings of the 40th Annual International Symposium on Computer Architecture. Tel-Aviv, Israel; ACM. 2013: 559-570.

[5] AWEKE Z B, YITBAREK S F, QIAO R, et al. ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks [M]. Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems. Atlanta, Georgia, USA; ACM. 2016: 743-755.

[6] POPEK G J, GOLDBERG R P. Formal requirements for virtualizable third generation architectures [J]. Commun ACM, 1974, 17(7): 412-421.

[7] HOFMANN O S, KIM S, DUNN A M, et al. InkTag: secure applications on an untrusted operating system [M]. Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems. Houston, Texas, USA; ACM. 2013: 265-278.

[8] CRISWELL J, DAUTENHAHN N, ADVE V. Virtual ghost: protecting applications from hostile operating systems [M]. Proceedings of the 19th international conference on Architectural support for programming languages and operating systems. Salt Lake City, Utah, USA; ACM. 2014: 81-96.

[9] FOTHERINGHAM J. Dynamic storage allocation in the Atlas computer, including an automatic use of a backing store [J]. Commun ACM, 1961, 4(10): 435-436.

[10] KILBURN T, EDWARDS D B G, LANIGAN M J, et al. One-Level Storage System [J]. IRE Transactions on Electronic Computers, 1962, EC-11(2): 223-235.

[11] MAYER A J W. The architecture of the Burroughs B5000: 20 years later and still ahead of the times? [J]. SIGARCH Comput Archit News, 1982, 10(4): 3-10.

[12] SOUPPAYA M, SCARFONE K. Guide to malware incident prevention and handling for desktops and laptops [J]. International Journal of Computer Research, 2013, 20(4): 417.

[13] AYCOCK J. Computer viruses and malware [M]. Springer Science & Business Media, 2006.

[14] FOSSI M, EGAN G, HALEY K, et al. Symantec internet security threat report, vol. 16 [J]. Symantec Corporation, Apr, 2011,

[15] ORMAN H. The Morris worm: A fifteen-year perspective [J]. IEEE Security & Privacy, 2003, 99(5): 35-43.

[16] CHEN T M, ABU-NIMEH S. Lessons from stuxnet [J]. Computer, 2011, 44(4): 91-93.

[17] BACHER P, HOLZ T, KOTTER M, et al. Know your enemy: Tracking botnets [J]. The Honeynet Project & Research Alliance, 2005,

[18] NAZARI A, SEHATBAKHSH N, ALAM M, et al. EDDIE: EM-Based Detection of Deviations in Program Execution [M]. Proceedings of the 44th Annual International Symposium on Computer Architecture. Toronto, ON, Canada; ACM. 2017: 333-346.

[19] RAINS T, MILLER M, WESTON D. Exploitation trends: From potential risk to actual risk; proceedings of the RSA Conference, F, 2015 [C].

[20] HU H, CHUA Z L, ADRIAN S, et al. Automatic Generation of Data-Oriented Exploits; proceedings of the USENIX Security Symposium, F, 2015 [C].

[21] HU H, SHINDE S, ADRIAN S, et al. Data-oriented programming: On the expressiveness of non-control data attacks; proceedings of the Security and Privacy (SP), 2016 IEEE Symposium on, F, 2016 [C]. IEEE.

[22] YU Y. Write once, pwn anywhere [J]. Black Hat USA, 2014,

[23] VOGL S, PFOH J, KITTEL T, et al. Persistent Data-only Malware: Function Hooks without Code; proceedings of the NDSS, F, 2014 [C].

[24] SONG C, LEE B, LU K, et al. Enforcing Kernel Security Invariants with Data Flow Integrity; proceedings of the NDSS, F, 2016 [C].

[25] SNOW K Z, MONROSE F, DAVI L, et al. Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization; proceedings of the Security and Privacy (SP), 2013 IEEE Symposium on, F, 2013 [C]. IEEE.

[26] CODENOMICON, MEHTA N. The Heartbleed Bug. 2014. <http://heartbleed.com/>.

[27] CHEN H, MAO Y, WANG X, et al. Linux kernel vulnerabilities: State-of-the-art defenses and open problems; proceedings of the Proceedings of the Second Asia-Pacific Workshop on Systems, F, 2011 [C]. ACM.

