一种面向 UEFI 模块的形式化建模与验证方法

王 冠[1,2],郝晓星[1,2]
(1. 北京工业大学 信息学部,北京 100124;
2. 北京市可信计算重点实验室,北京 100124)

摘 要:固件作为一种固化在 ROM 中的特殊软件程序,主要负责加电自检,硬件设备初始化,引导操作系统等基础功能,运行级别和安全等级较高,亟需一种高效、可靠的 UEFI 模块安全检测方法。采用形式化方法对 UEFI 模块进行规约与验证,对于提高固件的安全性具有重要意义。基于现有的有限状态自动机和下推自动机基础,分别对 UEFI 模块中的安全漏洞属性和 UEFI 模块程序控制流进行形式化建模,利用模型检验对上述模型进行形式化验证。其中利用数据抽象思想将 UEFI 模块抽象为程序控制流且压缩其状态规模来缓解模型检验时的状态爆炸问题,并给出了相关模型的定义以及模型间转换、组合的算法。实验结果表明,对 UEFI 模块的抽象及压缩能够很好地缓解模型检验中的状态爆炸问题,并且该形式化验证方法能够实现对 UEFI 模块安全漏洞的自动化验证,且能够达到较低的漏报率。

关键词:UEFI;形式化方法;模型检验;安全漏洞;有限状态自动机;下推自动机
中图分类号:TP301 **文献标识码**:A **文章编号**:1673-629X(2021)12-0116-06
doi:10.3969/j.issn.1673-629X.2021.12.020

A Formal Modeling and Validation Method for UEFI Module

WANG Guan[1,2],HAO Xiao-xing[1,2]
(1. Faculty of Information Technology,Beijing University of Technology,Beijing 100124,China;
2. Beijing Key Laboratory of Trusted Computing,Beijing 100124,China)

Abstract:Firmware,as a special software program solidified in ROM,is mainly responsible for power-on self-check,hardware device initialization,guiding the operating system and other basic functions. The operating level and safety level are high,so an efficient and reliable UEFI module safety detection method is urgently needed. Formal methods are used to specify and verify UEFI modules,which is of great significance to improve the security of firmware. Based on the existing finite state automaton and push down automaton,the security vulnerability attributes and the program control flow of the UEFI module are formalized,and the above models are formally verified by model check. The UEFI module is abstracted into program control flow by data abstraction and its state scale is compressed to alleviate the state explosion problem in a model test,and the definition of related models and the algorithm of transformation and combination among models are given. The experiment shows that the abstraction and compression of the UEFI module can alleviate the state explosion problem in the model test,and the formal verification method can realize the automatic verification of the UEFI module security vulnerabilities and can achieve a low omission ratio.

Key words:UEFI;formal method;model checking;security vulnerabilities;finite state automaton;push down automaton

0 引 言

UEFI 作为定义在平台固件层与操作系统层之间的一个开放且抽象的业界标准编程接口,与传统的 BIOS 相比绝大部分代码采用 C 语言编写,提升了 UEFI 固件的开发效率,并且采用驱动模块化设计提高了 UEFI 系统的可扩展性,使得 UEFI 迅速取代 BIOS。与此同时,UEFI 固件程序容量几何数量增长,功能愈加复杂,新颖的接口以及驱动加载模式,再加上大量的 C 语言源码问题,使得 UEFI 固件的安全问题更加明显,更由于 UEFI 固件在计算机中的特殊位置,使其成为攻击操作系统的跳板[1]。目前针对 UEFI 固件的安全检测技术主要是模糊测试技术、污点分析技术、符号执行技术。模糊测试技术[2-3]在大体量软件漏洞检测中具有先天优势,可重用性高,但自动化程度不高,效

收稿日期:2021-01-08 **修回日期**:2021-05-12
基金项目:国家重点研发计划(2019YFB2102303);国家自然科学基金(61971014)
作者简介:王 冠(1968-),男,硕士生导师,副教授,研究方向为信息安全、可信计算、数据挖掘与智能信息系统;郝晓星(1995-),男,硕士研究生,研究方向为信息安全。

