The Application of Moving Target Defense to Field Programmable Gate Arrays

John Dombrowski
University of South Alabama
School of Computing
Mobile, AL 36688

jad1107@
jagmail.southalabama.edu

Todd R. Andel
University of South Alabama
School of Computing
Mobile, AL 36688

tandel@
southalabama.edu

J. Todd McDonald
University of South Alabama
School of Computing
Mobile, AL 36688

jtmcdonald@
southalabama.edu

ABSTRACT

Field Programmable Gate Arrays (FPGAs) are powerful and flexible pieces of hardware used in a variety of applications. These chips are used in monitoring network traffic, guidance systems, cryptographic calculations, medical devices, embedded systems, as well as many other varied uses. They can be used in a large number of ways as well as in a large number of areas, which allows for nearly limitless applications. Outside of being used as a cryptographic processor and network monitoring, these chips are not being used to directly provide software/hardware security. FPGAs are extremely widespread in addition to becoming more integrated into the systems that they are a part of. This leads to vulnerabilities in almost every system that uses these chips. One way to combat these FPGA based vulnerabilities in every system using them, is to implement a Moving Target Defense (MTD) on the chip itself. FPGA based MTD would allow each FPGA to enhance, rather than weaken, the security of a system. This paper incorporates previous applications of Field Programmable Gate Arrays, and explores potential software/hardware security implementations for these chips through the application of Moving Target Defenses.

CCS Concepts

• Security and privacy → Security in hardware • Security and privacy → Intrusion/anomaly detection and malware mitigation → Malware and its mitigation

Keywords

Field Programmable Gate Array; Moving Target Defense; Partial Dynamic Reconfiguration

1. INTRODUCTION

An Application Specific Integrated Circuit (ASIC) is a customized chip that has a program hardwired as its logical circuit structure. A Field Programmable Gate Array (FPGA) is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

CISRC '16, April 05-07, 2016, Oak Ridge, TN, USA

© 2016 ACM. ISBN 978-1-4503-3752-6/16/04..\$15.00

DOI: <http://dx.doi.org/10.1145/2897795.2897820>

a chip implementation that functions like an ASIC; however, it is able to be reprogrammed and changed whereas an ASIC is not.

Regardless of implementation, circuits (as well as software) remain vulnerable to various types of attacks. Of particular interest in this paper are MATE attacks. Man-at-the-End (MATE) attacks are where a piece of hardware or software is obtained legally, or illegally, for the specific purpose of being studied in order to replicate or reverse engineer in order to successfully perform an attack on that software or hardware. A relatively new approach to combating MATE attacks is through what is called a Moving Target Defense (MTD). The idea behind a MTD is to make a successful attack harder to mount by regularly changing what is being attacked, or the environment that the attack is targeting.

An approach to incorporating MTD techniques into hardware relies on the ability to dynamically change the hardware at runtime. Partial Reconfigurability (PR) and Partial Dynamic Reconfigurability (PDR) are both actions that can be taken by FPGAs in order to reprogram how the FPGA works without having to reboot the entire FPGA. PR can be used to change part of the FPGA that is not currently being used and dynamic reconfiguration is where the reconfiguration does not cause any problems with the performance of the circuit. Combining both PR and dynamic reconfiguration is considered PDR [1]. Configuration changes can come in the form of circuit variance, where multiple circuits (called a variant) are all different in structure, yet are still functionally equivalent. Circuit variance may prove useful when paired with PDR on an FPGA.

This paper explores the potential of using a FGPA partitioning technique combined with PR and PDR, along with circuit variance in order to provide a hardware-based MTD.

