Software Security and Randomization through Program Partitioning and Circuit Variation

Todd R. Andel
University of South Alabama
Mobile, Alabama USA
tandel@
southalabama.edu

Lindsey N. Whitehurst
University of South Alabama
Mobile, Alabama USA
lnw1222@
jagmail.southalabama.edu

Jeffrey T. McDonald
University of South Alabama
Mobile, Alabama USA
jtmcdonald@
southalabama.edu

ABSTRACT

The commodity status of Field Programmable Gate Arrays (FPGAs) has allowed computationally intensive algorithms, such as cryptographic protocols, to take advantage of faster hardware speed while simultaneously leveraging the reconfigurability and lower cost of software. Numerous security applications have been transitioned into FPGA implementations allowing security applications to operate at real-time speeds, such as firewall and packet scanning on high speed networks. However, the utilization of FPGAs to directly secure software vulnerabilities is seemingly non-existent.

Protecting program integrity and confidentiality is crucial as malicious attacks through injected code are becoming increasingly prevalent. This paper lays the foundation of continuing research in how to protect software by partitioning critical sections using reconfigurable hardware. This approach is similar to a traditional coprocessor approach to scheduling opcodes for execution on specialized hardware as opposed to running on the native processor. However, the partitioned program model enables the programmer the ability to split portions of an application to reconfigurable hardware at compile time. The fundamental underlying hypothesis is that synthesizing portions of programs onto hardware can mitigate potential software vulnerabilities. Further, this approach provides an avenue for randomization or diversity for software layout and circuit variation.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: GENERAL—*Hardware/software interfaces*; D.2.0 [**SOFTWARE ENGINEERING**]: General—*Protection mechanisms*

General Terms

Security; Theory

MTD'14, November 3, 2014, Scottsdale, Arizona, USA.

http://dx.doi.org/10.1145/2663474.2663484.

Keywords

Software partitioning; secure software; circuit variation; program protection; reconfigurable hardware

1. INTRODUCTION

Application-Specific Integrated Circuit (ASIC) designs are increasingly being replaced by Field Programmable Gate Arrays (FPGAs) due to their flexibility and lower cost. The commodity status of FPGAs has allowed computationally intensive algorithms, such as cryptographic protocols, to be able to take advantage of faster hardware speed while at the same time leverages the reconfigurability and lower cost of software. The majority of applications transformed from software to FPGA implementations target execution time and energy reductions [21, 24, 27]. Numerous security applications have been transitioned into FPGA implementations to allow security applications to operate at real-time speeds, such as firewall and packet scanning uses on high speed networks without performance degradations [6, 11, 14, 15]. FPGA security research exists into side-channel vulnerabilities and countermeasures [1, 2, 4], intellectual property protections [17, 18, 20, 25], and anti-tamper applications [5, 7].

The utilization of FPGAs to secure software vulnerabilities is seemingly non-existent. We have found one group that has looked at using FPGAs for software security by using dynamic verification of a program's integrity during run-time. In [10],they develop an FPGA approach to verify the integrity of a program while it is running and using a fully encrypted version of the program from memory. This can be viewed as an external monitor, or device, which supplies real-time encryption/decryption services for a program, but not a direct security solution for a given program.

The hypothesis in this paper is that partitioning security critical program sections to FPGAs may mitigate many software security risks that rely on jumping within a program's address space. Attacks that rely on addresses to be successful, such as stack overflow, heap overflow, and return-to-libc attacks, would be moot on an FPGA since software defined as circuit logic gates in effect do not have a address space or rely on a program counter. If a program with known vulnerabilities is taken and broken up into sections of invulnerable and vulnerable parts (with respect to address-based attacks), we can implement the invulnerable parts as usual on a traditional processor, and implement the vulnerable sections on an FPGA. In this way, we propose that the overall security of the program can be increased because attacks on these vulnerable sections can no longer be carried out. Additionally, since we utilize reconfigurable hardware, our

partition approach can be used to provide a dynamic and adaptive software layout, resulting in a continually changing target.

The remainder of this paper is as follows. Section 2 provides a background on current security approaches for existing implementations. Section 3 describes the partitioned program approach for securing software. Section 4 describes the use of program partitioning as an avenue for randomization or diversity for software layout. Section 5 identifies related research questions and finally, Section 6 provides a summary and discusses the focus of ongoing and future work.

