软件异构冗余执行系统的安全能力分析

马博林[1]，张铮[1]，任权[1]，张高斐[2]，邬江兴[1]
（1. 信息工程大学，河南 郑州 450001；2. 网络通信与安全紫金山实验室，江苏 南京 211111）

摘 要：软件冗余执行（SRE）基于故障随机发生的性质，实现对软硬件故障的容错处理，是常见的容错设计方法。软件异构冗余执行（SHRE）则在 SRE 的基础上利用软件多样化特征，通过冗余执行相同功能的异构软件副本，表决执行结果来抵御软件漏洞和同质化威胁。基于此，提出了 SHRE 系统的分类方法，引入了 SHRE 系统的安全能力概念，考虑 N 模冗余、I/O 操作模式以及受攻击软件副本的恢复能力，分析了不同结构 SHRE 系统的安全性。分析结果显示，SHRE 系统在三模冗余且受攻击软件副本具备恢复能力的情况下安全能力表现最好，缩短受攻击软件副本的恢复时间能够提高系统安全性。

关键词：软件异构冗余执行；软件漏洞和同质化；安全能力
中图分类号：TP393
文献标识码：A
DOI: 10.11959/j.issn.1000−436x.2021176

Security capability analysis of software-based heterogeneous redundant execution system

MA Bolin[1], ZHANG Zheng[1], REN Quan[1], ZHANG Gaofei[2], WU Jiangxing[1]
1. Information Engineering University, Zhengzhou 450001, China
2. Purple Mountain Laboratories, Nanjing 211111, China

Abstract: Software-based redundant execution (SRE) is a popular fault-tolerant design method which makes use of faults occurring randomly to achieve fault-tolerance. Software-based heterogeneous redundant execution (SHRE) uses heterogeneous redundant software replicas with identical function based on SRE and diversity of software. By comparing the results of heterogeneous redundant software replicas, SHRE can resist threats from software vulnerabilities and homogenization. The classification method of SHRE was proposed, and the security capability of SHRE was introduced. Based on N-modular redundancy, I/O operation mode and the recovery capability of attacked software replica, resistance to attack of different structures were analyzed. The analysis shows that the security capability of SHRE performs best when it is triple-mode redundancy architecture and attacked software replica can be recovered. Besides, by shortening the recovery time of attacked software replica, security to SHRE can be improved.

Keywords: software-based heterogeneous redundant execution, software vulnerabilities and homogenization, security capability

1 引言

在网络空间安全被高度重视的今天，软件产品越来越丰富的同时，新的软件漏洞也常出现在人们的视野当中。漏洞的存在主要受以下几个因素影响：1) 软件功能复杂、代码量庞大增加了漏洞存在的可能性，并且有限的测试能力难以应对；2) 全球化市场背景下，软件产品的供应链条越来越长，供

收稿日期：2021−03−24；**修回日期**：2021−06−20
通信作者：张铮，ponyzhang@126.com
基金项目：国家自然科学基金资助项目（No.61521003）；国家重点研发计划基金资助项目（No.2018YFB0804003）
Foundation Items: The National Natural Science Foundation of China (No.61521003), The National Key Research and Development Program of China (No.2018YFB0804003)

应链上的每个环节都可能成为预置漏洞的地方，美国发布的《全球供应链安全国家战略》就提到，信息技术和网络的发展是供应链风险发生的一个重要原因；3) 当前漏洞发现理论和工程技术水平都存在缺陷，在解决漏洞问题方面都不够全面彻底。因此，无论是从信息技术的发展角度还是从供应链利益的博弈角度来解释，漏洞的存在都是必然的。与此同时，软件同质化[1-2]使漏洞可以快速、广泛地传播。信息系统的建设多采用架构技术，因此在软件选择和部署上存在大量相似程序，这就使某个漏洞后门会被攻击者应用到相同环境的目标系统中，为扩大攻击范围提供便利。

软件冗余执行（SRE, software-based redundant execution）[3]利用故障发生的时空随机性质，通过表决比较软件副本的执行结果，可对软硬件故障引起的计算错误实现容错处理。吴斌等[4]率先分析了软件双冗余容错系统的容错能力和性能影响因素，在非极端情况下软件同构冗余执行的容错能力高于单个软件的容错能力。由于SRE的容错能力是建立在随机性故障的基础上，因此其无法应对由软件漏洞和同质化带来的安全威胁。因为冗余的软件副本之间具有完全相同的设计缺陷或漏洞，在相同的攻击输入条件下，同构冗余的软件副本会遭受同样的网络攻击，产生一致的输出结果或同态故障，导致错误可以通过表决比较，失去容错能力。

软件异构冗余执行（SHRE, software-based heterogeneous redundant execution）是结合SRE和软件多样化[5]，用于解决软件漏洞不可避免和同质化安全问题的主要方法。软件副本之间在结构设计和实现上存在不同，甚至是采用了完全不同的技术路线，这就保证了软件副本之间存在相同安全漏洞的概率是极低的。因此，当网络攻击输入时，软件副本产生一致的攻击成功结果，并通过表决比较是极小概率事件。SHRE从机理上既可感知随机性故障，也可抵抗利用软件漏洞的网络攻击。

SHRE抵抗软件漏洞攻击是否可靠，并且如何定量地分析SHRE系统抵抗攻击的能力，仍是需要解决的问题。本文根据SHRE抵抗软件漏洞攻击的原理，通过软件副本不同输出结果的概率分布，证明SHRE不存在误报，抵抗攻击是可靠的；提出以多样化方式、时空维度、发生位置为基准的SHRE系统分类方法；考虑影响SHRE系统结构的关键因素，包括N模冗余、I/O操作模式和受攻击软件副本的恢复能力，同时引入SHRE系统的安全能力概念，分析对比软件单副本与S_0～S_4不同结构SHRE系统的安全性。

S_0表示软件单副本。S_1表示双模冗余结构SHRE系统。S_2表示三模冗余结构、主从模式、受攻击副本无恢复能力SHRE系统。S_3表示三模冗余结构、代理模式、受攻击副本无恢复能力SHRE系统。S_4表示三模冗余结构、受攻击副本有恢复能力SHRE系统。其中，受攻击副本在有恢复能力的情况下，软件副本会在多种状态间转移，利用可达集连续时间马尔可夫链计算S_4的安全性。

分析结果表明，采用三模冗余且受攻击软件副本具备恢复能力设计的SHRE系统更安全，同时缩短受攻击软件副本的恢复时间能够有效提高其安全性。

2 SHRE抵抗攻击的可靠性分析

SHRE是在N模冗余技术（NMR, N-modular redundancy）[6]的基础上，为解决软件同质化和漏洞不可避免的安全问题发展而来的，目前国内外研究成果主要基于双模冗余（DMR, dual modular redundancy）[7]和三模冗余（TMR, triple modular redundancy）[8]。基于DMR的SHRE通过比较双软件副本的结果发现异常，但是无法识别受攻击软件副本，只适用于攻击检测。基于TMR的SHRE能够在三软件副本的结果间进行投票表决，假设至少有2个相同的结果被认为是正确的，系统将输出多数一致的结果，或者以这个结果继续执行，不会影响程序的正确运行，防御效果要优于DMR结构，能够检测攻击并识别受攻击软件副本，更进一步地还可以对其进行恢复。

