/* OptDyn */

# Open Hardware Security Framework

A **MUST** for Mission Critical IoT Applications

**By: Alex Karasulu**
CEO/Founder, OptDyn

**Professor Miguel Diogenes Matrakas**
Manager of Celtab, Latin American Center of Open Technologies
Itaipu Technological Park

**22 March 2018**

# Projects Overview

Mission critical industrial applications, especially those involving the Internet of Things (IoT) require hardware based security capabilities above and beyond the base trusted computing platform. The suite of hardware projects in this proposal fill in these gaps to provide an ultra secure computing environment specifically needed for IoT gateways[1] in all mission critical sectors: i.e. national power grids, defense, aerospace, and financial sectors. Furthermore, the independent, and open nature of these FOSSH projects protect governments and institutions from all kinds of malicious actors including state sponsored cyber warfare programs.

## Rationale and Drivers

With all its inspiring advantages, the Internet of Things (IoT), also brings legitimate security concerns with increased device connectivity. Security experts have warned the U.S. congress of the consequences ("the serious risk to life and property") of the increasing numbers of poorly secured devices on the Internet of Things [2]. These concerns increase dramatically, especially as the liabilities resulting from system compromises increase. It is reasonable to presume that many mission critical sectors will indefinitely defer or completely avoid any IoT rollout until absolutely certain that IoT security levels approach that of present day closed-loop (unconnected) systems.

The latest and most promising advances in the primitives used to build intrusion detection and prevention systems require fast hardware implementations for their feasible applicability. Software based approaches introduce too much latency and slow down processing to the point where systems become absolutely unusable. These techniques involve line rate stream (fuzzy) scanning for polymorphic malware and real time introspection into the processing system that trap malicious operations during execution. Respectively, the techniques referred to are context triggered piecewise hashing (CTPH), and dynamic information flow tracking (DIFT). Subprojects elaborate on how these critical primitives for intrusion detection and prevention systems could be feasibly implemented using hybrid processing systems (SoCs) with FPGA material on the same chip.

The timing for this hardware security framework proposal is optimal. ARM recently announced and published its Platform Security Architecture (PSA) which now provides even more robust foundations to support the higher level security functions proposed here. Intel started shipping Xeon processors with FPGA fabric on the same die to support hardware acceleration in the data center and consequently in the cloud. The entire value chain from IoT devices to cloud applications can now be protected using the same framework across multiple architectures.

---

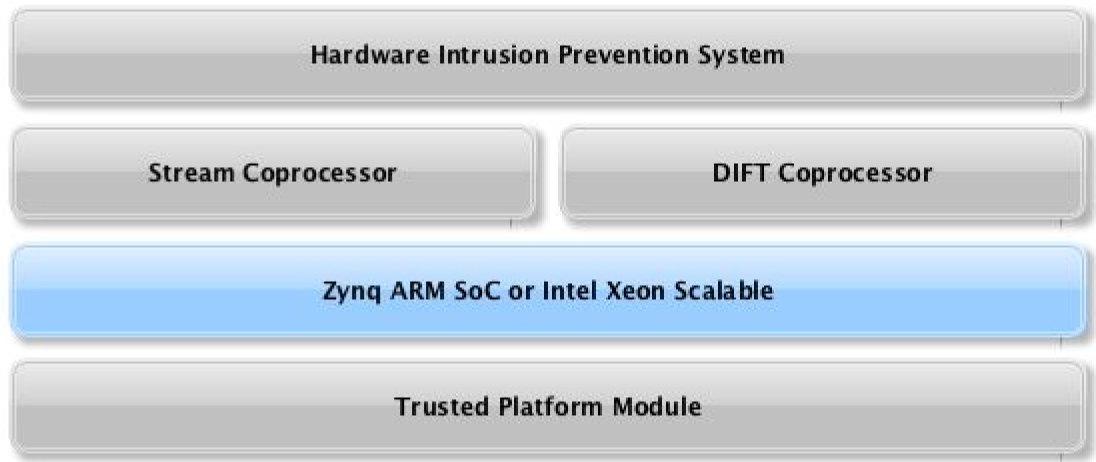
[1] The framework can and should also be applied to mission critical server systems in the data center. This is possible now due to the availability of Intel's Xeon+FPGA processors and Amazon's F1 FPGA instances.

[2] Security experts warn congress that the Internet of Things could kill people

Without these enhanced hardware security features to protect systems, organizations providing mission critical services and products will delay or abandon IoT efforts. At best, their IoT agenda will remain a fringe research topic or a talking point for marketing presentations.

## Projects and Relationships

The diagram below shows the high level components of the hardware security architecture in gray. The blue block represents the FPGA based platforms on which components will be developed for (minus the TPM module):




The dependency relationships are top down. For example the hardware IPS depends on the Stream Coprocessor and the DIFT Coprocessor. Both these components depend on their underlying platform which in turn depends on the TPM used.