[28] SHACHAM H, PAGE M, PFAFF B, et al. On the effectiveness of address-space randomization; proceedings of the Proceedings of the 11th ACM conference on Computer and communications security, F, 2004 [C]. ACM.

[29] COWAN C, PU C, MAIER D, et al. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks; proceedings of the USENIX Security Symposium, F, 1998 [C]. San Antonio, TX.

[30] ABADI M, BUDIU M, ERLINGSSON U, et al. Control-flow integrity; proceedings of the Proceedings of the 12th ACM conference on Computer and communications security, F, 2005 [C]. ACM.

[31] EVANS I, LONG F, OTGONBAATAR U, et al. Control jujutsu: On the weaknesses of fine-grained control flow integrity; proceedings of the Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, F, 2015 [C]. ACM.

[32] NECULA G C, MCPEAK S, WEIMER W. CCured: Type-safe retrofitting of legacy code; proceedings of the ACM SIGPLAN Notices, F, 2002 [C]. ACM.

[33] JIM T, MORRISETT J G, GROSSMAN D, et al. Cyclone: A Safe Dialect of C; proceedings of the USENIX Annual Technical Conference, General Track, F, 2002 [C].

[34] NAGARAKATTE S, ZHAO J, MARTIN M M, et al. CETS: compiler enforced temporal safety for C; proceedings of the ACM Sigplan Notices, F, 2010 [C]. ACM.

[35] LIU Z, LEE J, ZENG J, et al. CPU transparent protection of OS kernel and hypervisor integrity with programmable DRAM [M]. Proceedings of the 40th Annual International Symposium on Computer Architecture. Tel-Aviv, Israel; ACM. 2013: 392-403.

[36] BANABIC R, CANDEA G, GUERRAOUI R. Finding trojan message vulnerabilities in distributed systems [M]. Proceedings of the 19th international conference on Architectural support for programming languages and operating systems. Salt Lake City, Utah, USA; ACM. 2014: 113-126.

[37] CHEN S, WANG R, WANG X, et al. Side-channel leaks in web applications: A reality today, a challenge tomorrow; proceedings of the Security and Privacy (SP), 2010 IEEE Symposium on, F, 2010 [C]. IEEE.

[38] KOCHER P C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems; proceedings of the Annual International Cryptology Conference, F, 1996 [C]. Springer.

[39] BONNEAU J, MIRONOV I. Cache-collision timing attacks against AES; proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, F, 2006 [C]. Springer.

[40] LIU F, YAROM Y, GE Q, et al. Last-level cache side-channel attacks are practical; proceedings of the Security and Privacy (SP), 2015 IEEE Symposium on, F, 2015 [C]. IEEE.

[41] XU Y, BAILEY M, JAHANIAN F, et al. An exploration of L2 cache covert channels in virtualized environments; proceedings of the Proceedings of the 3rd ACM workshop on Cloud computing security workshop, F, 2011 [C]. ACM.

[42] ZHANG Y, JUELS A, REITER M K, et al. Cross-VM side channels and their use to extract private keys; proceedings of the Proceedings of the 2012 ACM conference on Computer and communications security, F, 2012 [C]. ACM.

[43] NADEHARA K, IKEKAWA M, KURODA I. Extended instructions for the AES cryptography and their efficient implementation; proceedings of the Signal Processing Systems, 2004 SIPS 2004 IEEE Workshop on, F, 2004 [C]. IEEE.

[44] TROMER E, OSVIK D A, SHAMIR A. Efficient cache attacks on AES, and countermeasures [J]. Journal of Cryptology, 2010, 23(1): 37-71.

[45] LIU F, GE Q, YAROM Y, et al. Catalyst: Defeating last-level cache side channel attacks in cloud computing; proceedings of the High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on, F, 2016 [C]. IEEE.

[46] PAGE D. Partitioned Cache Architecture as a Side-Channel Defence Mechanism [J]. IACR Cryptology ePrint archive, 2005, 2005(280):

[47] WANG Z, LEE R B. New cache designs for thwarting software cache-based side channel attacks; proceedings of the ACM SIGARCH Computer Architecture News, F, 2007 [C]. ACM.

