计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20131382
52(3):702-717, 2015

基于散列时间有效性的轻量级完整性监测方法

徐钦桂[1] 秦 勇[1] 杨桃栏[2]
[1](东莞理工学院计算机学院 广东东莞 523808)
[2](国防科学技术大学计算机学院 长沙 410073)
(dgxuqg@126.com)

Light-Weight Integrity Monitoring Based on Hashing Time Validity

Xu Qingui[1], Qin Yong[1], and Yang Taolan[2]
[1](*College of Computer, Dongguan University of Technology, Dongguan, Guangdong* 523808)
[2](*College of Computer, National University of Science Technology, Changsha* 410073)

Abstract Real-time monitoring of node integrity is effective means to protect resource-restrained nodes. By identifying main tampering attack modes against resource-restrained nodes, and analysiing the influence on hashing time, pure-software integrity monitoring means based on inspecting hashing time validity is suggested. On the basis of analysing testability condition of hashing time validity, checksum forging punishment coefficient is proposed to indicate tamper-resisting performance of monitor mechanism, and a light-weight hashing algorithm of merging program states is put forward. By simplifying hashing structure and integrating program states into checksum, checksum forging is made more difficult. Damaged nodes have to spend much more time on extra work like restoring legal code and program states than on hashing if they want to aquire the correct checksum. Hence, the proposed mechanism imposes much greater checksum forging punishment on damaged nodes than other approaches like SWATT and Shah. In order to prevent message forging or tampering during transmission over communication networks, a monitoring protocol supporting message authentication is designed. For tolerating influence from hashing time fluctuation and checksum guess, node integrity state is evaluated from results of both checksum comparison and hashing time validity statistics. The experiments show that the proposed approach achieves high reliabiliy in examining validity of checksum and hashing time with small cost. Toleration ability against fluctuation disturbance on hashing time from node multi-tasking environment and communication networks is improved, and hence tamper-resisting performance of resource-constrained nodes is enhanced.

Key words integrity monitoring; hashing time validity; light-weight hashing algorithm; low-overhead authentication algorithm; checksum forging punishment coefficient

摘 要 实时监测节点完整性状态是资源受限节点安全保护的有效手段.分析针对资源受限节点的主要篡改攻击模式及对散列时间带来的影响,提出基于散列时间有效性检验的纯软件完整性监测手段.基于对散列时间有效性可检验条件分析,提出采用验证值伪造惩罚系数描述散列模块抗篡改能力,设计一种融入程序状态的轻量级散列算法,通过简化算法结构与融入程序状态,增大验证值伪造难度,提高验证值伪造惩罚系数.设计支持消息认证的监测协议防止消息伪造,基于验证值比较与散列时间有效性统

收稿日期:2013-10-23;**修回日期**:2014-05-23
基金项目:国家自然科学基金项目(61300198);广东省自然科学基金项目(S2013010011858);广东省科技计划基金项目(2007A060304003);东莞市高等院校科研机构科技计划基金项目(2012108102035,2012108102007);清远市产学研项目(2012D021212005);广东省高等学校学科与专业建设专项基金项目(2050205)

计,判定节点完整性状态.实验结果表明:该方案以微小的节点开销为代价,获得了更高的散列时间有效性检验可靠性,增强了对散列时间与消息传输时间波动干扰的容忍能力,提高了资源受限节点防篡改攻击性能.

关键词 完整性监测;散列时间有效性;轻量级散列算法;低开销认证算法;验证值伪造惩罚系数

中图法分类号 TP309.1;TP393.08

近年来,信息网络中的安全威胁逐渐向物联网、工业控制网络等领域扩散,设备节点通常因计算能力、内存容量有限,难以部署功能强大的软硬件防护设施,而成为系统安全保护的薄弱环节之一[1].实时监测节点完整性是应对威胁的有效措施,根据验证值属性可将监测方法分为基于静态验证值与基于动态验证值2类方法.

基于静态验证值方法每次对整个节点或指定模块计算SHA-1或MD5验证值,只要节点代码不被更改,验证值就保持不变.该类方法一般需在节点设置一个可信的专用防篡改硬件模块,正确计算、如实报告验证值,并采用加密、签名方法传输验证值,防止录制重放攻击.如张焕国等人[2]借鉴可信计算技术,设计适合嵌入式设备可信平台模块ETPM[3],在节点启动时直接访问内存,计算、验证各软件模块SHA-1验证值,仅允许ETPM判定可信的模块执行,实现设备节点可信启动功能.Basile等人[4]将SHA-1散列算法、加解密算法、身份认证等功能集成到用FPGA实现的硬件监测模块,实时计算指定模块验证值,经加密与模糊化处理后可靠传输到监测方(Verifier)进行验证.这类方法依靠外加的专用硬件模块执行节点验证值计算与报告,不但增加节点成本与设计复杂度,而且难以检测部署在CPU内置Flash中的代码模块.

在基于动态验证值方法中,节点通过报告动态变化的验证值来防范重放[5]与穷举检索[6]攻击,Verifier比较实际验证值与预期验证值,判定待监测模块完整性.伪造正确验证值需增加额外操作,带来散列惩罚时间(验证值计算时间增加值),Verifier可据此分辨验证值真假,验证值伪造惩罚系数(散列惩罚时间与原有散列时间之比)越大,伪造验证值越容易被检出[7].该类方法不要求节点增加防篡改硬件模块,适用性好,近年来受到关注与重视.SWATT[7]是基于动态验证值方法的重要模型,作者Seshadri[7]将验证值计算与报告模块(integrity measurement & report,IMR)设计成不可并行的优化结构,攻击者若欲受损节点伪造正确验证值,必须修改IMR,增加的额外操作带来13%验证值伪造惩罚时间,在散列时间与消息传输时间保持基本恒定条件下,Verifier可据此检验IMR完整性;为防止TOCTOU攻击[8],Seshadri等人[7]设计了一种不可中断代码执行环境(indisputable code execution, ICE),在关闭节点中断条件下,先验证IMR,再由IMR验证用户模块完整性,实现了针对传统系统可验证执行的Pioneer模型[9]与针对WSN节点的代码安全更新模型SCUBA[10],但验证值伪造惩罚系数并无明显改善.

不少学者针对动态验证值方法应用、散列算法与IMR保护等问题开展了研究.Li等人[11]将该类方法用于计算机外围设备固件完整性监测;Shah等人[12]对散列算法结构进行改进,对基于ARM7TMDI的SCADA设备,将验证值伪造惩罚系数提高到51%,但缺乏消息认证功能,易受伪造消息干扰,引起误判.另外,由于Flash程序存储器访存周期长,当CPU主频更高时,实际验证值伪造惩罚系数可能降低.Park等人[13]提出基于有限域上矩阵运算的WSN防篡改程序完整性验证(program integrity verification, PIV)模型,每次按固定顺序处理存储块内容,计算长度可变代码块的动态验证值,结合散列时间检测结果,监测节点完整性,但若攻击者对被篡改块作单独处理,减少因伪造验证值所需额外操作,会影响验证值伪造惩罚时间存在性的检测.Defrawy等人[14]提出动态信任根模型SMART,在关闭节点中断条件下,通过测定IMR能否在限定时间内计算SHA-1消息验证码来判定其完整性,然后以IMR为动态信任根,检测软件模块完整性,启动应用程序执行,使应用程序在受损设备节点上仍以可信方式执行,但需修改节点硬件结构,给实施带来不便.

综上所述,现有基于动态验证值方法虽然对硬件支持要求低、适用性好、支持节点运行时完整性实时监测,但仍需解决以下问题:

1) 监测结果受节点多任务波动干扰影响大,要求在关闭节点中断条件下执行验证值计算[5-6,11],容易导致外部事件或节点任务的处理被延误;

2) 散列算法结构仍显复杂,验证值伪造惩罚系

数不高[7-9],若消息传输时间上下波动会影响监测可靠性;

3) 鲜少考虑验证值消息真实性认证问题[5-6,12],若受消息伪造或篡改攻击可能导致大量误判.

本文研究基于散列时间有效性检验的完整性监测方法,通过散列时间有效性可检验条件分析,提出一种融入程序状态的轻量级散列算法与相应的监测协议,增大验证值伪造惩罚系数,增强对多任务波动与消息传输波动干扰的容忍能力,设计低开销消息认证算法,抵抗消息伪造攻击,提高系统适应性与监测可靠性.

1 用散列时间有效性监测完整性

1.1 问题描述

仅篡改节点合法模块代码的攻击可通过 IMR 报告的验证值检测出来,但若攻击者同时篡改 IMR,设法伪造与预期一致的验证值,将大大增加检测难度. 本文探讨面临 IMR 攻击威胁环境下节点完整性监测方法. 鉴于很多 SCADA 数据采集监控节点倾向于选择无存储管理单元(memory management unit, MMU)微控制器设计,这里假定节点以实地址存储器模式工作,程序地址与物理地址相同,程序存储器、非程序存储器(包括 I/O 接口)统一编址. 程序存储器中烧写各种合法模块,剩余空间用随机数填充. 这样,我们将 IMR 攻击抽象为存储器复制(memory copy attack)攻击与代理(proxy attack)攻击 2 钟模式. 如图 1(a)所示,前者将合法模块 M_i 代码移至非程序存储器区域(包括片外辅存等),用腾出的空间装载恶意模块 MalModule,同时用 MalIMR 替换 IMR,MalIMR 可从备份处取回 M_i 原有代码来伪造正确验证值. 如图 1(b)所示,后者让 MalIMR 委托一个合谋节点的 ProxyIMR 来计算正确验证值.

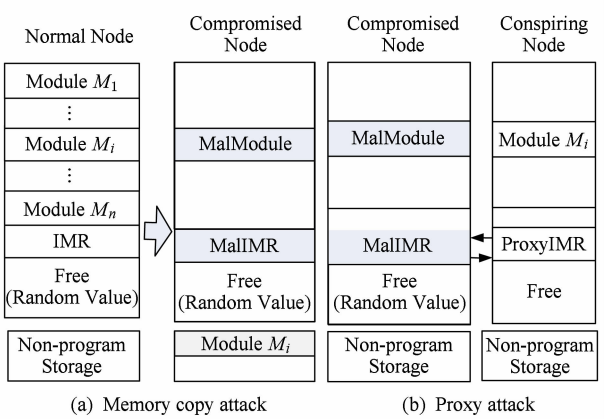


Fig. 1 Tamper mode against IMR.

