THE CYBERSECURITY STACK

WHITE PAPER

How to Secure Embedded Systems with a Defense-in-Depth Approach

WRITTEN BY:

Jothy Rosenberg

Founder & CEO

JUNE 2019

DOVER MICROSYSTEMS

Someone just used a new weapon, and this weapon will not be put back into the box.

Michael Hayden
FORMER DIRECTOR OF THE NSA & CIA

Michael Hayden, the former head of the NSA and CIA, stood at a podium and recounted his experience learning of the Stuxnet computer worm and its ability to cause real damage in the physical world.

"I understand the difference in destruction is dramatic, but **this has the whiff of August 1945**," Hayden said, CSPAN cameras rolling. "Someone just used a new weapon, and this weapon will not be put back into the box."

This quote from Hayden's speech puts into perspective the nation's mindset and posture in 2010 when the Pentagon compelled DARPA to fund its $100 million **CRASH program**—a program with a sole purpose of finding a solution that could defend against cyberattacks, like Stuxnet, that come across the network.

As Hayden would say, we're now living in a "post-Stuxnet world," and our cyberdefenses must evolve to meet this new reality. This is why Dover Microsystems exists. Our founding team led the largest prime contract in DARPA's CRASH program, and over the last nine years we've developed our CoreGuard® technology—that includes research conducted at **Draper**—into a product fit for commercial use.

The focus of this white paper, however, is not CoreGuard. Instead, it's the stack model we developed to make sense of today's cybersecurity solutions landscape, and to construct a framework for properly defending embedded devices through a defense-in-depth approach.

In this paper, we'll describe our inspiration for the stack, and give a detailed breakdown of how a defense-in-depth approach works. We'll also highlight a long existing gap in the stack, which DARPA's CRASH program was able to identify and CoreGuard technology is able to fill.

What Is The Cybersecurity Stack & Why Do We Need It?

A stack is a useful conceptual model

In the computer technology space, models are important conceptual tools for simplifying our understanding of large, complex problems. The cybersecurity landscape is particularly wide and diverse. By applying a stack model to the solutions in the cybersecurity space, we can decompose its large universe into smaller, distinct components represented by layers. The result is a cybersecurity stack model that simplifies the solutions landscape, and illustrates the defense-in-depth methodology required for securing today's embedded systems.

To get a better picture of how the cybersecurity stack functions, let's consider what is perhaps the most famous computer technology conceptual model represented as a stack—the Open Systems Interconnection (OSI) model.

The OSI model is a framework that illustrates the interoperability of diverse communication systems with standard telecom and computing protocols. It takes something very complicated—seven layers of technology—and organizes it into a stack based on the way each layer communicates with other layers.

Each layer in the OSI model passes and receives information to and from the layers it is sandwiched between, or the layer directly above or below it respectively in the case of the top and bottom layers.

The classic OSI model is shown on page five.

We used the OSI model as the inspiration for our cybersecurity stack because it is built from the ground up, and it demonstrates how each piece of a complicated puzzle fits together.

defense-in-depth | dē,fens-in-depTH |

phrase of defense

1 the use of multiple layers of security to protect a system. Also known as "The Castle Approach," this type of strategy used with IoT aims to build an impenetrable fortress around a vast and valuable landscape of embedded systems.

OSI MODEL

APPLICATION HTTP	High-level APIs
PRESENTATION XML	Translation of data between a networking service and an application
SESSION Web application dialogs	Managing a continuous exchange of information between two nodes
TRANSPORT TCP, Ipsec, UDP	Reliable transmission of data segments between points on a network
NETWORK Packets, Routing, IP	Addressing, routine and traffic control for a multi-node network
DATA LINK MAC, Ethernet, Zigbee, WiFi	Reliable transmission of data frames between two nodes connected by a physical layer
PHYSICAL Pins, Voltage, Half Duplex, Full Duplex	Transmission and reception of raw bit streams over a physical medium

Today's Cybersecurity Stack

There are two important truisms about cybersecurity today that are rarely discussed. One is that almost all cybersecurity solutions are based in software, and software is inherently vulnerable to attack (think "patch and pray"). The other is that there is no one-size-fits-all solution to cybersecurity, meaning a layered approach is a must.

The cybersecurity stack we present in this white paper is meant to address both of these truisms, by:

1. Demonstrating what kinds of solutions are needed to secure an embedded system;
2. Highlighting the pros and cons of each solution; and
3. Explaining how each layer works to increase the security of the next layer above it, thus closing some of the holes that land in the "cons" category.

The cybersecurity stack is the ideal tool for a System on Chip (SoC) designer who needs security but is unsure where to start. The stack gives a road map to what security layers they need, as well as the risks associated with leaving certain layers unfilled.

Let's take a look at each layer of the stack, starting at the bottom and moving up.

CYBERSECURITY STACK

SOFTWARE	**APPLICATION** Credentials, Sanitization	Manages credentials for authorized users and runs sanitation routines that check SQL queries and other data integrity issues
	KERNEL Operating System, Intrusions, Virus Scans	Scans for malware signatures, detects intrusions, and prevents unauthorized access to a network
	ENCRYPTION Communications, Data-in-Motion	Prevents data theft by converting plaintext data into ciphertext code that can only be reconstituted using an authorized key
	COMPARTMENTALIZATION Hypervisors, Zones, TEE	Isolates critical pieces of software in a sandbox so they cannot be corrupted
HARDWARE	**ROOT OF TRUST** Keys, Secure Boot, Crypto	Validates all the hardware and software on the system at boot time
	PHYSICAL Tamper, Supply Chain, Rad-hard, Fault Tolerant	Ensures unauthorized personnel cannot touch or tamper with the system in the real world

PHYSICAL

Tamper Protection, Supply Chain, Rad-hard, Fault Tolerant

Cybersecurity, like a fortress, must be built from the ground up because without a secure foundation, the rest of the defense structure will fall. The physical layer of protection is a lot like our defense-in-depth foundation—and it is unique to each use case.