[48] WANG Z, LEE R B. A novel cache architecture with enhanced performance and security; proceedings of the Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture, F, 2008 [C]. IEEE Computer Society.

[49] LIU F, LEE R B. Random fill cache architecture; proceedings of the Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on, F, 2014 [C]. IEEE.

[50] DEMME J, MARTIN R, WAKSMAN A, et al. Side-channel vulnerability factor: A metric for measuring information leakage [J]. ACM SIGARCH Computer Architecture News, 2012, 40(3): 106-117.

[51] YAN M, GOPIREDDY B, SHULL T, et al. Secure Hierarchy-Aware Cache Replacement Policy (SHARP): Defending Against Cache-Based Side Channel Atacks [M]. Proceedings of the 44th Annual International Symposium on Computer Architecture. Toronto, ON, Canada; ACM. 2017: 347-360.

[52] STANDAERT F-X, MALKIN T G, YUNG M. A unified framework for the analysis of side-channel key recovery attacks; proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, F, 2009 [C]. Springer.

[53] HE Z, LEE R B. How secure is your cache against side-channel attacks? [M]. Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. Cambridge, Massachusetts; ACM. 2017: 341-353.

[54] STEFANOV E, VAN DIJK M, SHI E, et al. Path ORAM: an extremely simple oblivious RAM protocol; proceedings of the Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, F, 2013 [C]. ACM.

[55] ZHANG X, SUN G, ZHANG C, et al. Fork path: improving efficiency of ORAM by removing redundant memory accesses [M]. Proceedings of the 48th International Symposium on Microarchitecture. Waikiki, Hawaii; ACM. 2015: 102-114.

[56] LIU C, HARRIS A, MAAS M, et al. GhostRider: A Hardware-Software System for Memory Trace Oblivious Computation [M]. Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems. Istanbul, Turkey; ACM. 2015: 87-101.

[57] YU X, HAIDER S K, REN L, et al. PrORAM: dynamic prefetcher for oblivious RAM [M]. Proceedings of the 42nd Annual International Symposium on Computer Architecture. Portland, Oregon; ACM. 2015: 616-628.

[58] SHI E, CHAN T-H H, STEFANOV E, et al. Oblivious RAM with O ((logN) 3) worst-case cost; proceedings of the International Conference on The Theory and Application of Cryptology and Information Security, F, 2011 [C]. Springer.

[59] KIM Y, DALY R, KIM J, et al. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors; proceedings of the ACM SIGARCH Computer Architecture News, F, 2014 [C]. IEEE Press.

[60] SEABORN M, DULLIEN T. Exploiting the DRAM rowhammer bug to gain kernel privileges [J]. Black Hat, 2015, 7-9.

[61] LAMPSON B W. A note on the confinement problem [J]. Communications of the ACM, 1973, 16(10): 613-615.

[62] SHAH G, MOLINA A, BLAZE M. Keyboards and Covert Channels; proceedings of the USENIX Security Symposium, F, 2006 [C].

[63] MAURICE C, NEUMANN C, HEEN O, et al. C5: cross-cores cache covert channel; proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, F, 2015 [C]. Springer.

[64] NAGHIBIJOUYBARI H, KHASAWNEH K N, ABU-GHAZALEH N. Constructing and characterizing covert channels on GPGPUs [M]. Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. Cambridge, Massachusetts; ACM. 2017: 354-366.

[65] MARTIN R, DEMME J, SETHUMADHAVAN S. TimeWarp: rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks [J]. ACM SIGARCH Computer Architecture News, 2012, 40(3): 118-129.

[66] CHEN J, VENKATARAMANI G. CC-Hunter: Uncovering Covert Timing Channels on Shared Processor Hardware [M]. Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture. Cambridge, United Kingdom; IEEE Computer Society. 2014: 216-228.

[67] YAN M, SHALABI Y, TORRELLAS J. Replayconfusion: detecting cache-based covert channel attacks using record and replay [M]. The 49th Annual IEEE/ACM International Symposium on Microarchitecture. Taipei, Taiwan; IEEE Press. 2016: 1-14.