2. EXISTING HARDWARE IMPLEMENTATIONS AND PROTECTIONS

The lines between software and hardware have been traditionally clear. Software applications were written in a high level language and compiled for operation on general purpose processors (GPPs) and embedded hardware solutions were used for custom applications. While this model maintains as the standard, today's lines are not quite as clear. Figure 1 indicates how this traditional model is blurred, as now software can be specified as hardware and hardware specified as software. This transition stems from the growth in ASICs to speed up computationally intensive algorithms, such as cryptographic applications.

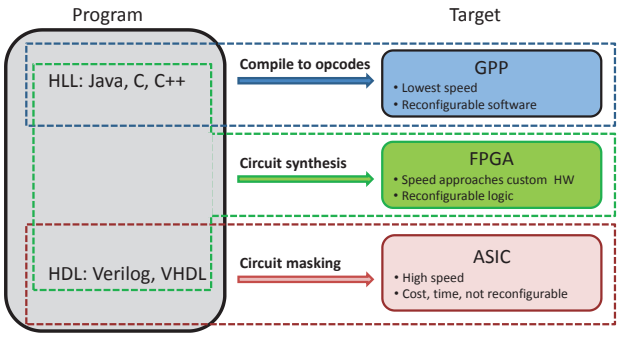

Figure 1: New Model of Software and Hardware

Even more momentum towards this new model comes from reconfigurable devices, such as FPGAs and Systems-on-Chip (SoC). Current SoC devices, such as the*Xilinx* [1] or *Altera* [2] based products, provide an ARM-based mobile processor with a limited amount of reconfigurable logic to provide customized solutions. Full FPGA implementations rely solely on reconfigurable logic. Some FPGA devices have enough reconfigurable logic to implement a fully virtualized (or "soft") version of the general purpose microprocessor. In the remainder of this paper we focus on FPGAs as our reconfigurable hardware, however, the research may be equally applied to SoC implementations. FPGA configurability, as well as much cheaper cost than customizing production ASICs, is making FPGAs the product of choice. FPGAs are now appearing as a viable option for software applications either on a soft processor, customized circuit implementation, or as a hybrid co-design/partition/processing of an application between a traditional general purpose processor and a customized portion of the software running as realized hardware on the FPGA device. *Xilinx* and *Altera* both produce boards that can be easily added to most desktop computers. Additionally, the *Intel Atom E6x5C* processor series [3] includes an *Altera Arria-II-GX* series FPGA embedded on the same chip die.

With the trend of offloading software to FPGA implementations, it remains to be seen if this causes new vulnerabilities or if software can be partitioned in such a way to provide enhanced software security against malicious code and adversaries. Up until this point, security in the context of FGPAs has looked at using FPGAs to increase speed in security applications for networking and cryptographic implementations, securing side-channel emanations, intellectual property protections and anti-tamper, and dynamically monitoring of programs. The following sections provide a brief overview of this current work.

2.1 Speed and Efficiency in Security Applications

FPGAs have proved useful for network security applications. In this context FPGAs have not been used to provide security of the program itself, but instead to provide speedup for security applications. The majority of work in this area has focused on providing real-time packet analysis for intrusion detection and anti-virus signature scanning [6,11,14,15]. Network-based packet scanning is limited to placement of devices such that the scanning rate can handle the maximum throughput of a given network link. With a traditional software based approach running on a general purpose processor, the sequential process greatly dictates placement closer to the end-point of most networks. The offloading of computationally expensive pattern-matching/signature based packet search processing to reconfigurable devices allows implementation of search techniques to be done in hardware in a parallel nature. This parallel analysis approach provides increased speed enhancements, allowing the network-based scanning to be done at a much higher position in the network hierarchy. Analysis at a higher level can restrict malicious packets from reaching a larger portion of internal networks. Implementations developed in [15] and [11] show achieved packet scanning rates around 2.4 Gbps, which is sufficient to analyze network links up to OC-48 ATM circuits.

Not only can FPGAs provide the means to speed up applications, but they can also decrease energy consumption. If blocks that take up the most execution time can be identified and implemented on an FPGA, energy savings can be increased up to 70% [24]. Other studies have also shown this level of energy savings as well as high speedups when compared to implementations solely on a general purpose processor.

FPGA solutions are not the only approach to speedup and energy savings for software. Application-specific integrated circuits (ASICs) also provide the parallelism that a reconfigurable fabric provides, and they have the advantage of consuming less power [27]. However, there is considerable amount of time costs in developing an ASIC. Furthermore, if parts of the application being executed need to be changed, the hardware cannot be changed to accommodate it, which may cause a degradation on its performance. Finally, the

[1]http://www.xilinx.com
[2]http://www.altera.com
[3]http://www.intel.com/content/www/us/en/intelligent-systems/stellarton/atom-e6x5c-platform-brief.html?wapkw=atom-e6x5c

mask costs for ASICs have also increased substantially compared to reprogrammable hardware [9]. Therefore, for cost efficiency, reprogrammable hardware is more commonly used than ASICs for increasing performance and decreasing energy consumption of programs.

