计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(9):253−258 [doi: 10.15888/j.cnki.csa.005974]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于代码审计技术的 OpenSSL 脆弱性分析①

杜　江, 罗　权

(重庆邮电大学 网络与信息安全技术重庆市工程实验室, 重庆 400065)

摘　要: 本文讨论应用代码审计技术, 分析 OpenSSL 源代码,进行脆弱性分析, 并作出针对性修补建议. 在进行源码级分析时, 主要采用数据流分析技术, 动态污点分析技术, 定理证明等. 各类代码审计技术由于都主要采用形式化手段分析软件构架的安全需求, 通常都对某种特定场景有较好效果, 但实用性较差. 在审计 linux, xen 等大型成熟软件项目时, 存在效率低下, 误报率高等缺陷, 甚至可能根本无法挖掘出有效漏洞. 为此通过采用搭配使用各种不同代码审计技术, 同时使用一种新的安全属性定义手法, 从底层角度定义安全属性, 以提升其对软件安全需求描述的准确度, 避免其审计缺陷. 在保留代码审计技术自动化程度高的优点同时提升其审计效率以及降低误报率, 深层次发掘代码脆弱性.

关键词: 漏洞挖掘; 代码审计; 形式化; OpenSSL

引用格式: 杜江,罗权.基于代码审计技术的 OpenSSL 脆弱性分析.计算机系统应用,2017,26(9):253–258. http://www.c-s-a.org.cn/1003-3254/5974.html

Vulnerability Analysis of OpenSSL Based on Code Audit Technology

DU Jiang, LUO Quan

(Chongqing Engineering Laboratory of Network and Information Security Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China)

Abstract: This paper discusses the process of applying code audit to analyze the vulnerabilities of OpenSSL source codes and proposes some specific fixing advice for OpenSSL. Source level analysis mainly contains data flow analysis, dynamic taint analysis and path constraint solving proof method, etc. Because various code audit techniques adopt formal analysis on software architecture based on their own security requirements, they usually produce good effects when aiming at some particular scenes, but they lack universality. When auditing important mature projects like linux and xen, it is even impossible to exploit vulnerabilities efficiently with using these code audit techniques with high false rate. In this case, the collocation use of different code audit techniques is applied, as well as a new method of the security attributes definition from the bottom to improve the accuracy of software security requirements description and to avoid the defects in its audit. These methods increase audit efficiency, decrease false positive and process deep vulnerability exploitation while retaining the advantages of the high degree of automation of code audit.

Key words: code audit; vulnerability minin; formal methods; OpenSSL

安全是相对的, 危险是绝对的. 现如今网络安全的两大威胁分别为软件漏洞和恶意代码, 其中软件漏洞对网络安全的影响尤为巨大. SSL 作为为网络通信提供安全及数据完整性的一种安全协议, 而 OpenSSL 作为 SSL 协议的实现, 一旦 OpenSSL 出现未被披露但是已被黑客使用的漏洞或者 OpenSSL 的开发公司——Netscape 在 OpenSSL 中植入后门, 势必将对我国的安全领域造成巨大影响. 对 OpenSSL 做一次脆弱性分析

① 收稿时间: 2016-12-31; 采用时间: 2017-02-08

是完全有必要的, 对开源项目的源代码进行安全审计必定对我国的网络安全有重大意义.

在编程语言不断高级化, 库函数不断封装, 使编程越发便捷的同时, 在代码进行抽象的过程中带来的安全风险也越来越多, 而代码审计的目标就是消除软件工程之中不断封装, 采用更高级语言编写代码时带来的安全风险. 同时消除一些已有的一些大型项目存在的安全风险, 如 linux、xen、OpenSSL 等. 代码审计技术对信息安全具有重大意义. 代码审计技术的最终目标是消除代码中所有的漏洞.