SHRE系统中软件副本的执行结果不一致，一定存在软件副本产生了异常结果。下面，以DMR结构的SHRE过程为例进行证明。DMR结构的SHRE如图1所示，具有冗余的软件副本α_0和α_1，设定以下集合。

集合S为SHRE系统所有可能的执行结果$\mathrm{Out}[O(\alpha_0),O(\alpha_1)]$组成的集合，其中$O(\alpha_0)$和$O(\alpha_1)$分别为软件副本$\alpha_0$和$\alpha_1$的执行结果。

集合C_α^0和C_α^1分别为软件副本α_0和α_1的正常执行结果集合。

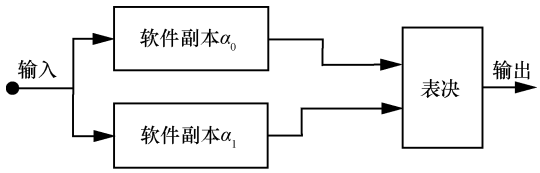


图 1　DMR 结构的 SHRE

集合 E_α^0 和 E_α^1 分别为软件副本 α_0 和 α_1 的异常执行结果集合。

集合 $X=\{x \mid x \in S \wedge x[0] \neq x[1]\}$ 为软件副本 α_0 和 α_1 的执行结果不一致。

集合 $Y=\{y \mid y \in S \wedge y[0] \in E_\alpha^0\}$ 为软件副本 α_0 产生异常执行结果。

集合 $Z=\{z \mid z \in S \wedge z[1] \in E_\alpha^1\}$ 为软件副本 α_1 产生异常执行结果。

当 SHRE 系统发现软件副本 α_0 和 α_1 的执行结果不一致时，若存在没有软件副本产生异常结果的可能，则该情况出现的概率为

$$P(X \cap \complement_S(Y \cup Z)) = P(X) - P(X \cap (Y \cup Z)) \quad (1)$$

根据全概率公式可得

$$P(X) = P(X \cap (Y \cap Z)) + P(X \cap (\complement_S Y \cap Z)) + P(X \cap (Y \cap \complement_S Z)) + P(X \cap (\complement_S Y \cap \complement_S Z)) \quad (2)$$

SHRE 系统在设计实现时充分考虑 I/O 操作、系统调用序列等[9]在一次执行过程中影响软件副本执行结果一致性的因素。通过共享缓存、代理模式、冗余边界[10]等方法保证 SHRE 系统在正常功能范畴内的一次执行过程中，软件副本的正常执行结果是一致的。如果 $\text{Out}[0] \in C_\alpha^0$ 且 $\text{Out}[1] \in C_\alpha^1$，那么 $\text{Out} \in X$ 一定不成立，所以

$$P(X \cap (\complement_S Y \cap \complement_S Z)) = 0 \quad (3)$$

软件副本 α_0 和 α_1 相互独立运行，产生的执行结果是独立的事件，因此

$$P(Y \cup Z) = P(Y \cap Z) + P(\complement_S Y \cap Z) + P(Y \cap \complement_S Z) \quad (4)$$

将式(2)～式(4)代入式(1)，得到

$$\begin{aligned} &P(X \cap \complement_S(Y \cup Z)) = P(X) - P(X \cap (Y \cup Z)) = \\ &P(X \cap (Y \cap Z)) + P(X \cap (\complement_S Y \cap Z)) + \\ &P(X \cap (Y \cap \complement_S Z)) + P(X \cap (\complement_S Y \cap \complement_S Z)) - \\ &P(X \cap (Y \cup Z)) = P(X \cap (Y \cap Z)) + \\ &P(X \cap (\complement_S Y \cap Z)) + P(X \cap (Y \cap \complement_S Z)) - \\ &P(X \cap (Y \cap Z)) + P(X \cap (\complement_S Y \cap Z)) + P(X \cap (Y \cap \complement_S Z)) - \\ &P(X)P(Y \cap Z) - P(X)P(\complement_S Y \cap Z) - \\ &P(X)P(Y \cap \complement_S Z) = P(X)P(Y \cap Z) + \\ &P(X)P(\complement_S Y \cap Z) + P(X \cap (Y \cap \complement_S Z)) - \\ &P(X)P(Y \cap Z) - P(X)P(\complement_S Y \cap Z) - \\ &P(X)P(Y \cap \complement_S Z) = 0 \end{aligned} \quad (5)$$

所以，当 DMR 结构的 SHRE 表决发现软件副本的执行结果不一致时，一定存在软件副本产生了异常结果。由此可得，当具有多个软件副本的 SHRE 表决发现结果不一致时，至少存在 2 个副本（符合 DMR 双模结构）之间不一致，也一定存在软件副本产生了异常结果。因此无论是 DMR 结构还是具有更多副本的 SHRE 抵抗攻击，都可以发现异常。

3　SHRE 的分类方法

SHRE 通过比较软件副本执行结果是否一致来发现软件是否遭受了恶意攻击。SHRE 的方式可以对单一的软件副本进行时间维度上的重复执行，也可以在空间维度上同时执行软件多副本。当然，只有时间或空间资源的冗余配置是不完全的，还需要相同功能作为时间和空间维度上的主线进行关联处理，并且考虑软件多副本间的差异化，以及冗余执行发生的位置，才能有效利用 SHRE 感知异常。下面，按软件多样化、时空维度和冗余执行的发生位置对 SHRE 系统进行分类。

3.1　软件多样化

SHRE 实现软件副本异构的途径通常采用软件多样化技术，依据技术实施的不同来源和动机，软件多样化方法可以分为三类[11]，分别是自然多样化、自动多样化和可控多样化。因此，按照软件多样化方法的不同，将 SHRE 分为基于自然多样化的 SHRE、基于自动多样化的 SHRE 和基于可控多样化的 SHRE。

1) 基于自然多样化的 SHRE

自然多样化是受使用需求、技术约束、市场竞争等因素的影响，软件在开发过程中自然形成的多样化，例如操作系统软件既包含了以 Windows、Linux 为主流的国外操作系统，也包含了深度、普华等国产操作系统。被保护的程序运行在由多样化商用货架产品构建的冗余环境中运行，通过表决机制，抵御利用运行环境中商用货架产品漏洞和后门