2.2 Security against Side-Channel Exploitation

With this increased ability of speed in computationally expensive operations, it is likely vendors and government contractors will start offloading cryptographic applications to FPGA devices. The speeds, time to delivery, and device configurability will drive this transition. However, even secure cryptographic algorithms can be exploited based on rudimentary implementations.

In their seminal 1999 paper [13], Kocher et al. present a method to recover a secret key from cryptographic hardware by monitoring the hardware's power consumption. In recent years, these so called side-channel analysis (SCA) attacks have become a focus of the cryptographic community. These attacks are conducted by collecting power consumption data of the hardware, referred to as power traces, over many cryptographic cycles and statistically correlating this data to the likely cryptographic key. These attacks allow an attacker to recover keys much faster than traditional cryptanalysis or a brute-force approach. Since Kocher's discovery, much work has focused on developing countermeasures to such side-channel attacks. Two such countermeasure approaches were developed in [1, 4] and in [2]. These countermeasures focused on securing the two most commonly used cryptographic algorithms, namely RSA and AES.

The countermeasure work in [1, 4] focused on protecting hardware FPGA implementations of RSA by flattening the power spikes normally associated with differential power analysis on squaring and multiplying operations in the RSA algorithm. This work developed a dynamic countermeasure which constantly varies the timing and power consumption of each operation, making correlation between traces more difficult than for static countermeasures. This protection was achieved by randomizing the radix of encoding for Booth multiplication and randomizing the window size used in exponentiation operations.

The countermeasure work in work [2] focused on protecting hardware FPGA implementations of AES by randomizing a circuit's clock frequency and simultaneously performing bit-balancing by encrypting (or decrypting) the inverse of the input data with an inverse AES circuit implementation. In this case the available area on the FPGA and the ability to parallelize both the primary AES stream and an inverse AES operation allowed the circuit to be protected from Hamming Weight and Hamming Distance type side-channel attacks.

2.3 Intellectual Property Protection and Anti-Tamper

Another area of concern for FPGA-based implementations is the protection of intellectual property and the ability to protect against physical tampering. By changing the hardware, generated circuits can cause an attacker's efforts to increase substantially without causing major increases in performance costs [18].

Physical tampering will commonly result in some type of temperature variation. For example, researchers in [12] showed that by freezing a chip, it could be removed and placed into a new circuit without loosing is dynamic data. With this type of threat along with potential heating impacts to changing power or physical handling of devices, protections based on thermal monitoring have been introduced. Thermal monitoring based on synthesized ring oscillator circuits have been proposed in [7] and [5]. The advantage to software synthesized embedded temperature monitoring as opposed to using standard physical thermal diodes is the fabric based sensors can be distributed within other synthesized circuits and cannot be physically bypassed. With this type of solution in place, physical attacks that result in temperature changes can trigger routines to wipe sensitive data in memory locations.

Another application of FPGAs in tamper resistance includes using execute only memory architecture, otherwise known as XOM. In these systems, the instructions within a program cannot be altered, but rather they can only be executed. Work in [26]describes a system in which all data is encrypted, and the only machine that can execute it is the processor with the correct key. They use public key cryptography along with XOM architecture so that anyone can encrypt a program to be run on a processor, but only the processor with the corresponding private key can run it. Using XOM in this way can successfully protect the intellectual property, but it alone cannot protect from a program's own vulnerabilities, such as those from buffer overflow attacks. To protect against unauthorized access to other parts of the processor, compartments can be set up and the system can halt a program if it attempts to read outside its compartment [26].

In a similar vein, protecting FPGA synthesized circuits, or intellectual property (IP) cores, has gained significant attention. The simple process of synthesizing a FPGA with an IP core allows anyone the ability to simply copy, or clone, and use a given system. While some IP cores are intended for open use, many are not and therefore require protection schemes. In such a situation, if an IP core can be restricted for use only on a given FPGA device then the core itself becomes useless to unauthorized users. This type of protective measure relies on the ability to positively identify a device, akin to DNA or fingerprint uniqueness. For example, synthesized circuits can be used to build a fingerprint based on minute imperfections of the underlying physical silicon device. Research following this approach can be seen in [20] and [25].

