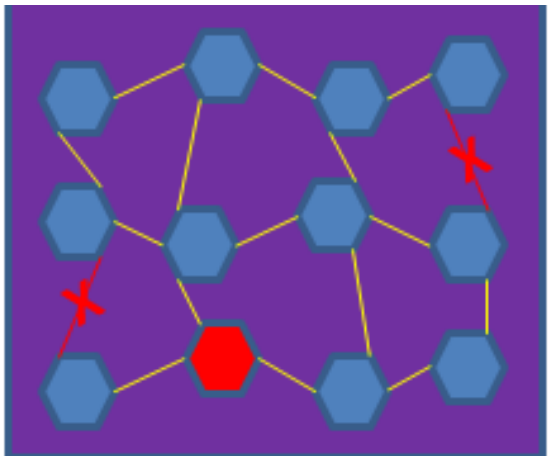
xRA Technical Overview



xRA Abstract:

After more than 30 years of academic research the principles of true state preserving Fault Tolerance (FT) have been applied to create the E**x**tended **R**esilience **A**rchitecture (**xRA**), a software architecture that improves the resilience, reliability, availability, and safety of mission critical applications.

xRA is a framework that creates distributed application replicas without a single point of control. It uses FT consensus management concepts to detect execution aberrations and continue application execution - before harm occurs - despite hardware failures or software anomalies.

By the very nature of its distributed independent replica operation, featuring both spatial and temporal diversity, xRA creates a high entropy randomized execution timing barrier to help defend against adversaries who try to change the execution path of applications.

xRA has been implemented completely in software. Its application focus, independent of the operating system, means it can be adapted to the broadest range of hardware and software platforms and will also complement many existing cybersecurity solutions. xRA may be used to implement solutions on existing platforms, from embedded IoT controllers, to edge servers, to cloud hosts and enterprise servers, even across heterogeneous platform configurations; on bare metal, in containers, or using virtual machines.

Architecture:

xRA can be thought of as an application comparison engine that instantiates multiple application replicas and controls their state space execution trajectory to ensure that the application performs as intended. xRA accomplishes this via two mutually enabling mechanisms: it enforces determinism of the application while simultaneously detecting inconsistencies (divergence) among the replicas. Determinism on one hand enables checking of consistency across replicas, while divergence detection enables the preservation of application determinism and correctness by removing the diverging replica. This "deadly embrace" combination of determinism enforcement and divergence detection makes the detection mechanism extremely sensitive to divergence and therefore an excellent indicator of a variance from the application's intended state space trajectory.

Core Mechanisms:

Distributed Replication (See **Figure 1**) – An xRA framework employs multiple replicas of the code to be protected. Each replica contains the complete application state, from which replicas unaffected by a failure can continue executing as if no failure occurred. Replicas operate in fully distributed mode, without any central deciding entity. The user may configure the number of replicas depending on mission requirements. Replicas exchange application replica state information aimed primarily at enabling determinism enforcement and consistency checking. Inter-replica communications are protected via secure channel mechanisms.

How Distributed Replication is used with xRA:

- xRA creates *n* (currently *n* = 2 to 31) replicas that appear to the user as a single logical application. (note: 2 replicas will detect failure; 3 or more replicas are needed for continued FT operation)
- xRA helps users create tools to deploy replicated xRA applications on designated hosts. (note: hosts may be located anywhere - data centers, cloud, edge, or any combination)
- xRA applications are created by:
  - modifying an existing application,
  - creating an application from scratch, or
  - creating an xRA service for an application that cannot be modified (ex: open source)