就目前而言, 漏洞挖掘技术主要指代码审计技术以及 fuzz 技术两种. 主流代码审计技术通常指的是数据流分析技术, 污点分析技术, 符号执行技术, 定理证明, 模式判别等. 这些技术通常都是基于形式化的漏洞挖掘手段, 其优点都是自动化程度高, 但是也存在各种各样的缺点, 如误报率高、“路径爆炸”之类的“维数灾难”等问题[1]. 通常这些技术都比较适合应用于比较单纯的某一场景之中, 在实际的工程应用中都存在实用性较差、效率低下、误报率高等致命缺陷. 对于脆弱性分析, 目前在工业界比较流行的是 fuzz 技术, 但 fuzz 技术理论上存在一定缺陷, 如无法深度探测代码脆弱性、测试用例编写困难, 自动化程度低, 样本间关联性差等[2-5]. 未来必将属于理论上更先进的代码审计技术, 但代码审计技术目前而言并未成熟, 难以在实际中应用.

在完整使用代码审计技术对软件工程进行脆弱性分析时, 分为两个过程: 1) 前端——代码解析部分; 2) 后端——形式化脆弱性分析部分.

对于前端部分来说, 主要指数据流分析, 污点分析等技术. 在代码解析过程中, 主要是对数据的流向, 执行过程做出分析, 其精度很大程度上在于是否可以准确描述程序的实际运行过程. 在分析过程中, 条件判断语句, 相关变量大小都将影响语句执行顺序. 在分析过程中, 数据流分析技术分为流敏感与流不敏感分析技术, 而污点分析技术又分为显式污染及隐式污染. 在大型的软件项目的实际分析过程中, 代码量过大导致无法构架出完整的代码模型, 调用流程图等相关信息, 通常会使用忽略隐式污染, 采用不敏感分析方法等降低分析难度, 但这都是以牺牲精度为代价的.

对于后端部分, 主要指定理证明技术, 模式判别技术等. 在对代码构架做形式化判别的过程中, 代码模型中包含的大量语义信息难以做出有效处理, 从编译角度说, 有限的语法, 词法可以表达出无限的语义, 这一点严重影响代码审计技术对代码的脆弱性分析能力. 另外在实际的处理过程中, 在对大型软件项目的状态有穷枚举数据集过大, 导致空间爆炸等问题; 在实际脆弱性判别过程中, 对代码的安全属性以及安全需求的边界难以确定. 最后形式化判别的代码模型与代码本身存在一定误差, 导致大量误报, 漏报等情况[6].

总体而言, 代码审计技术并不成熟, 实用性相对较差, 且误报率高, 但理论上较 fuzz 技术更为卓越自动化, 具有形式化程度高, 挖掘层次深等优点. 代码审计技术具有较高的研究价值.

1 OpenSSL 的审计过程

代码审计技术作为漏洞挖掘技术领域的重要组成部分, 其重要程度不言而喻. 在对 OpenSSL 的审计过程中, 主要使用的是: 数据流分析, 污点分析, 以及定理证明技术. 代码审计技术存在实用性差的缺点, 在实际的应用过程中, 需要搭配使用, 在保留形式化挖掘优点的基础上, 提升其实用性, 真正达到一定的审计效果.

在本次代码审计过程之中, 所采用的方法是利用数据流分析技术分析 OpenSSL 代码构架, 获取软件执行路径信息, 随后在其基础上使用污点分析技术勾勒出受输入信息影响部分, 消减无关代码分支干扰, 最后使用定理证明技术分析脆其弱性, 进行漏洞挖掘工作, 最终实现形式化漏洞挖掘. 在定理证明过程中采用一种新的证明手法, 从底层角度描述代码的安全需求, 消除常规定理技术需要手动在代码加入断言缺陷, 同时提高定理证明技术准确性. 图 1 给出了 OpenSSL 代码审计的流程图.

1.1 数据流分析

数据流分析技术是一种代码审计的支撑技术, 用于获取在程序运行过程中的变量参数等相关信息[1].