2.4 Dynamic Monitoring of Programs

In order to ensure that programs run within its permissible bounds, FPGAs have be used to monitor them during runtime. For example, CODESSEAL uses external hardware to help prevent tampering, structural, and data attacks [10]. The approach taken in CODESSEAL was to protect programs by having the program and data in encrypted form when loaded in the program memory space. A FPGA is placed between the memory and cache to provide high speed encryption/decryption of the code and data as well as provide protected instruction flow before being executed within the processor. Only encrypted data is sent between the external hardware and the processor, so that addresses and data lines cannot be sniffed. Because they implemented this systems using an FPGA the overhead was not substantial, on average a 3.97% penalty. Although this protects against tampering, it may not protect against struc-

tural attacks. Other systems extract the normal behavior of programs while they are running in order to determine guidelines for correct usage of the software [3]. In these types of systems, extra hardware is attached to the GPP to monitor the program and prevent attacks by detecting abnormal behavior. Although this system prevents attacks on trusted software, it requires an external hardware monitor.

3. SOFTWARE PROTECTION THROUGH HARDWARE PARTITIONING

Noticeably absent from the literature is the use of FPGAs to provide direct security against software vulnerabilities. Research is needed to determine if software protections can be realized from FPGA commodity availability. We are currently developing a way to protect software through through partitioning of critical sections, or components, onto reconfigurable hardware. To our knowledge this approach appears novel and not covered in current literature. The closest approach similar to our method is the CODESEAL [10] system previously described, which utilized a combined compiler/hardware system to support software security, but in a different approach than software partitioning

3.1 Partitioning Approach

Software, or program, partitioning can be visualized according to Figure 2 as a program now consisting of both traditional operations on a general purpose processor and synthesized logic on an FPGA or other reconfigurable logic. This approach is similar to a traditional coprocessor approach to scheduling opcodes for execution on specialized hardware as opposed to running on the native general purpose processor. For instance, the introduction of math coprocessors, such as the *Intel 8087* floating point unit, enabled complex mathematical operations to be executed on customized hardware, thus freeing up the general purpose processor up to continue simpler operations. The coprocessor approach was a manufacture time decision by the hardware designer and is not tailorable by the software programmer once burned into a physical chip implementation. However, the partitioned program model combined with reconfigurable logic enables the programmer the ability to split off security related portions of an application to reconfigurable hardware at compile time. With this option now available, it is hypothesized that synthesizing portions of programs onto hardware can mitigate potential software vulnerabilities. The challenge is to identify program portions that are best suited, from a software security standpoint, for partitioning and implementation onto hardware.

While the idea of partitioning software between a general purpose processor and reconfigurable device is not new, using the approach for software security is. Traditionally the partitioning problem has focused on identifying program portions for hardware synthesis such that speedup enhancements according to Amdahl's Law outweigh the extra costs of triggering the hardware and the cost of transmitting program state between the general purpose processor and the hardware [22]. A prime example in the security realm is the offloading of computationally expensive cryptographic operations where the context switch overhead between software and hardware is less than the computational time saved for such operations.

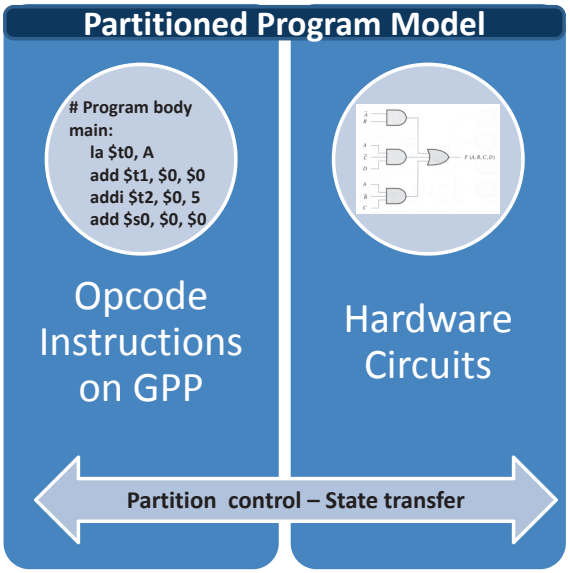

Figure 2: Program Partitioning

To illustrate our program partitioning approach for software security, consider the well understood problem of stack buffer overflows, the source of many program vulnerabilities (Morris, Blaster, and Slammer worms to name a few). In a traditional buffer overflow attack, an attacker attempts to work outside of the memory boundary of a program data value to an alternate location within the program address space. If the address jumped to had operations that invoked a shell or other malicious code, the attacker may be able to obtain unauthorized access to the system. This process is illustrated in Figure 3, where a function accepting command line input can be subverted by attacker input that both overwrites the original function return address and injects potential attacker code in place of the repeated "A's" at the new jump location. It is hypothesized that by following the partitioned program approach and synthesizing these critical portions of code to circuit logic onto FPGAs, jumping to a separate part of the program address space may have no contextual meaning. That is, programs or program portions synthesized into circuit logic do not rely on a program counter or the program address space. Thus, the partitioned program approach may prevent this type of vulnerability.