### Implementation Platforms

There are two primary platforms: an ARM based embedded system platform for low power IoT gateways in the field, and a Xeon based server platform for cloud applications in data centers. Both platforms carry programmable logic cells on the same die in conjunction with hard processor cores with debugging interfaces to introspect processing without impacting performance. Both also provide extensive security primitives to simplify and reduce the effort to implement the higher layers. The ARM based Xilinx Zynq SoC provides the ideal power to processing ratios for IoT gateways, while Xeon Scalable is perfect for HPC and cloud workloads in the data center. Both platforms cover the full value chain from IoT gateway to the IoT based cloud applications that leverage them.

## Trusted Platform Module

This is an optional subproject to locally manufacture an open and discrete TPM device that implements TPM functionality, and nothing else, in its own tamper proof semiconductor package. Although several commercial TPM device manufacturers already exist, every national IT security effort should require local manufacturing under government scrutiny to prevent state actors from compromising TPM endorsement keys which are generated and burned into the silicon during manufacturing. If this key is compromised the entire system can be compromised.

If local commercial manufacturers already exist, then there's no need for implementing this project. Simply auditing their processes would suffice. If a local provider does not exist, this project should be started immediately. To distribute risk and lower costs multiple countries can combine resources to produce an open project or support an existing open TPM project such as the OpenTPM Project.

**Integrated TPMs**

---

Intel provides its own integrated TPM inside some processor models often referred to as Intel Platform Trust Technology (PTT). The endorsement key pair's private key is burned into the processor die during the manufacturing process. The private key should never be collected or recorded. With ever more corporate cooperation with national security agencies, there's no way to determine if these keys are kept to compromise foreign systems for intelligence gathering purposes. If these keys are compromised, then even non-state actors could potentially tamper with mission critical systems.

At a minimum, the security framework **MUST** provide a TPM interface which can switch the underlying implementation based on the availability of TPM devices. The TPM interface should prefer external discrete TPM implementations if both an integrated and discrete TPM implementation are present on the system. Obviously any TPM is better than no TPM, so the facade should fall back to using an integrated TPM if that is the only available option.

## Other Projects

The three other topmost projects, the hardware IPS, Stream Coprocessor, and the DIFT Coprocessor, are described in their own dedicated sections to follow. They're the primary components which add the enhanced security capabilities to the primitives offered by the underlying platforms.

# Stream Coprocessor

Pattern matching is a fundamental operation required by almost every computing discipline. It is the basis to lexical analysis and parsing which is needed for virtually everything from protocols, to file formats, to languages and even intrusion detection and prevention systems.

> ⓘ **Any slight advance or advantage in stream pattern matching has the ability to dramatically impact all other areas that depend on it.**

## Latency and Software

It is no longer sufficient to find patterns in stationary data once it comes to rest. Patterns in flowing streams must be matched while data is in motion, and this must occur at line rate to prevent the introduction of latency. Sometimes streams in transit should be trapped or dropped to prevent security breaches before malware enters a system or before unauthorized sensitive information can exit from it.

I/O rates have been steadily increasing to the point where software based solutions simply cannot keep up and they unnecessarily overload the main system processing units. Data marshalling, serialization and deserialization occurs all the time and up to 30% of the CPU cost is consumed with these operations. Data marshalling over high speed streams could easily overwhelm any software based solution. Significant latency is introduced by software at the mercy of the kernel scheduler under load.

## Hardware Investment

It is very expensive to solve a problem using hardware. Before making the decision to do so you have to consider the return on the investment. It is wise to get answers to the following questions:

- Will it amortize to recover the investment?
- Will it solve problems when software no longer can?
- Is it applicable to a wide range of problems, so its utility and ROI can justify the investment over time?

Traditional processors are generic work pistons optimized for branch prediction logic used to execute procedural code. Processors are not optimized to filter or match patterns against data streams. There's at least two orders of magnitude of performance lost by executing software on generic processors to match patterns in data streams versus using custom pattern matching hardware. The processor has to execute program instructions, which in turn loads data from the stream, performs comparisons, and switches state to continue processing inputs. That's a lot of layers to match a pattern in a low level data stream.

**A pattern matching engine is an ideal candidate for using a dedicated coprocessor architected specifically to provide the common and ubiquitous function of pattern matching over data streams.**

Increasing I/O rates make it impossible to use software only solutions without introducing unpredictable amounts of latency. Processes share time on the CPU using context switching and this could prevent the program from being able to process I/O in time to prevent delays. This is a major problem with networking applications and real time systems. This is also the reason why most enterprise switches and routers use FPGA hardware instead of software only solutions.