对程序的恶意攻击。马海龙等[12]在路由系统的应用层采用 Cisco、Maipu、Quagga 等 7 种异构冗余路由套件，实现了路由系统的主动防御。张铮等[13]基于 Web 服务器的层次化结构，在操作系统层、虚拟化层、服务器软件层等进行异构冗余设计，有效抵御 Web 攻击。宋克等[14]基于盛科 CTC5160 交换芯片，采用异构冗余的 Atom E3930-Ubuntu、QorIQ T1042-VxWorks、龙芯 2K1000-Linus 模组处理上行数据，抵御未知威胁。

基于自然多样化的 SHRE 具有的优势是能够在大量的商用货架产品中快速地选择出功能等价的组件，构建 SHRE 运行环境。但由于软件的定制化开发、商用产品的闭源保护、运行环境的特殊依赖等原因，软硬件的自然多样化无法为一切软件提供功能等价且异构冗余的运行环境，因此还需要基于自动多样化、可控多样化的 SHRE 方法，来扩大 SHRE 的应用范围。

2) 基于自动多样化的 SHRE

基于自然多样化的 SHRE 只能通过多样化的商用货架产品构建 SHRE 运行环境，而该方法无法针对应用程序本身进行多样化变换。自动多样化就是为解决该问题提出的，其在程序的可执行文件加载到内存前，对应用程序的代码层面进行多样化变换。

第一种是对程序的源代码进行多样化变换，例如，马博林等[15]基于指令集随机化提出了抗 PHP 代码注入攻击的方法，通过修改应用源程序与解释器实现原型系统，攻击者由外部注入的恶意代码与可执行代码不一致，攻击无法成功。第二种是在程序的源代码编译生成可执行文件的过程中进行多样化变换，例如，张宇嘉等[16]提出了绑定程序变量的动态不透明谓词实现方法，在 LLVM[17]基础上对中间代码生成阶段进行混淆，并对源程序多次编译生成异构冗余的目标代码。第三种是针对可执行程序进行二进制重写，例如，姚东等[18]提出了基于冗余执行架构的 CFI（control flow integrity），通过二进制重写技术，在间接跳转/调用指令和返回指令处增加表决机制，通过反馈学习优化得到执行频率较高的子图 Sub-CFG，以此对比程序当前运行状态，判断是否遭受攻击。由于编译器的修改很容易继承到大部分平台中，而二进制重写又需要对目标程序进行逆向，增加了难度，因此编译过程的多样化方法要优于对可执行程序进行二进制重写。

3) 基于可控多样化的 SHRE

可控多样化相比于自然多样化和自动多样化，更注重人为控制程序产生多样性，并且可以摆脱对源代码的依赖。一方面在程序功能等价的前提下尽可能产生相对独立的程序版本，从“相对独立”的更高要求出发，以多版本程序（N-version programming）[19]为代表，广泛应用于软件容错、可靠性设计等方面，而由于云服务中的应用程序本就存在多版本特征，该方法也被用来解决云环境下虚拟机间共存攻击[20]。

另一方面控制程序加载时的运行环境，从而使冗余程序在应对攻击行为时会产生不一致的输出，该方面以多变体（N-variant）[21]执行架构为代表。例如，Cavallaro[22]设计了一种多变体架构，当 2 个变体加载到该系统运行时，其中一个变体在地址空间分区的基础上，将地址空间位移多个字节，在 2 个变体同时冗余执行的情况下，地址空间的相对距离产生了变化，当攻击者破坏指针进行地址覆盖攻击时，覆盖结果不同，无法取得一致的攻击结果；Salamat 等[23]则利用堆栈空间设计出 Orchestra 多变体架构，冗余执行的 2 个变体分别采用正向和反向的堆栈生长方式，因此，基于堆栈的攻击无法通过架构中对于 I/O 操作结果的表决机制；Volckaert 等[24]基于不相交的代码布局来保证变体间的异构性，设计出 GHUMVEE 系统，通过变体冗余执行和表决机制有效地防御控制流劫持漏洞攻击。

基于可控多样化的 SHRE，异构性多集中在代码布局、堆栈布局、地址空间布局等软件运行时的相关因素，架构轻量、部署便捷是其主要的优势，但冗余软件副本的同步和调度等工作是实现过程中面临的主要困难。

3.2 时空维度

SHRE 还可以根据时空维度进行分类，3.1 节提及的研究成果均属于软件在空间维度上冗余执行，并行同时处理相同的输入。而阿里安全猎户座实验室联合浙江大学[25]率先提出了一种新技术，即差异重播，在 vanilla memory、poisoned memory 这 2 种差异的情况下串行地重放程序执行，实现了 TimePlayer 原型系统，通过比较串行执行的结果有效地检测未初始化变量的使用。

软件副本在表决点的等待关系会降低 SHRE 的性能，DAFT[26]假设软件副本产生异常执行结果的概率很小，改进软件副本在空间维度上需要并行同

时执行，消除主副本与从副本之间在表决点的等待关系。主副本不等待表决结果继续执行，由从副本对执行结果进行表决，并发送异常处理信号。DAFT 将线程级 SHRE 的性能损耗降低至 1.38x。

3.3　发生位置

SHRE 可从不同的位置实现，从粒度上划分可分为指令级、进程级和应用级。不同的位置和粒度会影响能够抵御的攻击类型，也决定了通过表决发现安全威胁的时机。

1) 指令级 SHRE

指令级 SHRE 通过指令级复制与比较实现异常数据流检查，异构化方法有等效指令替换、等效指令序列随机化、寄存器分配随机化、垃圾指令插入等。SWIFT[27]在同一线程中以指令粒度地重复执行，store 和 branch 指令之前对结果进行表决，发现异常数据流，但是该方法在不同架构处理器中开销差距较大，适用范围有限。VAR3+[28]为了避免在冗余执行过程中创建多个内存位置，复制除 store 和 branch 指令之外的所有指令，在 load、store 和 branch 指令之前表决执行结果，检查数据流是否存在不一致的情况，保证安全性的基础上有效减小开销。ILDC[29]使用冗余的寄存器副本复制除 branch 之外的所有指令，并在 store、branch 指令之前和 load、move 指令之后设置表决点，比较原始操作和重复操作的结果是否一致。实验结果表明，相比于 VAR3+和 SWIFT，ILDC 能够检测到更多的异常，并且开销更低。

2) 进程级 SHRE

进程级 SHRE 采用 2 个独立的进程复制应用程序，并在 I/O 操作或系统调用进行表决同步，由于现代操作系统中程序与底层交互通过系统调用完成，且粗粒度 I/O 过程缺乏有效的安全保障，因此大多数进程级的 SHRE 将系统调用作为表决同步点。