2. APPLICATION SPECIFIC INTEGRATED CIRCUITS

An Application-Specific Integrated Circuit (ASIC) has the logical structure of a program hardwired as its design. This allows the ASIC to perform the program quickly and efficiently, mostly due to the fact that the circuit doesn't need to follow the normal instruction lifecycle as well as that the logical structures on the hardware can be optimized and tailored for their intended function. Let's say that an ASIC is being used to always handle floating point numbers, then the arithmetic logic structures can be made to handle floating point numbers as efficiently as possible. This is because the

ASIC will never have to handle integers, so if the ASIC is inefficient at handling integers it is a moot point. Since the ASIC has been optimized to handle the program's execution as efficiently as possible, the ASIC is using the least amount of power necessary to complete the operation. An ASIC can be used anywhere a circuit board, or even software, is doing the same task repetitively, such as vending machines, printers, radios, and cars for a few examples. The downside to an ASIC is that the cost required to develop one is so high as to be extremely prohibitive unless the production amount is at least in the thousands [10]. On top of the high cost of creating an ASIC, if any change needs to be made to the program running on it, then the entire structure of the ASIC will need to be reassigned and the new ASICs will need to be manufactured and distributed. Most hardware applications do not need the top end efficiency and speed provided by an ASIC, which leads to industry using FPGAs instead.

3. FIELD PROGRAMMABLE GATE ARRAYS

A Field Programmable Gate Array (FPGA) is an integrated circuit which has general logical structures that can be combined to execute a logic of a given circuit or piece of software. This is capable since the FPGA can specify how the logic structures are connected, which allows the FPGA to functionally produce a circuit design for any program that can fit within the number of logic structures available. FPGAs can be limited since every logical structure has to be created within the device, meaning that FPGAs cannot make use of pre-fabricated hardware, such as a General Purpose Processor (GPP). This has led to the creation and widespread use of an FPGA with a GPP already as part of combined integrated circuit. That type of FPGA is called a System on a Chip (SoC) and the industry is transitioning from FPGAs to SoCs because a SoC has all the benefits of both an FPGA and GPP, providing the ability to provide seamless hardware/software partitioning which could be used to increase software efficiency and even security.

FPGAs, as well as SoCs, have been used in security measures due to their increased execution speed and flexibility of application [9]. FPGAs are used in firewalls, high speed packet scanning [8], missile guidance, ASIC prototyping, radios, car infotainment systems, car backup-camera image processing, printers, routers, super computers, as well as many more applications. In an overall view of the differences between ASICs and FPGAs, it is clear that ASICs are faster, smaller, and more efficient than FPGAs [7]. However, FPGAs are much less expensive, have a wider scope of use compared to an ASIC, and can change what the logic circuit does without replacing it. All technology used today is vulnerable to attack and FPGAs or SoCs are no exception, which means that defensive measures must be put in place to mitigate attacks. One Such attack is a MATE attack, which is a common attack method used in software and hardware reverse engineering.

4. Man-at-the-End Attacks

A MATE attack is an attack performed by someone who has acquired a copy of the hardware or software and the license necessary to operate it, but is using that copy to find a way to attack the hardware or software in order to perform an act which the software or hardware that would not normally be

allowed. A few common MATE attacks are circumvention, brute force, entropy reduction, and probing attacks.

4.1 Circumvention Attack

A circumvention attack is an exploit that avoids the diversified parts of an obfuscated program. The attack is accomplished by using a *return-to-libc* attack, which uses code already in the program, to make code injection unnecessary and circumvents instruction set randomization defense, and other similar defenses [5].

4.2 Brute Force Attack

Brute force attacks are an exploit which tries every possible key value until it reaches one that works. This type of exploit works best with a smaller key space, and is practical with a small enough key space [5]. An Example of a brute force attack is a dictionary attack, which is where a list of possible usernames, passwords, or other keys are compiled. Once the dictionary is complete a program simply tries every entry, or every combination of entries, in the dictionary until it runs out or gains access.

4.3 Entropy Reduction Attack

An entropy reduction attack is an exploit which reduces a large key space, effectively shrinking the key space and allowing a brute force attack to be used. An example of an entropy reduction attack is a NOP (No Operation Performed) sled, which is where the attacker precedes the malicious code with a series of NOP instructions [5]. The reason for the NOP sled, the series of NOP instructions, is that if the code executes any of the NOP sled then it will slide from NOP instruction to NOP instruction until the malicious code begins. At which point the malicious code will be executed, which is the entire point of the attack.

