**基于Ghidra的自动化二进制漏洞分析工具项目**

**详细设计说明书说明书**

# 引言

随着软件应用程序的复杂性和数量的增加，二进制漏洞的发现和修复变得至关重要。传统的手动漏洞分析方法耗时且容易出错，因此需要一种自动化工具来加速漏洞检测和分析的过程。本项目旨在开发一款高效、准确且易于使用的二进制漏洞分析工具，以帮助安全专业人员和开发人员提高漏洞检测的效率和准确性。该项目旨在开发一款强大的二进制漏洞分析工具，能够自动化检测和分析二进制程序中的漏洞，并提供相关的报告和建议。该工具将结合使用Ghidra反编译器、Semgrep分析工具和OpenAI模型，以提供全面的漏洞检测和代码分析功能。

# 设计理念

这是一个基于自定义的Semgrep C规则、semgrep工具和反编译器的伪代码，用于自动发现漏洞的设计理念。

设计理念如下：

1. 自定义Semgrep C规则集：自定义的Semgrep C规则集，以识别特定的漏洞模式或安全问题。这些规则可以根据实际需求编写，并涵盖各种可能的漏洞类型，如缓冲区溢出、格式化字符串漏洞等。
2. 反编译成为伪代码： 使用Ghidra反编译将目标程序转换为伪代码，对伪代码表示应用Semgrep规则集进行静态分析，以检测潜在的漏洞；根据Semgrep的输出结果，识别和记录发现的漏洞，包括漏洞类型、位置和其他相关信息。
3. 集成Semgrep和反编译器： 将Semgrep工具和反编译器集成到设计中。通过调用Semgrep工具执行静态分析，并使用反编译器将目标程序转换为可供分析的中间表示。这可以通过命令行调用、API调用或其他适当的方式完成。
4. 自定义规则更新和维护： 随着漏洞的不断演变和新的安全问题的出现，设计人员应定期更新和维护自定义的Semgrep规则集。这可以包括添加新的规则、更新现有规则以适应新的漏洞模式，或者根据实际情况进行调整。

通过以上设计理念，可以建立一个自动化的漏洞发现系统，有效地识别和报告存在的安全漏洞。

# 模块解析：

## Callisto入口文件

* controller(self): 控制整个分析过程的方法。
* decompiler(self): 运行Ghidra反编译器及相关工具的方法。
* semgrep(self, function): 对给定函数进行Semgrep分析的方法。
* openAI(self, functions, semgrep): 对给定函数进行OpenAI分析的方法。
* config(self): 解析配置文件的方法。
* argHandler(self, args): 处理命令行参数的方法。

Callisto().controller()创建了一个Callisto类的实例，并调用了controller()方法来启动整个分析过程。在controller()方法中，代码首先解析命令行参数，然后运行Ghidra反编译器，将二进制文件转换为C代码。接下来，它将C代码分割为函数，并对每个函数进行Semgrep分析以查找潜在的漏洞。如果开启了OpenAI分析，它还会调用OpenAI模型对每个函数进行进一步的分析。最后，它将分析结果打印出来，并将结果写入输出文件（如果指定了输出文件）。

## Ghidra

decomp.py脚本，使用Ghidra的反编译接口（DecompInterface）来对当前程序进行反编译。

步骤如下：

1. 导入所需的模块和类：sys、DecompInterface、ConsoleTaskMonitor。
2. 获取当前程序（currentProgram）。
3. 创建DecompInterface对象并打开程序。
4. 获取程序中的所有函数。
5. 创建一个ConsoleTaskMonitor对象用于监视反编译过程。
6. 打开一个名为output.c的文件，用于存储反编译后的C代码。
7. 打开一个名为callGraph.txt的文件，用于存储函数调用图。
8. 对每个函数进行遍历：
   * 获取被调用的函数列表。
   * 将函数名称和被调用函数列表写入callGraph.txt文件。
   * 使用DecompInterface对函数进行反编译，返回反编译后的DecompiledFunction对象。
   * 将反编译后的C代码写入output.c文件，并在函数之间添加分隔符""。
9. 关闭callGraph.txt和output.c文件。