In some instances, like with sensitive government servers, the physical layer of protection might go beyond the device itself to include a locked cabinet or room, or even bodyguard protection.

In most cases, however, the physical layer contains tamper-protection for when the device cannot be secured under lock and key. It goes even deeper with supply-chain protections and tamper detection, as well as fault tolerant mechanisms like redundant processors to handle hardware failures. If the device goes into space, for example, it must be radiation-hardened to prevent cosmic radiation from causing faults that look like attacks.

The main objective of this physical layer is to make sure that unauthorized personnel cannot touch or tamper with the device in the real world in order to cause alterations to the device's digital assets, like its operating system and/or application software.

ROOT OF TRUST

Keys, Secure Boot, Cryptography

While the foundation of our cybersecurity defenses might be built and deployed in the physical world, Root of Trust (RoT) is where our traditional understanding of cybersecurity begins.

This layer of the stack is aimed at protecting a device during its most vulnerable of states: boot time. The RoT contains the public/private key pair for asymmetric encryption, and includes basic cryptographic functions needed for security operations like hash functions and random number generation.

The RoT also validates that all of the hardware is the hardware it says it is, and that all the software running on that hardware is valid and unaltered.

Aside from ensuring the device doesn't fall into the wrong hands (physical layer), the RoT layer is one of the most important.

THE GAPING HOLE IN THE STACK

There is a dangerous chasm between the hardware foundation and the software stack above it. We'll come back to how Dover's CoreGuard solution fills this hole, but first let's continue climbing through the software layers.

COMPARTMENTALIZATION

Hypervisors, Zones, Trusted Execution Environment

One of the most popular security solutions deployed today, in one form or another, is compartmentalization.

Compartmentalization can be thought of as sandboxing, where critical pieces of software that must not be corrupted are isolated from less critical pieces of software, such as user applications. The use of compartmentalization was made popular by Android smartphones, whose app store was not curated and as a result offered applications that were infected with malware that could potentially corrupt the OS and carrier software. By putting the Android OS and carrier software in one "trusted" compartment, and the user applications in another "untrusted" compartment, manufacturers could avoid the possibility of cross-contamination.

Compartmentalization is still used in a similar way today, although some vendors have gone beyond bifurcating into single trusted and untrusted zones to instead allow for a seemingly endless number of compartments. While compartmentalization is supported by hardware, it is still part of the software stack because the compartments contain software that is controlled by the software application layer.

ENCRYPTION
Communications, Data-in-Motion

Encryption has become essential in today's cybersecurity stack, as more and more of our communication takes place in the digital realm. Encryption is done by a user with an authorized key that is shared only with the intended recipient of the encrypted data. Encryption takes plaintext—data that can be read and understood—and converts it into a jumble of letters, numbers, and special characters called "ciphertext." That ciphertext is undecipherable by anyone without the matching authorized key.

When data is encrypted on one end of a system (from the sender) and travels out of that system (to the receiver), without being decrypted and re-encrypted along the way, that is known as end-to-end encryption.

KERNEL
Operating System, Intrusions, Virus Scans

The kernel layer is where our operating system lives.

Cybersecurity solutions in this space consist of the built-in protections that ship with most operating systems. These include memory partitioning, as well as system-level protections like the strict separation of one process from another. Additionally, the kernel layer is where popular security solutions like firewalls, intrusion detection, and virus scanning are deployed.

Both the encryption and compartmentalization layers below the kernel layer help reinforce many of the traditional solutions that are deployed at this level. For example, compartmentalization helps ensure the strict separation of processes.