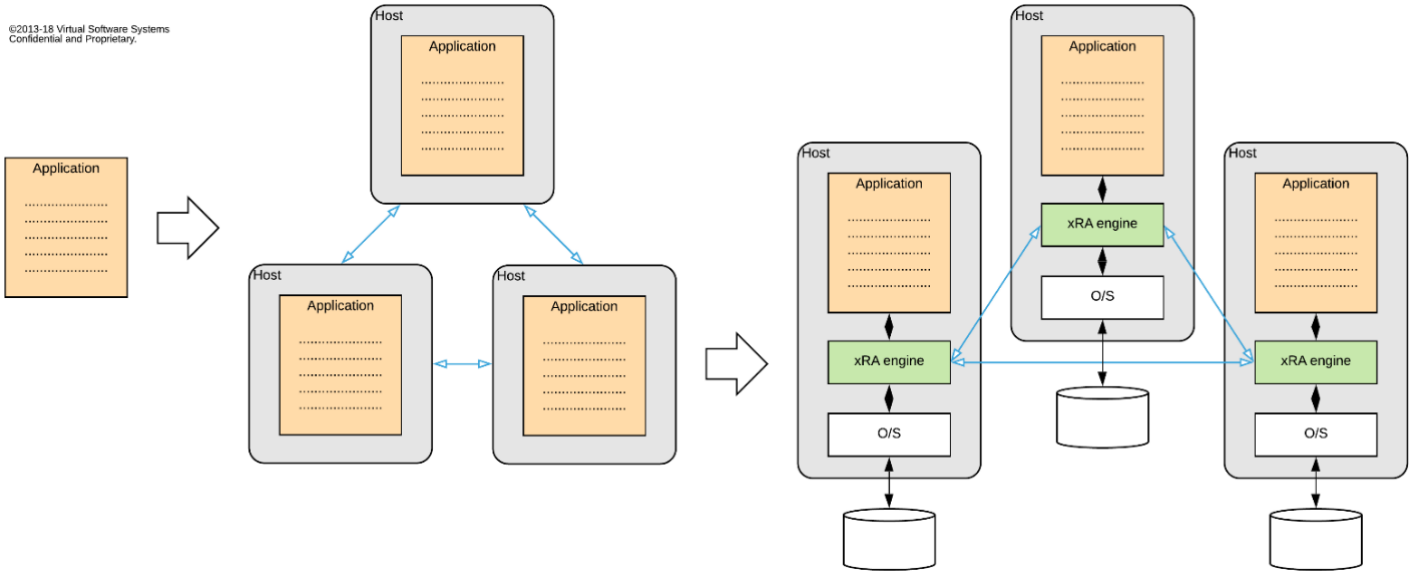


Figure 1 – Distributed Replication

Thread-Level Determinism – Determinism of an xRA application is enforced by requiring that critical system calls, thread creation and destruction, inter-thread concurrency control, and I/O, pass through the xRA software which is currently implemented as an API library. (Thread-Level Determinism means that each replicated thread will go through exactly the same sequence of xRA calls, regardless of I/O and inter-thread concurrency control)

Cross-replica entropy – Each replicated application thread is scheduled by its host operating system at different times across the replicas. xRA allows threads to assume states relative to one another that will differ markedly from replica to replica at any one time. The resulting entropy injection makes it harder for an adversary to successfully coordinate an undetected attack across the replicas.

Consistency Checking (See **Figure 2**) – Each xRA API call effectively separates blocks of application code into "islands of resilience" and executes them atomically from the perspective of resilience. After each block is executed its state is deterministic and comparable. xRA exchanges and checks each block execution state for consistency of both code and data across replicas before returning control to the calling application thread. A code or data divergence occurs when a consistency check fails. Given the deterministic properties of the system, divergence can only be caused by influences external to the intended function of the application: hardware faults, intrusions, pathological system function (ex: halting), or poor coding.

