Morpheus II: A RISC-V Security Extension for Protecting Vulnerable Software and Hardware

Austin Harris[†], Tarunesh Verma[‡], Lauren Biernacki[‡], Alex Kisil[∅], Misiker Aga[∅], Valeria Bertacco[‡∅], Baris Kasikci[‡], Mohit Tiwari[†], and Todd Austin[‡∅]
University of Michigan[‡], University of Texas[†], Agita Labs[∅]

I. Toward More Durable Security

With the growth of cloud computing and IoT, data security has never been more important. With cloud computing, we hand over our personal and private information to cloud providers and their customers, and we can only hope that they steward our data well. For IoT devices, we install them everywhere in our home, cars, and workplace, and then we trust these devices to not spy on us. We are all extending much trust to the manufacturers and vendors of computing systems today, and in many cases they are letting us down. Yet, the world of computing is replete with examples of data breaches and poor stewardship of sensitive data, suggesting that stronger security measures are surely needed.

Today's Defenses Lack Durability: In modern computer security, they are two primary means be which systems are protected. The first is ***patch-based security*** approach where software and hardware vulnerabilities are addressed by patching the system's software. The key challenge with this approach is that attacks will not stop until the system is free of vulnerabilities. Unfortunately, the complexity of modern software and hardware, combined with the rate at which new software is created, ensures that patched systems always have plenty of additional vulnerabilities for motivated attackers to exploit.

A more powerful approach is to outfit the system's software or hardware with a ***targeted defenses against well-known attacks***. Examples of these defenses include no-execute stacks, which prevents code injection on the stack, or Intel's Cache Allocation Technology (CAT), which silences covert channels in the last-level cache. These defenses are superior to patch-based defenses because they can typically defend against entire class of attacks. However, targeted attacks often have limited scope, thus, attackers will devise ways to step around these defenses. For example when no-execute stacks where introduced, the attack community quickly perfected techniques to inject code into the heap with heap-spray attacks.

Unfortunately, the sum total of today's security defense only throws moderately-strong barriers into the face of oncoming attackers for existing attacks. If new "zero-day" vulnerabilities are discovered, systems are completely unprotected. As such, there is a great need for new thinking in computer security, in particular, for defenses that are more durable against a fast-growing slate of security attacks.

Morpheus Defenses Work Despite Vulnerabilities: The Morpheus security technology works in a vulnerability-agnostic fashion [1], allowing it to stop attacks on vulnerable software and hardware. Where traditional security defense focus on specific vulnerabilities, Morpheus defenses instead obfuscate the information assets required by attacks. This approach denies attackers timely access to the critical information assets needed to attack systems. The critical information assets that Morpheus can protect include the following:

- code representation
- code and data pointer representation
- code and data layout (both absolute and relative)

Information assets are protected using ***encryption and churn*** to protect critical information assets. By encrypting code and pointers, attacks lose the ability to find code gadgets, inject pointers into the stack, perform relative address attacks, and so on. Yet, savvy attackers can adapt to even high-entropy encryption by utilizing memory disclosures and side-channel attacks to eventually acquire the information assets they need to attack a system. Consequently, Morpheus uses churn to re-key the cipher used to protect information assets on a regular basis, thereby destroying any disclosed information assets.

A key aspect of Morpheus is that it only protects information assets within the program and microarchitecture implementation. These assets possess undefined semantics because they are the internal workings of compilers and microarchitectures. Consequently, encrypting and churning these assets, while breaking attacks, has little to no effect on normal software. This property allows Morpheus to imbue defenses into vulnerable software and hardware without putting undue burdens on programmers and system software.

It is quite challenging to find any attack that doesn't utilize some subset of these critical information assets. Thus, protecting these information assets broadly stops security attacks [2], ranging from control-flow attacks, to privilege escalation, to side-channel attacks and beyond. Moreover, it is likely that attacks discovered in the future will also utilize some subset of these information assets, thus, it is possible that Morpheus systems could have some measure of immunity from attacks in the near future.