此脚本的目的是使用Ghidra的反编译功能将程序转换为C代码，并生成函数调用图。生成的C代码将存储在output.c文件中，函数调用图将存储在callGraph.txt文件中。

## Semgrep规则

* **argvenvpaccess：**规则raptorargvenvpaccess用于检测程序访问命令行参数（argv）或环境变量（envp）的潜在安全风险。应用该规则时，会生成一个信息级别的消息，指出程序访问了由潜在攻击者控制的命令行参数或环境变量。该规则适用于C和C++编程语言。规则使用模式匹配来检测代码中是否出现了argv或envp。
* commandinjection：规则中的模式提示程序可能使用了system(...)或popen(...)函数，如果使用不当可能存在潜在危险。规则还包括一个模式，检查是否以特定格式调用了潜在危险的函数。此规则旨在识别和防止C和C++程序中的命令注入漏洞。它强调了在构造操作系统命令时，正确中和特殊字符和用户控制输入的重要性，以减轻注入攻击的风险。
* doublefree：doublefree.yaml的内容是一个规则，用于检测C和C++代码中的double free错误。该规则的主要目的是防止在同一内存地址上调用两次free()函数，从而导致内存管理数据结构的破坏和潜在的内存安全问题。规则的消息部分解释了double free错误的含义和潜在影响。当程序对同一内存地址调用两次free()时，可能导致程序的内存管理数据结构被破坏。这种破坏可能导致程序崩溃，或者在某些情况下，导致两次后续的malloc()调用返回相同的指针。如果malloc()两次返回相同的值，并且程序稍后允许攻击者对这个双重分配的内存写入数据，那么程序就容易受到缓冲区溢出攻击。规则中还包含了匹配模式，用于检测代码中是否存在double free错误的模式。其中，第一个模式用于匹配连续两次调用相同指针的free()函数。第二个模式用于排除在第一次free()之后对指针进行重新赋值的情况，以避免误报。
* formatstringbugs：规则指出，当软件使用一个接受外部来源的格式字符串作为参数的函数时，可能会导致安全问题，如缓冲区溢出、拒绝服务或数据表示问题。该规则列出了常与格式字符串一起使用的函数列表，包括printf、scanf、fprintf、sprintf、syslog和各种err/warn函数。它还提到格式字符串可以作为这些函数的第一个、第二或第三个参数出现。该规则强调，如果格式字符串的来源是可信的且不可外部修改的，例如在国际化的情况下，格式字符串包含在由系统管理员管理的库文件中，那么外部控制可能不会构成漏洞。
* incorrectordersetuidsetgidetc：涉及到setuid、setgid、seteuid和setegid函数调用顺序的问题。如果函数调用的顺序不符合模式，可能会导致软件的行为出现问题和潜在的安全弱点。这些模式主要关注setuid、setgid、seteuid和setegid函数的调用顺序是否正确
* **incorrectunsignedcomparison：**是为了检测C和C++编程语言中不正确无符号比较的实例而定义的。规则中的模式识别特定的比较操作，这些操作很可能是不正确的。这些操作包括检查无符号变量是否小于或等于零，或者大于或等于零，这在大多数情况下被视为无效的比较。
* **incorrectuseofsizeof：**分析了代码中对malloc指针类型错误使用sizeof()运算符的情况。它检测到在指针上调用sizeof()的情况，如果意图是确定已分配的内存大小的话，这可能导致意外的结果。
* **incorrectuseofstrncat：**分析了代码中对strncat()函数错误使用的情况。strncat()函数旨在成为strcat()函数的安全替代品。然而，strncat()函数与strcat()函数几乎一样危险，因为很容易被误用。具体而言，strncat()函数的size参数可能令人困惑，因为它表示缓冲区中剩余的空间量。应用程序开发人员常犯的第一个常见错误是提供整个缓冲区的大小，而不是剩余的大小。还可能发生更微妙的错误。size参数应该是缓冲区中剩余的空间量减一；否则，NUL字节将被写入缓冲区末尾的下一个字节。
* incorrectuseofstrncpystpncpystrlcpy：软件在从源缓冲区读取或写入目标缓冲区时使用了源缓冲区的大小，这可能导致访问超出缓冲区边界的内存。当目标的大小小于源的大小时，可能会发生缓冲区溢出。
* insecureapiaccessstatlstat：规则旨在识别涉及access、stat或lstat函数的代码模式，这些函数常与时间检查时间使用（TOCTOU）竞态条件相关联。Raptor工具会将这些模式标记为潜在的安全漏洞，
* insecureapialloca：于分析使用C和C++编写的程序。该规则与使用alloca()函数相关，该函数被认为是潜在危险的。该规则强调，如果不正确使用alloca()，它可能会引入漏洞。尽管alloca()可以安全使用，但由于其不安全的特性，它并不推荐使用。该函数无法确保返回的指针指向有效可用的内存块。分配的内存可能超出栈的边界，甚至可能侵入其他内存中的对象，但alloca()本身无法检测此类错误。