图 1 针对 IMR 的攻击模式

IMR 遭篡改后的节点很难仅通过验证值比较来准确判断其完整性状态,但 MalIMR 搜寻合法模块 M_i 原有代码或与合谋节点 IMR 间传输消息会带来验证值伪造惩罚时间. 若能通过监测机制将该时间放大,迫使受损节点散列时间超过其正常数值范围,就可通过散列时间有效性检验来推断 IMR 与其节点的完整性状态[5-7,9]. 因此,对于面临存储器复制攻击或 Proxy 攻击威胁的节点,可采用基于散列时间有效性检验的监测手段,在没有专用硬件模块支持的环境下,可高效可靠地监测节点完整性状态.

1.2 散列时间有效性可检验条件

在节点保持完整可信条件下,不受多任务切换、系统中断干扰的散列时间是 CPU 实际执行散列算法计算验证值所需时间,称为净散列时间 $TGCI$. 期望散列时间 $TGCE$ 是节点执行一次验证值计算所需实际时间,可看成净散列时间 $TGCI$ 与中断处理、任务切换、其他任务处理引起的散列波动时间 T_{MT} 之和:

$$TGCE = TGCI + T_{\mathrm{MT}}, \tag{1}$$

其中,$T_{\mathrm{MT}} \in [0, \Lambda_{\mathrm{MT}}]$. 用 t_{unit} 表示验证值计算过程中处理一个单元所需 CPU 时间平均值,则一次监测交互中计算 n 个单元验证值所需净散列时间为

$$TGCI = n \times t_{\mathrm{unit}}. \tag{2}$$

若对节点实施存储器复制攻击,期望散列时间 $TMCE$ 为 MalIMR 伪造正确验证值的净散列时间 $TMCI$ 与节点波动时间 T_{MT} 之和:

$$TMCE = TMCI + T_{\mathrm{MT}}. \tag{3}$$

将 $TMCI$ 可看成在 $TGCI$ 基础上增加搜寻被更改合法模块原有代码所需额外时间 T_{extra},减去攻击者对散列算法进行优化而节省的时间 T_{opt}:

$$TMCI = TGCI + T_{\mathrm{extra}} - T_{\mathrm{opt}}. \tag{4}$$

令 MalIMR 处理每单元需增加额外时间 t_{extra},算法优化可节省时间 t_{opt},则 $T_{\mathrm{extra}} = n \times t_{\mathrm{extra}}$, $T_{\mathrm{opt}} = n \times t_{\mathrm{opt}}$.

若对节点实施 Proxy 攻击,伪造正确验证值所需的期望散列时间 $TPCE$ 为合谋节点计算正确验证值所需散列时间 $TOCI$、监测消息转发与接收时间 T_{SR}、因消息排队等引起传输波动时间 T_{Queue} 构成:

$$TPCE = TOCI + T_{\mathrm{SR}} + T_{\mathrm{Queue}}, \tag{5}$$

其中,$TOCI \in [0, TGCI]$, $T_{\mathrm{Queue}} \in [0, \Lambda_{\mathrm{Queue}}]$.

节点期望监测时间 TVE 是在节点期望散列时

间 TCE 基础上,增加消息收发时间 T_{SR},消息传输波动时间 T_{Queue}所得:

$$TVE = T_{SR} + T_{Queue} + TCE. \quad (6)$$

将 TCE 为 $TGCE$,$TMCE$ 与 $TPCE$ 时的期望监测时间分别记为 $TGVE$,$TMVE$ 与 $TPVE$.

为基于散列时间有效性监测节点 IMR 完整性,定义有效散列时间为正常节点 IMR 如实测量、如实报告的散列时间,非有效散列时间为受损节点 MalIMR 伪造预期验证值所需的实际散列时间或凭空伪造的虚假散列时间,对应的监测时间分别称为正常监测时间与非正常监测时间. 一种可靠区分有效与非有效散列时间所需的条件是:最小非有效期望散列时间大于最大有效期望散列时间,最小非正常监测时间大于最大正常监测时间. 其中监测时间条件更强,它可涵盖散列时间条件. 若存在存储器复制攻击威胁,散列时间有效性可检验条件 $\min(TMVE) > \max(TGVE)$可表示为

$$TGCI + T_{extra} - T_{opt} + T_{SR} > TGCI + \Lambda_{MT} + T_{SR} + \Lambda_{Queue}. \quad (7)$$

若存在 Proxy 攻击威胁,散列时间有效性可检验条件 $\min(TPVE) > \max(TGVE)$可表示为

$$2T_{SR} > TGCI + \Lambda_{MT} + T_{SR} + \Lambda_{Queue}. \quad (8)$$

记验证值伪造惩罚系数 $\rho = \dfrac{T_{extra} - T_{opt}}{TGCI} = \dfrac{t_{extra} - t_{opt}}{t_{unit}}$,用相对散列波动系数 κ_1、绝对散列波动系数 κ、传输波动系数 κ_2 分别表示 Λ_{MT} 与散列时间 $TGCI$ 的比值、Λ_{MT} 占 CPU 时间比率、Λ_{Queue} 与消息收发时间 T_{SR} 的比值,则 $\Lambda_{MT} = \kappa_1 \times TGCI$,$\kappa = \dfrac{\kappa_1}{1+\kappa_1}$,$\Lambda_{Queue} = \kappa_2 \times T_{SR}$,并可将不等式(7)(8)变换为

$$\kappa_2 T_{SR} < n(\rho - \kappa_1) t_{unit}, \quad (9)$$

$$n(1+\kappa_1) t_{unit} < (1-\kappa_2) T_{SR}. \quad (10)$$

式(9)(10)成立分别要求 $\rho > \kappa_1$,$\kappa_2 < 1$,2 式同时成立的条件为 $\dfrac{\kappa_2 T_{SR}}{(\rho-\kappa_1) t_{unit}} < n < \dfrac{(1-\kappa_2) T_{SR}}{(1+\kappa_1) t_{unit}}$. 减小 κ_1,κ_2,增大 ρ 都有助于增大不等式(9)(10)两边的差值,提高散列时间有效性可检验性能. 较小的 ρ 值要求系统保持微小的 κ_1,κ_2 系数,监测机制只能容忍微小的波动干扰,甚至需在关闭节点中断、通信网络具有良好实时性的环境下工作,适用性差. 较大 ρ 值能放宽对系统 κ_1,κ_2 系数的限制,容忍较大的散列波动与传输波动干扰、适用性好.

1.3 融入程序状态轻量级散列算法

t_{unit}越小、t_{opt} 越小、t_{extra} 越大,比例系数 ρ 越大,散列时间有效性可检测性越好. 系数 ρ 主要由散列算法结构决定. 散列算法结构越简洁、验证值越简短,算法流程越简单,所需运算操作次数越少,t_{unit} 越小;采用优化的不可并行迭代结构,可防止攻击者利用多个受损节点合谋计算散列值来提高优化时间 t_{opt};验证值中融入动态特性越多,伪造验证值难度越大,额外操作时间 t_{extra}越多,散列时间有效性可检测性越好. 基于这些考虑,我们设计一种融入程序状态轻量级散列算法 LW-Hash,图 2 是散列结构,其执行过程如下:

1) 将长度为 lz 的指定单元序列 $\Psi_C = \langle x_1, x_2, \cdots, x_{lz} \rangle$划分为长度为 l 个单元的 z 个子序列 $X_1, X_2, \cdots, X_z$,其中 $X_i = \langle x_{l(i-1)+1}, \cdots, x_{li} \rangle$.

2) 链接变量初始值 h_0 置为 0,以表示程序状态的程序计数器 PC、程序状态寄存器 PSW、数据堆栈指针 DP 为盐值(salt value),对序列 Ψ_C 中各单元子序列 X_i 内容 $VR(X_i)$进行压缩迭代,第 i 次压缩迭代可表示为

$$h_i = f(h_{i-1}, VR(X_i), PC, PSW, DP). \quad (11)$$

3) 第 z 次压缩迭代输出长度为 l 个单元字长的链接变量值 h_z 作为验证值 $LW\_Hash(\Psi_C)$.

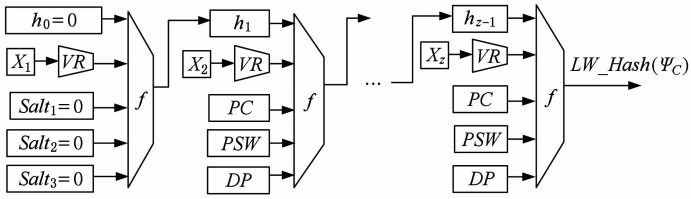


Fig. 2 Hash framework of light-weight hashing algorithm.

图 2 轻量级散列算法散列结构

图 3 是压缩迭代结构,第 i 次压缩迭代由迭代次数为 l 的循环实现,每个迭代产生链接变量的一个单元字,第 j 个迭代处理过程为:先用 PSW 对第 $i-1$ 次迭代产生的链接变量第 j 个单元字 $h_{i-1,j}$ 进行循环移位($SHIFT$),与序列 X_i 中单元 x_{li+j}的内容 $VR(x_{li+j})$相加,再对结果用堆栈指针 DP 进行移位,最后与程序计数器 PC 异或,得到本次循环的输出链接变量单元字 $h_{i,j}$:

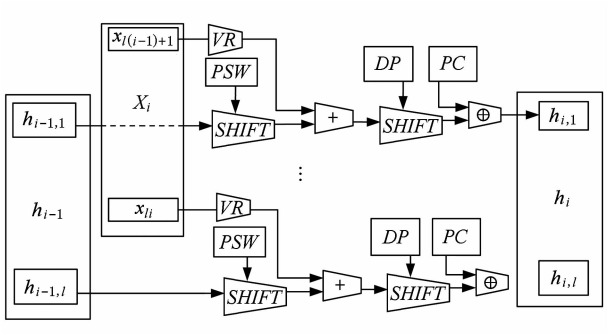


Fig. 3 Compression iteration diagram of LW-Hash.

图 3 LW-Hash 压缩迭代结构

$$h_{i,j} = SHIFT_{DP}(SHIFT_{PSW}(h_{i-1,j}) + VR(x_{l(i-1)+j})) \oplus PC. \quad (12)$$

第 i 次压缩迭代得到链接变量 $h_i = \langle h_{i,} , \cdots , h_{i,l} \rangle$.