率低下,代码覆盖率不高。污点分析技术[4]可提供精确的信息流传播跟踪,但需要额外的系统开销与大量的先验分析,检测条件苛刻。符号执行技术[5]由于对路径敏感,状态爆炸问题严重,没有很高的分析精度。

形式化方法[6-8]是基于严格的数学基础,对计算机硬件、软件系统进行形式规约、开发和验证的技术。形式化验证方法对程序模型和安全属性进行自动验证且验证时能覆盖整个程序空间,漏报率低,非常适合高安全级别系统或程序的安全验证及分析。为对 UEFI 模块中安全漏洞进行分析与检验,提高 UEFI 固件的安全性,该文提出一种针对 UEFI 模块的形式化建模与验证的方法。该方法先对 UEFI 模块中可能存在的安全漏洞属性和 UEFI 模块进行形式化建模分析,再利用数据抽象技术[8-9]缓解形式化验证过程中的状态爆炸问题。

1　相关技术

1.1　形式化方法

形式化方法的两项主要研究内容为形式规约与形式化验证。形式规约利用形式规约语言描述整个系统的模型或者系统需要满足的安全属性[7]。形式化验证就是验证现有的程序或者系统是否满足其形式规约。形式化验证有两种形式,基于演绎的定理证明和基于算法的模型检验。定理证明的基本思想为:程序或系统满足其形式规约,然后通过一系列推理规则,以演绎推理的方式来证明此逻辑命题的正确性[7]。模型检验[7,9-10]提出的基本思想是:随着计算机并发系统的高速发展,验证一个程序或系统是否满足一个公式要比证明该公式在所有情况下均被满足要容易得多,所以提出了在有限状态自动机模型上来验证公式的可满足性的新思路。模型检验通过自动遍历程序或系统的有限状态来检验系统模型与其性质规约的满足关系。如果程序或系统的形式化模型不满足给定的性质规约,模型检验算法会给出不满足的反例,可以根据反例对系统或程序进行分析和调试,如果模型检验算法未发现反例,则系统或程序一定满足所检验的性质[7]。

1.2　有限状态自动机

有限状态自动机(finite state automaton, FSA)[11-12]分为非确定和确定两种形式,两者是等价的,其唯一的区别就是状态转移函数不同,确定有限状态机一个输入对应一个状态转移,非确定有限状态自动机一个输入对应多个状态转移,然后从多个状态转移中非确定地选择其中一个。有限状态自动机识别正则语言[12],其所接受的所有字符串构成了自动机识别的语言 L(M)。确定有限状态自动机的定义如下:

定义 1:确定有限状态自动机(deterministic finite automaton, DFA)可以写成元组的形式(Q, Σ, δ, Q_0, F),其中:

(1) Q 表示状态的非空有限集合,$\forall q \in Q$, q 称为 DFA 的一个状态;

(2) Σ 表示输入字符的集合,$\Sigma = \{a_0, a_1, \cdots, a_n\}$;

(3) δ 表示状态转移或移动函数,$\delta: Q \times \Sigma^* \rightarrow Q$, $\delta(q_n, a_n) = q_m$;

(4) $Q_0 \in Q$ 表示 DFA 的初始状态;

(5) $F \subseteq Q$ 表示 DFA 的终止状态集合,$\forall q \in F$, q 称为 DFA 的一个终止状态。

确定有限状态自动机 DFA 从初始状态 Q_0 开始,逐个读入字符集 Σ 中的字符,根据当前状态 q_n、输入的字符 a_n 和状态转移函数 δ 来决定 DFA 的下一状态 q_m。当输入字符结束,如果 DFA 的当前状态 $q_m \in F$,表示 DFA 接受该字符集 Σ,否则表示 DFA 不接受该字符集。

1.3　下推自动机

下推自动机(push down automaton, PDA)[11-12]除了包含有限状态外,还包括一个长度不限的栈,下推自动机的状态转移不仅参考有限状态部分还要考虑栈的当前状态,状态转移过程还包括栈的出栈、入栈操作。下推自动机的定义如下:

定义 2:下推自动机 PDA 可以写成元组的形式(Q, Σ, Γ, δ, Q_0, Z_0, F),其中:

(1) Q、Σ、Q_0、F 与 DFA 中含义一致;

(2) Γ 表示有限的堆栈字母表,为能够推入堆栈的符号的集合;

(3) δ 表示转移函数,δ 为 $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ 到 $Q \times \Gamma^*$ 子集的映射,映射关系:$\delta(q, a, X) = \{(q_1, \gamma_1), (q_2, \gamma_2), \cdots, (q_n, \gamma_n)\}$,控制着自动机的行为。形式上 δ 的自变量可以表示为一个三元组 $\delta(q, a, X)$,其中 $q \in Q$, 表示 PDA 当前状态;$a \in \Sigma$ 或 $a = \varepsilon$, ε 为空串表示不是输入符号;$X \in \Gamma$,表示当前堆栈栈顶符号;

δ 的输出是序对 (q, γ) 的有穷集合,其中 q 表示新状态,γ 表示堆栈符号用来替代当前堆栈栈顶符号 X,如果 $\gamma = \varepsilon$,则弹出栈顶元素 X,如果 $\gamma = X$,堆栈不发生改变,如果 $\gamma = YZ$,那么栈顶符号 X 被 Z 代替,然后 Y 推入栈;

(4) $Z_0 \in \Gamma$,表示 PDA 栈中的初始符号。

2　形式化建模与验证方法设计

该文使用模型检验对 UEFI 模块进行形式化验证,其中 UEFI 模块代码采用下推自动机 PDA 进行建模,UEFI 模块中可能存在的安全漏洞属性采用确定有

限状态自动机 DFA 进行建模,最后采用模型检验算法验证 UEFI 模块是否满足给定的安全漏洞属性模型。UEFI 模块形式化验证流程如图 1 所示。

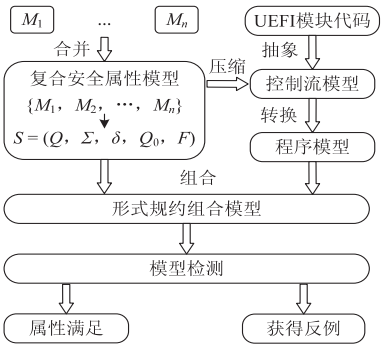

图 1 形式化验证流程

2.1 安全漏洞属性建模

利用有限状态自动机对 UEFI 模块的安全漏洞属性进行建模。首先,需要对安全漏洞进行形式化分析,确定安全漏洞的状态集合 Q、初始状态 Q_0、终止状态集合 F。其次,将安全漏洞所涉及的一系列可能的程序点采用抽象语法树(abstract syntax tree,AST)[13]进行描述,作为有限状态自动机的输入信息 Σ。最后,根据状态间的转移关系设计状态间的转移函数 δ。安全漏洞的形式化描述中每一次状态转移的结果是确定的,所以基于确定有限状态自动机 DFA 的 UEFI 模块安全漏洞属性模型(security vulnerabilities deterministic finite automaton,SVDFA)的定义如下:

定义 3:SVDFA 可以写成元组的形式(Q,Σ,δ,Q_0,F),其中:

(1) Q 表示安全漏洞所涉及的非空有限状态集合,$\forall\ q \in Q$,q 为在安全漏洞状态转移时可能达到的任意状态;

(2) Σ 表示可能引起安全漏洞状态转移的程序操作表达式集合,程序操作源码转为与上下文无关的 AST 语法描述的表达式;

(3) δ 表示安全漏洞状态转移函数集合,$\delta: Q \times \Sigma^* \rightarrow Q$,$\delta(q_n, a_n) = q_m$,$q_n \in Q$ 表示当前状态,$a_n \in \Sigma$ 表示输入的程序节点表达式,$q_m \in Q$ 表示转移后状态;

(4) $Q_0 \in Q$ 表示安全漏洞的初始状态;

