

GOALS

- Develop a low-cost, low-overhead hardware security engine to protect applications that compute critical data from malicious attacks and transient errors.
- Lower the barriers to applying the security engine. Specifically, achieve low runtime overhead, low hardware resource overhead, high source compatibility, and high binary compatibility.
- Apply the security engine on power grid applications such as DNP3 Client/Server to ensure the secure execution of these applications in the presence of memory corruption attacks and accidental errors.

FUNDAMENTAL QUESTIONS/CHALLENGES

- How to prevent attacks from different entry points (outsiders, normal users, or insiders).
- How to enforce both spatial memory safety and temporal memory safety so as to provide high detection coverage for memory corruption attacks.
- How to efficiently (with small overhead) transmit monitoring data collected from the main processor to the security engine.
- How to asynchronously check the memory safety without interrupting the normal execution of a program.
- How to apply the protection technique on existing applications so that minimum effort (on the developer's site) is required to convert unprotected programs into protected programs.
- How to design the protection scheme so that unprotected code (e.g., third-party libraries) can interoperate with protected code transparently.

RESEARCH PLAN

- The **AHEMS** (*Asynchronously Hardware-Enforced Memory Safety*) Framework is proposed to protect applications from memory corruption attacks by enforcing *spatial* and *temporal* memory safety. AHEMS has two major parts:
 - **Source Code Instrumentation:** The source code of a program is instrumented with `alloc` and `dealloc` instructions to establish the interface between the program and the hardware security engine.
 - **Hardware Security Engine:** The security engine receives the memory events from the runtime monitor, checks the memory safety *asynchronously* (i.e., does not stop the main processor), and raises exceptions if the memory events indicate violation of memory safety.

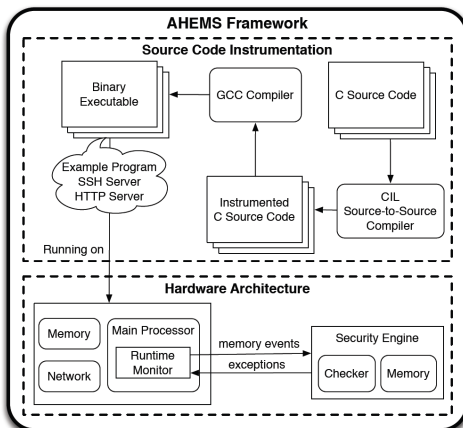


Figure 1: Architecture of AHEMS Framework

RESEARCH RESULTS

- The AHEMS prototype achieves high coverage (passing 676 out of 677 tests) with as low as 10.6% runtime overhead, and outperforms four other state-of-the-art approaches.

Table 1: Detection Coverage of AHEMS on Juliet Test Suite

Spatial Memory Errors			
CWE No.	Description	Tested	Detected
CWE121	Stack-based Buffer Overflow	209	208
CWE122	Heap-based Buffer Overflow	18	18
CWE124	Buffer Underwrite	102	102
CWE126	Buffer Overread	145	145
CWE127	Buffer Underread	33	33
CWE588	Attempt to Access Child of Non-structure Pointer	34	34
CWE680	Integer Overflow to Buffer Overflow	38	38
CWE761	Free Pointer Not at Start of Buffer	38	38
Subtotal		617	616
Temporal Memory Errors			
CWE No.	Description	Tested	Detected
CWE415	Double-free	38	38
CWE416	Use-after-free	20	20
CWE562	Return of Stack Variable Address	2	2
Subtotal		60	60
Total		677	676

Table 2: Runtime Overhead of AHEMS against Other Approaches

Programs	AHEMS	Mudflap	Softbound+CETS	SAFECode	AddressSanitizer
bh	0.2%	13655.2%	Compiler error	Runtime error	41.9%
bisort	38.4%	3114%	341.1%	154.3%	74.7%
em3d	10.5%	705.0%	473.0%	192.1%	89.5%
health	17%	13343.9%	737.8%	943.2%	361.5%
mst	4.3%	1169.2%	395.1%	644.1%	92.2%
perimeter	22.2%	32317.8%	443.1%	212.7%	142.8%
power	0.0%	263.9%	1.2%	1.0%	2.7%
treeadd	8.6%	106008.1%	448.1%	518.8%	398.7%
tsp	0.0%	1404.9%	206.9%	57.5%	76.8%
voronoi	4.6%	2224.2%	False alarm	Runtime error	156.5%
Average	10.6%	17420.6%	380.8%	340.5%	143.7%

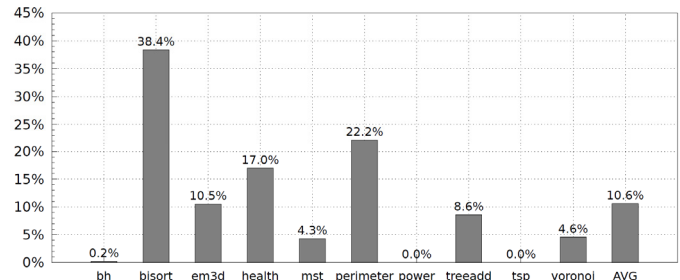


Figure 2: Runtime Overhead of AHEMS on Olden Benchmarks

BROADER IMPACT

- AHEMS can be applied to protect a broad spectrum of applications written in C.
- AHEMS architecture can be extended to support different types of monitoring, such as checking of the control flow integrity of a program.
- AHEMS essentially eliminates notorious attacks, such as buffer overflow or dangling-pointer problems.

INTERACTION WITH OTHER PROJECTS

- We utilized the methods and tools developed by the activity "Testbed-Driven Assessment" to set up our experimental environment.

FUTURE EFFORTS

- Develop a cache system in the security engine to further speed up the performance of memory safety checking.
- Embed AHEMS into power grid substation devices such as security gateways or data aggregators so that critical applications running on those devices are protected.
- Extend AHEMS to check control flow integrity.