Answers to the key hardware acceleration questions are all affirmative for the case of a stream pattern matching coprocessor. It is well worth the investment. In fact, it's perhaps a necessity considering the line rates and the inability to keep up with them using software. This use case for hardware acceleration is an ideal example of a focused optimization to a very common problem.

**Hardware based I/O throttling could also be performed by the stream processor while pattern matching on data streams. This is an important capability especially for multi-tenancy in cloud systems. Software unlike hardware may not be able to react in a timely fashion due to context switching and ensure tenants properly receive their share of network and disk bandwidth. Bandwidth should be treated like any other resource. It should be throttled across cloud tenants to prevent deprivation in the presence of greedy consumers.**

## Applications

The Stream Coprocessor could be applied to several problems across a wide range of disciplines and industries. We cannot possibly envision them all. However there are some sweet spots that constantly involve stream processing and pattern matching. Computing areas that benefit most are networking, security, cloud, storage and as a consequence big data analytics systems. The accelerator can even be used to hardware accelerate UNIX tools like grep and AWK.

**Say goodbye to the data diode or air gap fiasco**

Many sectors use data diodes and airgap systems to control data flows from connected networks to sensitive mission critical systems. These tools are heavily used and are still required by the defense industry: it's a NATO requirement for mission critical defense systems.

In combination with the intrusion prevention system proposal, the coprocessor has the ability to detect and stop the flow of malware across systems and storage devices in real-time. It obsoletes the data diode and the air gap used in mission critical systems. The coprocessor effectively makes the packet the minimum unit of data trapped for approval by an air gap.

The coprocessor is ideal for detecting and indexing content in high end flash storage devices such as Solid State Drives (SSD) and Open Channel Solid State Drives. When embedded into the device, the coprocessor can automatically maintain indices into data blocks without involving the main processing system. Storage devices themselves can conduct primitive search queries in parallel to push indexing and search activities deep down into peripheral storage instead of shuffling data back and forth between the main processing system across peripheral connect buses. The application of the stream coprocessor will have a profound impact on the performance of big data systems. The overwhelming IoT data tsunami resulting from billions of connected devices requires big data analytics acceleration and analytics on data in motion (data streams).

## How does it work?

The primary output of this project is a hardware based stream pattern matching coprocessor. The heart of the coprocessor contains an engine based on a binary adaptation of the Aho-Corasick string matching algorithm. The coprocessor context switches between streams based on the availability of I/O. Production rules in Backus Naur form use regular expressions to define complex patterns in streams. A paged state machine transition table enables the application of a limitless number of patterns on any given stream.

The coprocessor should be available on both server side big iron Intel platforms and embedded systems serving as IoT gateways on ARM. It will be used in conjunction with the DIFT Coprocessor to implement the Hardware Based Intrusion Prevention System. All components will be designed to operate on both ARM and Xeon FPGA based PS/PL systems.

### Aho-Corasick Algorithm

The Aho-Corasick string pattern matching algorithm is used by the command line UNIX grep command. It is also used by the AWK programming language (also invented by Alfred Aho, alongside Peter Weinberger, and Brian Kernighan at Bell Labs).

**AWK essentially adds procedural logic to combine the application of regular expressions on streams of input. It glues regular expressions to build more complex stream processing systems. It is the procedural equivalent of declarative production rules.**

A binary version of the Aho-Corasick string pattern matching algorithm will be implemented in hardware as the engine forming the core of the coprocessor. The engine executes Mealy State Machines in (a page-able) tabular form while consuming byte stream input. These state machines represent regular expressions specifying binary patterns. Higher level translation tables apply character sets or other kinds of encodings to the binary representations executed by the engine. The engine uses registers to track stream position and the current state of the state machine executed on the stream much like a traditional processor's program counter.

## Stream Context Switching

To be of any real world use, the coprocessor should be able to concurrently service an unlimited number of data streams without performance degradation. As long as resources permit there should be no limit to the number of streams that could be multiplexed using time slices on the coprocessor. Any kind of stream time scheduling should consider I/O availability to prevent the wasteful allocation of the coprocessor on streams without available I/O.

> **So long as the I/O does not overwhelm the bandwidth limitations of the coprocessor, there should be no significant decline in stream processing performance as the number of streams increase.**

Before switching from one stream to another, the coprocessor must save the state of regular expressions on streams being processed. It either loads or initializes (if new) the context of the next stream with available I/O into its registers to begin to process regular expressions on it. This is very similar to the way a normal processor deals with a program context switch.

## Complex Pattern Combinations

If limited by the number of patterns it could match on a data stream, the coprocessor would be ineffective in most real world use cases. The same is true when it comes to combining patterns to represent complex patterns. There should be enough flexibility to define language productions if desired. That is, if desired, it should be possible to implement complex lexers and parsers.