LW-Hash 算法对程序状态盐值的引入出于如下考虑:由于按固定顺序对指定单元序列 Ψ_C 执行迭代处理,未涉及选择控制结构,每次计算过程执行固定的指令序列,依次引用的 PC 寄存器值构成不变的地址序列;很多 SCADA 节点倾向于采用 μCOS, VXWorks 等实时操作系统实施任务管理,各任务具有独立堆栈,若算法 LW-Hash 实现能避免函数调用,DP 寄存器值在散列值计算期间可保持不变; PSW 值虽不断变化,但只受迭代运算影响,仅与序列 Ψ_C 各单元内容相关. 由于多任务环境下任务切换、中断处理能自动完成程序状态的保护与恢复,正常节点对同一单元序列 Ψ_C 计算散列值,能得到确定的结果. 但受损节点若欲获得正确验证值,则需增加大量保存与恢复这些寄存器内容的额外指令,遭受的散列惩罚时间将大幅增加.

算法设计从 4 个方面体现验证值防篡改能力与惩罚时间系数 ρ 的放大性能:

1) 交错执行加法、异或、移位运算,产生随机变化的动态验证值,必须以指定存储单元正确内容为输入,顺序执行各操作才能得到正确结果,防止攻击者采用分析手段快速推算验证值;

2) 指定单元序列 Ψ_C 某单元的变化通过 PSW 传递扩散到链接变量各单元字中,增大验证值破解难度;

3) 压缩迭代过程简洁,对每个存储单元 $x_{i,j}$ 的处理操作少,每个单元处理时间 t_{unit} 少,各运算操作前后依赖,具有反并行优化功能,使得 $t_{opt} << t_{unit}$;

4) 将程序状态 PC, SW, DP 融入验证值,受损节点若要伪造正确验证值,需增加搜寻被篡改单元正确内容的额外指令,添加保存与恢复程序状态的额外操作,这些都将导致 t_{extra} 增大.

1.4 基于散列时间有效性的监测协议

完整性监测协议创建一个框架,利用可认证消息传输监测命令与动态验证值,准确测量散列时间与监测时间,执行散列时间有效性检验与节点完整性评判.

1.4.1 监测协议流程

图 4 描述了针对正常节点的完整性监测协议,它依赖于生产过程中植入节点 ROM 的共享密钥 KS 执行消息认证,不定期对节点执行完整性监测,一次监测会话由多次监测交互构成,覆盖节点全部单元,每次监测交互包含验证值信息采集与认证 2 个阶段.

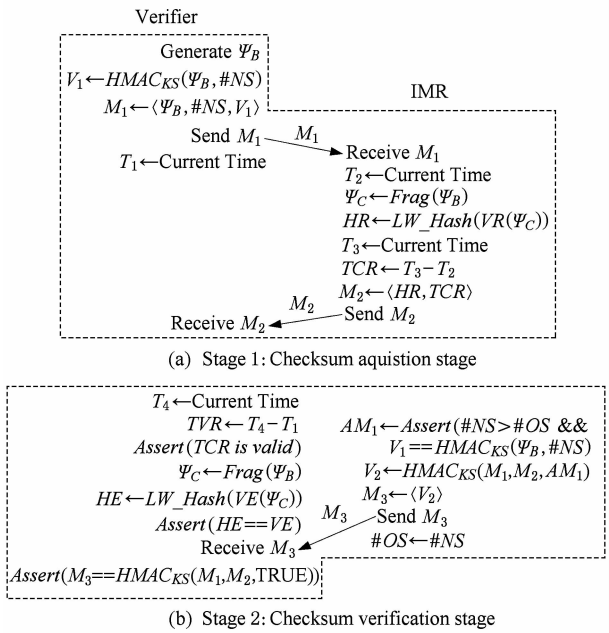


Fig. 4 Integrity monitoring protocol based on validity of hashing time.

图 4 基于散列时间有效性的完整性监测协议

1) 在验证值信息采集阶段,Verifier 生成块大小为 S 的随机化存储块序列 Ψ_B,为每次监测交互设置一个命令序号 $\#NS$,用节点共享密钥 KS 计算 Ψ_B 与 $\#NS$ 的消息验证码 $V_1 = HMAC_{KS}(\Psi_B, \#NS)$;将由 Ψ_B, $\#NS$, V_1 组成的监测命令消息 M_1 发送至 IMR,并启动计时;IMR 接收 M_1,解耦出 Ψ_B 并进行分片处理,产生待验证单元序列 Ψ_C,调用 LW-Hash 模块计算验证值 $HR = LW\_Hash(\Psi_C)$,测量散列时间 TCR,构建验证值消息 $M_2 = \langle HR, TCR \rangle$ 发回 Verifier.

2) 在验证值信息认证阶段,Verifier 首先接收验证值消息 M_2,测量监测时间 TVR,检验散列时间

TCR 有效性与验证值 HR 正确性;同时 IMR 认证消息 M_1 的真实性、完整性与新鲜性,计算消息 M_1,M_2 与认证信息 AM_1 的消息认证码 V_2,作为消息 M_3 发回 Verifier. 节点对 M_1 的认证推迟到验证值计算完成后执行,原因是为了避免影响正常节点 TCR 与 TVR 值准确性. 最后 Verifier 利用 M_3 检验 M_1 完整性与 M_2 的真实性,确认 IMR 对 M_1 的认证结果. 一次监测会话完成后,Verifier 利用收集到的验证值与散列时间信息,结合节点与网络环境状态信息,执行节点完整性分析与判断.

1.4.2 完整性监测机理

IMR 基于分片处理方法从存储块序列 Ψ_B 生成待验证单元序列 Ψ_C,当序列 Ψ_B 满足一定长度与随机性要求时,IMR 只有获得待验证单元序列正确内容,在规定时间内完成 LW-Hash 压缩,得到正确验证值,才能被 Verifier 判定为正常节点. 不管 LW-Hash 算法代码是否被攻击者获知,在监测机制满足一定条件下,Verifier 都能通过监测消息分析,获知受损节点完整性状态.

1) 攻击者未获得 LW-Hash 算法结构情况. 攻击者只能通过查表或随机猜测方法来获取正确验证值,增大随机单元序列 Ψ_C 长度方法可使查表手段失效,通过增加验证值长度降低验证值猜测成功率,使随机猜测手段失灵. Verifier 从受损节点收集的实际验证值与预期验证值必然不同,据此就可断定节点遭受篡改.

2) 攻击者已获得 LW-Hash 算法结构情况. 若存在待验证单元内容被更改,原有内容已备份至它处,MalIMR 可找到 Ψ_C 单元序列原有内容,执行 LW-Hash 算法得到正确验证值. 但若监测机制满足式(9)(10)散列时间有效性可检验条件,其实际散列时间 TCR 必然明显增大. 1.7.1 节证明了在监测消息能可靠传输的环境下,不管 MalIMR 是否如实报告 TCR,Verifier 都能可靠分辨 TCR 有效性,1.7.2 节推导了基于多次监测交互统计数据来判定阶段完整性状态的准则.

1.4.3 监测协议安全性

满足式(9)(10)散列时间有效性可检验条件的通信网络应具有一定的保护功能,我们假设攻击者能监听、收集监测消息,向网络注入消息,但不能拦截监测消息或推迟其传输. 监测协议的安全性可以分 3 种情形讨论.

情形 1. 共享密钥未泄露. 攻击者在 Verifier 与受损节点间伪造、注入监测消息. 当监听到监测命令消息 M_1,由于攻击者不持有节点密钥 KS,即使伪造与注入监测消息 M_2' 或 M_3',这些消息也很难恰好能与 M_1 相互认证而被丢弃,因而难以干扰 Verifier 对受损节点验证值与散列时间异常性的判断,监测协议是安全的.

情形 2. 共享密钥 KS 未泄露. 攻击者回放监测消息. 攻击者通过监听收集了监测消息 M_1,M_2,M_3,在监听到来自 Verifier 新监测命令消息 M_1' 后,先于正确验证值消息 M_2' 或 M_3',向网络注入重放响应消息 M_2,M_3,由于 M_2 或 M_3 无法与 M_1' 相互认证而被 Verifier 丢弃,因而监测协议具有防范对正常监测过程干扰破坏的能力.

情形 3. 共享密钥泄露,攻击者伪造监测消息. 节点 N_i 收到监测命令消息 M_1 后,在正常散列时间内算出正确的验证值,并向 Verifier 发送正确的监测消息 M_2,M_3. 但攻击者抢先向 Verifier 发送能与 M_1 相互认证的伪造消息 M_2' 或 M_3',致使 Verifier 将 N_i 判为受损节点. 对于这种情况,由于攻击者获得了受保护的共享密钥 KS,若排除暴力破解、Verifier 泄露情形,可以认为攻击者对 N_i 实施了成功入侵,将 N_i 视为受损节点是合适的,监测协议并未导致误判.

1.5 按比例选新产生随机存储块序列

在监测命令消息中指定随机化存储块序列 Ψ_B,进而产生随机化存储单元序列 Ψ_C,应适当控制消息长度,降低通信开销,提高散列时间测量的准确度. PRG 方法基于 Coupon 原理产生随机数构造 Ψ_C,随机性好,但对具有 N 个单元节点,访存与计算复杂度高达 $O(N\ln N)$[7]. T 函数方法按单圈结构产生随机化存储单元序列 Ψ_C,访存与计算复杂度低至 $O(N)$,但键值易被猜出而实施查表攻击[6]. 本文给出一种按比例选新策略随机数序列构造方法,每次按比例从未覆盖集随机新选若干元素,与从全集中随机选择的元素一起,产生随机化存储块序列 Ψ_B,进而产生随机化单元序列 Ψ_C,在保证单元序列 Ψ_C 随机性条件下,将访存与计算复杂度控制在 $O(N)$.

具体方法是:首先将节点全部待监测单元集合 SC 划分为 M 个大小为 S 个单元的存储块 B_1,B_2,…,B_M,$SC=B_1\cup B_2\cup\cdots\cup B_M$,记 $SB=\{B_1,B_2,\cdots,B_M\}$. 用 SBU 表示当前监测会话中尚未覆盖的存储块集合,初值为 SB,定义选择系数 $\gamma(0\leqslant\gamma\leqslant1)$,每次监测交互先从 SBU 中按不回置方式随机选取 $\lceil m\gamma\rceil$ 个存储块,再从全集 SB 中随机选择 $m-\lceil\gamma m\rceil$ 个存储块,组成一次监测交互中存储块序列 $\Psi_B=$

$\langle B_{(1)},\cdots,B_{(m)}\rangle$,作为消息 M_1 的组成部分在网上传输. 节点对 Ψ_B 进行分片处理,产生待随机化监测单元序列 $\Psi_C=\langle B_{(1)},B_{(2)},\cdots,B_{(1)}+1,B_{(2)}+1,\cdots,B_{(m)}+S-1\rangle$.