OpenSSL 的主执行流程为: 新建 SSL 结构体之后, 选择加密模式, 之后进入 ssl3_connect 或者 ssl3_accept 之中的握手流程, 在握手流程之中, 会调用密码库、io 库、证书库、随机数生成库等等. 而数据流分析的作用就是自动分析出数据在代码之中的流向, 同时收集输入数据在 OpenSSL 执行过程之中的信息.

简单起见, 以流敏感、路径不敏感方法获取程序执行路径, 以及记录中途的参数处理. 由于现当代的代码构架风格, 通常程序在最接近底层的部分只做最基本的操作, 如 send、recv、read、write 等等, 所有的上层操作只做抽象、封装以及逻辑判断等. 数据流分析

技术用于收集程序执行流程中的完整抽象过程以及最终的执行过程的相关信息. 这是一个代码审计的准备过程, 是一个完善程序代码模型的过程.

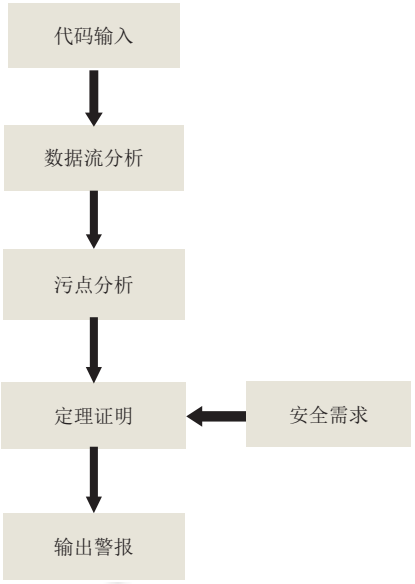

图 1 OpenSSL 代码审计流程图

在这一过程之中, 主要工作是: 1) 建立代码模型; 2) 收集变量信息.

1.1.1 建立代码模型

仅以 ssl3_connect 此主功能函数为例说明:

ssl3_connect->多个握手状态->调用的不同函数.

由于此函数调用流程过于复杂, 图 2 中流程密集到难以观察, 此过程若以手动方式归档处理将无法实现, 现对其中的一小部分放大再做观察.

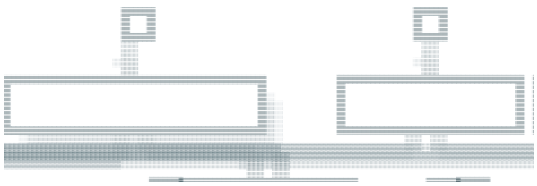

图 2 ssl3_connect 函数调用流程图

图 3 为_send_server_key_exchange 函数在 OpenSSL 中的引用流程图, 使用数据流分析技术对其进行分析.

简单起见, 使用 ida 等静态分析工具直接获取函数调用流程, 但此类调用流程路径不敏感, 需进一步分析. 在拥有源代码的情况下, 使用判别分析等手段进行路径再次分析, 约束语句判断, 如 if 语句, while 语句, switch 语句等. 消减分支, 将分支数据归档, 此过程可以依靠符号执行技术完成[4]. 在此分析过程中, 建立调用流程图供后续分析时使用, 简单来说, 即在对底层代码中任意一个变量或者函数感兴趣时候, 可以得到一个完整的, 从 OpenSSL 获取数据包开始一直到此函数的所有调用过程, 以及所有的此函数继续调用的所有代码路径. 在要求最低情况下, ida 已经可以达到要求.