GHUMVEE[24]是在进程副本发生系统调用时进行表决，直到进程副本都到达表决同步点并且系统调用结果一致才会恢复执行，阻止恶意访问进程空间外的代码和数据。ReMon[30]改进了 GHUMVEE 的表决粒度，为安全无紧密关系的系统调用设置宽松的表决策略，既保证了安全防御能力，又提高了性能。潘传幸等[31]基于等价异构程序的表决判定，保证内存地址不泄露，有效防御控制流劫持攻击，并且额外性能开销不超过 13%。进程作为系统进行资源分配和调度的基本单位，进程级 SHRE 的异构方法主要来自程序加载时操作系统的随机化技术，不仅可以给攻击者带来更大的不确定性，还可以与操作系统紧密配合，例如对系统调用号、动态加载库的加载位置、栈生长方向、堆布局等关键对象的随机化方法。线程是比进程更小的程序执行的基本单位，冗余线程共享同一进程下的资源，不能像进程级一样利用操作系统自身的随机化技术，异构方法较为有限，因此线程级 SHRE 研究成果较少。

然而，许多程序具有不确定性行为，冗余进程间可能不具备完全相同的系统调用序列，因此会产生假阳性错误，这也是进程级 SHRE 面临的主要难题之一。

3) 应用级 SHRE

应用级 SHRE 以整体应用为副本，更粗粒度地比较冗余应用的输出是否一致。大多数基于自然多样化的 SHRE 就是在应用级实现，以提供具体应用服务的商用货架产品为冗余目标。

4　SHRE 安全能力分析

4.1　SHRE 安全能力

方滨兴院士[32]提出了网络空间安全是通过实现一组准确“控制”所获得的特定能力，根据 SHRE 原理，该“控制”不再是安全策略、惯例规程等，而是通过软件副本间的异构性，实现的一种内生安全构造。为准确地分析 SHRE 安全能力，本节给出 SHRE 安全能力的定义。

定义 1　SHRE 系统的安全能力是指系统在遭受漏洞攻击的情况下，漏洞利用不成功且保持系统正常可用的能力。

定义 2　SHRE 系统的安全性是指系统在某时刻具备安全能力的概率。

第 1 节中分析了软件不可避免地会存在漏洞，Humphrey[33]对约 13 000 个代表程序进行了多年研究发现，开发者每编写 1 000 行代码，就会产生 100～150 个错误，因此做出如下假设。

假设 1　冗余的软件副本都存在未知漏洞。

对于单程序副本，攻击者能够根据经验及结果不断调整攻击方法。

假设 2　攻击者不受专业能力、知识储备、实验环境等主客观条件的约束。

SHRE 通过表决软件副本的输出结果来发现异

常，斩断攻击链，而面对内部人攻击或者内部违规操作等来自软件副本内部的风险，SHRE 无法有效抵御。

假设 3 不考虑由 SHRE 系统内部发起的攻击或者违规操作。

当攻击者尝试的攻击次数无穷大时，可以成功对软件单副本实施有效的攻击，所以认为随着攻击时间 t 增加，对软件单副本攻击成功的可能性也在增加。因此软件单副本（记为 S_0）在持续攻击下，t 时刻还未被攻击成功的可能性，即软件单副本 t 时刻的安全性表示为

$$\mathrm{AP}_{S_0} = R_\alpha^i(t) = \mathrm{e}^{-\lambda_i t}, 0 < \lambda_i < 1 \tag{6}$$

4.2 影响 SHRE 安全能力的结构

当 SHRE 系统通过表决发现异常后，系统能否继续保证安全可用，主要受系统的 I/O 操作模式和异常处理方法影响。

SHRE 的 I/O 操作模式是为避免冗余的软件副本在重复多次执行同一读写操作后产生影响表决的假阳性问题采取的关键设计，将与读写操作相关的系统调用、功能函数或代码片段只执行一次，如图 2 所示。I/O 操作模式主要分为主从模式和代理模式，其中，主从模式中冗余的软件副本分为主副本和从副本，主副本负责执行读写操作，并将执行结果复制给其他的从副本，保证了所有软件副本可以获得一致的读写结果。代理模式则由分发器代替软件副本执行读写操作，将执行结果复制给所有软件副本。

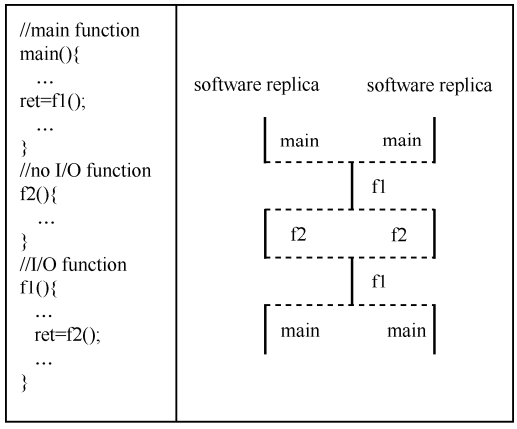


图 2 解决假阳性问题的关键设计

当表决发现差异时，SHRE 系统可以选择停止所有软件副本来阻止攻击，这种情况虽然有效抵御了攻击，但影响了软件功能的可用性。为了保证软件功能持续可用，异常处理也可以选择停止或隔离异常的软件副本，加入新生成的副本，并复制获得正常副本状态。该方式对于代理模式，仅需要考虑新生成副本与异常副本之间的异构性，避免遭受连续攻击。对于主从模式，需要相对复杂的处理过程，如果主副本发生异常，因为其控制着读写操作，首先需要选择正常的从副本作为新的主副本，例如，VARAN[34]通过 Unix 域套接字将文件描述符复制给新的主软件副本，同时加入新生成的从副本；如果从副本发生异常，则按照代理模式的方法进行异常副本替换。

4.3 SHRE 安全场景

SHRE 除了发现和阻断攻击以外，还可以进行反馈调整，对受攻击软件副本进行恢复，提高系统的安全性，本文第 2 节分析了 TMR 比 DMR 结构的优势。例如，N-Variant[21]、Orchestra[23]、Buddy[35]在识别到受攻击软件副本后会通过重启的方式来恢复，Exterminator[36]则能够在运行时生成内存补丁来纠正发现的内存错误，对受攻击软件副本进行恢复。本节以受攻击软件副本是否具备恢复能力为基准，结合主从/代理模式和 DMR/TMR 结构，对多种 SHRE 结构进行安全能力分析。

1) 受攻击副本无恢复能力

受攻击副本无恢复能力的情况下，SHRE 不存在状态的转移和恢复，主要利用软件单副本对攻击的抵抗能力度量 DMR/TMR 结构 SHRE 的安全性。

DMR 结构无论是主从模式还是代理模式，任何单软件副本被攻击成功，系统通过表决感知到输出不一致后都会失去安全能力，安全场景如下。

场景 1 DMR 结构，双冗余软件副本均未被攻击成功。DMR 结构 SHRE 系统（记为 S_1）的安全性可以表示为

$$\mathrm{AP}_{S_1} = R_\alpha^0(t)R_\alpha^1(t) \tag{7}$$

其中，$R_\alpha^0(t)$ 和 $R_\alpha^1(t)$ 为双冗余副本抵抗攻击的安全性。

TMR 结构在主从模式下，由于主副本负责执行读写操作，是其余 2 个从副本运行的基础，因此一旦主副本被攻击成功，系统将失去安全能力；同时为不影响程序正确运行，系统必须保证不少于 2 个副本未被攻击成功。安全场景如下。