> ⚠ **There should be no limit to the number and combination of patterns matched by the engine on any given stream.**

The engine will execute state machines representing patterns or combinations of patterns which can be logically combined. These pattern combinations will be specified using productions in Backus Naur form: extended or augmented may be supported. This will provide sufficient flexibility to match any language, protocol or format in a stream using a grammar.

### Paged Transition Tables

Patterns (the regular expressions and their combinations) are compiled into deterministic finite state machines. A nondeterministic finite automaton is first generated using Thompson's Construction Algorithm, followed by a powerset construction and reduction. The final representation is a storage efficient sparse state transition matrix in a page-able table.

The engine accesses state transition data stored in a table backed by block memory in the FPGA. This allows for very quick access however BRAM is very limited, and can best be used as a primary cache for state transition data. The number of patterns matched against a stream would be extremely limited if BRAM was the only state transition store.

> ⚠ **There should be little degradation in stream processing performance as the number of patterns matched against a stream increases. And there should be no limit to the number of patterns that could be used to match against a stream.**

This is why the BRAM must act as the primary cache. On a cache miss, state transition tables will be paged in from the secondary cache, which will most likely be processing system main memory. The primary storage for pattern state transition tables will be non-volatile storage (disk or flash). A simple indexed file format (a primitive b+tree db) on the non-volatile storage may be needed for random access to quickly lookup the needed transition tables.

## Linux Integration and Ecosystem

The coprocessor functionality is exposed to user space processes through system calls. Device handlers and kernel modules provide the underlying infrastructure to interface with the coprocessor in the PL. The usage semantics would be very similar to the epoll() system call. File descriptors are registered along with patterns to match on IO. Instead of raw IO notifying processes, the detection of patterns in the stream notifies processes of an available match.

> **This entails the use of a specific operating system. We will target Linux and perhaps optionally an**

**existing open RTOS implementation available for both Xilinx Zynq for ARM and on Xeon FPGA for Intel.**

The subsystem would be enabled if the proper configuration is detected. If on a Xilinx Zynq or Xeon FPGA the system could load and enable the proper drivers and kernel modules to activate the system API.

### Mainline Kernel Contribution

The system has great potential as a mainline kernel contribution to extend the capabilities of the Linux Operating System. It's general purpose application makes acceptance by Linus highly probable. The spread of FPGA based SoCs and Intel's new Xeon processor with FPGA will make the widespread use of such a pattern matching coprocessor possible. Meaning this is not a proposal for a niche application and has extreme utility on the latest systems already being sold in the market.

### Linaro Participation

David Rusling is an ARM fellow and distinguished engineer. He is also the CTO of Linaro and on the board of OptDyn advisors.

We've already discussed the prospect of including the coprocessor system API in the Linaro distribution which flows straight into the Linux Kernel mainline.

# DIFT Coprocessor

Information Flow Tracking (IFT) is a data flow tracking technique promising comprehensive security vulnerability protection. Dynamic Information Flow Tracking (DIFT) uses tags to dynamically track any flow in the system. This enables the detection of inconsistent and illegal conditions giving rise to several families of vulnerabilities including:

- Command Injection
- Authentication and Authorization Bypass
- Format String Attacks
- Cross-site Scripting
- Buffer Overflows
- SQL Injection
- Directory Traversal

The primary output of this project is a zero overhead hardware DIFT implementation using a monitoring software IP core as coprocessor in the PL of the Zynq SoC. All components in the system will operate in the PL in parallel to monitor and track data flow to trap illegal conditions leading to vulnerabilities.

The project aims to require no bytecode instrumentation, nor source code instrumentation for tag introduction. Unlike the research in Appendix A this project aims to also avoid static analysis to be a fully transparent hardware DIFT implementation. Instead an inference engine in the Coprocessor will be used to dynamically deduce the results of static analysis.

The coprocessor will be implemented on an ARM UltraScale Zync SoC and on the Intel Scalable processor families. In terms of operating systems we will support Linux and at least one open RTOS operating system. These two hardware platforms and operating systems should enable us to secure and satisfy most if not all use cases.

## Previous Research

The potential of such techniques were discovered by OptDyn while researching alternative mechanisms to implement United States Patent 7,971,255 on **"Detecting and preventing malcode execution"** by Alfred Aho.

ⓘ **United States Patent 7,971,255 Abstract**

A system for detecting and halting execution of malicious code includes a kernel-based system call interposition mechanism and a libc function interception mechanism. The kernel-based system call interposition mechanism detects a system call request from an application, determines a memory region from which the system call request emanates, and halts execution of the code responsible for the call request if the memory region from which the system call request emanates is a data memory region. The libc function interception mechanism maintains an alternative wrapper function for each of the relevant standard libc routines, intercepts a call from an application to one or more libc routines and redirects the call into the corresponding alternative wrapper function.