(5) $F \subseteq Q$ 表示满足安全漏洞的最终接受状态集合。

SVDFA 的状态迁移从初始状态 Q_0 开始,如果当前程序执行点的 AST 表达式 a_n 与当前状态 q_n 能满足状态转移函数 $\delta(q_n, a_n) = q_m$,SVDFA 当前状态由 q_n 变为 q_m,如果程序检验完毕,SVDFA 在状态转移过程中有状态 $q \in F$,则表示该程序可能出现安全漏洞模型描述的安全问题,否则没有出现安全漏洞属性模型描述的安全问题。UEFI 模块中的安全问题可能非常复杂,如果一开始就对一个复杂的安全问题进行形式化建模将非常困难且不精确。通常一个复杂的安全问题由几个简单的安全问题组成,采用笛卡尔乘积的方式将一些简单安全漏洞属性模型$\{M_i = (Q_i, \Sigma_i, \delta_i, Q_{0i}, F_i)\}$集合组合成一个复杂的安全漏洞属性模型 $S = (Q, \Sigma, \delta, Q_0, F)$,这种建模方式使得安全漏洞描述模块化,并且使得模型扩展、复用成为可能。

2.2 程序控制流模型

数据抽象作为一种解决状态爆炸问题的有效手段,其基本思想就是在对系统进行模型检验时剔除系统中与检验无关的信息,仅保留有用信息,最后对抽象系统进行分析与验证就会变得容易,进而缓解模型检验时的状态爆炸问题。将 UEFI 模块近似抽象为对应的控制流模型后,对于复杂的 UEFI 模块,状态空间数量仍然很多,而要检验的安全漏洞属性往往是有限状态空间。该文根据安全漏洞属性再次压缩控制流模型,将控制流模型中与安全漏洞属性无关的控制流进行删除,进一步压缩状态空间,最终得到只与安全漏洞属性相关的 UEFI 模块抽象模型。将 UEFI 模块代码抽象为与程序节点相关的控制流模型,只对抽象出的控制流进行建模,降低整个模型的复杂度。本建模方式关注点在于程序执行路径中有哪些可能被执行的函数或方法以及执行顺序,并不考虑真实执行下的具体数据流,是一种对原系统的近似抽象。UEFI 模块的程序控制流图(UEFI control flow graph,UCFG)定义如下:

定义 4:UCFG 可以写为元组的形式(N,E,IE,CE),其中:

(1) N 表示所有程序节点的集合,$\forall\ n \in N$,n 表示程序控制流中的一个程序节点,n 具有三种属性 n(NAME,OP,TYPE),其中 NAME 表示程序点的名称,OP 表示程序节点源码转为抽象语法树 AST 描述后的表达式,TYPE 表示程序节点的类型。程序节点 TYPE 有三种类型;

· CALL,表示调用普通过程(方法或函数)的程序节点;

· RETURN,表示此程序节点为 return 操作;

· OTHER,表示除 CALL 和 RETURN 以外的其他程序节点。

(2) $E \subseteq N$,表示程序入口点集合;

(3) IE 表示传输边,为程序控制流中普通过程内的程序节点传输边,IE = $\{< n_1, n_2 > \mid n_1, n_2 \in N$ 且 n_1, n_2 属于同一过程内$\}$;

(4) CE 表示调用边,为程序控制流中普通过程间的程序节点调用边,CE = $\{< n_1, n_2 > \mid n_1, n_2 \in N$ 且 n_1,

n_2 不属于同一过程内}。

例如,在对如图 2 所示的函数调用序列进行控制流建模时会有:传输边 IE={(n_1,n_2)},表示一条从程序点 n_1 到程序点 n_2 的过程内传输边;调用边 CE={(n_1,n_x),(n_2,n_y)},表示会有从程序点 n_1 到函数 FunctionA()定义节点 n_x 和 n_2 到函数 FunctionB()定义节点 n_y 两条调用边。

```
EFI_STATUS FunctionA(...){          //程序点n_x
}
...
EFI_STATUS FunctionB(...){          //程序点n_y
}
...
EFI_STATUS Function(...){
    ...
    Status = FunctionA(...);        //程序点n1
    Status = FunctionB(...);        //程序点n2
    ...
}
```