* raptorinsecureapiatoiatolatof：涉及到C和C++语言。它强调了应避免使用atoi()、atol()和atof()函数，这些函数用于将字符串转换为数字。当字符串无法转换时，这些函数会导致未定义行为，因此应该避免使用它们。规则的模式匹配部分列出了三个模式，分别是atoi(...)、atol(...)和atof(...)，表示匹配对应的函数调用。总之，该规则警示不要使用atoi()、atol()和atof()函数，因为它们在字符串无法转换时会导致未定义行为。
* insecureapigets：分析了在C和C++编程语言中使用gets()函数的情况。这些规则特别针对gets()函数，因为它本质上是危险的，并可能导致缓冲区溢出漏洞。该规则旨在检测代码中使用gets()函数的情况。它强调了使用gets()函数存在的固有安全风险，因为它缺乏边界检查，可能导致缓冲区溢出漏洞。该规则的严重程度设置为错误，表示应该解决和修复该问题，以确保代码的安全性。
* insecureapimktemptmpnamtempnam：内容强调了使用诸如mktemp()、tmpnam()和tempnam()等函数创建临时文件的风险。这些函数存在竞争条件漏洞，即在应用程序打开文件之前，生成的文件名可能会被攻击者拦截并替换。此外，生成的文件名通常没有足够的随机性，使其易于猜测。该弱点的后果包括潜在的敏感信息泄漏，如读取或修改文件或目录。内容还提供了一个使用C语言的示例，演示了使用不安全临时文件的漏洞。为了减轻此弱点，建议使用安全的替代方法来创建临时文件，并确保正确的文件权限和处理方式，以防止未经授权的访问。
* raptorinsecureapirandsrand：识别在安全环境中使用了非密码学强度的伪随机数生成器（PRNG）。该规则适用于使用 C 和 C++ 编写的软件。该规则特别检查代码中是否存在 "rand()" 和 "srand()" 函数。当在密码学上下文中使用非密码学强度的 PRNG 时，可能会使密码学受到某些类型的攻击。为确保系统的安全性，使用密码学强度的 PRNG 非常重要。弱 PRNG 可能无法提供足够的随机性，并且可能会被利用来破坏密码学安全。
* **insecureapiscanfetc：**提供的内容是与不安全的API使用相关的规则片段，具体涉及到scanf、fscanf、sscanf等函数的使用。该规则强调了缓冲区溢出的潜在漏洞，当程序尝试在缓冲区中存储超过其容量的数据或在缓冲区边界之外写入数据时，就会发生缓冲区溢出。规则强调程序员应该实施适当的安全保护措施来防止此类漏洞。
* **insecureapisprintfvsprintf：**识别可能存在安全漏洞的潜在危险函数调用有关，如果使用不当，这些函数可能导致漏洞。具体涉及到的函数是sprintf和vsprintf。如果复制的数据量超过缓冲区的容量，这些函数可能引发缓冲区溢出漏洞。规则强调程序员应考虑基本的安全保护措施，以防止此类漏洞的发生。该规则的严重程度被分类为错误，表示存在重要的安全问题。
* raptorinsecureapistrcpystpcpystrcat：该规则针对潜在危险的函数，如strcpy、stpcpy、strcat、wcscpy、wcpcpy和wcscat。这些函数如果使用不当，特别是在未正确处理缓冲区溢出条件的情况下，可能会引入漏洞。该规则强调了考虑安全保护的重要性，因为经典缓冲区溢出漏洞的存在表明缺乏基本的安全措施。
* **integertruncation：**截断错误发生在将一个原始类型转换为较小大小的原始类型并在转换过程中丢失数据时。 当将原始类型转换为较小的原始类型时，大值的高位比特将在转换中丢失，可能导致一个与原始值不相等的意外值。这个值可能被用作缓冲区的索引、循环迭代器或仅仅是必要的状态数据。无论哪种情况，这个值都不能被信任，系统将处于未定义的状态。虽然这种方法可能可行地用于隔离值的低位，但这种用法很少见，截断通常意味着发生了实现错误。
* integerwraparound：提供了一些模式，用于识别可能存在漏洞的代码片段，涉及诸如malloc、valloc、calloc、realloc、reallocf和aligned\_alloc等函数。这些模式包括乘法运算（$X \* $Y）和赋值语句（$N = $X \* $Y），其中结果被用作内存分配或重新分配函数的参数。这些规则强调了处理整数溢出和环绕漏洞的重要性，因为当所得值用于资源管理、执行控制、安全决策或内存操作（如分配和复制）时，可能会导致安全问题。
* interestingapicalls：分析涉及到查找代码中调用的可能存在安全问题的API函数。
* **memoryaddressexposure：**分析软件中潜在的内存地址泄露漏洞。具体而言，它检查软件是否生成包含环境、用户或相关数据敏感信息的错误消息，这可能被利用来破解地址空间布局随机化（ASLR）。
* mismatchedmemorymanagement：用于分析代码中可能存在的内存管理例程不匹配情况。当软件试图将内存资源返回给系统时，但调用的释放函数与最初用于分配该资源的函数不兼容时，就会触发此规则。内存管理函数不匹配可能导致严重后果，如代码执行、内存损坏或程序崩溃。后果的严重性和利用难度将取决于例程的实现和所管理的对象。用于检测内存管理不匹配的模式涉及free函数。规则查找free($PTR)的实例，但排除了使用malloc、calloc、realloc、strdup或strndup等函数分配的指针$PTR的情况。还处理了在释放之前将指针显式转换为$CAST的情况。
* **mismatchedmemorymanagementcpp：**