第 i 次监测交互产生 Ψ_B 的候选存储块序列数为 $M^{m-\lceil m\gamma\rceil}C_{m\gamma}^{M-(i-1)\lceil m\gamma\rceil}$. 一次监测会话包括 $K=\frac{M}{m\gamma}$ ($\gamma\neq 0$)次监测交互,覆盖所有 N 个待监测单元所需访存次数为$\frac{N}{\gamma}$. 当 $\gamma=1$ 时,每次监测交互仅从 SBU 随机选取存储块,访存与计算效率达到 T 函数方法水平,但 Ψ_B 有 $C_m^{M-(i-1)m}$ 个候选值,随机性远优于 T 函数方法;当 $\gamma=0$ 时,完全从 SBU 随机选择存储块,退化为 PNG 方法;设置 $0<\gamma<1$,可在覆盖性与随机性间折中.

1.6 低开销消息认证算法

为防止攻击者通过伪造监测命令消息或验证值消息,诱导 Verifier 将正常节点误判为受损节点,我们设计一种低开销消息认证算法 LO-Auth,对监测消息真实性、完整性进行认证. LO-Auth 以节点密钥 KS 与待认证消息 Q 为输入,通过加法、乘法、异或等基本运算,计算消息认证码 $HMAC$,计算与存储开销低,具有防伪造、防破解功能. 算法执行流程有 3 步:

1) 算法初始化. 将长度为 k 的共享密钥 KS 看成 k 个单元字 $KS_1,KS_2,\cdots,KS_k$,将待认证消息 Q 划分为 p 个长度为 k 个单元的消息分片 $Q_1,Q_2,\cdots,Q_p$(Q_p 不足部分用 0 填充),每个消息分片 Q_i 由 $Q_{i1},Q_{i2},\cdots,Q_{ik}$ 等 k 单元字组成,以共享密钥 KS 为消息认证码初值 $HC_0=\langle KS_1,KS_2,\cdots,KS_k\rangle$.

2) 计算部分消息认证码. 以迭代方式依次对消息分片 $Q_1,Q_2,\cdots,Q_p$ 计算部分消息认证码 $HC_1,HC_2,\cdots,HC_p$,设 $HC_i=\langle HC_i,HC_i,\cdots,HC_{ik}\rangle$,$HC_i$ 计算公式为

$$\begin{bmatrix}HC_{i1}\\ \vdots\\ HC_{ik}\end{bmatrix}=\begin{bmatrix}\prod_{j=1}^{k}(HC_{i-1,j}+Q_{i1})\oplus\sum_{j=1}^{k}(HC_{i-1,j}\times Q_{i1})\oplus\prod_{j=1}^{k}(HC_{i-1,1}+Q_{ij})\\ \vdots\\ \prod_{j=1}^{k}(HC_{i-1,j}+Q_{ik})\oplus\sum_{j=1}^{k}(HC_{i-1,j}\times Q_{ik})\oplus\prod_{j=1}^{k}(HC_{i-1,k}+Q_{ij})\end{bmatrix},\tag{13}$$

其中,所有+,×运算仅保留低 $w(b)$ 运算结果. 式(13)利用乘法、加法、异或混合运算,将消息融入部分消息认证码中,既能认证消息真实性与完整性,又能防止根据消息及其认证码反推节点密钥.

3) 取得消息 Q 的认证码. 整个消息 Q 的消息验证码是完成分片 Q_p 处理后得到的 $HC_p=HMAC_{KS}(Q)$.

LO-Auth 算法对部分认证码每个单元字用左右子式的交叉和之积与中间子式的交叉积之和通过异或运算得到,算法运算操作少,计算开销低,适合资源受限节点,满足动态验证方法对消息真实性认证要求:①利用乘积运算将不同消息之间的细微差异放大,再利用异或运算将其模糊化,防止采用差分攻击手段破解共享密钥 KS;②引入右子式,使一个消息单元字的变化立即扩散到消息认证码所有单元字中;③左中子式中部分消息认证码各单元字以先加后乘与先乘后加 2 种方式参与运算,便于打破单一模式的规律性.

1.7 节点完整性状态分析方法

1.7.1 散列时间有效性检验

在监测机制满足式(9)(10)可检验条件情况下,Verifier 能可靠区分正常节点报告的有效散列时间与受损节点报告的非有效散列时间.

对于正常节点报告的有效散列时间 $TGCR$,与 Verifier 测得的实际监测时间 $TGVR$,有 $TGCR\in[TGCI,TGCI+\Lambda_{MT}]$,$TGVR=TGCR+T_{SR}+T_{Queue}$,因此有 $TGCR\leqslant TGCI+\Lambda_{MT}\wedge TGVR-TGCR\leqslant T_{SR}+\Lambda_{Queue}$.

对于受损节点报告的非有效散列时间 $TMCR$,$TPCR$,分 2 种情况:

1) MalIMR 伪造节点验证值. 但如实测量与报告散列时间,由式(7)(8)所示,实际散列时间数值将超过有效散列时间最大值,$\min(TMCR,TPCR)>TGCI+\Lambda_{MT}$.

2) MalIMR 给出处于正常数值范围的虚假散列时间. 不妨设 MalIMR 实际散列时间是为 $TMCR=TMCE_{min}=(1+\rho)\times TGCI$ 或 $TPCR=TPCE_{min}=T_{SR}$,报告最大有效散列时间 $TCR=TGCE_{max}=(1+\kappa_1)\times TGCI$,具有最小消息传输波动时间 $T_{Queue}=0$.

若受损节点遭受存储器复制攻击,则:

$$TMVE\geqslant TMCR+T_{SR}=(1+\rho)\times TGCI+T_{SR},$$

$$TMVE\text{-}TCR\geqslant(1+\rho)\times TGCI+T_{SR}-(1+\kappa_1)\times TGCI=n\times(\rho-\kappa_1)\times t_{unit}+T_{SR}>\kappa_2\times T_{SR}+T_{SR}=T_{SR}+\Lambda_{Queue}.$$

若受损节点遭受 Proxy 攻击,则:

$$TPVE \geqslant TPCR + T_{SR} = T_{SR} + T_{SR},$$

$$TPVE\text{-}TCR = T_{SR} - (1+\kappa_1) \times TGCI + T_{SR} > (1+\kappa_1) \times TGCI + \kappa_2 \times T_{SR} - (1+\kappa_1) \times TGCI + T_{SR} = \kappa_2 \times T_{SR} + T_{SR} = T_{SR} + \Lambda_{Queue},$$

故有 $\min(TMVR - TMCR, TPVR - TPCR) > T_{SR} + \Lambda_{Queue}$.

综合以上 2 种情况,Verifier 根据 IMR 报告的散列时间 TCR 与直接测得的监测时间 TVR,判断 TCR 为有效散列时间的条件为

$$\begin{cases} TCR \in [TGCI, TGCI + \Lambda_{MT}] \\ TVE - TCR \leqslant T_{SR} + \Lambda_{Queue} \end{cases}. \tag{14}$$

1.7.2 节点完整性状态判定准则

由于节点任务与网络负载具有突发性特征,实际上 T_{MT} 偶尔超过 Λ_{MT},T_{Queue} 偶尔超过 Λ_{Queue} 都是可能的,仅凭一次监测交互中散列时间有效性检测结果来判定节点完整性状态,容易导致误判. 为提高监测结果可靠性,我们基于多次监测交互的统计数据进行节点完整性判定.

首先,允许实际应用中节点散列波动时间 T_{MT} 与消息传输波动时间 T_{Queue} 偶尔超过预先设定的上限值,用 $\beta_1 = Pr(T_{MT} \leqslant \Lambda_{MT})$,$\beta_2 = Pr(T_{Queue} \leqslant \Lambda_{Queue})$ 分别表示 2 个时间位于相应设定区间内的概率.

在此基础上,统计一次监测会话中具有有效散列时间的交互次数:

$$U = |\{i \mid i \in [1,K] \wedge [TCR \in [TGCE, TGCE + \Lambda_{MT}] \wedge TVE - TCR < T_{SR} + \Lambda_{Queue}]\}|, \tag{15}$$

$K-U$ 为因 T_{MT} 或 T_{Queue} 超限而导致非有效散列时间交互次数.

设置比例系数 $\theta(0<\theta\leqslant 1)$,表示在一次监测会话中,Verifier 能容忍正常节点 IMR 报告非有效散列时间的监测交互比率达 $1-\theta$,采取以下准则判断节点完整性状态:

$$Result = \begin{cases} \text{Tampered}, \left(\exists j \in [1,K]:[HE_j \neq HR_j]\right) \wedge \left(\forall j \in [1,K]:[HE_j = HR_j] \wedge \frac{U}{K} < \theta\right), \\ \text{Intact}, \forall j \in [1,K]:[HE_j = HR_j] \wedge \frac{U}{K} \geqslant \theta. \end{cases} \tag{16}$$

式(16)含义:①若 IMR 在某次监测交互中报告的验证值与预期值不一致,即可断定节点受损;②虽然所有验证值与预期值一致,但若具有有效散列时间的交互比率 $\frac{U}{K}$ 低于阈值 θ,也认为节点受损;③仅当各次监测交互的验证值全部匹配,具有有效散列时间的交互比率 $\frac{U}{K}$ 不低于 θ,才认为完整性节点保持完整.

2 实验与性能分析

2.1 LW-Hash 算法实现与性能分析

2.1.1 LW-Hash 算法实现

以一款 32 b 嵌入式系统评估板为待监测节点,采用基于 Cortex M3 核 STM32F103VET6 处理器,片内含 512 KB Flash,48 KB RAM,最高主频为 72 MHz,运行 Thumb-2 指令集,是一种存储资源与计算能力受限节点. 节点提供 A/D,D/A,PWM,CAN,SP2,I2C 等接口,支持环境监测、能量计费、工业控制等应用系统的数据采集功能,提供 10BASE-T 以太网与波特率为 9600 bps 的 RS-485 两种通信方式与 Verifier 通信. Flash 存放节点所有软件模块,空闲部分用不可压缩随机数填充,节点完整性监测就是监测节点 Flash 各单元内容是否与预期值一致.

验证值伪造惩罚时间放大系数 ρ 是影响散列时间有效性可检验性的关键因素之一,为防止攻击者通过改造 LW-Hash 算法,用算法优化时间 t_{opt} 来抵消验证值伪造惩罚值时间 t_{extra},我们根据 Cortex 体系结构特点,构造了如图 5 所示 LW-Hash 算法在 Cortex M3 节点的优化实现.