图 2　函数调用序列

由于静态分析只是对语法语义的分析,并不涉及真实数据流,对于条件判断语句如 if-else 或 while 等控制结构,并不会做出精准判断,目前采用的建模策略为默认每条分支都有可能执行,如图 3 的代码序列,会产生传输边 IE={(n_1,n_2),(n_1,n_3)}。

通过上述 UEFI 模块程序控制流模型 UCFG,就可以描述一个程序中所有的程序节点转换路径,源程序被抽象为一个从入口程序点出发,沿着传输边 IE 与调用边 CE 的一条有限或无限的有向图。

```
EFI_STATUS Function(...){
    ...
    Status = FunctionA(...);            //程序点n1
    if … then
        Status = FunctionB(...);        //程序点n2
    end
    else
        Status = FunctionC(...);        //程序点n3
    end
    …
}
```

图 3　控制结构序列

UEFI 模块安全漏洞属性的有限状态机模型 SVDFA 中的 Σ 集合数量有限,且在大多数情况下远远少于程序控制流 UCFG 中的程序点集合 N 的数量,所以在抽象的程序控制流中仍有大量与安全漏洞检测无关的控制流节点,该文利用安全漏洞属性模型 SVDFA 对程序控制流 UCFG 进行压缩,删减无用节点,节点类型的判定算法描述如下:

1. initialize: Uesful Node UF←{},Uesless Node UL ←{}
2. for ∀ $n \in N$ do
3.　　if $n \in E$ or n. OP ∈ SVDFA. Σ then
4. UF←{ n }
5.　　end if
6. end for
7. for ∀ $n \in N$ do
8. if $n \notin$ UF and (∀ $n \in$ DFS(n, {IE,CE})) $\notin$ UF then
9. UL←{ n }
10.　　end if
11. end for

其中 DFS(n ,{IE,CE})函数表示找出程序节点 n 在边集{IE,CE}中直接或间接调用的所有节点。

2.3　程序建模

将 UEFI 模块代码抽象的程序控制流模型 UCFG 转换为下推自动机 PDA 时, Q , Q_0, F 都只包含一个虚拟状态{ s },即表示在 UCFG→PDA 时 PDA 不存在状态的转换,只是利用 Σ , Γ , δ 来记录程序节点的执行路径。UEFI 模块代码的下推自动机模型(UEFI push down automaton,UPDA)定义如下:

定义 5:UPDA 可以表示为元组形式(Q , Σ , Γ , δ , Q_0, Z_0, F),其中:

(1) Q ={ s },表示状态的有限集合;

(2) Σ ={IE∪CE},表示输入符号的有限集合,实际为控制流模型 UCFG 中所有类型边的集合;

(3) Γ ={ N },表示有限的堆栈字母表,实际为控制流模型 UCFG 中所有程序节点的集合 N ;

(4) δ 表示转移函数集合, δ 为 $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ 到 $Q \times \Gamma^*$ 子集的映射,映射关系: $\delta(s,a,X) = \{(s,\gamma_1),(s,\gamma_2),\cdots,(s,\gamma_n)\}$。转移函数生成算法如下:

1. for ∀ $e \in \Sigma$, $e = \{<n_1,n_2> | n_1,n_2 \in \Sigma\}$ do
2.　　if e. n_1. TYPE=RETURN then
3.　　UPDA. δ add δ (s , e ,e. n_1)= {(s , ε)}
4.　　end if
5.　　if e. n_1. TYPE=CALL then
6. UPDA. δ add δ (s , e ,e. n_1)= {(s ,e. n_2 e. n_1)}
7.　　end if
8.　　if e. n_1. TYPE=OTHER then
9.　　UPDA. δ add δ (s , e ,e. n_1)= {(s ,e. n_2)}
10.　　end if
11. end for

(5) Q_0 = s ,表示初始状态;

(6) $Z_0 \in$ { E },表示 UPDA 堆栈中的初始符号,实际为控制流模型 UCFG 中的程序入口节点;