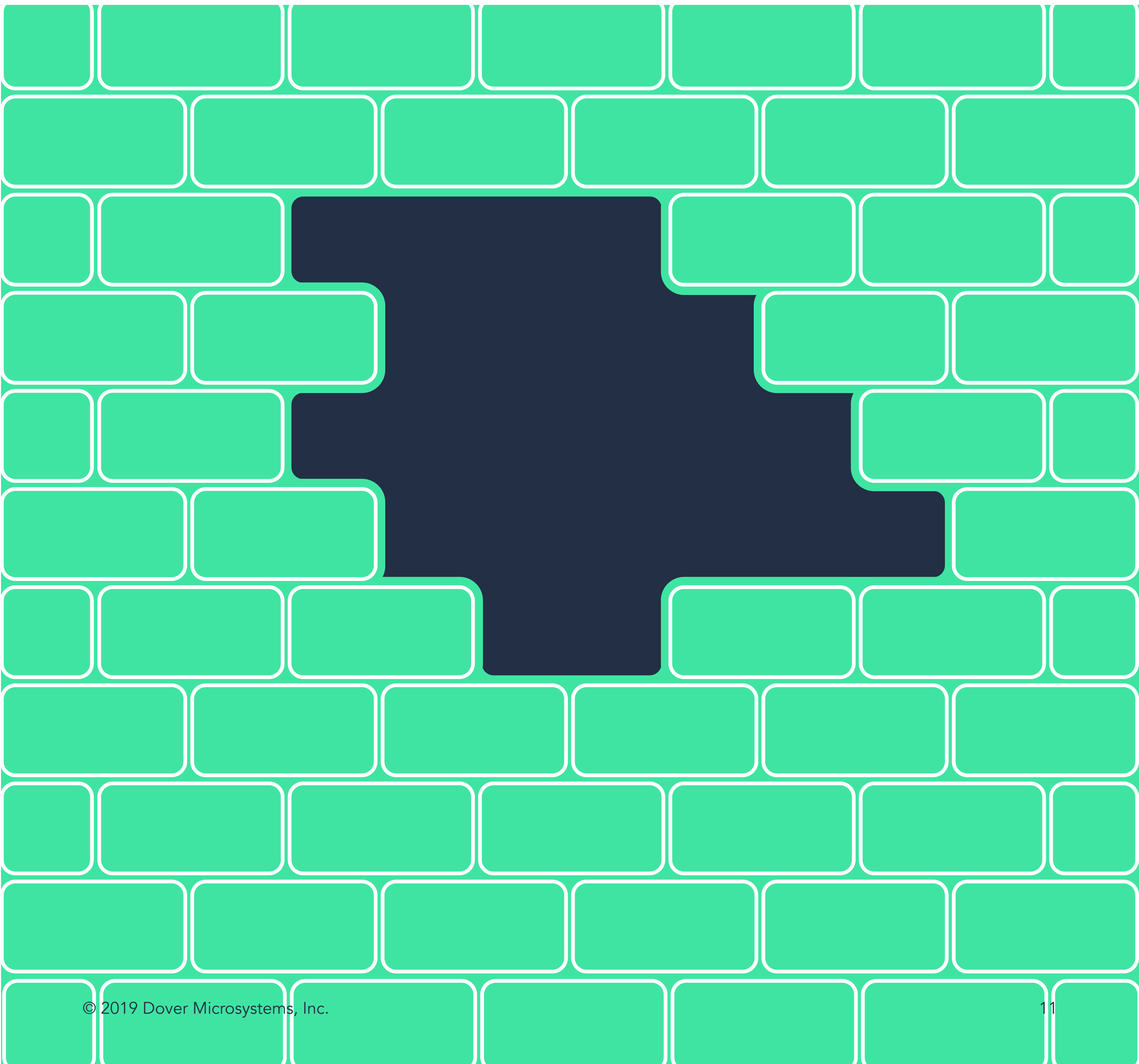
APPLICATION
Credentials, Sanitization

The application layer is where users interact with systems.

Cybersecurity solutions at this layer are responsible for a number of functions, such as managing credentials for authenticated users, running sanitation routines that check SQL queries, calling privileged encryption functions, and checking external data to be used in critical decision-making functions, such as those used in AI and machine learning applications.

As already mentioned and shown in the diagram, today's cybersecurity stack is incomplete. The missing layer makes it vulnerable because the stack's integrity depends on each layer being filled. If we leave a layer unfilled, each layer above it is vulnerable. For example, if we can't verify the supply chain, then we can't verify RoT—and without RoT we can't be sure anything our device is doing is safe.

Because the missing layer bridges the hardware and software layers of the stack, neglecting it leaves every software layer above it vulnerable to attack. To best illustrate this risk, and to shed light on the exploitable bugs the dangerous chasm exposes to attackers, let's look at how each software layer of the cybersecurity stack might be attacked today.



How Attackers Target The Upper Layers Of The Stack

While attackers certainly target the RoT and physical layers of cybersecurity, it's important that we stop for a second and put into context just how often the upper software layers are hit. The compartmentalization, encryption, kernel, and application layers aren't just where named attacks like Stuxnet, Heartbleed, and **WannaCry** occur. They're also where many of the most common categories of attacks—like buffer overflows, code injection, control-flow hijacking, SQL injection, and data exfiltration—occur.

This has led to an environment where cybersecurity breaches happen 44 times per second, where ransomware attacks occur every 14 seconds, and where annual cybercrime damages are projected to **approach $6 trillion by 2021**, according to Cybersecurity Ventures.

With that in mind, let's examine these software layers and how bad actors are infiltrating them.

APPLICATION ATTACKS

VULNERABLE? YES, VERY.

Most of today's cybersecurity ecosystem is made up of software solutions that run at the application level. This is likely due to the fact that attacks against applications are very common.

An application's greatest vulnerability is that it is based in software and all software is vulnerable. According to Steve McConnell in his book, *Code Complete*, there are **15 to 50 bugs per 1,000 lines of delivered code**. To give that statistic some context, consider this: the Android operating system has 15M lines of code, Windows 7 has 40M, and a Ford F-150 is off the charts with 150 million lines of code. Furthermore, the FBI has found that at least two percent of all bugs can be turned into exploitable vulnerabilities, aka cyberattack vectors.

ATTACK PATTERNS

Attackers use automated tools bought or accessed for free on the Dark Web to probe applications for exploitable vulnerabilities, including those in the software security applications running on the system.

Microsoft researchers have discovered that **70 percent of all exploitable bugs in their products** are memory safety vulnerabilities. This aligns with the research reflected in MITRE's Common Weakness Enumeration (CWE) list, which identifies the "classic buffer overflow" as the most common memory safety vulnerability. A buffer overflow occurs when a program is able to write more data to a buffer—or fixed-length block of computer memory—than it is designed to hold.

What attackers desire most is to find a memory safety or other type of vulnerability in software that is running at a high-privilege level. Many security software packages run at root privilege so they are highly-prized targets. If the attacker can complete a buffer overflow attack and overwrite a return address such that the CPU is directed to execute code they injected, and if the target is running at root privilege, then the injected code will run as root as well; at that point, all bets are off and the attacker can do pretty much anything they want on the system.