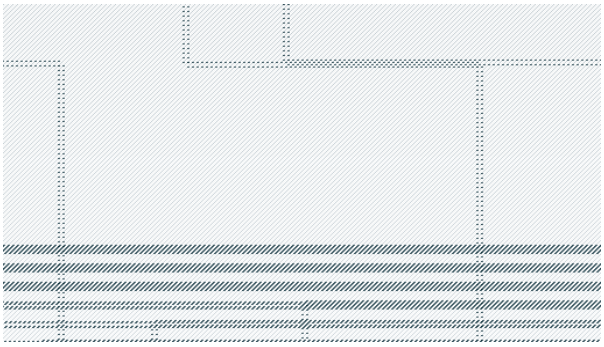

图 3 部分调用流程图

1.1.2 收集变量信息

以输入信息为起点, 获取所有调用过程之中作为参数传递的变量. 实际而言, 所有的参数变量信息将变成一棵巨大的树, 这个树上的每个节点, 都代表一个参数变量. 随着代码执行的深入, 这棵树会越来越大. 而在后续代码审计的步骤中需要查询代码树中任意结点的可能状态以及程序执行路径, 用于支持其他代码审计技术.

其中结点即为函数的参数, 每个结点包含子函数中敏感信息. 以此树来描述代码中的流程信息.

```
Struct{
Int type;
Char[20] name;
Int max;
Int min;
Void* choice;//参数在多种不同情况下的可能不同取值数组.
Int len;
Int fromnode;
Int[20] nextnode;
……;
}node;
```

1.2 污点分析

污点分析是一种跟踪并分析污点信息在内存中流动的技术. 所有能够被输入影响到的变量及被影响变

量所影响到的变量都属于污点变量, 在污染变量的过程被称为污点扩散. 污点分析技术在漏洞挖掘之中意义重大, 分析污点数据是否可以影响代码关键操作, 判别是否存在漏洞, 常被用于 sql 注入漏洞的检测. 此技术容易受编译器影响, 同时构造复杂, 在这里, 同样将其当做漏洞挖掘的支撑技术看待, 规避这些缺陷.

并非所有的可以影响关键操作都是漏洞, 若可影响关键操作的数据被净化, 即对污点数据进行验证, 约束则不存在脆弱性, 当然也有可能验证不严, 在污点分析中称为净化不完全. 如果, 此数据与污点数据无直接关系, 但是有逻辑关系, 如:

```
Char ch[]={"……"}//污点数据
If(strlen(ch))
{int type=0;}
Else
{int type=1;}
```

此类污染被称为隐式污染, 此类污染造成漏洞的一个典型类型是 UAF 类型漏洞, 典型漏洞例子为 linux 的 CVE-2016-0728, 在引用计数整数溢出变为 0 时, 导致内核对象被释放, 在被释放空间内伪造虚假内核对象, 调用原释放空间保留的引用执行任意代码.

实现污点分析, 隐式污染的分析是一个难点. 显式污染, 可以使用语法树分析处理, 对于隐式污染处理较为复杂. 好在隐式污染较为罕见, 对于隐式污染, 由于隐式污染一定伴随于显式污染, 这里主要依靠其后的定理证明技术排除显示污染附近代码上下文可能的危险操作, 一定程度上可以缓解隐式污染所造成的不利影响.

污点分析技术的最大难点在于净化的过程, "净化"顾名思义即对受污染变量的约束过程. 真正的问题在于如何确定净化完全, 即确定是否存在验证不严问题. 形式化判别净化是否完成是一个巨大难题, 这里将此任务将交由定理证明技术完成.

1.3 定理证明技术

定理证明的方法是指, 以公式的形式描述软件安全属性, 若代码可以突破安全属性范围, 则发出警报. 而这个安全属性由安全工程师以自己的经验提炼. 此技术优点在于严格的推理证明控制检测的进行, 误报率低, 方便辅助安全工程师编写测试用例等[7,8].

常规定理证明技术包括数学定理证明、硬件验证、协议验证等. 定理证明技术的实现关键在于证明器的编写, 用于判断过程内代码是否存在漏洞, 过程间代码是否存在漏洞. 通常需要在程序中插入大量断言、注释等辅助证明器工作[3,6]. 其缺陷在于定理规定复杂, 扩展性不强, 需要添加断言之类在代码审计过程中进行交互, 增加程序员工作量等.