(7) F ={ s },表示 UPDA 的接受状态或终结状态集合。

2.4　形式化验证

基于自动机理论[12]的形式化验证框架如下:假设采用 Σ 表示程序中所有程序节点表达式集合, $S \subseteq \Sigma^*$ 表示所有符合安全漏洞属性的程序节点执行序列, $\gamma \in \Sigma^*$ 表示程序中一条可行的程序节点执行序列。$\Gamma \subseteq \Sigma^*$ 表示程序中所有可行程序节点执行序列的集合。判断 $S \cap \Gamma$ 是否为空,如果为空则说明没有出现安全漏洞,不为空则可能出现安全问题。

通常程序可能的执行序列 Γ 是不可计算集,因此 $S \cap \Gamma$ 为一个不可判定问题,但可以通过限定 S 和 Γ

的形式将问题具体化来判定。首先,在 UEFI 模块中通常的时序安全漏洞属性的程序节点序列都能使用正则语言描述,假设 S 为正则语言,则存在 SVDFA 可以识别 S,即 $S \subseteq L$(SVDFA)。其次,程序的可行执行路径需要一个栈来记录返回调用者地址,需要用上下文无关语言描述,所以假设 Γ 为一个上下文无关的语言,则存在 UPDA 可以识别 Γ,即 $\Gamma \subseteq L$(UPDA)。判定 $S \cap \Gamma$ 变为判定 L(SVDFA)$\cap L$(UPDA)是否为空,则定义 $P = L$(SVDFA)$\cap L$(UPDA),因为 L(SVDFA)为正则语言,L(UPDA)为上下文无关语言,则 P 为上下文无关语言。同理,P 可以被一个下推自动机(security vulnerabilities UEFI push down automaton,SVUPDA)接受,且 SVUPDA 为 SVDFA 与 UPDA 的交集。SVUPDA 可由 SVDFA 与 UPDA 组合而成[14],定义如下:

定义 6:SVUPDA 可以表示为元组形式(Q,Σ,Γ,δ,Q_0,Z_0,F),其中:

(1) Q 表示状态的有限集合,实际值为 SVDFA 中安全漏洞的状态集合 Q;

(2) Σ 表示输入符号的有限集合,实际值为 UPDA 中所有类型边的集合 Σ;

(3) Γ 表示有限的堆栈字母表,实际值为 UPDA 中所有程序节点的集合 Γ;

(4) δ 表示转移函数集合,δ 为 $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ 到 $Q \times \Gamma^*$ 子集的映射,映射关系:$\delta(q,a,X) = \{(q_1,\gamma_1),(q_2,\gamma_2),\cdots,(q_n,\gamma_n)\}$。转移函数生成算法描述如下:

```
1. for ∀ i ∈ UPDA. δ, i(s,e,n1) = {(q1, X)}, e = {<n1, n2> | n1, n2 ∈ Σ} do
2.   if i. X = i. n2 then
3.     for ∀ j ∈ SVDFA. δ, j(qn, an) = qm do
4.       if i. n1. op = j. an then
5. SVUPDA. δ add δ(j. qn, i. e, i. n1) = {(j. qm, i. n2)})
6.       end if
7.     end for
8.   end if
9.   if i. X = i. n2 i. n1 then
10.    for ∀ q ∈ SVDFA. Q do
11. SVUPDA. δ add δ(q, i. e, i. n1) = {(q, i. n2 i. n1)})
12.    end for
13.  end if
14.  if i. X = ε then
15.    for ∀ q ∈ SVDFA. Q do
16.      SVUPDA. δ add δ(q, i. e, i. n1) = {(q, ε)})
17.    end for
18.  end if
19. end for
```

从上述转移函数算法中可知,当 UPDA 中的程序节点类型 TYPE 为 CALL 或 RETURN 时,SVUPDA 的状态不发生变化,只是记录 SVUPDA 的堆栈变化,当 UPDA 中的程序节点类型 TYPE 为 OTHER 时,既记录 SVUPDA 的堆栈变化,也记录了状态的转移。

(5) Q_0 表示初始状态,实际值为 SVDFA 中安全漏洞的初始状态 Q_0;