LW-Hash 算法的优化实现采取了 3 项优化措施:1)设置参数 $l=1$,产生 32 b 长度验证值,进一步简化迭代压缩结构,在减小 t_{unit} 同时增大获取较大 t_{opt} 的难度;2)用 26 条指令处理 8 个待监测单元,平均每单元仅需 3.25 指令,留给攻击者的优化空间小,使 $t_{opt} \ll t_{unit}$;3)在验证值中融入 PC 与 PSW 中进位 C 标志,内循环用完所有可用通用寄存器 R0~R12,若实施存储器复制攻击,需增加存储地址检查与调整、PC 与 PSW 的保存与恢复操作,以及为之准备通用寄存器的操作,散列惩罚时间 t_{extra} 大.

2.1.2 LW-Hash 散列算法随机性

验证值随机性是基于动态验证值监测机制可靠工作的基础,散列算法具有良好防差分攻击[15]能力,才能使验证值伪造行为受到散列时间延长的惩罚.

```
Assembly Code
  MOV R0,#Random   /* Checksum is initialized with ramdom */
  MOV R11,#m/8   /* ΨB is divided into 8 groups */

loop1:
  LDMIA R12!,{R2-R9}   /* R12 is pointer of cells storing address of memory blocks to be hashed */
  LDR   R10,#S   /* Load to R10 Size of memory block */

loop2:
  LDR   R1,[R2],#4   /* Read a memory cell */
  ADDS R0,R1,R0,RRX   /* h_i←x_i+(h_{i-1}≫PSW.C) and update PSW */
  XOR   R0,R14,R0   /* h_i←h_i XOR PC */
  ⋮
  LDR   R1,[R9],#4
  ADDS R0,R1,R0,RRX
  XOR   R0,R14,R0
  SUBS R10,R10,#1
  BNE   loop2   /* Jump to process the next cell of same memory block */
  SUBS R11,R11,#1
  BNE   loop1   /* Jump to process the next memory block */
```

Fig. 5 Optimized implementation of LW-Hash on Cortex nodes.

图 5 LW-Hash 在 Cortex 节点上优化实现

由于算法特性独立于节点平台,本文仅在 Cortex M3 节点平台进行 LW-Hash 散列算法随机性测试实验,实验结果具有代表性.对内容初始化为随机数的 2,256,4 096,65 536 个单元内容,随机选择一个单元,翻转其中一个比特,计算 32 b 验证值,连续测试 4096 次.定义击中次数为 2 个验证值对应位置数值相同的字节数,距离为 2 个验证值对应位数值不同的比特数[16],计算击中次数、最小距离、平均距离,实验结果如表 1 所示.可以看出,由于随机移位、交替运算融合打乱了验证值与其输入数据间联系,在4 096 次随机测试中,击中 1 次的比率在 1.8%以内,击中 2 次的比率微小,验证值间最小距离为 5,平均距离围绕理想值 16 作微幅波动,偏差都在 1%以内,表现出良好的随机性.受损节点获取正确验证值的最快方法是执行 LW-Hash 散列算法,若实施存储器复制攻击伪造正确验证值,必须增加搜寻被更改模块原有代码的操作,这样就为基于散列时间有效性检验可靠监测节点完整性创造了条件.

Table 1 Randomness Properties of LW-Hash

表 1 LW-Hash 算法随机特性

Cell Number n	0 Hit	1 Hit	2 Hit	Minimum Distance	Mean Distance	Deviation /%
2	4 041	55	0	5	16.014	0.09
256	4 038	58	0	6	15.907	0.58
4 096	4 036	59	1	6	16.058	0.36
65 536	4 019	75	2	5	16.036	0.40

2.1.3 验证值伪造惩罚时间放大性能

验证值伪造惩罚系数是衡量系统对验证值伪造惩罚时间存在性检测能力的指标[7,9,14].在图 5 算法实现中,顺序处理的前后单元地址呈随机变化特性,若对节点实施存储器复制攻击,需对 Hash 模块代码进行至少 4 处修改:1)对 Rd(为 R2,R3,…,R9 之一)中访存单元地址 x_i 进行检查,若所在区域[$\#A_1$,$\#A_2$]的原有模块代码已搬至他处保存,则需加上偏移量 $\#offset$,增加 5 条指令:"CMP Rd,$\#first$;BLO next;CMP Rd,$\#last$;BHI next;ADD Rd,Rd,$\#mod\_offset$;next:";2)由于执行 CMP 指令更改了 PSW 中条件标志,需在每条 ADDS 指令前后分别添加 4 条用于恢复与保存 PSW 的指令:"MRS R11,cpsr;PUSH R11","POP R11;MRS R11,cpsr";3)因指令地址发生变化,每条 XOR 指令前需增加 1 条从偏移地址 $\#pc\_offset$ 处获取预期 PC 值的指令"LDR R11,[R10,$\#pc\_offset$]";4)因 R10 本来存放存储块中未处理单元数,内层循环开始处需增加保存 R10 的指令 PUSH R10,恢复 R10 内容指令"POP R10".理论上一次内层循环至少需多执行 58 条指令,处理每单元平均需多执行 7.25 条,验证值伪造惩罚指令数放大系数为 $\rho_{\text{instruction}}=223\%$.

图 6 通过比较正常节点与遭受存储器复制攻击的受损节点净散列时间随单元数变化关系,展现了对验证值伪造惩罚时间的放大性能.受损节点因执

行更多操作,其所需净散列时间 $TMCI$ 高出正常节点净散列时间 $TGCI$ 达 75%,即验证值伪造惩罚系数达 $\rho=75\%$. 虽然系数 ρ 因 Flash 单元访存周期长而明显低于 $\rho_{\text{instruction}}$,但显著高于 SWATT 模型获得的惩罚系数 13%[7] 与 VIPER 模型获得的 13.7%[11],为容忍节点散列波动与消息传输波动时间干扰,提高监测可靠性,创造了条件.

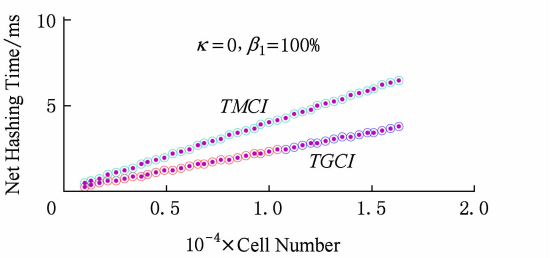


Fig. 6 Scaling performance against checksum forging punishment.

图 6 验证值伪造惩罚时间放大性能

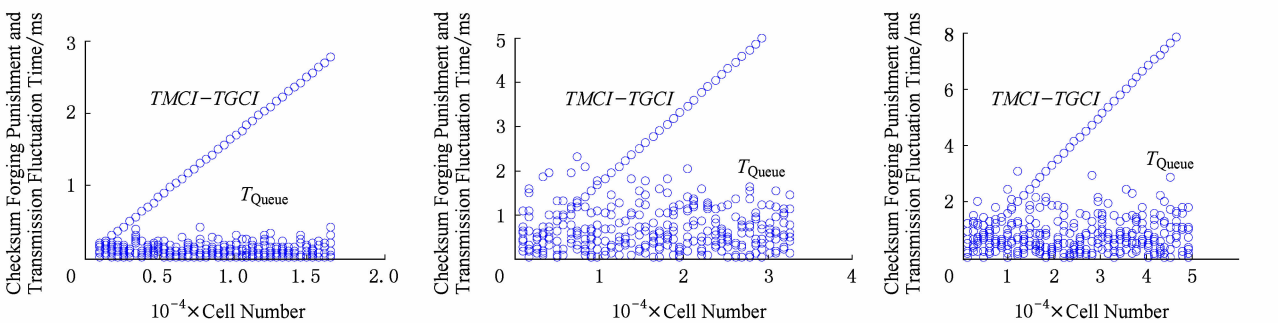
2.2 散列时间有效性可检验性能

2.2.1 对波动时间干扰的容忍性能

本文监测机制在存在散列波动时间与传输波动时间环境下工作,具有容忍波动干扰的能力. 对于节点散列波动干扰,根据式(9)散列时间有效性可检验条件 $\kappa_1<\rho$,理论上在散列波动系数 $\kappa_1<75\%$ 的条件下,遭受存储器复制攻击的受损节点要获得正确的 LW-Hash 验证值,其最小期望散列时间 $TMCE$ 会超过正常节点最大期望散列时间 $TGCE$. 若消息传输波动 T_{Queue} 可忽略,能容忍节点在验证值计算期间用 $\kappa<50\%$ 的 CPU 时间处理中断、任务切换或执行其他任务.

对传输波动干扰的容忍性能与节点所受攻击威胁种类有关. 若节点遭受 Proxy 攻击威胁,最小散列值伪造惩罚时间为监测消息转发与接收时间 T_{SR},监测机制仅能容忍小于 T_{SR} 的消息传输波动时间.

若仅受存储器复制攻击威胁,图 7 给出了受损节点伪造验证值惩罚时间 $TMCI$-$TGCI$、传输波动时间 T_{Queue} 与涉及单元数 n 间变化关系. 图 7(a)为在消息传输时间呈较大幅度波动的 Ethernet 环境下对传输波动干扰的容忍性能,若设参数 $T_{\text{SR}}=0.24$ ms,$\kappa_2=100\%$,$\beta_2=95\%$,波动时间上限 $\Lambda_{\text{Queue}}=T_{\text{SR}}$,有 5%超过 Λ_{Queue} 的异常 T_{Queue},当一次监测交互中参与验证值计算的单元数 $n>2\,500$ 时,就能保证 $TMCI$-$TGCI>T_{\text{Queue}}$;图 7(b)中,波动时间上限增大到 $\Lambda_{\text{Queue}}=8T_{\text{SR}}$ 时,有 1%超过 Λ_{Queue} 的异常 T_{Queue},设置 $n>15\,000$ 时,能保证 $TMCI$-$TGCI>T_{\text{Queue}}$;图 7(c)为在传输时间较为稳定的 RS-485 总线环境下,对传输波动干扰的容忍性能,消息收发时间 $T_{\text{SR}}=124$ ms、波动系数 $\kappa_2=2\%$、波动时间上限 $\Lambda_{\text{Queue}}=2.50$ ms 时,有 1%超过 Λ_{Queue} 的异常 T_{Queue},设置 $n>20\,000$ 时,可保证 $TMCI$-$TGCI>T_{\text{Queue}}$.



(a) Ethernet, $\kappa_2=100\%$, $\beta_2=95\%$, $T_{\text{SR}}=0.24$ ms (b) Ethernet, $\kappa_2=800\%$, $\beta_2=99\%$, $T_{\text{SR}}=0.24$ ms (c) RS485, $\kappa_2=2\%$, $\beta_2=99\%$, $T_{\text{SR}}=124$ ms

Fig. 7 Tolerance against fluctuation of message transmission time.