IN THE WILD

Application-level attacks are the ones we see most frequently in the wild, and often the ones that rise to the level of the consumer's consciousness. Most famously, a SQL injection was used in the enormous **Equifax breach that affected 145 million consumers.**

In a SQL injection attack, data from outside the system is used to construct a command that is sent to a back-end database system. If the incoming data is not sent through a sanitization routine, potentially harmful commands can be sent to the database.

In this case, attackers were able to find a weakness in the open-source Apache Struts 2 framework. Apache closed the hole in their software with a patch just three days after it was discovered, but Equifax never implemented the patch. This gave attackers an opening to use SQL injections, with relatively unsophisticated payloads, to demand sensitive financial information from the system and compromise hundreds of millions of users in the process.

KERNAL ATTACKS

VULNERABLE? YES.

The kernel of an operating system runs at a high-privilege level, which makes it a prime target for attack. As with security software running as a highly-privileged application, finding an exploitable vulnerability in the operating system is the holy grail among bad actors because it gives the attacker near total control over the device they've targeted.

ATTACK PATTERNS

An operating system is a sophisticated piece of software, comprised of millions of lines of code. This translates to an average number of exploitable vulnerabilities that increases with each additional line of code written. Because of this, bad actors will target the kernel level much in the same way they target software vulnerabilities at the application level—the difference here being a potentially bigger payoff.

A successful attack on an OS, that results in root-level access, allows attackers to do anything they like with the system at hand. This might include installing malware for an advanced persistent threat attack, encrypting the entire disk in a ransomware attack, spreading an attack to connected machines, or becoming part of a bot army to perform a distributed denial of service (DDoS) attack.

IN THE WILD

Kernel-level attacks can be used to compromise a single system or, as is becoming more frequent, to compromise a number of systems to create a botnet that can do even greater damage in the real world. Such was the case in October 2016, when kernel-level attacks were used to compromise IoT devices and use them in a **DDoS attack on Dyn**.

Dyn, one of the world's largest DNS providers, was targeted by bad actors who, using the Mirai framework, had strung together a botnet that sent bogus traffic to Dyn in an attempt to overload their servers and crash the DNS service. They were successful, and it resulted in a denial of access for everyday users to popular websites, like CNN, Twitter, and Netflix.

The Dyn attack was possible because the perpetrators targeted connected surveillance cameras that had no security features and ran the Linux OS for which Mirai was designed. This made it easy for attackers to commandeer these devices and use them in a way that was not intended.

ENCRYPTION ATTACKS

VULNERABLE? YES, INDIRECTLY.

Cryptography—the process of encrypting and decrypting data—is technically impenetrable, and has only been broken in a purely academic sense. It would require the use of a quantum computer.

But that doesn't stop bad actors from trying to hijack encryption routines.

In fact, the **dirty little secret about encryption is that it's vulnerable** to over-the-network attacks. This is because whether encryption is realized in a hardware element on the SoC, or as a runtime software routine, it is always accessed as a function call from the application that requires its services.

When done through software, that call vectors directly to the software runtime encryption routine. When done through hardware, the operating system provides the encryption hardware with the data that must be protected, as well as the call routine that notifies the OS that the requested encryption is complete. The OS also provides the location of the buffer containing the now encrypted data.

ATTACK PATTERNS

An attacker who wishes to exfiltrate data from such a system cannot do so when the calls to encryption are allowed to operate as designed. So instead they try to find a flaw in a software component (application, operating system, runtime, security software, etc.) where they can launch a **buffer overflow**.

Once they have gained access, they can modify the prescribed flow of the application such that it skips over the call to encryption altogether. This is easily done by finding and modifying the call instruction for the encryption function and turning that into a "no operation" instruction. Alternatively, if they send out fake data instead of stealing the legitimate data that was encrypted, they can change the input buffer to be encrypted with their data and leave the call to encryption intact.

IN THE WILD

While one would be hard pressed to find any instance of encryption being cracked in the wild, it's not so difficult to make the argument that encryption is involved in most cyberattacks. Although we can't point to an exemplary attack as a rule of thumb, understanding encryption's role in cyberattacks will help us understand how bad actors are approaching this layer of the stack.

There are two ways in which encryption is used in cyberattacks today: it's either a vehicle or the prize. Attackers are either using encryption to send malicious traffic to unwitting victims, or the goal of the attacker is to get their hands on whatever is being protected by encryption on the device.

In the latter sense, most software-based attacks aim at getting access to a device at a level that will allow them to control the calls to the encryption routine. This allows attackers to circumvent said routine and either steal sensitive data or disguise malicious data as legitimate.

In the former sense, attackers use encryption to conceal and launch attacks. This then feeds into the second goal of obtaining data that should be off limits.

COMPARTMENTALIZATION ATTACKS

VULNERABLE? YES (AND MORE SO THAN PREVIOUSLY THOUGHT).

Compartmentalization is embodied in Arm's TrustZone or Intel's SGX. The intent of compartmentalization is to prevent code running in one compartment from being able to access (and in particular, to write to) memory in other compartments.

In the case of Arm's TrustZone—the most widely deployed compartmentalization technology in embedded systems—there are just two compartments: "trusted" and "untrusted." It is in TrustZone's trusted compartment that the kernel and small bits of other highly-critical code run.

The untrusted compartment runs code that is less critical, and it is this untrusted code that we do not want accessing code or data in the trusted compartment.

ATTACK PATTERNS

Compartmentalization accomplishes that goal very well, but it **doesn't protect the code and data inside each compartment** from network-based attacks. In fact, a 2018 study published in the **IEEE IoT Journal** notes that, although an absolute necessity for protecting SCADA systems, compartmentalization is limited in that "compromised access to the central hub will leave all data vulnerable."