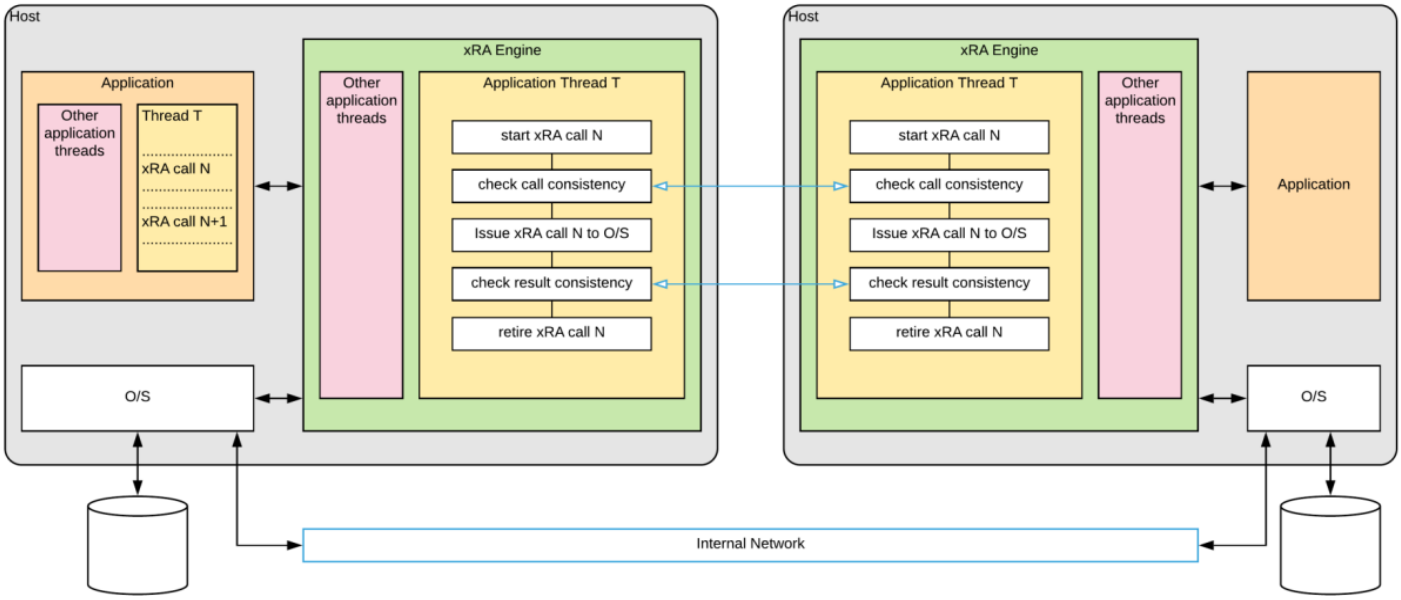


Figure 2 – Consistency Checking Between Two Replicas

Fault Handling – When a divergence from the intended application state trajectory is detected, the xRA Fault Handler runs real time diagnostics to determine which replica(s) contains the source of the divergence. The Fault Handler then reconfigures the system by logically removing that replica from the set, thus preserving thread-level determinism. Like all other xRA replicated mechanisms, the Fault Handler is also fully distributed and operates without any centralized voting entity.

Properties:

Replication – enables the ability to provide continued processing in the face of a fault, without loss of state or loss of "in-flight" (transaction) data. High availability applications can be developed to take advantage of this capability. Replication removes single points of failure: failure of a single replica or a single inter-replica communication channel does not result in overall application failure because the surviving replicas hold the full application state from which they can resume normal operation.

Threat agnostic protection – the mechanisms that enable divergence detection do not have nor need to have any prior knowledge of what caused the divergence.

Platform independence - Since the xRA API library is written at the application level, it can be ported to virtually all computing platforms that support deterministic thread scheduling. Currently it supports Windows and Linux.

Use Case Abstracts:

As an architecture, with a focus solely on protecting the application at the user level, xRA has the flexibility to be applied across many different hardware and software platforms as well as many different user applications. Below is a list of some example xRA use cases that are further described in another document, available by contacting VS^2.

Process Control Systems Protection - xRA process control applications detect and isolate control system failures and enable continued control operations despite a replica failure.

Autonomous, Fault Tolerant Control Systems - xRA applications distribute control functions to multiple control elements ("replicas") that operate independently yet appear to non-xRA software or application users as one logical unit. xRA replicas may be spatially distributed ensuring that operating control is maintained despite any site failure or disruption, without loss of transactions or operating 'state'.

Simultaneous Multi-platform Software Testing - xRA-enabled testing simultaneously tests a software release on multiple heterogeneous platforms – without requiring the involvement of experienced software developers until divergences have been detected and a root cause analysis is required.

Hardware Testing - xRA-enabled testing of electronic circuit boards can be used to test hardware only or the hardware under operating conditions with software. Testing hardware as it executes software enables quality control personnel to test for and expose problems that might otherwise not be detected until field deployment.

Conclusions:

xRA brings new dimensions of resilience, reliability, availability, and safety to mission- and life-critical applications.

xRA is an innovative framework based on the proven principles of Fault Tolerance. By creating distributed application replicas without a single point of control, xRA enables continued applications' operation despite hardware failures and software anomalies.

xRA's application focus means it can be adapted to the broadest range of hardware platforms and operating systems. It can be implemented across heterogeneous platform environments on the edge, in the cloud, or in the data center as well as in embedded systems such as those in the IoT. xRA provides new levels of both physical safety and security - through high availability, fault detection, and continued operation - before harm occurs.