图 7 对消息传输波动时间的容忍性能

虽然 SWATT,VIPER 等模型也具有时间波动干扰容忍能力,但由于其惩罚系数 ρ 较低,抵抗散列波动能力弱;尽管理论上可通过增大单元数 n 来掩盖较大的传输波动干扰,但 $TGCI$ 增大会影响正常任务和系统中断的及时处理.

2.2.2 有效与非有效散列时间区分度

非有效散列时间包括受损节点如实报告的真实非有效散列时间与伪造非有效散列时间 2 种. 图 8 给出了正常节点如实报告的期望散列时间 $TGCE$ 取值区间与受损节点如实报告期望散列时间 $TMCE$ 取值区间间关系. 在散列波动高达占用 $\kappa=33\%$ CPU 时间条件下,遭存储器复制攻击受损节点期望散列时间 $TMCR$、正常节点期望散列时间 $TGCE$ 数值处于 2 个不重叠的区间内. 随着 n 增大,2 个区间之间

隔离带越来越宽,有效与非有效区分度越来越好.虽然 κ 增大会使 $TGCE$ 值增大,导致隔离带变窄,但只要满足式(9)(10)可检验条件,2 个区间就不会重叠.

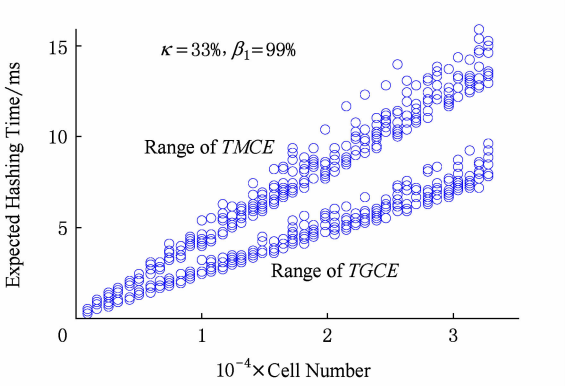


Fig. 8 Differentiation between effective hashing time and ineffective ones.

图 8 有效与非有效散列时间区分度

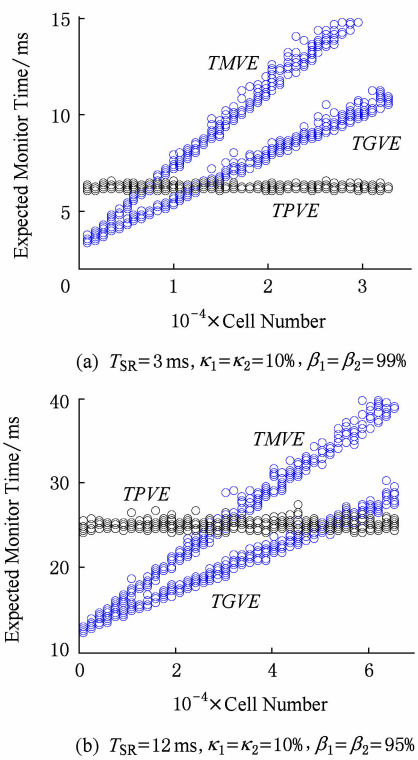
我们在待监测节点上运行 μcos 实时系统,并进行任务切换时间与中断响应时间进行测试,得到结果分别仅为 35μs 与 14μs.若将验证值计算任务设置为最高优先级任务,则该任务执行期间不会发生任务切换,时钟中断为主要散列干扰源,即使设置 SysTick 定时器 1 ms 产生一次中断,绝对散列波动系数仅为 κ=1.4%.由此可知,在一般情况下,采用本文监测机制,可在任务调度、时钟中断、数据采集照常工作条件下,在 $TGCE$ 与 $TMCR$ 取值区间之间形成一个很宽的隔离带,Verifier 能可靠鉴别受损节点完整性状态.

由于 $\max(TCR)=(1+\kappa_1)\times TGCI$,若受损节点报告虚假散列时间 TCR,则 Verifier 推算的最小消息传输时间为 $\min(TMVE)-\max(TCR)>(1+\kappa_2)T_{SR}$;由于 $\min(TCR)=TGCI$,$\max(TMVE)=\max(TMCE)+T_{SR}+\Lambda_{Queue}=(1+\rho+\kappa_1)\times TGCI+(1+\kappa_2)T_{SR}$,推算的最大消息传输时间为 $\max(TMVE\text{-}TCR)=(\rho+\kappa_1)\times TGCI+(1+\kappa_2)T_{SR}$;因此 $TMVE\text{-}TCR\in((1+\kappa_2)T_{SR},(\rho+\kappa_1)\times TGCI+(1+\kappa_2)T_{SR}]$.而正常节点的期望消息传输时间为 $TGVE-TGCR\in[T_{SR},(1+\kappa_2)T_{SR}]$,显然 $TMVE\text{-}TCR\notin[T_{SR},(1+\kappa_2)T_{SR}]$,Verifier 可正确鉴别受损节点报告的虚假散列时间.

2.2.3 散列时间有效性可检验性能

散列时间有效性可检验性能由期望正常监测时间 $TGVE$ 与期望非正常监测时间 $TMVE$ 或 $TPVE$ 间差值大小来表征.图 9 给出了 2 种不同配置条件下 3 种监测时间随单元数 n 变化关系.图 9(a)中监测消息收发时间为 T_{SR}=3 ms,散列波动系数 κ_1 与传输波动系数 κ_2 均为 10%,散列波动时间 T_{MT} 与传输波动时间 T_{Queue} 位于相应取值区间的比例 β_1,β_2 均为 99%.若节点仅受存储器复制攻击威胁,当 $n>5\,000$ 时,有 $\min(TMVE)>\max(TGVE)$,Verifier 能可靠检验散列时间 TCR 有效性;若仅受 Proxy 攻击威胁,则当 $n<12\,000$ 时,$\min(TPVE)>\max(TGVE)$,Verifier 能可靠检验 TCR 有效性;若同时受 2 种攻击威胁,图 9 中 3 条粗直线围成的三角形对应的参数 n 取值区间为散列时间有效性可检验条件,即当 $n\in(5\,000,12\,000)$时,才能可靠检验 TCR 有效性.随着波动干扰增加,κ_1,κ_2 变大,粗直线进一步变粗,图 9 中三角形越来越小,能使 Verifier 可靠监测 TCR 有效性的 n 取值区间不断缩小,直到宽度为 0,不等式(9)(10)无解.

图 9(b)中将消息收发时间提高到 T_{SR}=12 ms 时,波动参数 κ_1,κ_2 保持不变,T_{MT} 与 T_{Queue} 位于相应取值区间的比例 β_1,β_2 均放宽至 95%.在存在 Memory Copy 攻击威胁、Proxy 攻击威胁与 2 种威胁并存的环境下,Verifier 可靠监测 TCR 有效性的条件分别为 $n>12\,000$,$n<40\,000$ 与 $n\in(12\,000,$



(a) T_{SR}=3 ms, $\kappa_1=\kappa_2$=10%, $\beta_1=\beta_2$=99%

(b) T_{SR}=12 ms, $\kappa_1=\kappa_2$=10%, $\beta_1=\beta_2$=95%

Fig. 9 How monitor time varies with memory unit number n under two scheme configurations.

图 9 监测时间随单元数 n 变化关系

40 000). 由于消息收发时间增加,图 9 中三角形明显增大,能满足 2 种威胁并存环境下散列时间有效性检验的单元数 n 取值区间变大.

综合图 9(a)(b)实验结果,若节点仅受存储器复制攻击威胁,采用快速的通信网络或总线允许将 n 设置为较小数值,降低监测交互开销,减轻对节点任务、系统中断实时性影响;若仅受 Proxy 威胁,采用低速通信网络或总线,将单元数 n 设置为较小数值,可使 $TPVE-TGVE$ 增大,提高散列时间有效性检验可靠性;若同时存在 2 种威胁,采用低速通信网络或总线,有利于增大不等式(9)(10)中参数 n 解区间宽度,增大正常监测时间与非正常监测时间差异.

2.3 模型相对性能分析

我们以动态验证值方法中的经典模型 SWATT[7]、改进模型 Shah[12]、基于矩阵运算模型 PIV[13]为参照,分析本文模型性能.

2.3.1 散列算法性能分析

散列值伪造惩罚系数是反映监测模型对伪造验证值检测能力的性能指标,它还反映监测模型对波动干扰容忍能力,对有效散列时间正确性判断能力. 由于 PIV 模型未对验证值伪造时间进行实验与分析,这里仅给出 SWATT 模型、Shah 模型与本文模型散列算法的性能对比,如表 2 所示. 3 种模型都对散列算法进行简化设计,以提高验证值伪造惩罚系数 ρ. 由于 Shah 模型采取融入程序状态增大散列值伪造难度等措施,将惩罚系数 ρ 提高到 51%. 而本文模型通过优化 LW-Hash 散列算法结构,将 ρ 进一步提升到 75%. 惩罚系数 ρ 越大,模型对散列波动容忍性能越好. LW-Hash 算法在验证值计算期间,能容忍节点将最多 42% CPU 时间用于中断处理、系统管理与其他任务处理,抗散列波动干扰性能明显强于 SWATT 与 Shah 模型散列算法.

Table 2 Performance Comparison of Hashing Algorithms

表 2 散列算法性能对比

Monitoring Model	Checksum Forging Punishment Coefficient ρ/%	Tolerance to Percentage of CPU time not used for Hasing/%	Probability of Misjudging Valid Hashing Time	
			$\Pr(T_{MT}\leqslant 0.1\times TCGI)=0.95$	$\Pr(T_{MT}\leqslant 0.3\times TCGI)=0.95$
SWATT	13	11	0.02	0.63
Shah	≤51	33	2.3×10^{-7}	0.037
Proposed Model	75	42	1.7×10^{-10}	5.5×10^{-4}

在存在散列波动干扰环境下,Verifier 判定正常节点报告非有效散列时间概率为 $\Pr(T_{MT}\geqslant\rho\times TCGI)$. 在 T_{MT} 不超过 $10\%\times TGCI$ 的概率为 95% 环境下,设置 3 种模型散列波动时间上限 Λ_{MT} 分别为 $TCGI$ 的 13%,51%,75%,Verifier 判定正常节点报告非有效散列时间概率分别为 0.02,2.3×10^{-7},1.7×10^{-10}. 若应用到 T_{MT} 不超过 $30\%\times TGCI$ 概率为 95%的环境,本文模型对有效散列时间误判概率低于 5.5×10^{-4},但 SWATT 与 Shah 模型中该概率已分别增大到 0.63 与 0.037. 因此,在散列波动干扰较大的节点环境中,本文模型区分有效与非有效散列时间的性能显著优于 SWATT 与 Shah 模型.