This means, once access is gained to a trusted compartment, an attacker is free to launch whichever flavor of attack they like on the data and/or applications held within the compromised compartment.

Considering there's already ample fodder, **publicly and easily available**, about attacking one of the most popular compartmentalization solutions on the market, it's easy to see why this layer needs the added protection that the rest of the cybersecurity stack provides.

IN THE WILD

Since compartmentalization is applied through software, it's vulnerable to all of the types of attacks to which software is vulnerable. As such, compartmentalization solutions also need to be upgraded and patched as new vulnerabilities are discovered.

One way in which attackers have compromised compartmentalization in the past is by **downgrading the compartmentalization solution** to an earlier version of the software where vulnerabilities are known to exist. From there, it's rather simple to choose an attack vector and exploit a system using someone else's playbook.

As is evident above, the software layers of our cybersecurity stack, while important, are also the most vulnerable to attack. But why are our software layers weak spots, especially when compared to the hardware layers below them?

The answer has to do with the hole in our cybersecurity stack.

Filling The Hole: Enforcement & Its Critical Importance To Cybersecurity Today

We know there is a hole in the cybersecurity stack because all the software layers in the stack—encryption, compartmentalization, kernel, and application—are highly vulnerable to network-based attacks. This is because they all contain a seemingly infinite number of software vulnerabilities that attackers can exploit to gain access and control.

The one thing all of these attacks have in common, however—regardless of the software layer in which they occur—is that they're aimed at a very specific goal: getting the host processor to act in a way the developer did not intend.

Security solutions that live in the missing layer of the stack between the hardware foundation and the software layers above it ensure the intent of all of the software running on a system. This layer, that we've named the "Enforcement" layer, is where Dover's CoreGuard solution lives—and it lives there alone, for now, as the only technology of its kind on the market.

A fatal flaw in processors created this hole

Our computing devices have processors with an architecture that dates back to the 1940s—an era long before the World Wide Web, when network intrusions could not even be imagined, never mind prevented. The simplicity of this architecture, designed by physicist and mathematician John von Neumann, enabled "Moore's Law," an observation that helped processors become smaller, faster, and cheaper with each passing generation. The **von Neumann architecture** prized performance above all else, including security.

Think of von Neumann processors like "simple computers": if you put good in you'll get good out and vice versa. When processors based on this legacy architecture are sent malicious code, they execute it without question. Conventional processors don't know the difference between good and bad instructions, and they can't enforce what they don't know.

The Enforcement layer aims to solve this problem by equipping von Neumann processors with a bodyguard that can stop them from hurting themselves.

Enforcement aims to stop cyberattacks at the core

The processor's inability to distinguish between good and bad is what makes software the go-to attack vector for cybercriminals. If you can manipulate the software than you can make the machine do whatever you want.

A solution that provides Enforcement-level security works by ensuring processors are only executing instructions that are consistent with the developer's intended system behavior.

CYBERSECURITY STACK

SOFTWARE	**APPLICATION** Credentials, Sanitization	Manages credentials for authorized users and runs sanitation routines that check SQL queries
	KERNEL Operating System, Intrusions, Virus Scans	Scans for signatures, detects intrusions, and prevents unauthorized access to a network
	ENCRYPTION Communications, Data-in-Motion	Prevents data theft by converting data into a code that can only be accessed using an authorized key
	COMPARTMENTALIZATION Hypervisors, Zones, TEE	Isolates critical pieces of software in a sandbox so that it cannot be corrupted
HYBRID	**ENFORCEMENT** Instruction-level Correctness	Immunizes processors against entire classes of network-based attacks, including zero-days
HARDWARE	**ROOT OF TRUST** Keys, Secure Boot, Crypto	Validates all the hardware and software on the system at boot time
	PHYSICAL Tamper, Supply Chain, Rad-hard, Fault Tolerant	Ensures unauthorized personnel cannot touch or tamper with the system in the real world

How CoreGuard Provides Enforcement-level Security

Since today's processors can't defend themselves, they need an Enforcement layer solution that will do it for them. They need a bodyguard. CoreGuard—as the only solution for embedded systems that provides enforcement of instruction-level correctness and prevents the exploitation of software vulnerabilities—is this bodyguard. CoreGuard immunizes processors against entire classes of network-based attacks, including zero-day threats, to stop attacks in real-time before any damage can be done.

CoreGuard accomplishes this by combining three key components of technology: enforcement hardware, micropolicies, and metadata.

Enforcement **HARDWARE**

CoreGuard's hardware component, known as the Policy Enforcer, sits directly next to the host processor, and controls communication between it and the outside world. The Policy Enforcer allows CoreGuard to provide protection at the lowest possible level by ensuring nothing is sent out to peripherals without first being checked against micropolicy rules.