While there are other solutions to the buffer overflow problem, such as using a type safe language and safe libraries, the program partitioning technique can provide another avenue for protection where other safeguards are not available. The buffer overflow problem offers a simple view into the potential of the partitioning theory for program protection. Additionally, other safety critical program areas may benefit from such an approach. Identification and potential elimination of program vulnerabilities through program partitioning may provide a foundational ground breaking approach to program vulnerability mitigation. This solution may also provide an avenue for protection against zero-day exploits.

Additionally, related to program partitioning between a GPP and a FPGA is the ability to fully synthesize a complete GPP onto an FPGA device. The approach is generally referred to as a "soft-core" processor and processor cores are

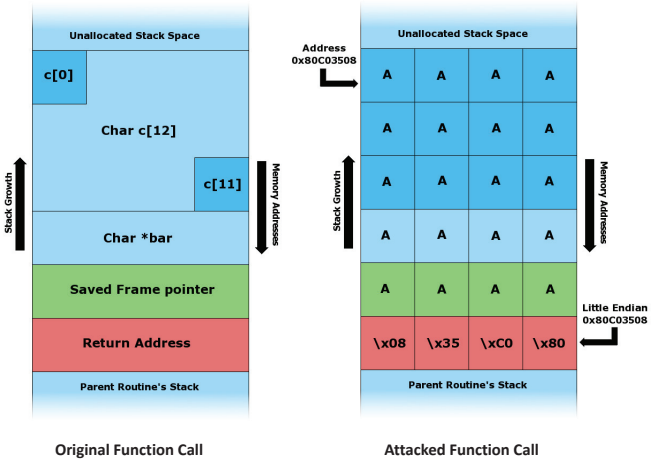

Figure 3: Buffer Overflow Attack

readily available for FPGA implementation; standard soft-cores readily available for synthesis are the *Xilinx Microblaze* and *Altera Nios* soft-core processors. The approach of software partitioning for program security would be equally applicable to soft-core processors.

3.2 Current Progress

Our research is a current work in progress, but we have made steady advancements towards testing the hypothesis of our partitioned program approach. The research plan follows a systematic phased approach to test the hypothesis that partitioning a program between hardware and software may mitigate address based software vulnerabilities.

Our first goal was to identify classical software vulnerabilities to be partitioned into software/hardware code portions. The starting focus is on the stack buffer overflow vulnerability as previously discussed. Since this vulnerability is well known, it serves as an ideal starting point to test the hypothesis. Given source code with this known vulnerability, critical sections of the code are readily identifiable for partitioning. We created a simple program that relies on user input into a *license check* routine before proceeding. The program was written in C and provides no array bounds checking on the user input and is therefore vulnerable to a buffer overflow attack. The procedure was successfully compiled to both an Intel based GPP and onto a *Microblaze* soft-core processor on a Xilinx Virtex-5 FPGA. A buffer overflow attack to subvert the license check is successful in both instances as expected. This current progress serves as a baseline for the buffer overflow vulnerability.

We are currently porting the vulnerable program to the respective partitioned code into software for the GPP and into circuit definitions for the FPGA. This is achieved through transitioning the baseline code into basic blocks and then identifying the critical section vulnerable to the buffer overflow. The critical portion is being coded directly as a circuit definition using a mixture of structural and behavioral VHDL. As previously depicted in Figure 2, the GPP software will need to trigger the critical partition on the FPGA as required. Additionally, any required data must be passed via state transfer between the GPP registers and virtual registers synthesized within the FPGA circuit definition, followed by the FPGA transferring control back to the GPP controlling program portion. As part of validation, the newly partitioned program will be executed to ensure normal operation.

Once our partitioning is successful, we will evaluate attacker success in both the traditional software environment vs. the new partitioned program approach. Timing metrics will also be collected to ensure the added overhead does not inhibit the approach's use. These concepts will be further tested against full soft processor and partitioned approach using soft processors as the GPP and the partitioned section onto another portion of the same FPGA fabric as an independent circuit definition.