4.4 Probing Attack

Probing attacks are an exploit which tries to find information about the randomized execution necessary to create a successful attack. One example of a probing attack takes place in a client-server environment, where the client keeps sending packets which jump to random addresses in an attempt to find the *libc* function *usleep*. The reason this function was the one targeted is that it would cause the connection to hang, which would let the attacker know it was found, and that all other *libc* function addresses can be calculated based upon the *usleep* location [5].

5. MOVING TARGET DEFENSE

Various methods for defeating MATE attacks have been created, and a few commonly used methods are obfuscation, tamper resistant hardware, and moving target defenses. Obfuscation is the process of making a piece of code, or a circuit, as difficult to comprehend as possible while still maintaining identical functionality. Tamper resistant hardware is hardware designed to make itself useless if tampered with, usually through physical destruction of the hardware or information dumps. Moving Target Defense is used to give attackers a more difficult environment to mount a successful attack in.

Basic idea behind a Moving Target Defense (MTD) is to reduce one of the largest weaknesses of any software or system. Once most pieces of software, or systems, are put in place, they are left in static form to run and are not changed unless it fails or

needs maintenance. This means that an attacker can spend as much time as they need collecting data, probing, learning, and analyzing the system to find a weakness. Since the program or system doesn't change, it means that the data being collected is still valid and useful months after it was gathered [4]. An MTD has enough of what is normally stored statically in a system stored dynamically, effectively creating a non-static environment. A system employing MTD is in constant change and doesn't allow attackers to learn about one particular system until a vulnerability is found, but will render the data collected about it useless since it might not be current information about the system once an attack vector has been discovered. Normally MTDs have been implemented only in software, but there is a feature of FPGAs and SoCs which may allow for a MTD to be implemented at the hardware level. This enabling feature is referred to as partial dynamic reconfiguration.

6. PARTIAL / PARTIAL DYNAMIC RECONFIGURATION

An FPGA is able to be reprogrammed by updating the files used to tell the FPGA how to operate. Traditionally, the entire circuit must be powered down in order for the changes to be implemented. Partial Reconfigurability and Partial Dynamic Reconfigurability is the ability of an FPGA to change its logic functionality without having to be powered down. There are a few ways that PR is currently supported, the two main types are difference based, and module based reconfigurations [6].

Difference based PR is where the changes being made are implemented by telling the FPGA what the difference is between the current circuit and the new one. Module based PR is where there are modules of code, which have the same input, output, and functionality. The modules of code then are able to be swapped in and out without interfering with the rest of the circuit. PDR allows the non-operating logic portions of the FPGA to be reprogrammed without requiring a power cycle.

An FPGA can undergo PR and PDR without stopping execution, as long as the part being reconfigured isn't in use while the update is happening [3]. This allows FPGAs to be changed dynamically during execution, and when paired with circuit variants or program partitioning [2] allows for MTD at the circuit level. Having a partitioned program on a SoC could even be made to mimic scheduling specific parts of code to run on a co-processor for performance increases [8]. This provides the ability for having partitions designed to run on the GPP of the SoC, as well as having partitions designed to be synthesized and run on the FPGA part of the SoC. The gain in partitioning afforded by this will increase the mutability of the program running on the SoC. This only works if there are multiple ways to implement the same code as different circuitry on an FPGA, and preferably in similar amounts of real estate that the circuit variants occupy.

PR and PDR combined enabled as an MTD is the basis for our proposed MATE attack security measures.

7. Using PDR to counter MATE attacks

A program can be represented logically as a circuit, which also means that equivalent circuits can be created where all of the circuits will have the same functionality [2]. Combining PDR with circuit variants, and possibly program partitioning, will

allow a SoC to have a dynamically, and probably constantly, changing circuitry implementation of a program without effecting the functional aspect. The original circuit will be broken up into partitions, each partition will have circuit variants created and loaded onto the memory accessible by the SoC as illustrated by Figure 1 [8].