2.3.2 监测开销分析

节点监测开销与散列算法结构、验证值长度、节点内存大小、模型参数设置等多种因素有关,主要包括节点存储开销、消息通信开销、节点计算开销 3 个方面. 图 10 给出 SWATT,Shah,PIV 与本文模型的

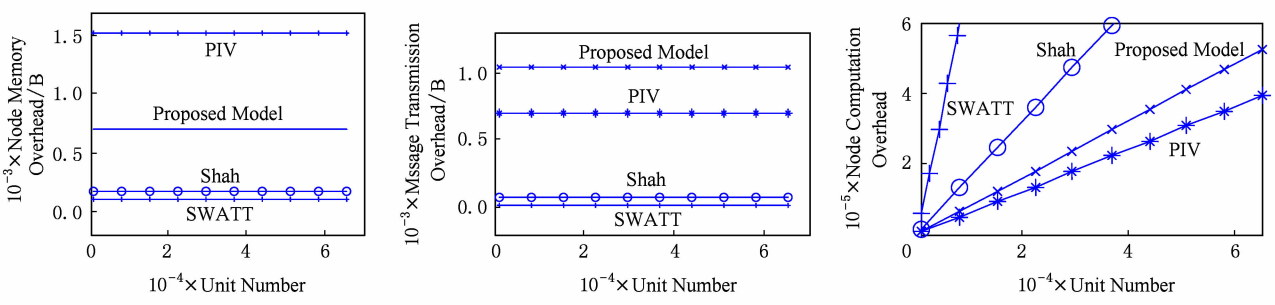


Fig. 10 Node monitoring overhead comparison.

图 10 节点监测开销对比分析

监测开销随节点待监测单元数变化状况对比.

4 种模型的节点存储开销为常量,其中,SWATT 与 Shah 模型散列算法结构简洁,存储开销在 0.2 KB 以内[7,9]. PIV 模型基于矩阵、向量运算方法计算验证值,加上随机密值矩阵所需存储较大,其节点存储开销达 1.5 KB[13]. 本文模型实现验证值计算、消息认证功能仅需 0.7 KB 存储空间,优于 PIV 模型,虽然高于 SWATT 与 Shah 模型,但对于具有数百 KB 或数 MB 内存配置的节点来说,其存储开销几乎可以忽略.

4 种模型的消息通信开销都具有不变特性,SWATT 与 Shah 模型仅需传输随机的存储地址初始值与验证值,消息通信开销低于 0.1 KB[7,9]. PIV 通过消息发送散列模块、随机密值矩阵、散列值,所需开销接近 0.7 KB[13]. 本文模型在一次监测交互中需传输存储块地址序列、验证值、消息验证码,通信开销约 0.11 KB,一次监测会话开销为 1.1 KB,虽然高出其他 3 种模型,但由于完整性监测执行频度不高,不会给通信网络或总线带来明显的额外负担.

节点计算开销随待监测单元数增加而增大,SWATT 模型基于 PRG 产生随机化访存地址,覆盖节点 N 个单元所需运算次数[7]为 $P_{SWATT}=8\times N\times \ln N$,它随 N 增加而快速增长. Shah 模型处理一个单元需要 16 次运算,计算一次验证值所需开销 $P_{Shah}=16N$[9]. PIV 模型计算 9B 长验证值需 $P_{PIV}=6N$ 次加法运算或乘法运算[13]. 本文模型处理一次内存访问需 4 次运算,每单元平均访问 2 次,执行一次监测会话计算开销为 $P_{LW\text{-}Hash}=8N$,虽然略高于 PIV 模型,但轻量级散列算法 LW-Hash 将处理每单元所需指令数减至 3.25 条,实际验证值计算效率并不逊色. 另外,本文模型每次监测交互仅处理 10%~40%单元,占用 CPU 时间较短,对节点正常任务影响更小.

2.3.3 安全性分析

基于动态验证值监测模型安全性可用其防攻击性能来反映,这里主要分析比较防范穷举搜索(exhaustive search attack)[5]、重放(replay)[4]、猜测(guessing)[17]、存储复制(memory copy)[7-8]、共谋(collusion)[18]、代理(proxy)[7,9]等攻击手段的性能,比较结果如表 3 所示:

Table 3 Comparison of Anti-Attack Performance

表 3 防攻击性能对比

Model	Resist Exhaustive Search Attack	Resist Replay	Resist Guessing	Resist Memory Copy	Resist Collusion	Resist Proxy	Resist Message Forging
SWATT	Good	Excellent	Excellent	Moderate	Excellent	Good	Poor
Shah	Good	Excellent	Excellent	Good	Excellent	Good	Poor
PIV	Excellent	Excellent	Excellent	Moderate	Weak	Poor	Poor
Proposed Model	Excellent	Excellent	Excellent	Excellent	Excellent	Good	Excellent

穷举搜索攻击需建立基于挑战值 Challenge 的验证值检索表,通过查表伪造验证值,检索表规模反映防范攻击的性能. 对于 Challenge 长度为 $L_{Challenge}$(单位为 b)的监测模型,完整的检索表大小为 $2^{L_{Challenge}}$,SWATT,Shah,PIV 与本文模型 Challenge 长度分别为 32 b,32 b,2 304 b 与 240 b[7,12-13],PIV 与本文模型所需检索表规模大于任何设备存储能力,防穷举检索性能极高,SWATT 与 Shah 模型检索表大小达 4×10^9 项,具有较高防穷举检索能力.

重放攻击是指受损节点通过重放不久前收集到的验证值消息来冒充正常节点,4 种模型都通过随机变化的 Challenge 使正常节点验证值以非预期方式随机变化,防重放攻击性能高.

猜测攻击是受损节点随机选择一个验证值发回 Verifier 实施欺骗,对于验证值宽度为 L_{Hash}(单位为 b)的模型,攻击成功概率为 $2^{-L_{Hash}}$. SWATT,Shah,PIV 的验证值宽度分别为 64 b,544 b,72 b,随机猜测成功概率最高为 10^{-19},防猜测攻击性能高. 本文模型虽采用 32 b 验证值,但基于"多次交互、一错否决"策略进行节点完整性评判,可将随机猜测成功概率降至同样水平.

存储器复制攻击主要通过散列时间有效性验证来检测,监测模型获得的威胁惩罚系数 ρ 越大,Verifier 正确区分正常节点与受损节点的概率 $Pr(T_{MT}<\rho\times TCGI)$ 越高. SWATT 模型对这种攻击给予的惩罚系数仅 13%[7],对这种攻击防范能力有限. Shah 模型将 ρ 提高到 51%[12]时,检测可靠性大为增强,但由于 Flash 单元访问周期长,随着 CPU 主频提高,实际惩罚系数会降低. PIV 模型按照固定顺序对存储块进行处理来计算验证值[13],攻击者可

以将处理正常存储块与处理被篡改块的代码分离出来,大幅减小散列惩罚时间 T_{extra},降低对受损节点完整性检测可靠性.本文模型将 ρ 提升到 75%,抗波动干扰与防攻击性能强明显高于其他 3 种模型.

在共谋攻击中,多个恶意节点合作完成受损节点验证值计算.PIV 模型基于矩阵与向量运算计算验证值,散列算法具有高并发特性,容易划分子任务分布到合谋节点执行,加快验证值计算过程,抗共谋攻击性能低.SWATT,Shah 与本文模型散列算法都具有反并行特性,仅能由一个节点按串行方式运行来计算正确验证值,防共谋攻击性能好.

代理攻击需增加消息传递时间,根据散列时间检测节点完整性的基本条件为 $T_{\text{SR}}+TOCI>TCGI$.PIV 与 Shah 模型在相应实验环境下计算节点验证值所需散列时间[12-13] $TCGI$ 分别为 40 s 与 8.1 s,代理节点容易采用更快硬件、算法优化等方法能在 $TOCI<TCGI-T_{\text{SR}}$ 时间内完成验证值计算,2 种模型对代理攻击的防范能力都较弱,SWATT 模型也存在同样的问题[7].本文模型可通过调节存储块尺寸来控制 $TCGI$ 大小,在传输时间波动较小、通信速率较低的网络环境下,防代理攻击能力较强.

消息伪造的主要目的之一是诱导 Verifier 对节点完整性状态做出误判.SWATT,Shah 与 PIV 模型都不支持对监测命令与验证值消息的认证功能,攻击者容易伪造验证值消息使 Verifier 将正常节点误判为受损节点.本文模型基于节点密钥计算监测命令、验证值消息的认证码,能认证消息真实性、完整性、时鲜性,防消息伪造能力强.

通过与 SWATT,Shah,PIV 等模型的性能对比与分析可知,本文模型以微小的存储、通信开销,以更高的计算效率提高了验证值伪造惩罚系数,增强了对节点散列波动与消息传输波动干扰的容忍能力,提供了消息认证功能,提高了对多种篡改威胁的防范性能,更适合工作于存在波动干扰、运行实时任务、资源受限、安全关键节点的完整性监测.

2.4 完整性监测可靠性评估

本文模型可靠性可用节点面临不同攻击威胁环境下,对受损节点的漏检概率与对正常节点误诊概率来评估.

1) 防穷举搜索攻击

若攻击者预先计算所有随机化存储块序列 Ψ_{B} 对应的验证值节点外存检索表中,该检索表大小至少为 $\min\limits_{M\geqslant(i-1)\lceil m\gamma\rceil}(M^{m-\lceil m\gamma\rceil}C_{m\gamma}^{M-(i-1)\lceil m\gamma\rceil})$.若 $M\geqslant 50$,$m\geqslant 20$,检索表大小超过 10^{32},无论计算开销与存储开销都对与设备节点都难于承受,即使用 40 GB 的 SD 卡保存 10^{10} 个验证值,受损节点漏检概率仍低于 10^{-20}.

2) 防重放攻击

若攻击者将从网络监听收集到的 L 个 Ψ_{B} 及对应验证值 HR 建立检索表,每次监测交互 Ψ_{B} 与检索表匹配概率低于 $\dfrac{L}{\min\limits_{M\geqslant(i-1)\lceil m\gamma\rceil}(M^{m-\lceil m\gamma\rceil}C_{m\gamma}^{M-(i-1)\lceil m\gamma\rceil})}$,一次监测会话中散列值全部匹配的概率低于 $\left[\dfrac{L}{\min\limits_{M\geqslant(i-1)\lceil m\gamma\rceil}(M^{m-\lceil m\gamma\rceil}C_{m\gamma}^{M-(i-1)\lceil m\beta\rceil})}\right]^{\frac{M}{\gamma}}$,即使检索表大至 $L=10^{10}$ 项,当 $M\geqslant 50$,$m\geqslant 20$ 时,受损节点漏检概率低至 10^{-60}.