4. DYNAMIC AND ADAPTIVE SOFTWARE LAYOUT

Since we are working with reconfigurable hardware, we are also actively researching topics related to a dynamic or polymorphic circuit variation, thus allowing us to continually change the adversary's perceived target, as now it is not statically burned into a customized chip implementation. We are working on two options for dynamic hardware reconfigurability. The first follows the idea of program partitioning, the second approach utilizes circuit variation through the generation of equivalent circuit variants.

4.1 Partitioning Variants

We can utilize our partitioning approach in more of a broad way to provide the ability to dynamically change our software layout. Instead of defining security critical portions to partition to hardware that we are currently investigating, we can also randomly select basic blocks for partitioning, thus providing different software/hardware compositions of the same program. Based on our current manual partitioning process, we can attest to the tediousness of manual development of the critical sections into logic blocks using hardware description languages (HDLs). This manual process would more than likely make generating many random program partition variants infeasible. Fortunately, we can rely on recent advancements of High Level Language (HLL)-to-HDL compilers enabling the direct translation of programs from High Level Language to HDL. Such HLL-to-HDL compilers that currently exist include SystemC [4], Streams-C [8] and Impulse C [5] to name a few.

4.2 Circuit Variants

Another approach to dynamic variation is through circuit variants. Since any program can be represented logically as a circuit, we can generate equivalent circuits to produce dynamic polymorphic variants. We have produced extensive work on program and circuit obfuscation related to this approach [16, 18, 19]. As Figure 4 depicts, we let P be a program, defined as a circuit, from which want to generate a set of equivalent circuits. δ is a set of programs or circuits with a common input/output size $[X, Y]$ and a bounded size $[S]$ (gates, lines of code, etc.). δ_P is a subset of programs or circuits with the same signature (functional behavior / truth table) as P. An obfuscating transformation, $O(\cdot)$, at best can only select an equivalent program/circuit from the

[4]http://www.accellera.org/downloads/standards/systemc/
[5]http://www.impulseaccelerated.com/

same set ($C_{|X|-|Y|-[S]}$) or a set with a larger (polynomially-bounded) size ($C_{|X|-|Y|-[f(S)]}$). Figure 4 also depicts that a randomly selected program (P_R) is chosen from the same I/O class set as P that would not be considered equivalent since it has a different I/O relationship.

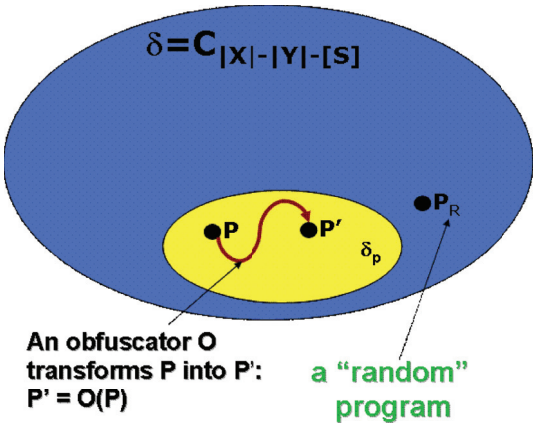

Figure 4: Polymorhipc Set Selection

We have been successful using this approach in taking a given combinational circuit and producing a family of equivalent circuits that maintain the functional program properties (i.e., input-output relationships), but look entirely different from a structural point of view. This pool of equivalent circuits provide polymorphic random variants that can be randomly chosen and loaded onto FPGA to provide a moving target. We are current extending this work to include the ability to generate sequential circuits, which will allow us to capture a greater subset of program variations.

4.3 Dynamic FPGA Configurations

Both of our variant approaches can be continually loaded to an FPGA device prior to run-time to create an artificial diversity for the attack space, thus creating confusion for the attacker since the polymorphic target space is in a continual flux. Normal FPGA operation bootstraps the circuit design upon startup or system reset, thus providing the ability for a moving target. This approach is somewhat limiting in that the system must be restarted for the new variant to take effect. However, there has been recent advancements in dynamic partial reconfiguration techniques which allow FPGAs to implement on-the-fly changes without having to shut down or re-set the device. Partial dynamic reconfiguration is currently supported by both Xilinx [6] and Altera [7] as well as the open source tool OpenPR [23]

5. ADDITIONAL RESEARCH AREAS

The commodity status of FPGAs being incorporated in common desktop devices will provide significant speed enhancement possibilities to highly computational programs. As the pervasiveness of these devices continues, e.g., integration of an *Altera* FPGA into the *Intel Atom E6x5C* product line, consumers may have these devices and not even be aware they are present. The increased vulnerability space must therefore be investigated to ensure new undiscovered attack vectors do not leave current systems more vulnerable and if so, determine what new security measures may be needed.