**NOTE**: Ironically this is the very same Alfred Aho from Columbia University whose Aho-Corasick Algorithm we are implementing in hardware with the Aho-Corasick Coprocessor project. This algorithm is the cornerstone of several technologies. Obviously pattern matching for recognition (detection) is the first step to prevention.

## Hardware is a **MUST**

Research shows that software DIFT implementations perform 25-37x slower than their untracked original systems. Software solutions require bytecode modifications and some access to the source code. Software solutions are too inefficient and intrusive to be feasible.

A series of recent research in the past 2-3 years shows that only hardware based solutions are viable. Hardware solutions themselves were shown to be very difficult to implement until recently on FPGA SoC platforms that combine hard cores monitored by coprocessors in the programmable logic (PL) of the SoC. See Appendix A for DIFT research conducted on the Xilinx Zynq-7000 series SoC published in June of 2016.

This same DIFT Coprocessor architecture is portable to the Intel Xeon with FPGA platform. Debugging interfaces for processor introspection as with ARM's CoreInsight exists for the Intel platform. This and the fact that it completely removes almost all overheads from the processing system by operating in parallel on the PL side make it the most attractive approach. Other hardware approaches exist in the state of the art, however they consume an entire hard core and have far too many moving parts which further complicates implementations.

Intel and the University of California have conducted alternative research on implementing hardware based DIFT using multi-core processing systems with a core dedicated to DIFT calculations. The core is responsible for performing information flow tracking operations.

# Hardware Based Intrusion Prevention System

This project represents both a host and network intrusion detection and prevention system. The output of this project is an order of magnitude more efficient, more accurate, and faster network and host intrusion detection and prevention system than anything currently available on the market today. It is a combined hardware and software solution that relies on the Aho-Corasick Stream Coprocessor and the DIFT Coprocessor components specified earlier.

## Fuzzy Hashing

The engine will detect polymorphic malware using the latest context based fuzzy hashing algorithms. These fall into the class of fuzzy hashing techniques called Context Triggered Piecewise Hashing (CTPH).

Most intrusion detection mechanisms use cryptographic hashes which do not have the flexibility of handling shape shifting malware.

For example, much like a real virus, a digital polymorphic virus or worm can change its shape folding into itself and use compression to be unidentifiable. Like the immune system of the body, anti-malware systems must relearn about the threat to prevent infection.

## Targeting Sequences in Streams

Specific attention will be applied to identifying packer instruction sequences and other relatively slow changing regions of polymorphic malware.

To a large extent fuzzy hashing driven by the modified Aho-Corasick Coprocessor can easily detect these sequences. Context triggered piecewise hashing mechanisms apply hashes to regions of a byte stream to calculate their fuzzy hashes. To do so requires a degree of pattern matching and parsing to find and target those regions for applying the hashing algorithms. The Aho-Corasick Coprocessor will be used to find such regions in streams in real time without latency introduction.

Whatever makes it through will get trapped by the DIFT Coprocessor on execution. This will automatically modify the threat database and update it's identifying hash sequences which allows the fuzzy hashing mechanism to catch it or any other mutated descendants before entering the system.

The fuzzy hashing based detection algorithms will be implemented in hardware. The intrusion detection system has the ability to detect and trap malware long before it has a chance to infect the running system. Another important characteristic of a hardware solution is to operate at line rate independently of the main processing system. The input output rates of networks and storage devices have increased so much that a software solution alone is not feasible without introducing significant latency.

Unlike traditional software based scanning solutions which must churn the disk, or slow down the network, this solution works as the data is flowing and is event driven.

The design specifically targets stream based operation. From where the stream originates or goes should make no difference to how the mechanism operates. The stream may come from locally attached storage, or from the network: the user maybe accessing a file from a newly attached USB storage device or is visiting a website page over the Internet. Regardless of the source or the sink, the filtration mechanism should work in the same way.

## CRITS Integration

CRITS is a threat database designed for social collaboration. By leveraging the knowledge of threats across a population of systems, greater knowledge of potential attacks are gained without the risk of experiencing them.

CRITS works with multiple security systems to capture threat information including the signatures of the malware vectors used for the attack. These signatures are tracked and shared across a global network of CRITS databases for communal threat intelligence.

These fuzzy hashing algorithms will allow for more accurate detection and threat information sharing via CRITS. We will integrate the system with CRITS to improve threat detection algorithms. However unlike the standard mechanisms of detection propagation in CRITS, we have the ability to propagate detection through gossip protocols based on locality.

## SIEM Integration

Command and control of millions of devices is essential especially where threats can arise and spread rapidly. SIEM integration is critical to detect abnormal behavior after calibrating to a normal baseline.