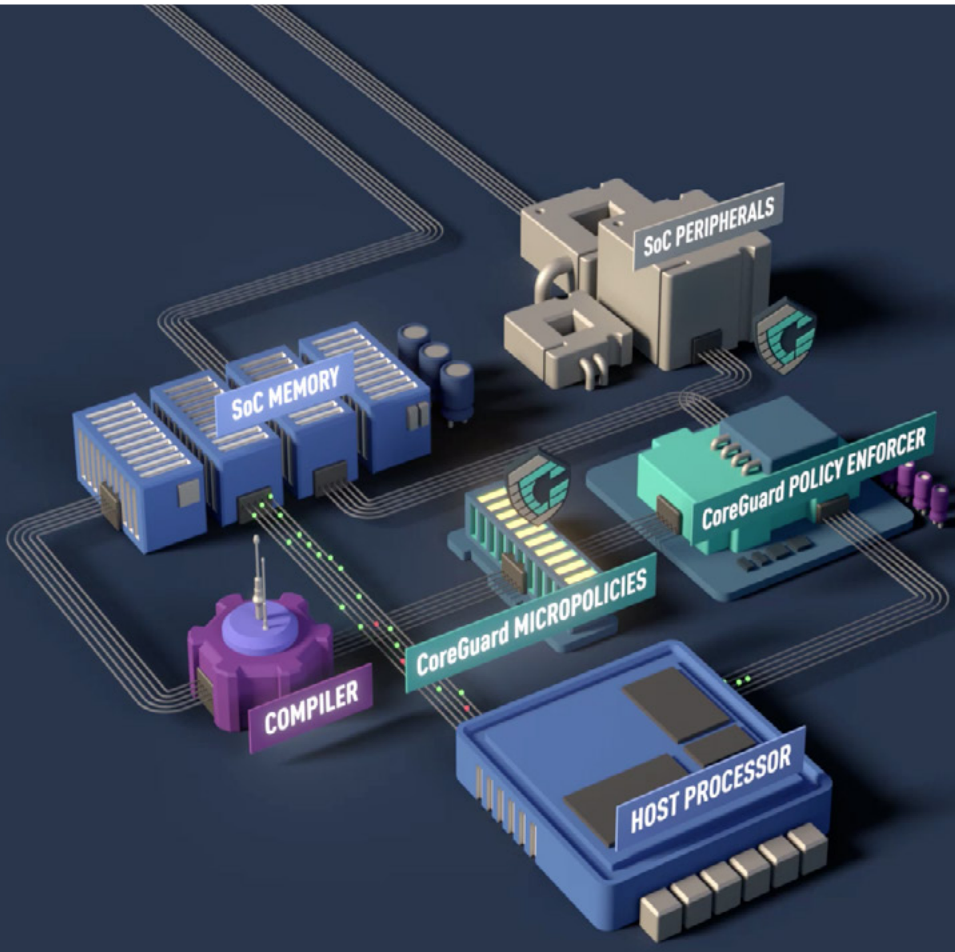
Rules **MICROPOLICIES**

Micropolicies define **security, safety, and privacy** rules that enable CoreGuard hardware to determine which instructions to execute and which to block. When an instruction is being executed by the host processor, the CoreGuard Policy Enforcer crosschecks it against the set of micropolicies installed on the SoC to ensure it isn't malicious. If the instruction complies with all the micropolicies, it is allowed to execute normally. If it violates any micropolicy, the Policy Enforcer sends an exception to the host processor and stops the instruction from completing before any damage can be done. CoreGuard **micropolicies are designed to stop entire classes of attacks**—not just specific attacks—including buffer overflows, code injection, data exfiltration, and even safety violations. They're able to accomplish all of this thanks to the rich metadata CoreGuard collects from the embedded systems it protects.

Information

METADATA

Metadata is information about information, and CoreGuard collects metadata on every piece of data and every instruction that is handled by the host processor. CoreGuard does this in two ways. First, CoreGuard collects metadata that is created in the binary when the application is compiled. CoreGuard then combines this static metadata with dynamic metadata that it collects during run-time. A typical processor wastes this valuable information, but CoreGuard uses it to determine whether an instruction should be executed or blocked.



How the Enforcement layer secures the upper layers of software

Thanks to lists like **MITRE's Common Weakness Enumeration (CWE)**, we have a vast amount of empirical data on the weaknesses in our software layers. We use this knowledge of vulnerabilities to write CoreGuard micropolicies that ensure that software, in any layer, only works the way it was intended.

Take a buffer overflow, for example. When a buffer overflow attack is successful, it's because no check was made to ensure the data being written to memory fits into the buffer size created. It is deterministic that the correct behavior is to never allow data to flow over the end of a buffer. CoreGuard's Heap and Stack Micropolicies ensure this correct behavior and protect the heap blocks and stack frame in memory.

And CoreGuard can close other attack vectors in similar ways.

When a function return does not follow the path out of the function that it followed in, for example, the program control-flow can be hijacked by overwriting the return address. It is deterministic that the correct behavior is to return to the function from which the current function was called. CoreGuard's Control Flow Integrity Micropolicy is designed to do just that—it stops attackers from redirecting the flow of execution of a program.

Another common attack pattern, in the face of a successful buffer overflow, occurs when the processor is forced to jump to a location where over-the-network data was written, and then begin executing the attacker's injected machine instructions. It is deterministic, in this scenario, that the correct behavior is to not allow the execution of data from an unknown external source. CoreGuard's RWX Micropolicy enforces this behavior by establishing read/write/execute permissions for code and data with fine-grained resolution.