1. 在 bad1() 函数中，存在不匹配的内存管理问题。ptr 对象是使用 new 进行分配的，但是却使用 free 进行释放。在 C++ 中，释放使用 new 分配的内存的正确方式是使用 delete，而不是 free。因此，正确的做法是将 free(ptr) 替换为 delete ptr。
2. 在 bad2() 函数中，存在另一个不匹配的内存管理问题。ptr 指针是使用 malloc 进行分配的，但是却使用 delete 进行释放。在 C++ 中，使用 malloc 分配的内存应该使用 free 进行释放，而不是 delete。因此，正确的做法是将 delete ptr 替换为 free(ptr)。
3. 在 bad3() 函数中，存在潜在的不匹配的内存管理问题。根据 heap 参数的值，p 指针可以指向一个局部数组或者是使用 new 动态分配的数组。如果 heap 为 true，则 p 被赋值为动态分配数组的地址。然而，在释放内存时，无论内存是在堆上分配还是在栈上分配，都使用了 delete[]。这可能导致未定义的行为。为了解决这个问题，您可以始终在使用 new[] 进行分配时使用 delete[]，或者对于使用 new 或者从栈上分配的内存，使用 free 进行释放。
4. good1()、good2() 和 good3() 函数没有任何不匹配的内存管理问题。它们正确地使用相应的函数（new 和 delete，或者 malloc 和 free）进行内存分配和释放，保持一致性。