场景 2 TMR 主从模式结构，主副本未被攻击成功且 2 个从副本未被全部攻击成功。TMR 结构、主从模式、受攻击副本无恢复能力 SHRE 系统（记

为 S_2）的安全性可以表示为

$$\mathrm{AP}_{S_2} = R_\alpha^0(t)(1-(1-R_\alpha^1(t))(1-R_\alpha^2(t))) \tag{8}$$

其中，$R_\alpha^0(t)$ 为主副本抵抗攻击的安全性，$R_\alpha^1(t)$ 和 $R_\alpha^2(t)$ 为从副本抵抗攻击的安全性。

TMR 结构在代理模式下，副本间不存在主从关系，只要有不少于 2 个副本被攻击成功，系统将会失去安全能力。安全场景如下。

场景 3　TMR 代理模式结构，不少于 2 个程序副本未被攻击成功。TMR 结构、代理模式、受攻击副本无恢复能力 SHRE 系统（记为 S_3）的安全性可以表示为

$$\mathrm{AP}_{S_3} = 1-\left[\prod_{i=0}^{2}(1-R_\alpha^i(t)) + \frac{1}{3}\sum_{i=0}^{2}\left(R_\alpha^i(t)\left(1-R_\alpha^{(i+1)\%3}(t)\right)\left(1-R_\alpha^{(i+2)\%3}(t)\right)\right)\right] \tag{9}$$

2) 受攻击副本有恢复能力

受攻击副本有恢复能力的情况下，副本状态会在未被攻击成功和被攻击成功之间转移，因此利用可达集连续时间马尔可夫链（CTMC, continuous time Markov chain）模型度量 TMR 结构 SHRE 的安全性。

TMR 结构 SHRE 的 CTMC 模型如图 3 所示，模型的稳定状态如表 1 所示。其中，λ_0、λ_1、λ_2 分别表示副本 α_0、α_1、α_2 被攻击成功的速率，即每小时可成功攻击的次数（次/小时），其倒数表示副本被攻击成功的平均时间；μ_0、μ_1、μ_2 分别表示副本 α_0、α_1、α_2 被攻击成功后恢复的速率，即每小时可完成恢复的次数（次/小时），其倒数表示副本恢复的平均时间，μ 值越大恢复速率越快。

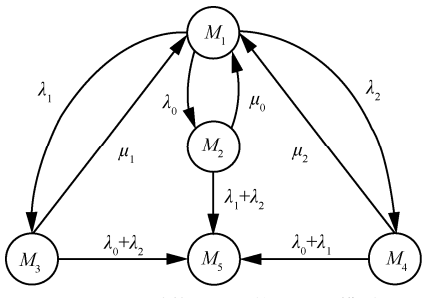


图 3　TMR 结构 SHRE 的 CTMC 模型

表 1 中，状态 M_1 表示各副本正常运行，副本漏洞均未被利用成功，称为漏洞休眠态；状态 M_2、M_3、M_4 分别表示副本 α_0、α_1、α_2 被攻击成功，此时 SHRE 系统通过表决能够感知异常输出，阻断攻击，统称为异常感知态；状态 M_5 表示在状态 M_2、M_3、M_4 的基础上，又产生了新的被攻击成功副本，系统失去安全能力，称为系统失效态。安全场景如下。

表 1　TMR 结构 SHRE 系统的 CTMC 模型稳定状态

状态序号	含义
M_1	各副本正常运行，处于漏洞休眠状态
M_2	副本 α_0 被攻击成功
M_3	副本 α_1 被攻击成功
M_4	副本 α_2 被攻击成功
M_5	SHRE 系统失去安全能力

场景 4　受攻击副本有恢复能力的情况下，系统处于漏洞休眠态和异常感知态。TMR 结构、受攻击副本有恢复能力 SHRE 系统（记为 S_4）的安全性可以表示为

$$\mathrm{AP}_{S_4} = P_1 + P_2 + P_3 + P_4 \tag{10}$$

其中，P_1、P_2、P_3、P_4 为 CTMC 模型瞬时处于状态 M_1、M_2、M_3、M_4 的概率。

4.4　SHRE 安全性计算

S_1～S_3 结构的安全性计算，只需要将式(6)代入到式(7)～式(9)中即可。S_4 结构的安全性计算需要先求解 P_1、P_2、P_3、P_4，系统由状态 M_e 转移至当前状态 M_e 和下一状态 M_f 的转移概率分别记为 $\boldsymbol{P}^{ee}$ 和 $\boldsymbol{P}^{ef}$，因此状态之间的转移矩阵为

$$\boldsymbol{U} = \boldsymbol{P}^{ee} + \boldsymbol{P}^{ef}(1-\boldsymbol{P}^{ff})^{-1}\boldsymbol{P}^{fe} \tag{11}$$

由 $\boldsymbol{U}$ 构造 CTMC 模型转移速率矩阵[37]，状态 M_e 到状态 M_f 的转移速率可以表示为

$$q_{ef} = \begin{cases} \lim\limits_{\Delta t\to 0}\dfrac{u_{ef}(\Delta t)}{\Delta t}, e \neq f \\ \lim\limits_{\Delta t\to 0}\dfrac{u_{ef}(\Delta t)-1}{\Delta t}, e = f \end{cases} \tag{12}$$

$\boldsymbol{Q}$ 是以 q_{ef} 为元素的矩阵。概率向量 $\boldsymbol{P}(t) = (P_1(t), P_2(t), \cdots, P_5(t))$，其中 $P_e(t)$ 为系统处于状态 M_e 的瞬时概率，可得到微分方程

$$\begin{cases} \boldsymbol{P}'(t) = \boldsymbol{P}(t)\boldsymbol{Q} \\ \boldsymbol{P}(0) = (P_1(t), P_2(t), \cdots, P_5(t)) \end{cases} \tag{13}$$

根据 CTMC 模型状态转移方程的推导方法，TMR 结构 SHRE 的 CTMC 模型状态转移方程为

$$\begin{bmatrix}\dot{P}_1(t)\\ \dot{P}_2(t)\\ \dot{P}_3(t)\\ \dot{P}_4(t)\\ \dot{P}_5(t)\end{bmatrix}=\begin{bmatrix}-\lambda_0-\lambda_1-\lambda_2 & \mu_0 & \mu_1 & \mu_2 & 0\\ \lambda_0 & -\mu_0-\lambda_1-\lambda_2 & 0 & 0 & 0\\ \lambda_1 & 0 & -\mu_1-\lambda_0-\lambda_2 & 0 & 0\\ \lambda_2 & 0 & 0 & -\mu_2-\lambda_0-\lambda_1 & 0\\ 0 & \lambda_1+\lambda_2 & \lambda_0+\lambda_2 & \lambda_0+\lambda_1 & 0\end{bmatrix}\begin{bmatrix}P_1(t)\\ P_2(t)\\ P_3(t)\\ P_4(t)\\ P_5(t)\end{bmatrix} \tag{14}$$