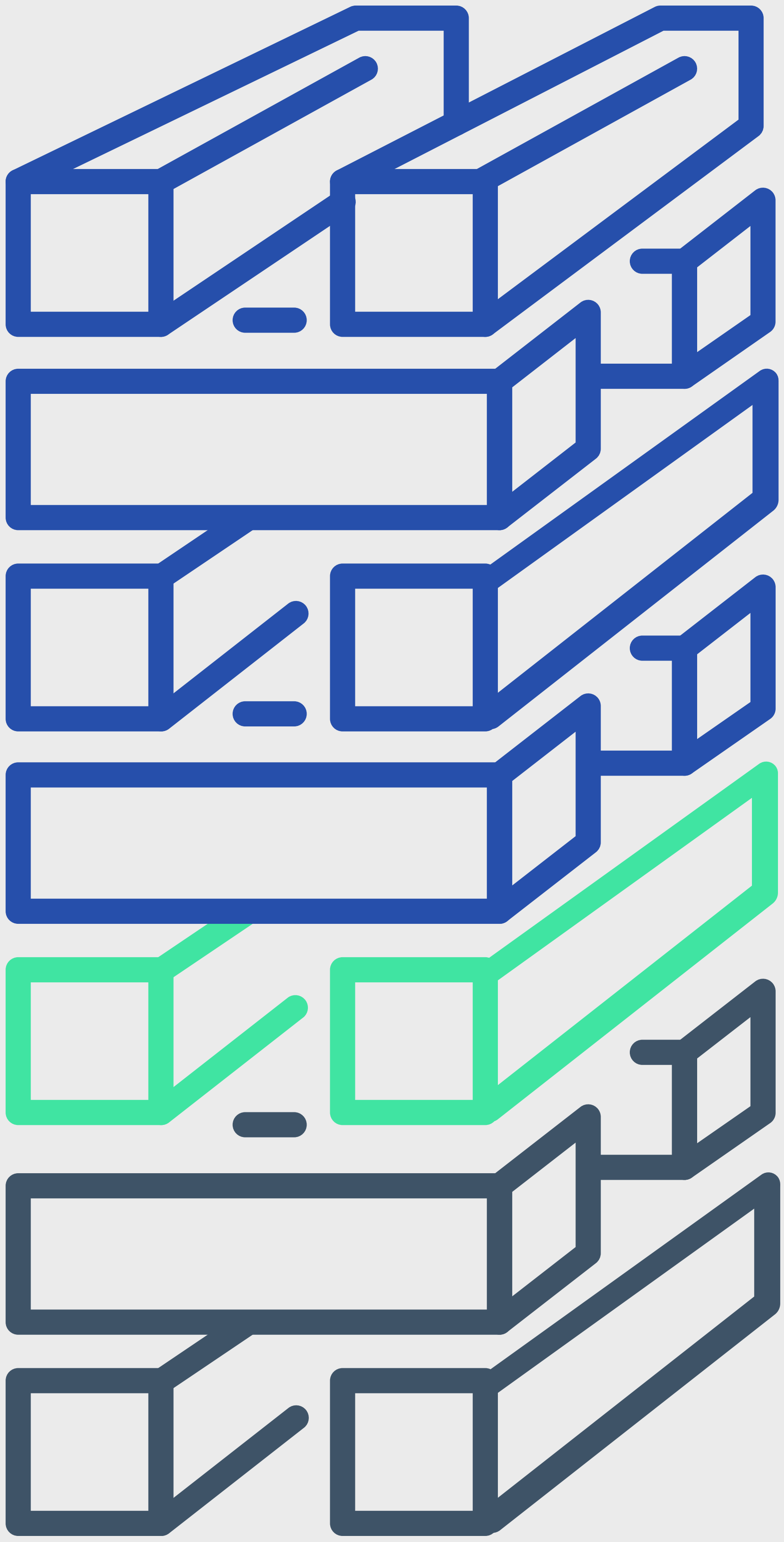
With just these few examples, you can see how CoreGuard's Enforcement layer solution can close the gap in our cybersecurity stack to ensure that embedded systems securely run code in the software layers.

How CoreGuard can eliminate redundancies to optimize Power, Performance & Area (PPA) impact

In addition to securing the software layers, CoreGuard provides an opportunity to eliminate redundancies in the cybersecurity stack, thus making systems more efficient. While having some redundancies is part of a defense-in-depth strategy, device makers are in a constant tug-of-war between often-conflicting goals around performance, power, and area. By making some solutions in other layers redundant, CoreGuard can decrease PPA costs while improving security.

Let's look at buffer overflows again. In particular, let's look at a code modification technique that programmers use to try to protect against stack smashing attacks where attackers use a buffer overflow to overwrite data in a special area of memory ("the stack") that keeps track of all the functions running in a program. To monitor buffer overflows on the stack, programmers sometimes insert "canaries," which are known values placed between a buffer and control data on the stack. When a buffer overflows, the canary is usually the first data corrupted. Later, when that canary value is checked, action (such as terminating the program) can be taken if the value has changed. Thus, like a canary in the coal mine can warn miners of potential danger, a canary on the stack can alert systems of a stack smashing attack. This monitoring comes at a cost, however. The program needs to call a function every time it checks the canary value, and scanning the full stack to check all canaries can impose significant performance overhead. Canaries require memory to be used on the front and back of every buffer, which can result in a large amount of extra memory utilization. Plus, like encryption functions, canary verification functions can be subverted or bypassed by attackers.

CoreGuard can block stack smashing attacks, thus eliminating the need for canary code modifications. Additionally, CoreGuard micropolicies provide broader protection than canary-based verification by preventing buffer overflows in multiple areas of memory, not just the stack.



The Hole In The Cybersecurity Stack Must Be Filled

Device makers, especially those developing products for critical industries like Automotive, Communications, Industrial IoT, and Medical Devices, have reached an inflection point. Whether due to growing competition, an ever-increasing risk of attack, a burgeoning sense of responsibility and liability, or all of the above, device makers are realizing the vital importance of a robust and actionable cybersecurity strategy.

Along with this realization, however, is the stark fact that software-based cybersecurity strategies are no longer sufficient. This has been proven time and time again by attacks like Stuxnet, Heartbleed, WannaCry and all of their derivatives out in the wild. As we've outlined in this paper, we need a defense-in-depth approach—one that goes beyond traditional methods—if we are to truly secure our devices.