Apache Metron is a highly scalable advanced security analytics framework build by the Apache Hadoop Community. It is an open source project at the Apache Software Foundation. It evolved from Cisco's OpenSoC project which was open sourced and contributed to the Apache Software Foundation.

Apache Metron forms the basis to intelligent analysis of site wide threat intelligence and anomaly detection. It is the software used to establish a security operations center (SOC) which reacts to threats. Every IoT deployment needs a SOC to monitor and respond to threats. This is a clear management and maintenance requirement for all organizations with mass IoT deployments.

Apache Metron will be configured to interoperate with the intrusion prevention system. This will allow security operations to take full advantage of the framework at the organizational level. This also implies the integration of CRITS to produce the most comprehensive threat intelligence system available.

## Machine Learning

Apache Metron already has machine learning algorithms design for institution wide anomaly detection. We also see the potential for machine learning capabilities at the level of each node in the system. If not extensive at least inferences can feed directly into the Apache Metron SIEM intakes.

## Total Remote Attestation

The Trusted Platform Module specifications talk about remote attestation capabilities. These are very low level capabilities that specifically attest the integrity of the system's boot process. The Trusted Platform Module do not attest the system's overall integrity during runtime. A higher level remote attestation framework (RAF) in concert with an IPS is needed to cover all host components and subsystems. These frameworks usually provide the higher level file verification and system verification functions with secure network communications for systems to remotely confirm integrity before transacting with external systems.

Small devices and sensors in an IoT deployment need to be able to query their gateway to determine if it is compromised before using it to pass sensitive information or take commands to drive an actuator.

This is where the TPM->IPS->RAF connection is crucial for complete overall remote attestation. Several frameworks already exist and are based on the Linux Security Module of the Linux Kernel. These frameworks build on top of this kernel module and often interact directly with a TPM device. We're proposing the involvement of the IPS as an intermediary with a custom IPS-RAF implementation.

## Open Source Approach

This is a big system with already existing open source components. It makes sense to participate in multiple consortia to pool know how and resources to realize at least parts of the overall intrusion prevention system. The parts will be broken down into separate distinct open source projects. These sub-projects should be hosted under the umbrella of organizations like:

- The Linux Foundation
- The Apache Software Foundation
- Linaro
- Mitre

Several Apache Software Foundation projects are used to implement Apache Metron. Metron is the top level security operations center product and interface to system wide monitoring. It rolls up SIEM information collected from all nodes in a large system and performs the necessary analytics to achieve threat intelligence. It will be modified for operation with CRITS to leverage threat information from several sites in a global threat intelligence network.

The ARM Linux system interfaces belong in the Linux Kernel. They're going to make there way into the Kernel through the ARM Linux consortium, Linaro. Participants should become members of Linaro and collaborate directly with ARM. There may also be potential for collaboration with Xilinx specifically since one of the target hardware platforms is the latest Xilinx UltraScale Zynq SoC family.

We've already discussed the prospect of including the Aho-Corasick Stream Coprocessor system API in the Linaro distribution which flows straight into the Linux Kernel mainline.

# Appendix A - State of the Art Hardware DIFT Research

The following research paper was recently submitted on June 2016 by Muhammad Abdul Wahab, Pascal Cotret, Mounir Nasr Allah, Guillaume Hiet, Vianney Lapotre, and Guy Gogniat. It shows their preliminary research on developing a hardware based DIFT implementation on the Zync-7000 SoC platform. The paper is directly available here on the Web.

# A portable approach for SoC-based Dynamic Information Flow Tracking implementations

Muhammad Abdul Wahab[α], Pascal Cotret[α], Mounir Nasr Allah[β], Guillaume Hiet[β]
Vianney Lapôtre[γ], Guy Gogniat[γ]
[α] IETR / SCEE research group, firstname.lastname@centralesupelec.fr
[β] INRIA / CIDRE research group, firstname.lastname@centralesupelec.fr
[γ] Lab-STICC / University of South Brittany, firstname.lastname@univ-ubs.fr

## Abstract

*This work introduces an efficient approach for DIFT (*Dynamic Information Flow Tracking*) implementations on reconfigurable chips. Existing solutions are either hardly portable or bring unsatisfactory time overheads. This work presents an innovative implementation for DIFT on reconfigurable SoCs such as Xilinx Zynq devices. Even though the feasibility of this approach is currently being studied, the first results are promising.*

## 1 Introduction

Security threats still remain a major concern in high technology systems. Regarding software security breaches, recent efforts such as DIFT have been proposed. DIFT aims to track the application control flow by adding metadata (also known as *tags*) to information containers (e.g. registers, memory addresses), propagating and checking it at runtime. These approaches have been successfully used against a wide range of attacks including buffer overflow, SQL injections and so on.