为便于计算，AP_{S_1}、AP_{S_2}、AP_{S_3}、AP_{S_4} 中不妨设定[38-39]

$$\lambda_0=\lambda_1=\lambda_2=\lambda \tag{15}$$

$$\mu_0=\mu_1=\mu_2=\mu \tag{16}$$

相比于精心构造并实施一次网络攻击的速率，软件副本的恢复速率要快很多，其中 λ=1/4，μ=15、30、60、120。计算得到 TMR 结构 SHRE 的 CTMC 模型各状态的瞬时概率为

$$\begin{cases}P_{M_1}=\dfrac{k_0}{2k_2}\mathrm{e}^{-\frac{k_1t}{2}}-\dfrac{k_1}{2k_2}\mathrm{e}^{-\frac{k_0t}{2}}+1\\ P_{M_2}=\dfrac{k_0\left(\frac{k_1}{4\lambda}-1\right)}{2k_2}\mathrm{e}^{-\frac{k_1t}{2}}-\dfrac{k_1\left(\frac{k_0}{4\lambda}-1\right)}{2k_2}\mathrm{e}^{-\frac{k_0t}{2}}\\ P_{M_3}=\dfrac{k_0\left(\frac{k_1}{4\lambda}-1\right)}{2k_2}\mathrm{e}^{-\frac{k_1t}{2}}-\dfrac{k_1\left(\frac{k_0}{4\lambda}-1\right)}{2k_2}\mathrm{e}^{-\frac{k_0t}{2}}\\ P_{M_4}=\dfrac{k_0\left(\frac{k_1}{4\lambda}-1\right)}{2k_2}\mathrm{e}^{-\frac{k_1t}{2}}-\dfrac{k_1\left(\frac{k_0}{4\lambda}-1\right)}{2k_2}\mathrm{e}^{-\frac{k_0t}{2}}\\ P_{M_5}=\dfrac{k_0k_1}{24\lambda k_2}\left(\mathrm{e}^{-\frac{k_0t}{2}}-\mathrm{e}^{-\frac{k_1t}{2}}\right)\end{cases} \tag{17}$$

其中，

$$\begin{cases}k_0=5\lambda+\mu-(\lambda^2+10\lambda\mu+\mu^2)^{\frac{1}{2}}\\ k_1=5\lambda+\mu+(\lambda^2+10\lambda\mu+\mu^2)^{\frac{1}{2}}\\ k_2=(\lambda^2+10\lambda\mu+\mu^2)^{\frac{1}{2}}\end{cases} \tag{18}$$

因此，将式(17)和式(18)代入 S_4 结构的安全性计算式(10)中，设 μ=60，可得到系统瞬时处于漏洞休眠态、异常感知态、系统失效态的概率，计算结果如图 4 所示，纵轴表示系统处于曲线所代表状态的概率，横轴表示系统遭受攻击的时间。系统处于漏洞休眠态的概率随攻击时间的持续而下降，处于系统失效态的概率随攻击时间的持续而上升。

当 μ=15、30、60、120 时，漏洞休眠态、异常感知态、系统失效态的对比结果分别如图 5～图 7 所示，相应地纵轴表示在不同的副本恢复速率下系统分别处于漏洞休眠态、异常感知态、系统失效态的概率，横轴均表示系统遭受攻击的时间。系统的副本恢复速率越快，在持续攻击下系统失效越慢，反之，系统的副本恢复速率越慢，在持续攻击下系统失效越快。

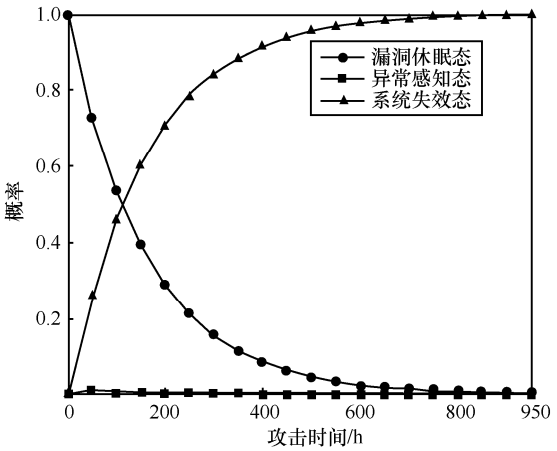


图 4　TMR 结构 SHRE 系统瞬时各状态整体变化曲线

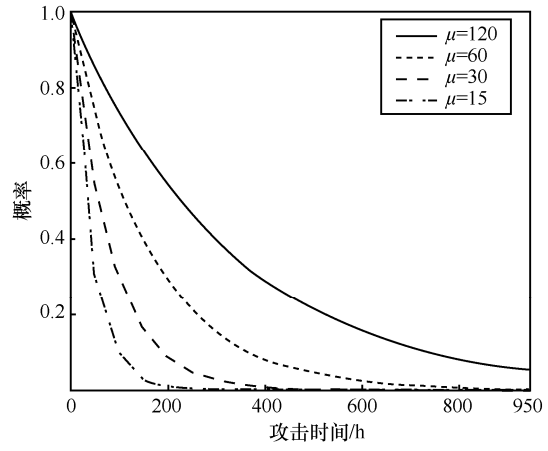


图 5　TMR 结构 SHRE 系统瞬时漏洞休眠态变化曲线

完成 S_4 结构的安全性计算分析后，令 λ=1/4，μ=60，计算 S_1～S_3 结构的安全性。S_0 结构和 S_1～S_4 结构的安全性结果如图 8 所示，纵轴表示软件单副本

及不同结构 SHRE 系统的安全性（即未被攻击成功）具备安全能力的概率，横轴表示系统遭受攻击的时间。

DMR 结构 SHRE 系统中任何单软件副本被攻击成功，系统通过表决感知到输出不一致后都会失去安全能力，通过观察 S_1 和 S_2 的结果曲线可知，DMR 结构 SHRE 系统的安全能力要低于单软件副本安全能力，这是由于系统感知异常后服务不可用造成的。但 SHRE 在感知异常后能阻断攻击结果输出或基于数据流的恶意控制。而单软件副本会遭受持续的信息泄露或恶意控制，比起服务的不可用，影响更严重。