II. The Morpheus II Architecture

Morpheus II is a refinement of the original Morpheus design [1]. Morpheus II was developed in the DARPA SSITH program, which had a requirement that designs be placed into the DARPA FETT program, where it was to be red-teamed for potential vulnerabilities. Thus, while original Morpheus had significant concerns and design aspects related to performance, the goal of Morpheus II was to capture the full security strength of Morpheus defenses even if occasional sacrifices were made in overall performance. Despite this focus on security, the performance impacts of Morpheus II are moderate, and we are confident they could be minimized with the attention of an industrial-strength design team. The Morpheus II architecture is implemented as a RISC-V extension applied to the Rocket Core pipeline.

Additionally, Morpheus II was tuned to primarily stop remote code execution (RCE) attacks, which are a high-value class of attacks that allow remote attackers to inject code into vulnerable machines. For example, Morpheus II RCE defenses would handily stop the Microsoft Exchange Server RCE attacks that were in the news as we the time of this writing. As another example, one of the largest security breaches in US history, the 2017 Equifax breach, was also initiated via an RCE attack on a vulnerability in the Apache Struts library that would have been easily stopped with Morpheus II defenses.

Morpheus II Architectural Features: Morpheus II implements ***always-encrypted code pointers***. A single-cycle twelve-round Simon cipher uses a randomly generated key to strongly encrypt all code pointers (*i.e.*, function pointers and return pointers). From attacker's perspective, all code pointers are encrypted all the time. As such, code pointers in DRAM, caches or registers are encrypted. The only time

Instruction Class	Example	Semantics
Arithmetic	enc_add r1, r2, 4	r1 = enc(dec(r2) + 4)
Relational	enc_sleq r1, r2, r3	r1 = dec(y) $\leq$ dec(r3)
Indirect Jump	enc_jalr r2, LR	LR = enc(PC), PC = dec(r2)

Table I: Morpheus II RISC-V ISA Extensions. Morpheus II adds three classes of instructions: *i)* always-encrypted pointer ALU operations, *ii)* decrypting pointer relational tests, and *iii)* decrypting indirect jumps and returns. Note that the *enc()* and *dec()* interfaces encrypt and decrypt always-encrypted pointers within the pipeline.

that code pointers are decrypted are in the cycle immediate before a function pointer or return address is placed into the PC register. In the Rocket core pipeline, this requires an additional stage in the JALR instruction implementation, extending its latency by one cycle.

When code pointers are always encrypted, it complicates the attacker's ability to forge code pointers. Additionally, relative address attacks become difficult to synthesize because computing a relative address from an encrypted pointer is a cryptographically hard problem. Since all RCE attacks involve some form of code pointer injection or manipulation, these attacks become significantly more challenging.

In addition, Morpheus II implements always-encrypted code. Like code pointers, a twelve-round Simon cipher uses a randomly generated key to strongly encrypt all instructions. Code remains always encrypted in the binary, DRAM, and all caches. Thus, instructions are only decrypted in the pipeline for execution. This decryption is implemented with a single decryption stage immediately before the instruction decode stage in the Rocket core pipeline. By always encrypting code, it becomes very challenging for attackers to identify code gadgets or synthesize new code for injection.

To thwart disclosures and side-channel attack, Morpheus II churns encryption keys whenever the system boots, reboots, or when a security violation has been detected (*e.g.*, segmentation fault or misaligned instruction fetch). On each of these events, the system is very quickly warm-booted, which reload the code under a new encryption key, and reconstitutes all code pointers from the original binary, again under a new encryption key. The churn process ensures that any valuable information gathered by attackers since the last churn cycle will be lost due to the re-keying of the Simon ciphers. In addition, by limiting churn to system warm boots, Morpheus II did not require tagged memory, which significantly reduced the complexity of the changes needed for the Rocket core, which in turn led to a fast design cycle for a small academic design team.

III. Performance, Power and S/W Impact Analysis

The Morpheus II design was deployed into an Amazon F1 Xilinx UltraScale Plus FPGA. The Morpheus ISA extensions and extra decryption stages were added to the RISC-V Rock et Core design. The power increase due to the extra logic and ciphers was only 0.8% for the entire Rocket Core design, including the DRAM controller and the XDMA PCIE bus controller. If only the Rocket Core pipeline is considered (about 20% of overall power), the relative power overhead increases to about 4%. Area overheads were uniformly low, at only 0.2%, measured in increased FPGA resource utilization.