Figure 1: Hardware-Based Software Protection [8]

Each time a piece of code is executed, the corresponding circuitry will be swapped with a random variant. With a sufficiently large amount of variants, it will make a MATE attack infeasible due to the inability to gather valid information on how the program or the circuitry operates. A circumvention attack would be mitigated if the part of the circuitry used to perform *return-to-libc* attack was on the FPGA and not the GPP. A brute force attack would be mitigated by the circuitry changing after every execution, which means that, unless the attacker was extremely lucky and the first key tried was correct, it would not succeed due to it not being able to try multiple keys. An entropy reduction attack would be similarly mitigated by the circuitry changing. Even if the attacker was able to have their malicious code in the software before it was synthesized to the SoC, the circuitry variation would effectively spread out the NOP sled and disconnect the NOP sled from itself. A probing attack would be pointless because the point of a probing attack is to try and find current, useful information about the software, such as where a piece of *libc* is stored in memory. The attack will return out of date information, since the SoC will have changed once the probe is executed.

8. SUMMARY

MATE attacks pose serious threats to software and hardware systems. This paper's proposed solution to these attacks is to use program partitioning and circuit variance in conjunction with a SoCs PDR to mitigate MATE attacks. The program intended to run on the SoC would be partitioned into sections, each section would have multiple circuit variants of it created and loaded onto the SoCs memory. Once a partition of code was finished executing or an amount of time has passed since a partition was executed, the SoC would dynamically load a new circuit variant of that partition of the program. This will be a circuit level implementation of an MTD. This MTD should mitigate MATE attacks and be a functional, as well as novel, implementation of an MTD. It may also prove to be more tamper resistant than a software implementation of a MTD, due to the requirement for physical access to the SoC, the possibility of including tamper resistant hardware, and for the advanced knowledge required for circuit analysis and reverse engineering.

9. ACKNOWLEDGEMENTS

This material is based in part upon work supported by the National Science Foundation under Grant No. DUE-1241675.

10. REFERENCES

- [1] Abel, N., Manz, S., Grull, F. and Kebschull, U. Increasing Design Changeability Using Dynamic Partial Reconfiguration. *IEEE Transactions on Nuclear Science*, 57(2). 602-609.
- [2] Anandel, T.R., Whitehurst, L. and McDonald, J.T. Software Security and Randomization through Program Partitioning and Circuit Variation. in *Proceedings of 1st Workshop of Moving Target Defense*, (Scottsdale, Arizona, 2014), 79-86.
- [3] Bacon, D.F., Rabbah, R. and Shukla, S. FPGA Programming for the Masses. *Communications of the ACM*, 56(4). 56-63.
- [4] Carvalho, M., Bradshaw, J.M., Bunch, L., Eskridge, T., Feltovitch, P.J., Hoffman, R.R. and Kidwell, D. Command and Control Requirements for Moving Target Defense. *IEEE Computer Society*, May/June 2012. 79-85.
- [5] Jajodia, S., Ghosh, A.K., Swarup, V., Wang, C. and Wang, X.S. Moving Target Defense Creating Asymmetric Uncertainty for Cyber Threats. Springer Science Business Media, New York, 2011.
- [6] Kao, C. Benefits of Partial Reconfiguration. *Xcell Journal*, Fourth Quarter 2005. 65-67.
- [7] Kuon, I. and Rose, J. Measuring the Gap between FPGAs and ASICs. in *FPGA'06*, (Monterey, California, 2006), 1-10.
- [8] McDonald, J.T. and Anandel, T.R. Program Partitioning and Polymorphic Functions: A Two-Pronged Approach for Program Protection Research. in *ARO Workshop on Continuously Upgradeable Software Security and Protection*, (Scottsdale, Arizona, 2014), 1-2.
- [9] Whitehurst, L., Anandel, T., McDonald, J.T. and Al-Assadi, W. Enhancing Security against Software Attacks with Reprogrammable Hardware. *Journal of Information Security and Applications*. 1-20.
- [10] Woo, P. Structured ASICs – A Risk Management Tool. *eASIC® Corp*. 1-6.