* mismatchedmemorymanagementcpp：旨在检测C++代码中的不匹配内存管理问题。该规则适用于Semgrep工具，旨在识别将内存资源返回给系统时使用的释放函数与最初用于分配该资源的函数不匹配的情况。内存管理不匹配可能导致严重后果，如代码执行、内存损坏或程序崩溃。该规则涵盖了几种不匹配的内存管理模式，包括将free()与malloc()、calloc()、realloc()、strdup()和strndup()错误使用的情况。它还检测了delete或delete[]与new或new[]错误使用的情况。
* raptormissingbreakinswitch：检测在C和C++代码中在switch语句或类似结构中省略了break语句的情况。使用各种模式来识别在switch的case中缺少break语句的情况。它还检查在switch的case中是否缺少了return或exit语句，以及它们与break语句的组合。这些模式使用占位符编写，例如"VAR"表示switch变量，"VAL1"和"$VAL2"表示case的值。
* **missingdefaultinswitch：**包含用于识别问题的模式。其中包括一个用于匹配没有任何case或default语句的switch语句的模式，以及一个用于排除已经具有默认情况的switch语句的模式。
* **offbyone：**提供的规则是静态分析工具Raptor的一部分。这些规则旨在检测C和C++代码中的Offbyone错误。Offbyone错误是指软件程序计算或使用的最大值或最小值与正确值相差1。这些规则涵盖了几种可能表示存在Offbyone错误的模式。这些模式包括使用错误索引进行数组访问、可疑的循环超出预期限制、涉及缓冲区大小和字符串长度的比较，以及对strncat函数的潜在误用。
* pointersubtraction：分析软件在计算大小时从一个指针中减去另一个指针，但如果这些指针不在同一个内存块中，则此计算可能不正确。使用各种模式来检测指针减法，包括针对特定指针类型（如char、int、float和double）的基于类型的模式。还包括更通用的模式，涵盖任何指针类型以及指针顺序的变化。这些模式可以是独立的表达式，也可以是包含了指针变量声明和赋值的较大代码块。该规则涵盖了两个指针都明确定义的情况，一个或两个指针具有初始表达式的情况，以及将一个指针的值赋给另一个指针的情况。
* raptorretstackaddress：当函数返回指向局部变量的指针时，这意味着它返回的是一个栈地址。然而，一旦函数返回，栈帧将失效，后续的函数调用可能会重用相同的栈地址，从而覆盖指针的值。这可能导致指针的值出现意外变化，甚至在对指针进行解引用时导致崩溃。
* **signedunsignedconversion：**规则识别了一个特定的有符号到无符号转换实例。变量length是类型为off\_t的变量，通过使用static\_cast将其转换为size\_t类型。此转换的目的是获取内存对象的大小。然而，在进行转换时存在潜在问题。off\_t类型通常是有符号类型，而size\_t是无符号类型。如果length的值为负数，转换后的size将变为一个很大的正数，这可能导致意外行为或错误的结果。
* signedunsignedconversion：识别使用有符号基元并将其转换为无符号基元，或使用无符号基元并将其转换为有符号基元的情况，这可能导致意外值并违反程序的假设。它警告不要依赖有符号和无符号数字之间的隐式转换，因为结果可能是意外的，并引入漏洞。该规则提供了多个模式来检测此类转换，包括赋值、函数返回和特定函数（如strncpy、memcpy、malloc等）。它涵盖了从有符号到无符号和从无符号到有符号的转换。
* typos：旨在通过静态代码分析来捕获常见的编程错误，这些错误可能会引入安全漏洞到代码库中。
* uncheckedretmalloccallocrealloc：有助于识别未正确检查错误的内存分配函数，这可能导致NULL指针解引用或其他漏洞。它提供了对特定函数以及检查或未检查返回值的各种方式的模式。
* uncheckedretsetuidseteuid：强调了软件未检查方法或函数返回值的潜在漏洞，定义了模式以识别未正确检查setuid()和seteuid()函数返回值的情况。它包括用于检测调用这些函数但未检查返回值的模式，以及用于排除使用比较运算符检查返回值或将其赋值给变量的情况的模式。这种疏忽可能导致软件无法检测到意外的状态和条件。如果攻击者能够强制函数失败或返回意外值，那么随后的程序逻辑可能会导致漏洞。
* **unsaferetsnprintfvsnprintf：**检测C和C++代码中对snprintf()和vsnprintf()函数的潜在不安全使用。这些函数返回它们尝试创建的字符串的总长度，这个返回值可能大于目标缓冲区的大小。如果不安全地使用返回值，例如直接将其作为索引用于写入目标缓冲区，可能会导致内存损坏和漏洞。
* unsaferetstrlcpystrlcat：strlcpy()和strlcat()函数返回它们尝试创建的字符串的总长度。对于strlcpy()函数来说，这意味着源字符串的长度；对于strlcat()函数来说，这意味着目标字符串的初始长度加上源字符串的长度。因此，返回值可能大于目标缓冲区的大小。如果不安全地使用返回值，例如将其用作写入目标缓冲区的索引，就可能导致内存损坏和漏洞。尽管此类漏洞并不常见，因为通常会忽略这些函数的返回值，但忽略这些函数的结果可能导致数据截断。
* **unsafestrlen：**规则说明整数溢出或环绕漏洞的潜在后果，强调了它们对资源管理、执行控制和安全决策的影响。它强调了考虑用户提供的输入的重要性，并警告了意外环绕的安全关键性。受输入大小影响时，短整数相对于int更容易受到攻击。
* **unterminatedstringstrncpystpncpy：**规则涵盖了C和C++编程语言中的几个函数，包括strncpy、stpncpy、wcsncpy和wcpncpy。它检查这些函数的使用情况，但目标缓冲区未正确进行空终止。它特别查找函数调用之后缺少空终止的模式，使用不同变体的空字符赋值。
* useafterfree：解释了使用后释放错误的后果以及数据损坏或未定义行为的可能性。
* writeintostackbuffer：使用不同的模式和元变量来匹配具有堆栈缓冲区写入特征的代码模式。它通过分析这些模式的代码来识别潜在的缓冲区溢出漏洞。
* **badwords：**用于检测代码中包含的可疑评论，这些评论可能暗示存在错误、不完整的功能或弱点。规则中列举了一些可疑的评论关键词，如BUG、HACK、FIXME、LATER、LATER2、TODO等，这些关键词表明缺少安全功能和检查，或者表示程序员应该修复的代码问题，比如硬编码变量、错误处理、不使用存储过程和性能问题等。