这里不使用常规定理证明方法, 直接判别程序动作的底层行为. 安全属性定义方法示例: 写操作字节数不可以超过限定内存空间大小, 如 memcpy 的 n 不可以超过 dest 的内存大小; 任意一个函数不可以有两条路径返回成功. 摒弃高级语言的语义级意义分析, 从底层行为角度定义安全属性, 排除编程语言进行描述时的多样性干扰. 从底层角度针对漏洞类型定义的安全属性, 可以提升其对软件的安全需求描述的准确度, 且无需手动在代码中插入大量断言用以支持证明过程中的布尔计算. 证明器可在遍历代码状态时自动识别底层行为性操作进行, 验证是否存在脆弱性.

以 OpenSSL 之中一个瑕疵代码为例, 此瑕疵与 OpenSSL 提供的测试代码配合可造成信息泄漏, 一次可以泄漏 255 字节数据.

```
static char ssl_next_proto_validate (unsigned char *d, unsigned len)
{
  unsigned int off = 0;
  while (off < len) {
    if (d[off] == 0)
      return 0;
    off += d[off];
    off++;
  }
  return off == len;
}
```

此 size 变量为 0 时候可以直接越过验证, 实现异常路径, 也就是说参数可以影响程序执行路径, 导致代码流程改变, 最终导致程序漏洞的出现.

在 OpenSSL 中, 若以"所有堆块必须在使用完之后被释放, 且释放只能一次, 释放后的指针无法使用"为一条安全属性. 根据先验知识, 如此设置约束规则起码可以发现 CVE-2015-0206、CVE-2014-3571 等漏洞.

但在更加复杂的逻辑缺陷导致漏洞中, 如最新的 CVE-2015-4484 提交密码错误 93 次返回 root 用户, 目前整个学术界都没有较好的检测方法. 堆溢出、栈溢出、内存泄露等典型漏洞严格来说属于编码规范化问

题, 都存在一定程度上的内存破坏行为. 对复杂的逻辑漏洞而已, 其形式上已经接近于开发者刻意留下的后门, 不再具有典型漏洞的特征, 在漏洞的利用过程中所执行的是代码正常的逻辑行为, 并没有 shellcode 存在. 若需要检测出逻辑漏洞, 需要对代码语义进行解析, 做出逻辑级的判断. 就目前的人工智能水平而言, 尚无法对机器语言(或者说编程语言)背后的自然语义做出形式化分析, 此类漏洞通常在学习者学习研究逆向相关项目时发现.

2 OpenSSL 漏洞分析

2.1 漏洞概述

OpenSSL 其本质是一个库文件. 以 ssl3_connect、ssl3_accept 等函数封装 connect、accept 等通信函数, 以透明的方式在程序的运行过程之中加入握手加密等过程. 在 ssl3_accept、ssl3_accept 等函数中, 调用 c 标准库函数中的 connect、accept 等函数, 实现通信过程中收、发过程中的加、解密等功能.

此次审计过程之中, 挖掘的 0day 漏洞, 已提交, 获取漏洞编号 CVE-2016-2179, 网景通信公司(Netscape Communications Corporation)致谢声明网址为:

https://git.openssl.org/?p=openssl.git;a=commit;h=f5c7f5dfbaf0d2f7d946d0fe86f08e6bcb36ed0d

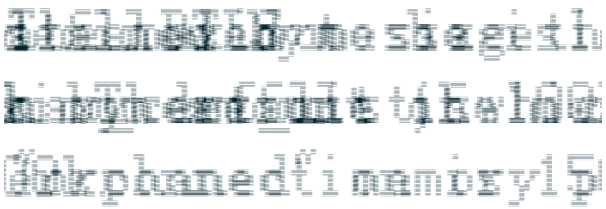

图 4 致谢声明

2.2 CVE-2016-2179

OpenSSL 通常作为服务器套件存在, 一般使用在 linux 服务器上, 但是由于使用 windows 平台之上的调试工具更为便捷, 本次实验环境为 win7 x86 系统, 以及 od2.1、openssl-1.0.2h, 辅助分析工具 Wireshark.