While vulnerability space investigation of incorporating these devices is beyond the scope of this paper, it is vital to identify such significant and related research areas. As FPGAs transform into a commodity device are included, or added to, general purpose processors, the vulnerability space needs to be fully understood. Research is therefore needed to to identify if malicious circuitry within the FPGA device may have introduced an attack vector against software running on the general purpose processor. This realization suggests two significant questions: 1) can malicious circuits on the hardware attack vulnerabilities on programs running on the general purpose processor; 2) can programs implemented as circuits be subjected to their own type of malware against the circuit?

Research focusing on the first area must consider that FPGA devices running in conjunction with GPPs may contain malicious circuits intended to attack traditional programs from within. Since it is local, malicious circuitry would in essence have local root level access to the processor. Malicious circuits could be embedded during FPGA fabric manufacturing, during circuit synthesis, or placed dynamically during runtime onto bootstrap memory to ensure circuit corruption upon restart.

Research focused on on the second area must be considered as programs are transitioned to circuit implementation onto FPGA devices. What types of vulnerabilities do these synthesized programs possess? Attacks against synthesized circuitry will be viable if run-time changes to the circuit fabric, or if dynamic changes to the bitstream storage are possible. Also, what types of vulnerabilities exists to general purpose processors fully implemented as soft processors within FPGAs? Typically, the circuit description is stored on the FPGA and reloaded as a bitstream program each time the system is powered on. An attacker may attempt to inject malicious code synthesized as a circuit to attempt code injection directly at the soft processor and at the bitstream storage location. Countermeasures to such attack vectors must also be investigated.

6. SUMMARY

The introduction of FPGAs as low cost and reconfigurable additions to general purpose processors has provided product developers compile time flexibility while nearing the speed of customized hardware devices. This realization has led to the use of FPGAs for highly computational algorithms, which make this paradigm highly attractive to computationally expensive, but easily parallelizable algorithms. While using this approach to efficiently run security applications is gaining traction, using FPGA and other reconfigurable devices to directly provide code security is lacking.

This paper has provided an approach to securing code vulnerabilities by partitioning software between typical opcodes on a general purpose processor and as synthesized circuitry for FPGA implementation. The hypothesis is that partitioning security critical program sections to FPGAs may mitigate many software security risks that rely on jumping within a program's address space. Further, we have presented an approach using reconfigurable logic and program

[6]http://www.xilinx.com/tools/partial-reconfiguration.htm
[7]http://www.altera.com/devices/fpga/stratix-fpgas/stratix-v/overview/partial-reconfiguration/stxv-part-reconfig.html

partitioning to provide an option for dynamic variability of program layout and circuit variation. Additionally, related research areas have been identified to investigate potential FPGA vulnerabilities.

7. ACKNOWLEDGMENTS

This material is based in part upon work supported by the National Science Foundation under Grants No. CNS-1305369, No. DUE-1241675, and No. DUE-1303384.

8. REFERENCES

[1] T. R. Andel, J. W. Barron, T. McDonald, and J. W. Humphries. RSA Power Analysis Obfuscation: A Dynamic Algorithmic Hardware Countermeasure. *International Journal of Computing and Digital Systems*, 3(2):69–78, May 2014.

[2] T. R. Andel, A. Fritzke, J. W. Humphries, and J. T. McDonald. Design and Implementation of Hiding Techniques to Obfuscate Against Side-Channel Attacks on AES. *International Journal of Computing & Network Technology*, 2(2):65–72, May 2014.

[3] D. Arora, S. Ravi, A. Raghunathan, and N. Jha. Secure embedded processing through hardware-assisted run-time monitoring. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 178–183 Vol. 1, March 2005.

[4] J. Barron, T. R. Andel, and Y. Kim. Dynamic Architectural Countermeasure To Protect RSA Against Side Channel Power Analysis Attacks. In *In Proceedings of 6th International Conference on Information Systems, Technology, and Management(ICISTM 2012)*, pages pp. 372–383, Grenoble, France, 28-30 March 2012.

[5] B. A. Brown, T. R. Andel, and Y. Kim. An FPGA Noise Resistant Digital Temperature Sensor with Auto Calibration. In *In Proceedings of 6th International Conference on Information Systems, Technology, and Management(ICISTM 2012)*, pages 325–335, Grenoble, France, 28-30 March 2012.