## Openai 接口

controller.py文件是漏洞分析工具的一部分。它利用OpenAI的自然语言处理模型来增强对C函数中安全漏洞的分析能力。

* 文件中定义了OpenAI类，用作与OpenAI模型交互的主要控制器。
* 该类具有一个初始化方法\_\_init\_\_(self, controller)，接受一个controller对象作为参数。该对象可能包含配置设置，包括用于与OpenAI进行身份验证的API密钥。
* davinci(self, promptData, model)方法利用OpenAI的Davinci模型分析给定的promptData。它使用提供的API密钥进行API调用，并返回分析结果。
* analyzeC(self, function, semgrep)方法负责分析C函数的安全漏洞。它接受一个C函数和semgrep静态分析工具的输出作为输入。它构建了一个提示，将semgrep输出和C函数组合起来，然后调用davinci方法执行AI分析。该方法根据提示的令牌数确定要使用的适当OpenAI模型。
* calcToken(self)方法计算提示数据中的令牌数。它使用与GPT3.5 Turbo模型使用的编码相匹配的tiktoken模块来确定令牌数。

将semgrep的静态分析结果与使用OpenAI模型进行的基于AI的分析相结合。它旨在通过利用自然语言处理能力，提高对C函数中安全漏洞的分析准确性和效果。