CVE-2016-2179 漏洞位于 d1_both.c 文件的 dtls1_get_message_fragment 函数中. 与基于 tcp 的加密不同, udp 有分片特性, 此函数作用是对于 udp 数据包所传送数据的抽象, 使上层函数不关心 udp 是否分片, 直接从接受缓存区读取数据. 此函数曾被爆出多个漏洞, 甚至包括一个堆溢出远程任意代码执行的高危漏洞.

此函数在接收数据过程之中, 若 udp 数据包分片, 则需要申请内存空间接收分片, 将所有分片数据接收完成, 排列整齐之后, 将数据拷贝至数据接收空间处. 在此抽象过程之中存在问题: 若在接收过程之中, 此次握手消息第一次接收到的是一个分片消息, 则申请内存空间, 将此次接收到的信息插入 buffered_messages 队列中, 若第二次接受到的是此次消息的完整数据包, 则此函数将误认为此次握手消息没有分片, 此函数将直接越过内存空间释放将消息挂入 ssl 结构体中, 引起内存泄露, 最终可导致服务端客户端双方的拒绝服务攻击. 每次握手最多可以申请出约 16883 个字节大小的内存块. 若同时向服务端申请多个会话, 每个会话的每次握手都可以申请.

10638 字节大小的内存空间, 最终将导致由于内存资源不足而引起的拒绝服务.

具体请看代码:

```
len = msg_hdr.msg_len;
frag_off = msg_hdr.frag_off;
    frag_len = msg_hdr.frag_len;
if (msg_hdr.seq != s->d1->handshake_read_seq
    &&!(s->d1->listen && msg_hdr.seq == 1))
Return dtls1_reassemble_fragment(s, &msg_hdr, ok);
```

无关代码片段省略, 由于是 udp 数据包, 数据包存在接受到的数据顺序和对方发送顺序不一致的情况, 所以 OpenSSL 提供的接收函数 dtls1_process_out_of_seq_message 将所接收到的 udp 数据包中数据取出后重新排序, 同时只要 udp 数据长度小于信息长度, 说明此数据包分分片报文, 则调用 dtls1_reassemble_fragment 函数, 此函数功能为组装分片 udp 数据包: 申请空间, 将 udp 数据包中数据在此中转, 组装成完整握手消息后输出. 被泄露的内存便是在此函数中被创建, 若此次握手消息第一次被接收, 便为其申请空间:

```
if (item == NULL) {
frag = dtls1_hm_fragment_new (msg_hdr->msg_len, 1); if (frag == NULL)
goto err;
memcpy(&(frag->msg_header), msg_hdr, sizeof(*msg_hdr));
frag->msg_header.frag_len = frag->msg_header.msg_len;
```

```
frag->msg_header.frag_off = 0;}
```

至此, 此将完成内存申请, 若再次发送此次消息的完整数据包之后, 便可从此函数返回接收数据成功, 从而越过这块内存数据的处理释放过程. 请看代码:

```
if (item == NULL) {
frag = dtls1_hm_fragment_new (msg_hdr->msg_len, 1); if (frag == NULL)
goto err;
memcpy(&(frag->msg_header), msg_hdr, sizeof(*msg_hdr));
frag->msg_header.frag_len = frag->msg_header.msg_len;
frag->msg_header.frag_off = 0;
}
(unsigned char *)s->init_buf->data + DTLS1_HM_HEADER_LENGTH;
i = s->method->ssl_read_bytes(s, SSL3_RT_HANDSHAKE &p[frag_off], frag_len, 0)
}
```

由代码可知, 如果第二次收到的数据包是完整的, 代码将直接进入消息接收阶段. 代码误认为此 udp 数据包是没有被分片的, 直接接收数据, 忽略了对分片数据接收缓冲区的处理与释放导致内存泄露.