[6] H. Chen, Y. Chen, and D. Summerville. A Survey on the Application of FPGAs for Network Infrastructure Security. *Communications Surveys Tutorials, IEEE*, 13(4):541 –561, quarter 2011.

[7] J. Franco, E. Boemo, E. Castillo, and L. Parrilla. Ring Oscillators as Thermal Sensors in FPGAs: Experiments in Low Voltage. In *Programmable Logic Conference (SPL), 2010 VI Southern*, pages 133–137, 2010.

[8] J. Frigo, M. Gokhale, and D. Lavenier. Evaluation of the Streams-C C-to-FPGA Compiler: An Applications Perspective. In *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*, FPGA '01, pages 134–140, New York, NY, USA, 2001. ACM.

[9] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu. An Overview of Reconfigurable Hardware in Embedded Systems. *EURASIP J. Embedded Syst.*, 2006(1):1–19, Jan. 2006.

[10] O. Gelbart, P. Ott, B. Narahari, R. Simha, A. Choudhary, and J. Zambreno. CODESSEAL: Compiler/FPGA Approach to Secure Applications. In *Proceedings of the 2005 IEEE international conference on Intelligence and Security Informatics*, ISI'05, pages 530–535, Berlin, Heidelberg, 2005. Springer-Verlag.

[11] N. B. Guinde and R. B. Lohani. FPGA Based Approach for Signature Based Antivirus Applications. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, ICWET '11, pages 1262–1263, New York, NY, USA, 2011. ACM.

[12] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Cal, A. J. Feldman, and E. W. Felten. Least We Remember: Cold Boot Attacks on Encryption Keys. In *In USENIX Security Symposium*, 2008.

[13] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag.

[14] S. Li, J. Torresen, and O. Soraasen. Exploiting Reconfigurable Hardware for Network Security. In *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on*, pages 292 – 293, April 2003.

[15] J. W. Lockwood, J. Moscola, M. Kulig, D. Reddick, and T. Brooks. Internet Worm and Virus Protection in Dynamically Reconfigurable Hardware. In *In Military and Aerospace Programmable Logic Device (MAPLD)*, page 10, 2003.

[16] J. McDonald. Capturing the essence of practical obfuscation. In S. Dua, A. Gangopadhyay, P. Thulasiraman, U. Straccia, M. Shepherd, and B. Stein, editors, *Information Systems, Technology and Management*, volume 285 of *Communications in Computer and Information Science*, pages 451–456. Springer Berlin Heidelberg, 2012.

[17] J. McDonald and Y. Kim. Examining Tradeoffs for Hardware-Based Intellectual Property Protection. In *Proc. of the 7th International Conference on Information Warfare (ICIW-2012)*, Seattle, USA., March 22-23, 2012.

[18] J. McDonald, Y. Kim, and D. Koranek. Deterministic Circuit Variation for Anti-Tamper Applications. In *Proc. of the Cyber Security and Information Intelligence Research Workshop (CSIIRW 2011)*, Oak Ridge, TN, USA., October 12-14, 2011.

[19] J. T. McDonald, Y. C. Kim, and M. R. Grimaila. Protecting Reprogrammable Hardware with Polymorphic Circuit Variation. In *Proceedings of the 2nd Cyberspace Research Workshop*, Shreveport, Louisiana, USA, June 2009.

[20] H. Patel, J. Crouch, Y. Kim, and T. Kim. Creating a Unique Digital Fingerprint Using Existing Combinational Logic. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 2693–2696, 2009.

[21] D. Pellerin and S. Thibault. *Practical FPGA Programming in C*. Prentice Hall, 2005.

[22] R. Sass and A. G. Schmidt. *Embedded Systems Design with Platform FPGAs: Principles and Practices*. Morgan Kaufmann, 2010.

[23] A. Sohanghpurwala, P. Athanas, T. Frangieh, and A. Wood. OpenPR: An Open-Source Partial-Reconfiguration Toolkit for Xilinx FPGAs. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 228–235, May 2011.

[24] G. Stitt and F. Vahid. Energy Advantages of Microprocessor Platforms with On-Chip Configurable Logic. *Design Test of Computers, IEEE*, 19(6):36 – 43, Nov/Dec 2002.

[25] G. Suh and S. Devadas. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pages 9–14, 2007.

[26] D. L. C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. Architectural support for copy and tamper resistant software. *SIGOPS Oper. Syst. Rev.*, 34(5):168–177, Nov. 2000.

[27] T. Todman, G. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung. Reconfigurable Computing: Architectures and Design Methods. *Computers and Digital Techniques, IEE Proceedings*, 152(2):193 – 207, Mar 2005.