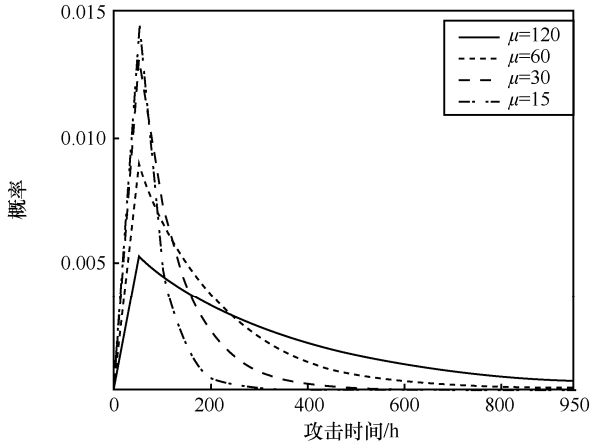


图 6　TMR 结构 SHRE 系统瞬时异常感知态变化曲线

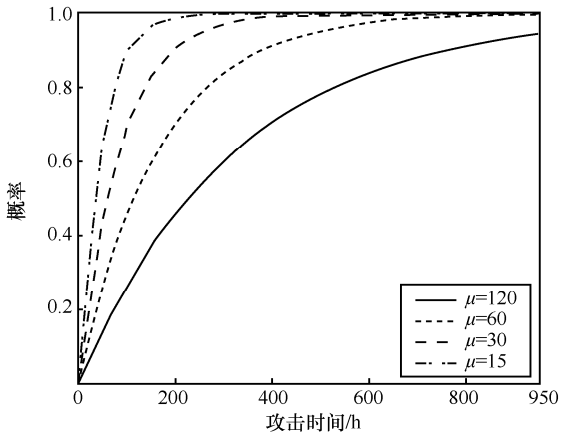


图 7　TMR 结构 SHRE 系统瞬时系统失效态变化曲线

通过观察 S_3 和 S_4 的结果曲线可知，TMR 结构 SHRE 系统的安全能力高于单软件副本安全能力，不仅可以阻断信息泄露或恶意控制，还能提高可用性，保证系统较高的安全能力。其中受攻击副本有恢复能力的 TMR 结构 SHRE 系统的安全性要远高于其他结构的安全性，是安全能力表现最好的 SHRE 系统结构。

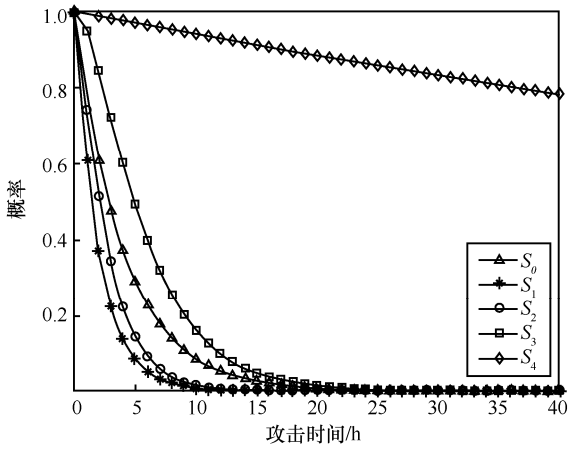


图 8　软件单副本与不同结构 SHRE 系统的安全性变化曲线

5　结束语

本文在研究 SHRE 原理和 SHRE 系统分类的基础上，引入 SHRE 系统的安全能力概念，总结包括 N 模冗余、I/O 操作模式以及受攻击软件副本恢复能力在内，影响系统安全能力的结构因素，分析不同结构 SHRE 系统下的安全场景。计算结果表明，受攻击副本有恢复能力的 TMR 结构 SHRE 系统的安全性最高，安全能力表现最好，并且缩短受攻击软件副本的恢复时间能够提高系统安全性。

参考文献：

[1] ZHANG Y G, VIN H, ALVISI L, et al. Heterogeneous networking: a new survivability paradigm[C]//Proceedings of The 2001 Workshop on New Security Paradigms. New York: ACM Press, 2001: 33-39.

[2] STAMP M. Risks of monoculture[J]. Communications of the ACM, 2004, 47(3): 120.

[3] CHEN Y S, CHEN P S. A software-based redundant execution programming model for transient fault detection and correction[C]//2016 45th International Conference on Parallel Processing Workshops (ICPPW). Piscataway: IEEE Press, 2016: 66-71.

[4] 吴斌, 高珑. 软件双冗余容错系统的容错能力和性能分析[J]. 计算机研究与发展, 2009, 46(z2): 129-136.
WU B, GAO L. Fault tolerance and performance analysis of software double redundant implemented hardware fault tolerance[J]. Journal of Computer Research and Development, 2009, 46(z2): 129-136.

[5] JUST J E, CORNWELL M. Review and analysis of synthetic diversity for breaking monocultures[C]//Proceedings of the 2004 ACM Workshop on Rapid Malcode. New York: ACM Press, 2004: 23-32.

[6] KOREN I, SU S. Reliability analysis of N-modular redundancy systems with intermittent and permanent faults[J]. IEEE Transactions on Computers, 1979, 28(7): 514-520.

[7] JEON H, ANNAVARAM M. Warped-DMR: light-weight error detection for GPGPU[C]//2012 45th Annual IEEE/ACM International Symposium on Microarchitecture. Piscataway: IEEE Press, 2012: 37-47.

[8] NEUMANN J V. Probabilistic logics and the synthesis of reliable organisms from unreliable components[J]. Automata Studies, 1956, 34: 43-99.

[9] 姚东, 张铮, 张高斐, 等. 多变体执行安全防御技术研究综述[J]. 信息安全学报, 2020, 5(5): 77-94.
YAO D, ZHANG Z, ZHANG G F, et al. A survey on multi-variant execution security defense technology[J]. Journal of Cyber Security, 2020, 5(5): 77-94.

[10] REINHARDT S K, MUKHERJEE S S. Transient fault detection via simultaneous multithreading[C]//Proceedings of 27th International Symposium on Computer Architecture. Piscataway: IEEE Press, 2000: 25-36.

[11] 仝青, 张铮, 邬江兴. 基于软硬件多样性的主动防御技术[J]. 信息安全学报, 2017, 2(1): 1-12.
TONG Q, ZHANG Z, WU J X. The active defense technology based on the software/hardware diversity[J]. Journal of Cyber Security, 2017, 2(1): 1-12.

[12] 马海龙, 伊鹏, 江逸茗, 等. 基于动态异构冗余机制的路由器拟态防御体系结构[J]. 信息安全学报, 2017, 2(1): 29-42.
MA H L, YI P, JIANG Y M, et al. Dynamic heterogeneous redundancy based router architecture with mimic defenses[J]. Journal of Cyber Security, 2017, 2(1): 29-42.

[13] 张铮, 马博林, 邬江兴. web 服务器拟态防御原理验证系统测试与分析[J]. 信息安全学报, 2017, 2(1): 13-28.
ZHANG Z, MA B L, WU J X. The test and analysis of prototype of mimic defense in web servers[J]. Journal of Cyber Security, 2017, 2(1): 13-28.