An effective cybersecurity defense-in-depth approach, however, has remained an elusive prize for device makers. With cybersecurity, as with solving any mammoth problem, it is hard to know where to start and even harder to know when you're finished. This is where the cybersecurity stack model comes into play. It serves as a blueprint for device makers as they assemble a layered cybersecurity solution that will adequately protect their systems from cyberattacks.

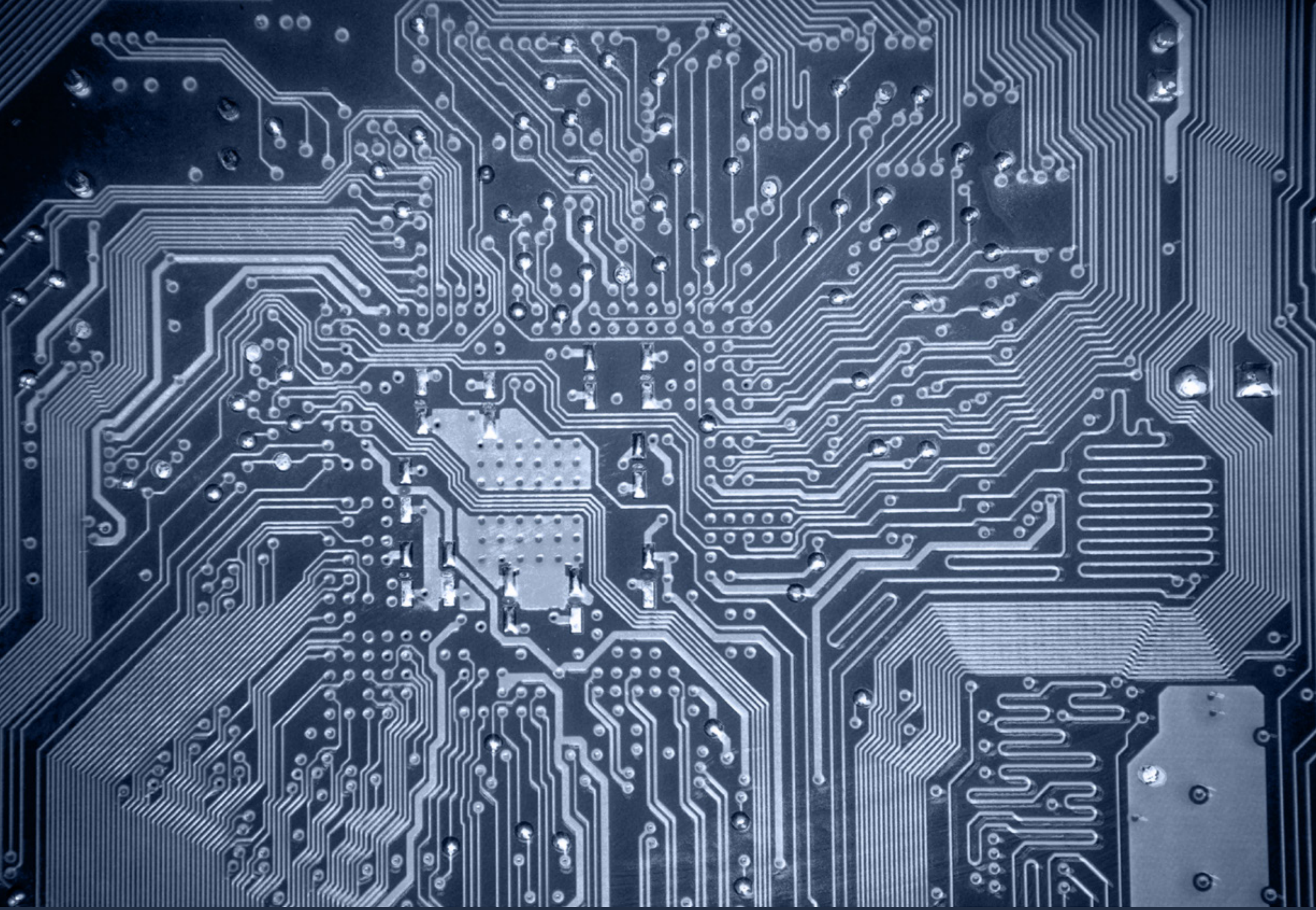
The cybersecurity stack model also clearly illustrates the importance of an Enforcement layer that will bridge the gap between hardware and software layers. Without Enforcement, all software layers in the stack are left dangerously exposed to network-based attacks via both known and unknown software vulnerabilities. Picture those Jenga® blocks we played with as kids. Each layer of the cybersecurity stack is like a Jenga block, and when you pull any block from the middle, the whole tower is at risk of tumbling down. Until Dover's CoreGuard solution, there was no "block" for the Enforcement layer; the tower was always teetering. Luckily, this is no longer the case.

Dover Microsystems CoreGuard technology is the only solution that can fill the hole in the cybersecurity stack by providing enforcement of instruction-level correctness that prevents the exploitation of software vulnerabilities and immunizes processors against entire classes of network-based attacks.

To learn more about how CoreGuard works,

REQUEST A DEMO

info.dovermicrosystems.com/request-a-demo



Learn More: info@dovermicrosystems.com

DOVER MICROSYSTEMS

About Dover Microsystems

Dover's lineage began in 2010 as the largest performer on the DARPA CRASH program. In 2015, Dover began development inside Draper before spinning out in 2017.

Based in Boston, Dover is the first company to fill the Enforcement layer of the cybersecurity stack. Dover's CoreGuard is the only solution for embedded systems that prevents the exploitation of software vulnerabilities and immunizes processors against entire classes of network-based attacks.

www.dovermicrosystems.com