(6) Z_0 表示堆栈中的初始符号,实际值为 UPDA 中栈初始符号 Z_0;

(7) F 表示 SVUPDA 的接受状态或终结状态集合,实际值为 SVDFA 中的安全漏洞的最终接受状态集合 F。

通过文献[15-16]中的模型检验算法判断 SVUPDA 接受的语言 $L(P)$ 是否为空就可判断是否出现安全漏洞,如果为空,则表示未出现安全漏洞模型 SVDFA 描述的安全漏洞,不为空则表示可能出现所描述的问题,需要确认是否为误报。之所以出现误报,是因为静态的语法语义分析并不考虑实际数据流,Γ 除了接受实际可行路径集合外,也会接受实际数据流时不会发生的路径。但是,因为 $\Gamma \subseteq L$(UPDA),可以得到 $S \cap \Gamma \subseteq L$(SVDFA)$\cap L$(UPDA),这保证了给定安全漏洞属性模型的完全性,即可能存在误报但无漏报。

3 实验结果与分析

基于自动机的模型检验主要能力取决于对 UEFI 模块中安全漏洞属性模型描述的丰富性、准确性,以及是否有状态爆炸引起的性能问题。在安全漏洞属性模型丰富性方面,该文考虑到 UEFI 模块中可以通过引入 StdLib 包调用 C 标准库及 GNU C 库,对 UEFI 标准库、C 标准库及 CNU C 库进行分析,总结了六种类型的安全漏洞模型,安全漏洞模型的分类与数量如下:缓冲区溢出类型 13 种;资源重复释放 5 种;空指针引用 7 种;格式化字符串 9 种;竞态条件 3 种;函数调用路径 7 种。该文共对 12 个 UEFI 模块代码混合植入上述六类,44 种的安全漏洞,共计 563 个由国家信息安全漏洞库 CNNVD 公布的 EDKII 中真实安全漏洞,对 UEFI 模块进行形式化验证,验证结果如表 1 所示。

表 1 安全漏洞检测结果

漏洞类型	植入数量	真实数量	误报数量	总量
缓冲区溢出	273	271	3	274
资源重复释放	89	89	15	104
空指针引用	52	52	2	54
格式化字符串	112	111	5	116
竞态条件	8	8	14	22
函数调用路径	29	26	8	34

从表 1 可以看出,在被检测的 UEFI 模块中,植入 563 个安全漏洞,检测报出 604 个可能存在的安全问

题,其中包括检测出 557 个植入的安全漏洞,误报 47 个,统计漏报率 1.06%,误报率 7.78%。经过分析发现,漏报的主要原因是安全漏洞属性模型的不丰富造成,符合安全漏洞模型的安全漏洞都可以检测出来,检测结果基本符合形式化验证中阐述的不会有漏报的理论,降低漏报率的主要方法是不断丰富和精确安全漏洞属性模型。误报率达 7.78%,经分析发现,大部分误报都是由于静态分析时并没有考虑真正数据流,在模型检验时有大量真实情况下不可达路径的原因,例如资源重复释放、竞态条件、函数调用路径类漏洞的误报,主要是因为真实数据流情况下会在条件判断的确定分支里进行资源释放,在程序抽象时会对条件判断的所有的分支进行统计,造成误报。对于性能分析,主要是验证在对程序控制流进行压缩和未压缩时的性能分析,来验证压缩算法可以缓解状态爆炸,提高检测性能。该文对代码规模约为 500 ~ 5 000 行范围的 UEFI 模块分别植入相同的 6 类,共 65 个安全漏洞进行 20 次检验,耗时平均值结果如图 4 所示。