在 od 中, 调试此过程, 在后续握手过程之中, 此内存块依旧处于占用态, 并没有被释放.

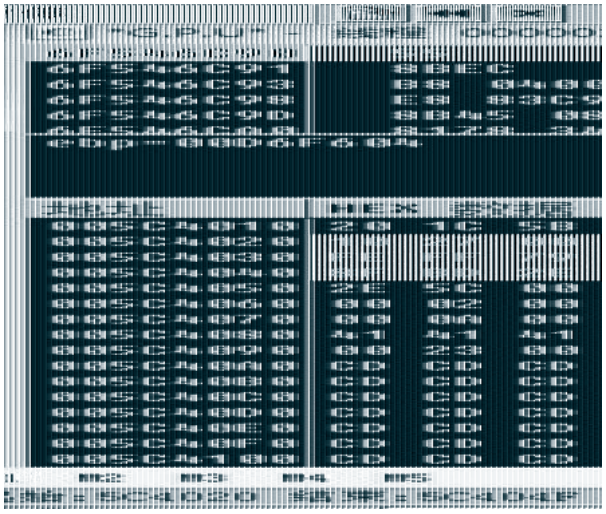

图 5 触发效果

3 修补方案

将每次接受到完整 udp 数据包消息时候检查 buffered_messages 队列, 若存在有关本次消息的内存信息存储, 则释放即可.

在每次握手过程之中检查上次握手的消息队列 buffered_messages 是否存在消息碎片未释放, 如果存在则释放未释放的消息碎片, 即可消除此漏洞造成的内存泄露情况.

4 结语

本文讨论的是多种代码审计技术的交互使用. 设计数据流分析, 污点分析以及定理判别三类技术, 三种技术相互配合提高审计效率以及审计准确性, 阐述了 OpenSSL 的一个漏洞的发现过程以及利用过程. 由于代码审计技术发展的局限以及本人水平有限, 代码审计手法自动化程度还未达到全自动目标, 通用性较差, 需要对此审计方案做较大改动. 由于此次工作主要目的为针对 openssl 进行代码审计并非实现代码审计工具, 且编写完整代码审计工具工作量过大, 此次审计采用编写多个工具, 分步骤手动执行, 采用半自动方式处理, 并未实现真正全自动审计, 另外, 对于非源代码原因引入漏洞, 如编译器缺陷、bug 导致软件漏洞, 代码审计技术将无法分析出其脆弱性.

参考文献

1 吴世忠, 郭涛, 董国伟, 等. 软件漏洞分析技术. 北京: 科学出版社, 2014.

2 文硕, 许静, 苑立英, 等. 基于策略推导的访问控制漏洞测试用例生成方法. 计算机学报, 2016, (39): 1–15.

3 牛伟纳, 丁雪峰, 刘智, 等. 基于符号执行的二进制代码漏洞发现. 计算机科学, 2013, 40(10): 119–121, 138. [doi: 10.3969/j.issn.1002-137X.2013.10.024]

4 朱正欣. 二进制程序的动态符号化污点分析[硕士学位论文]. 合肥: 中国科学技术大学, 2015.

5 马俊. 基于模拟器的缓冲区溢出漏洞动态检测技术研究[硕士学位论文]. 长沙: 国防科学技术大学, 2008.

6 吴世忠. 信息安全漏洞分析回顾与展望. 清华大学学报(自然科学版), 2009, 49(S2): 2065–2072.

7 张健. 精确的程序静态分析. 计算机学报, 2008, 31(9): 1549–1553.

8 肖庆. 提高静态缺陷检测精度的关键技术研究[博士学位论文]. 北京: 北京邮电大学, 2012.