[14] 宋克, 刘勤让, 魏帅, 等. 基于拟态防御的以太网交换机内生安全体系结构[J]. 通信学报, 2020, 41(5): 18-26.
SONG K, LIU Q R, WEI S, et al. Endogenous security architecture of Ethernet switch based on mimic defense[J]. Journal on Communications, 2020, 41(5): 18-26.

[15] 马博林, 张铮, 陈源, 等. 基于指令集随机化的抗代码注入攻击方法[J]. 信息安全学报, 2020, 5(4): 30-43.
MA B L, ZHANG Z, CHEN Y, et al. The defense method for code-injection attacks based on instruction set randomization[J]. Journal of Cyber Security, 2020, 5(4): 30-43.

[16] 张宇嘉, 庞建民, 张铮, 等. 基于软件多样化的拟态安全防御策略[J]. 计算机科学, 2018, 45(2): 215-221.
ZHANG Y J, PANG J M, ZHANG Z, et al. Mimic security defence strategy based on software diversity[J]. Computer Science, 2018, 45(2): 215-221.

[17] JUNOD P, RINALDINI J, WEHRLI J, et al. Obfuscator-LLVM: software protection for the masses[C]//2015 IEEE/ACM 1st International Workshop on Software Protection. Piscataway: IEEE Press, 2015: 3-9.

[18] 姚东, 张铮, 张高斐, 等. MVX-CFI: 一种实用的软件安全主动防御架构[J]. 信息安全学报, 2020, 5(4): 44-54.
YAO D, ZHANG Z, ZHANG G F, et al. MVX-CFI: a practical active defense framework for software security[J]. Journal of Cyber Security, 2020, 5(4): 44-54.

[19] FRANZ M. E unibus pluram: massive-scale software diversity as a defense mechanism[C]//Proceedings of the 2010 New Security Paradigms Workshop. [S.n.:s.l.], 2010: 7-16.

[20] LEVITIN G, XING L D, XIANG Y P. Co-residence data theft attacks on N-version programming-based cloud services with task cancelation[J]. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2020, PP(99): 1-10.

[21] COX B, EVANS D, FILIPI A, et al. N-Variant systems: a secret less framework for security through diversity[C]//USENIX Security Symposium. Berkeley: USENIX Association, 2006: 105-120.

[22] CAVALLARO L. Comprehensive memory error protection via diversity and taint-tracking[D]. Milan: University of Milan, 2007.

[23] SALAMAT B, JACKSON T, GAL A, et al. Orchestra: intrusion detection using parallel execution and monitoring of program variants in user-space[C]//Proceedings of the fourth ACM European Conference on Computer Systems. New York: ACM Press, 2009: 33-46.

[24] VOLCKAERT S, DE SUTTER B, DE BAETS T, et al. GHUMVEE: efficient, effective, and flexible replication[C]//Foundations and Practice of Security. Berlin: Springer, 2013: 261-277.

[25] CAO M C, HOU X T, WANG T, et al. Different is good: detecting the use of uninitialized variables through differential replay[C]// Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2019: 1883-1897.

[26] ZHANG Y, LEE J W, JOHNSON N P, et al. DAFT: decoupled acyclic fault tolerance[J]. International Journal of Parallel Programming, 2012, 40(1): 118-140.

[27] REIS G A, CHANG J, VACHHARAJANI N, et al. SWIFT: software implemented fault tolerance[C]//International Symposium on Code Generation and Optimization. Piscataway: IEEE Press, 2005: 243-254.

[28] THATI V B, VANKEIRSBILCK J, PISSOORT D, et al. Instruction level duplication and comparison for data error detection: a first experiment[C]//2018 IEEE XXVII International Scientific Conference Electronics. Piscataway: IEEE Press, 2018: 1-4.

[29] CHIELLE E, KASTENSMIDT F L, CUENCA-ASENSI S. Overhead reduction in data-flow software-based fault tolerance techniques[C]//FPGAs and Parallel Architectures for Aerospace Applications. [S.n.:s.l.], 2016: 279-291.

[30] VOLCKAERT S, COPPENS B, VOULIMENEAS A, et al. Secure and efficient application monitoring and replication[C]//2016 USENIX Annual Technical Conference. Berkeley: USENIX Association, 2016: 167-179.

[31] 潘传幸, 张铮, 马博林, 等. 面向进程控制流劫持攻击的拟态防御方法[J]. 通信学报, 2021, 42(1): 37-47.
PAN C X, ZHANG Z, MA B L, et al. Method against process control-flow hijacking based on mimic defense[J]. Journal on Communications, 2021, 42(1): 37-47.

[32] 方滨兴. 定义网络空间安全[J]. 网络与信息安全学报, 2018, 4(1): 1-5.
FANG B X. Define cyberspace security[J]. Chinese Journal of Network and Information Security, 2018, 4(1): 1-5.

[33] HUMPHREY W S. Personal software process (PSP)[M]. New York: John Wiley & Sons, Inc., 2002.

[34] HOSEK P, CADAR C. VARAN the unbelievable: an efficient N-version execution framework[C]//ACM Special Interest Group on Programming Languages. New York: ACM Press, 2015: 339-353.

[35] LU K. Securing software systems by preventing information leaks[D]. Atlanta: Georgia Institute of Technology, 2017.

[36] NOVARK G, BERGER E D, ZORN B G. Exterminator: automatically correcting memory errors with high probability[J]. Communications of the ACM, 2008, 51(12): 87-95.

[37] 任权, 邬江兴, 贺磊. 基于 GSPN 的拟态 DNS 构造策略研究[J]. 信息安全学报, 2019, 4(2): 37-52.
REN Q, WU J X, HE L. Research on mimic DNS architectural strategy based on generalized stochastic petri net[J]. Journal of Cyber Security, 2019, 4(2): 37-52.

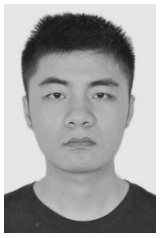
[38] SHI J, MENG Y X, WANG S P, et al. Reliability and safety analysis of redundant vehicle management computer system[J]. Chinese Journal of Aeronautics, 2013, 26(5): 1290-1302.

[39] WANG S P, CUI X Y, SHI J, et al. Modeling of reliability and performance assessment of a dissimilar redundancy actuation system with failure monitoring[J]. Chinese Journal of Aeronautics, 2016, 29(3): 799-813.

[作者简介]

马博林（1993− ），男，河北吴桥人，信息工程大学博士生，主要研究方向为网络空间安全。

张铮（1976− ），男，湖北黄梅人，博士，信息工程大学副教授，主要研究方向为网络空间安全、高性能计算。



任权（1994− ），男，湖南常德人，信息工程大学博士生，主要研究方向为新型网络体系结构。

张高斐（1996− ），男，河南许昌人，网络通信与安全紫金山实验室工程师，主要研究方向为网络空间安全。

邬江兴（1953− ），男，浙江嘉兴人，中国工程院院士，信息工程大学教授，主要研究方向为通信与信息系统、网络空间安全。