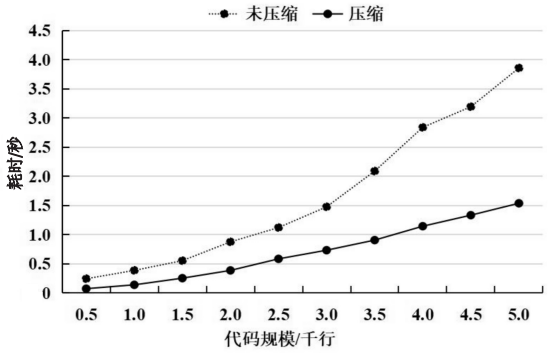

图 4　模型检验耗时

随着代码规模的增大,程序的状态空间会呈现指数增长趋势,在计算机性能不变的前提下,在未压缩的模型检验中,耗时与状态空间成正比,耗时趋势呈指数增长,预计随着代码规模持续增大会出现状态爆炸,内存不足的问题。在压缩后的模型检验中,状态空间是跟软件规模和安全漏洞属性状态数量组成的线性函数,随着软件规模持续增大,状态空间线性增长,达到缓解状态爆炸的目的。

4　结束语

固件主要负责计算机中硬件初始化和预备操作系统启动环境等功能,处于计算机中极其重要的位置,有着较高的安全级别,并且随着 UEFI 模块规模的逐渐庞大,被攻击面逐渐增加。该文提出一种针对 UEFI 模块的形式化建模与验证的方法,基于有限状态自动机和下推自动机分别对 UEFI 模块中的安全漏洞属性和代码进行形式化建模,采用模型检验算法进行形式化验证,同时利用数据抽象的思想压缩程序控制流的状态规模,缓解状态爆炸问题。对 44 种 UEFI 模块中可能出现的安全问题进行建模,12 个植入安全问题且不同代码规模的 UEFI 模块进行形式化验证,实验结果表明,该检验方法漏报率为 1.06%,误报率 7.78%,并较好地缓解了状态爆炸问题。

参考文献:

[1] 杨松松. 基于 UEFI 固件的漏洞分析与研究[D]. 北京:北京工业大学,2018.

[2] 郭东奥. UEFI 驱动程序的自动化模糊测试方法研究[D]. 南京:南京大学,2019.

[3] 张　雄,李舟军. 模糊测试技术研究综述[J]. 计算机科学,2016,43(5):1-8.

[4] 任玉柱. 固件代码动态污点分析技术[D]. 洛阳:战略支援部队信息工程大学,2019.

[5] SABBAGHI A,KEYVANPOUR M R. A systematic review of search strategies in dynamic symbolic execution[J]. Computer Standards and Interfaces,2020,72:103444.

[6] GLEIRSCHER M,MARMSOLER D. Formal methods in dependable systems engineering:a survey of professionals from Europe and North America[J]. Empirical Software Engineering,2020,25(6):4473-4546.

[7] 王　戟,詹乃军,冯新宇,等. 形式化方法概貌[J]. 软件学报,2019,30(1):33-61.

[8] WOODCOCK J,LARSEN P G,BICARREGUI J,et al. Formal methods:practice and experience[J]. ACM Computing Surveys,2009,41(4):1-36.

[9] 侯　刚,周宽久,勇嘉伟,等. 模型检测中状态爆炸问题研究综述[J]. 计算机科学,2013,40(S1):77-86.

[10] 林惠民,张文辉. 模型检测:理论、方法与应用[J]. 电子学报,2002,30(S1):1907-1912.

[11] 邹海明. 形式语言、自动机和语法分析[M]. 武汉:华中工学院出版社,1985.

[12] 蒋宗礼,姜守恒. 形式语言与自动机理论(第 3 版)[M]. 北京:清华大学出版社,2007.

[13] 高传平,谈利群,宫云战. 基于抽象语法树的代码静态自动测试方法研究[J]. 北京化工大学学报:自然科学版,2007,34(S1):25-29.

[14] HOPCROFT J E,MOTWANI R,ULLMAN J D. Introduction to automata theory,languages,and computation[M]. 3rd ed. USA:Addison - Wesley Longman Publishing Co., Inc., 2006.

[15] ESPARZA J,HANSEL D,ROSSMANITH P,et al. Efficient algorithms for model checking pushdown systems[C]//Lecture notes in computer science. [s. l.]:[s. n.],2000:232-247.

[16] JANCAR P. Equivalence of pushdown automata via first-order grammars[J]. Journal of Computer and System Sciences,2021,115:86-112.