3) 防猜测攻击

由于 LW-HASH 算法具有良好随机性,攻击者采用猜测方法猜中正确散列值的概率为 $(lw)^{-1}$,猜中一次监测会话中所有 $M(\gamma m)^{-1}$ 次监测交互散列值概率为 $\left(\dfrac{1}{lw}\right)^{\frac{M}{\gamma m}}$,与固件块数 M、散列值长度 w 成反比,若设置 $M=50$,$m=20$,$\gamma=0.5$,$w=32$,$l=1$,受损节点漏检概率不超过 10^{-48}.

4) 防存储器复制与代理攻击

若一次监测会话中验证值全部正确,节点保持完整,但因波动时间 T_{MT} 或 T_{Queue} 超过设定上限,使具有有效散列时间的监测交互比率低于 θ,有效散列时间交互数少于 $K\theta$,正常节点误判为受损节点,则误诊概率上限为 $\sum\limits_{j=0}^{K\theta-1}C_K^j(1-\beta_1\beta_2)^{K-j}(\beta_1\beta_2)^j$,随 K,β_1,β_2 增大而降低.

若 IMR 模块被更改,在一次监测会话中攻击者采用随机猜测方法给出不少于 $K\theta$ 个具有有效散列时间的正确散列值,但仍有 $K(1-\theta)$ 个交互具有非有效散列时间,受损节点漏检概率为 $\sum\limits_{j=K\theta}^{K}\left(\dfrac{1}{2^{lw}}\right)^j$,随 K,θ,w 增大而降低.将正常节点纳入一起考虑,若 $K=10$,$\theta=0.2$,$\Lambda_{\text{MT}}+\Lambda_{\text{Queue}}=\rho\times TCGI$,$\beta_1=\beta_2=0.95$,$w=32$,$l=1$,误诊与漏检概率可分别低至 10^{-10} 与 10^{-19} 以下,若 β_2 提高到 0.99,则误诊概率还可进一步降至 10^{-12} 以下.

表 4 给出了模型按指定参数配置后系统对 4 种威胁类型带来的攻击难度与系统漏检概率、误诊概

率.在允许少量散列波动时间、传输波动时间超过相应上限值的条件下,在存在穷举检索、重放、猜测、存储器复制与代理攻击威胁的环境中,误诊与漏检概率分别低至 10^{-12} 与 10^{-19} 以下.

Table 4 Theoretical Analysis of Resisting Attack Threats

表 4 理论上防攻击威胁能力

Thread Type	Parameter Settings	Attack Difficulty	Prob. of False Negative	Prob. of False Positive
Exhaustive Search Attack	$M\geqslant 50$ $m\geqslant 20$	Require Storage Capacity large enough to Prestore over 10^{32} Checksum	$\leqslant 10^{-20}$	
Replay Attack	$M\geqslant 50$ $m\geqslant 20$	Require Lookup Table as large as 10^{10} items	$\leqslant 10^{-60}$	
Guessing Attack	$M=50$ $m=20$ $\gamma=0.5$ $w=32$ $l=1$	Difficulty of guessing checksum of all Interaction session increase as checksum length and session number increase.	$\leqslant 10^{-48}$	
Memory Copy and Proxy Attack	$\gamma=0.5$ $K=10$ $\theta=0.2$ $\beta_1=0.95$ $\beta_2=0.99$ $w=32$ $l=1$ $\Lambda_{MT}+\Lambda_{Queue}=75\%c\times TCGI$	Normally, probability of fluctuation time exceeding upper limit is low, and difficulty of obtaining multiple correct checsums by guess increases as $K\theta$ increases	$\leqslant 10^{-19}$	$\leqslant 10^{-12}$

3 结论与展望

基于动态验证值完整性监测方法不要求节点配置防篡改专用硬件单元,基于纯软件手段,利用监测时间检测验证值计算与报告模块完整性,适用性强,适合为无特殊硬件支持的已有嵌入式系统节点完整性监测.

本文研究基于散列时间有效性检验的完整性监测方法,设计一种结合验证值与散列时间监测节点完整性状态的监测机制.通过监测时间构成分析,导出散列时间有效性可检验条件,提出一种融入程序状态轻量级散列算法 LW-Hash,降低验证值计算开销,增大验证值伪造难度,提高了验证值伪造惩罚系数,能在存在较大散列波动与传输波动时间干扰环境下,可靠检验节点散列时间有效性.以随机化存储块地址序列为输入,产生动态变化验证值,降低监测通信开销,设计低开销消息验证算法,执行监测消息认证,防范监测命令与验证值消息伪造攻击.监测模型以微小节点 CPU、存储开销与监测消息通信开销为代价,可靠监测资源受限节点完整性,有效抵抗字典、录制重放、随机猜测、存储器复制与代理攻击,适用工作于存在波动干扰环境、运行实时任务、资源受限、安全关键设备节点的完整性监测.

随着数据采集与监视控制系统、物联网节点规模的增长,监测方 Verifier 计算开销、监测消息通信开销成为不可忽视的问题,发往或来自不同节点的监测消息因竞争通信网络或通信总线会增大传输波动时间.下一步将基于动态验证值方法拓展到具有大量节点系统的完整性监测,采用分布式监测方案,重点研究多节点并发监测、监测节点计算开销优化、通信开销优化与合谋攻击防范等问题.

参 考 文 献

[1] Falliere N, Murchu L, Chien E. W32. stuxnet dossier [R/OL]. Cupertino: Symantec, 2011 [2014-10-01]. http://www.symantec.com/security_response/writeup.jsp?docid=2010-071400-3123-99

[2] Xu Mingdi, Zhang Huanguo, Zhao Heng, et al. Security analysis on trust chain of trusted computing platform [J]. Chinese Journal of Computers, 2010, 33(7): 1165-1176 (in Chinese)
(徐明迪,张焕国,赵恒,等. 可信计算平台信任链安全性分析[J]. 计算机学报, 2010, 33(7): 1165-1176)

[3] Zhang Huangguo, Li Jing, Pan Danling, et al. Trusted platform module in embedded system [J]. Journal of Computer Research and Development, 2011, 48(7): 1269-1278 (in Chinese)
(张焕国,李晶,潘丹铃,等. 嵌入式系统可信平台模块研究[J]. 计算机研究与发展, 2011, 48(7): 1269-1278)

[4] Basile C, Carlo V, Scionti A. FPGA-based remote-code integrity verification of programs in distributed embedded systems [J]. IEEE Trans on Systems, Man, and Cybernetics. Part C: Applications and Reviews, 2012, 42(2): 187-200

[5] Yen S M, Liao K H. Shared authentication token secure against replay and weak key attacks [J]. Information Processing Letters, 1997, 62(2): 77-80

[6] Premaratne K, Kulasekere E C, Bauer P H, et al. An exhaustive search algorithm for checking limit cycle behavior of digital filters [J]. IEEE Trans on Signal Processing, 1996, 44(10): 2405-2412

[7] Seshadri A, Perrig A, Van Doorn L, et al. Swatt: Software-based attestation for embedded devices [C] //Proc of 2004 IEEE Symp on Security and Privacy. Piscataway: IEEE, 2004: 272-282

[8] Bratus S, D'Cunha N, Sparks E, et al. TOCTOU, traps, and trusted computing [M] //Trusted Computing-Challenges and Applications. Berlin: Springer, 2008: 14-32

[9] Seshadri A, Luk M, Shi E. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems [J]. ACM SIGOPS Operating Systems Review, 2005, 39(5): 1-16

[10] Seshadri A, Luk M, Perrig A, et al. SCUBA: Secure code update by attestation in sensor networks [C] //Proc of the 5th ACM Workshop on Wireless Security. New York: ACM, 2006: 85-94

[11] Li Y, McCune J M, Perrig A. VIPER: Verifying the integrity of PERipherals' firmware [C] //Proc of the 18th ACM Conf on Computer and Communications Security. New York: ACM, 2011: 3-16

[12] Shah A, Perrig A, Sinopoli B. Mechanisms to provide integrity in SCADA and PCS devices [C] //Proc of the Int Workshop on Cyber-Physical Systems-Challenges and Applications(CPS-CA). Piscataway, NJ: IEEE, 2008

[13] Park T, Shin K G. Soft tamper-proofing via program integrity verification in wireless sensor networks [J]. IEEE Trans on Mobile Computing, 2005, 4(3): 297-309

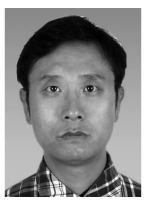
[14] Defrawy K, Francillon A, Perito D, et al. Smart: Secure and minimal architecture for(establishing a dynamic) root of trust [C] //Proc of the 19th Annual Network & Distributed System Security Symp(NDSS). San Diego: Internet Society, 2012

[15] Sun Bing, Zhang Peng, Li Chao. Impossible differential and integral cryptanalysis of Zodiac [J]. Journal of Software, 2011, 22(8): 1911-1917 (in Chinese)
(孙兵,张鹏,李超. Zodiac 算法的不可能差分和积分攻击[J]. 软件学报, 2011, 22(8): 1911-1917)

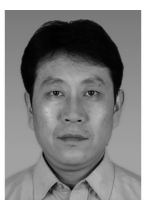
[16] Sheng Liyuan, Li Gengqiang, Li Zhiwei. One-way Hash function construction based on tangent-delay ellipse reflecting cavity-map system [J]. Acta Physica Sinica, 2006, 55(11): 5700-5707 (in Chinese)
(盛利元,李更强,李志炜. 基于切延迟椭圆反射腔映射系统的单向 Hash 函数构造[J]. 物理学报, 2006, 55(11): 5700-5707)

[17] Munilla J, Peinado A. Off-line password-guessing attack to Peyravian-Jeffries's remote user authentication protocol [J]. Computer Communications, 2006, 30(1): 52-54

[18] Saxena V, Gupta J P. Collusion attack resistant watermarking scheme for colored images using DCT [J]. IAENG International Journal of Computer Science, 2007, 34(2): 1-7



Xu Qingui, born in 1967. Professor and PhD. Member of China Computer Federation. His main research interests include computer architecture, information security and intelligent detetion and instrument.



Qin Yong, born in 1970. Professor and PhD. His main research interests include network and parallel distributed processing technology and application(mmcqinyong@126.com).

Yang Taolan, born in 1945. Professor and PhD supervisor. His main interests include computer architecture and compiler(yangtl@dgut.edu.cn).