Performance impacts were quite low. The decryption stages add on additional stage to the front-end of the pipeline for instruction decryption, which increases branch misprediction latency from 4 to 5 cycles. Additionally, an extra cycle of latency is exposed on all indirect jumps and returns. For a system with a Gshare branch predictor and infrequent indirection, the overheads will be well below 5% slowdown. For our deployed system running a network facing application, the performance impacts were in the range of our measurement noise.

Software and design impacts were also low. The changes necessary to LLVM to support Morpheus II compilation was less than 1k lines of code. In addition, the changes to the Chisel code to accommodate the Morpheus II extensions on the Rocket Core totalled only 369 additional lines, including the cipher engine. Finally, few software changes were required in the software running on the Morpheus II system for the FETT Challenge. Our platform was running the Michigan Mock Medical Database (M3DB) running on a FreeRTOS web server with SQLite database, totaling more than 200K lines of code. To accommodate Morpheus II defenses, only three lines of code needed to be changed.

IV. Morpheus II Security Analysis

In the summer of 2020, the Morpheus II architecture was entered into the DARPA FETT Challenge [3], along with three additional teams. The security firm Synack ran the challenge with their crowd-sourced "researchers" providing red-team testing. Galois adapted FireSim for the test hardness, which allowed the Morpheus II design to be deployed into the Amazon AWS F1 cloud instance. When a researcher attacked one of our machines, they would spin up an AWS F1 instance with a DARPA SSITH secure core.

All teams were required to stand up a challenge application with known software vulnerabilities. We built the Michigan Mock Medical Database (M3DB), which was a mock COVID-19 patient database for medical research, with a network-facing REST interface for querying the patient data in a differentially private manner. Attackers had to penetrate this database and modify or exfiltrate patient data to successfully attack the Morpheus II target.

All teams were requested to provide known software vulnerabilities in the code base, with ours being in the FreeRTOS and SQLite software packages. This requirement grew out of the DARPA SSITH program, which was focused on hardware techniques for protecting vulnerable software. In addition, each team had make the entire build environment available to the researchers, so they could install and build their own code for testing and study purposes. Each team also had to build a mission list, which was a step-by-step guide detailing easy-to-hard attack scenarios. Bug bounty payouts were gauged to the mission difficulty. DARPA paid bug bounties for finding vulnerabilities, which were never disclosed, but DARPA did say publicly that the bounties went at least to $50k.

During the three months of the FETT challenge, the Morpheus II target had 535 researchers trying to penetrate it. Despite being the second most attacked target in the FETT program, the ***Morpheus II target was never penetrated and no vulnerabilities were discovered***. By the end of the program, DARPA disclosed that 10 vulnerabilities were discovered by the research (on targets other than Morpheus II).

V. Conclusions and Future Directions

Our efforts are now turning toward defining a bona fide RISC-V extension to protect pointers. Additional considerations are entering into the design, including support for virtual memory and per-process keys with preemptive scheduling. In addition, an effort at Agita Labs is working to commercialize a derivative of the Morpheus technology for the Azure and AWS clouds.

References

[1] M. Gallagher, L. Biernacki, S. Chen, Z. B. Aweke, S. F. Yitbarek, M. T. Aga, A. Harris, Z. Xu, B. Kasikci, V. Bertacco, S. Malik, M. Tiwari, and T. Austin, "Morpheus: A Vulnerability-Tolerant Secure Architecture Based on Ensembles of Moving Target Defenses with Churn," in *Architectural Support for Programming Languages and Operating Systems*, 2019.

[2] L. Biernacki, M. Gallagher, Z. Xu, M. Tadesse Aga, A. Harris, S. Wei, M. Tiwari, B. Kasikci, S. Malik, and T. Austin, "Software-driven security attacks: From vulnerabilitysources to durable hardware defenses," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 2021.

[3] "Finding exploits to thwart tampering (FETT) bug bounty," https://fett.darpa.mil/.