Nevertheless, existing approaches cannot be implemented in modern SoCs due to their rigidity and strong time overhead. The purpose of this work is to find a more efficient method to implement DIFT features in embedded systems without compromising their security level. The chosen approach, including a dedicated hardware DIFT coprocessor, is discussed in this paper.

Section 2 presents related works on SoC-based DIFT solutions. Then, Section 3 explains the overall architecture of the approach proposed in this work. Section 4 introduces the DIFT coprocessor; especially, the interface between the PS (*Processing System*) and the coprocessor. Finally, Section 5 sums up the contributions of this work and looks at perspectives.

## 2 Related work

First and foremost, DIFT was implemented in software as in [7] which presents a flexible solution. However, the performance overhead is too high (from 300% up to 3700% as noted in [6]). Several hardware architectures were proposed to speed up DIFT processing time: [2, 8, 9] provide lower performance penalties at the expense of flexibility.

In [6], Kannan et al. proposed to decouple tags computation from the main application instructions towards a dedicated hardware coprocessor allowing applications on the CPU to run faster with multiple concurrent active policies. From that time, other solutions were proposed to add features or improve performances shown in [6]. For instance, Deng et al. ([3, 4]) proposed to use dynamic tainting to implement DIFT and other similar techniques such as UMC (*Uninitialized Memory Check*) or BC (*Boundary Check*).

In [5], Heo et al. proposed system-level approach to implement DIFT and other related techniques. Information required for tags computation by the coprocessor are added to the application source code through binary instrumentation. This information is executed at runtime: as a result, it sends data from the CPU to a FIFO queue read by the coprocessor. This approach, even though more realistic and generic, presents some drawbacks:

1. Information leakage at the interface between the CPU and the coprocessor (transmission of `load/store` memory addresses).
2. Code injection attacks may not be detected because the injected code is not instrumented (no information flow control will be done on this code).
3. It requires binary instrumentation to export memory addresses to the coprocessor (added instructions will be architecture-dependent).

## 3 Global architecture

Previous works were implemented using softcores (the CPU is implemented on FPGA logic): as a consequence, information required for tags computation was easily extracted by accessing internal signals. However, it is impossible with hardcores (where the CPU is an ASIC). This work proposes to use existing debug components in ARM CPUs to partially recover information required to decouple regular computation from tags computation. The remaining information is obtained through static analysis. The architecture proposed in this work is shown in Figure 1.

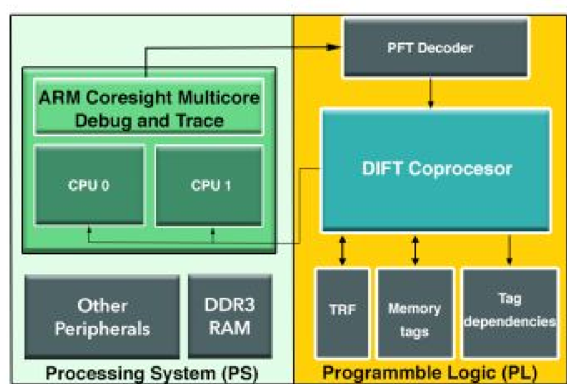



**Figure 1. Proposed Architecture for DIFT**

In this work, Zynq SoCs from Xilinx were used. However, it could be used with any architecture combining an ARM CPU with a FPGA. ARM debug components (called *CoreSight*) allow to generate and recover traces of applications ran by the CPU: traces contain information on instructions committed on the CPU. However, it uses a special protocol also known as PFT (*Program Flow Trace*): PFT outputs must be decoded to obtain human-

readable information on committed instructions.
The PFT decoder analyzes traces and sends them to the coprocessor implemented in the FPGA logic. The PFT decoder takes only 0.48% of FPGA logic. Moreover, during compilation phase of source code with LLVM, static analysis is done. PFT data is used alongside static analysis results that are loaded by the OS to Tag dependencies IP in the PL (*Programmable Logic*) when the program is launched. Tags are stored in TRF and in Memory tags (see Figure 1). The DIFT coprocessor checks tags according to user-defined security policies to verify if the CPU handles data in an unauthorized way.

## 4 DIFT Coprocessor

Decoupling DIFT operations from instruction decoding is possible by synchronizing both cores at system calls. The DIFT coprocessor requires at least three information from the CPU core obtained through CoreSight components and static analysis: PC (*program counter*), memory addresses (for *load/store* instructions) and instruction encoding.

### 4.1 DIFT Coprocessor interface

ARM CoreSight components allow to debug the code efficiently with negligible time overhead. Information obtained from CoreSight components of ARM Cortex-A9 (CPU included Zynq SoCs) are related to all the instructions modifying the PC register.

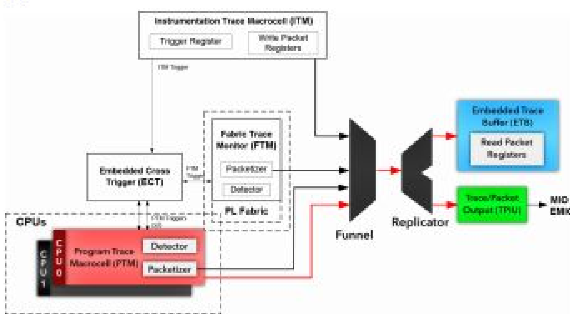



Figure 2. CoreSight Components[1]

Figure 2 shows the CoreSight components involved in the trace generation and export to the PL (FPGA area). PTM (*Program Trace Macrocell*) generates a trace for each committed instruction modifying the PC value. For instance, considering the code in Figure 3, PTM will generate a trace for instructions on lines 4 or 5 depending on the condition on line 3. The trace is transmitted through the funnel and the replicator and pushed in trace sinks (ETB and TPIU). ETB (*Embedded Trace Buffer*) allows to store traces in an on-chip RAM while TPIU (*Trace Port Interface Unit*) can send it to the programmable logic.

### 4.2 Static Analysis

Figure 3 shows the code and the result obtained through static analysis for this code. Static analysis allows to obtain tag dependencies shown in the control flow graph. The directives inside curly brackets indicate how tags should be propagated. Consider the node 2 in figure 3, the variable x should be tainted by the tag of "random" function output and y should be tainted by the tag of "input" function output.
The main core (ARMv7 architecture) commits instruction and waits, if necessary, on system calls until the DIFT coprocessor completes tags checking. Meanwhile the DIFT coprocessor reads tag dependencies, propagates and checks tags accordingly for all

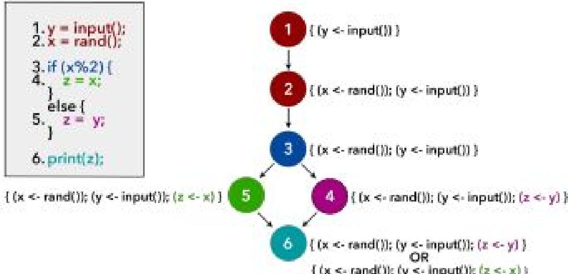



Figure 3. Example Code and CFG

the instructions. If a tag check fails, an exception is raised to alert the CPU.

## 5 Conclusion

The approach proposed in this work shows huge potential as it allows to efficiently implement DIFT on SoCs. This approach should not be limited to ARM-based SoCs: Intel also offers trace components, for debug purposes, allowing to retrieve information on committed instructions. Any SoC with hard cores including debug components can be used to implement DIFT with our approach. A prototype is under development to study feasibility of DIFT on Zynq SoC using our approach. Then, we plan to look at frequency issues between the CPU core and the DIFT coprocessor which is another reason why existing architectures are not commonly used.

## References


[1] Zynq technical reference manual. `www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf`.

[2] M. Dalton, H. Kannan, and C. Kozyrakis. Raksha: A flexible information flow architecture for software security. *SIGARCH Comput. Archit. News*, 35(2):482–493, June 2007.

[3] D. Y. Deng, D. Lo, G. Malysa, S. Schneider, and G. E. Suh. Flexible and efficient instruction-grained run-time monitoring using on-chip reconfigurable fabric. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 137–148. IEEE Computer Society, 2010.

[4] D. Y. Deng and G. E. Suh. High-performance parallel accelerator for flexible and efficient run-time monitoring. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–12. IEEE, 2012.

[5] I. Heo, M. Kim, Y. Lee, C. Choi, J. Lee, B. B. Kang, and Y. Paek. Implementing an application-specific instruction-set processor for system-level dynamic program analysis engines. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 20(4):53, 2015.

[6] H. Kannan, M. Dalton, and C. Kozyrakis. Decoupling dynamic information flow tracking with a dedicated coprocessor. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 105–114. IEEE, 2009.

[7] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. 2005.

[8] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas. Secure program execution via dynamic information flow tracking. In *Acm Sigplan Notices*, volume 39, pages 85–96. ACM, 2004.

[9] G. Venkataramani, I. Doudalis, Y. Solihin, and M. Prvulovic. Flexitaint: A programmable accelerator for dynamic taint propagation. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 173–184. IEEE, 2008.

# Appendix B - Project Proposal Copyright Notice

This project summary and all other projects proposed in this cyber security hardware suite are the intellectual property of OptDyn, Inc incorporated in the State of Delaware, USA.

Copyright © 2018 OptDyn, Inc.