GINSENG: KEEPING SECRETS IN REGISTERS WHEN YOU DISTRUST THE OPERATING SYSTEM

CS 5352: Advanced Operating Systems Design - Spring 2023

SAHAJA REDDY CHINTA

R11846157

ABSTRACT

- To develop a solution for protecting sensitive data in mobile and embedded apps from security threats, specifically from a potentially untrusted **operating system**. Ginseng, aims to do this by allocating sensitive data to registers at compile time and encrypting it at runtime before it enters the memory, reducing the amount of app logic in the TEE and requiring only minor modifications to existing apps.

SENSITIVE DATA IN MEMORY

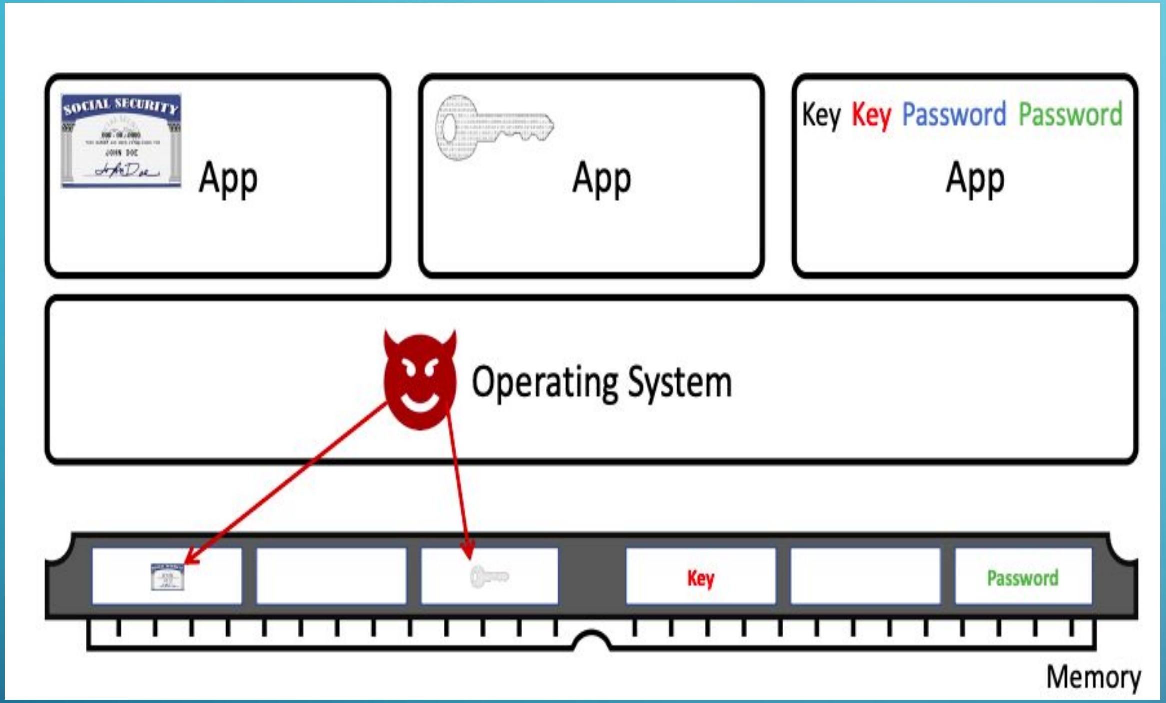


fig 1. Storing application data in memory

Many application have sensitive data in memory with the assumption that OS is trust worthy, But the problem is that the OS is complex software and has many vulnerabilities that can be exploited by attackers. when the OS is Compromised the OS can easily access Sensitive data from Memory

Many solutions have been proposed ,but
i)Don't work on ARM architecture
ii)Few need technologies not available of ARM
iii) Rely of TrustZone technology to run the complete app in secure environment which makes it vulnerable to attack.

GOAL OF GINSENG

- To protect sensitive data against a compromised OS
- What are we trusting ?
  - Hardware
  - ARM TrustZone
  - A chain of trust

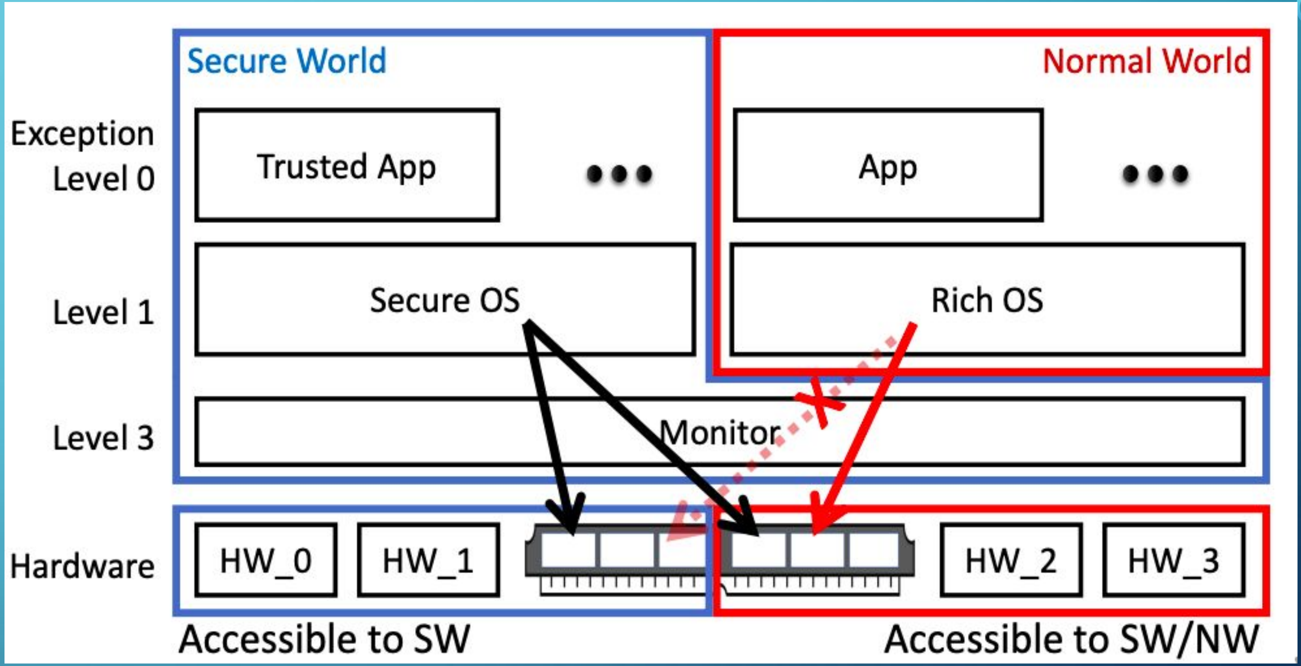
What we won’t trust ?

Anything not included in the trusting part like OS, System software ,applications..

ARM TRUSTZONE

We use ARM TrustZone to develop the solution. ARM TrustZone methodology to isolate security critical components in a system. It divides the system into two parts normal world , secure world.

- **Normal World:**
  - i) Rich OS: linux, Android
  - ii) App: third party apps
- **Secure World:**
  - i) Secure OS: **An operating system that manages data to make sure that it cannot be altered, moved, or viewed except by entities having appropriate and authorized access rights.**
  - ii) Trusted App



GINSENG

- Ginseng is a new approach to protecting the secrets of mobile and IoT apps from the threat of an untrusted operating system. This approach aims to support third-party apps without increasing the attack surface

Two Principles :

□ **It doesn't allow the app's logic to enter the Trusted Execution Environment**

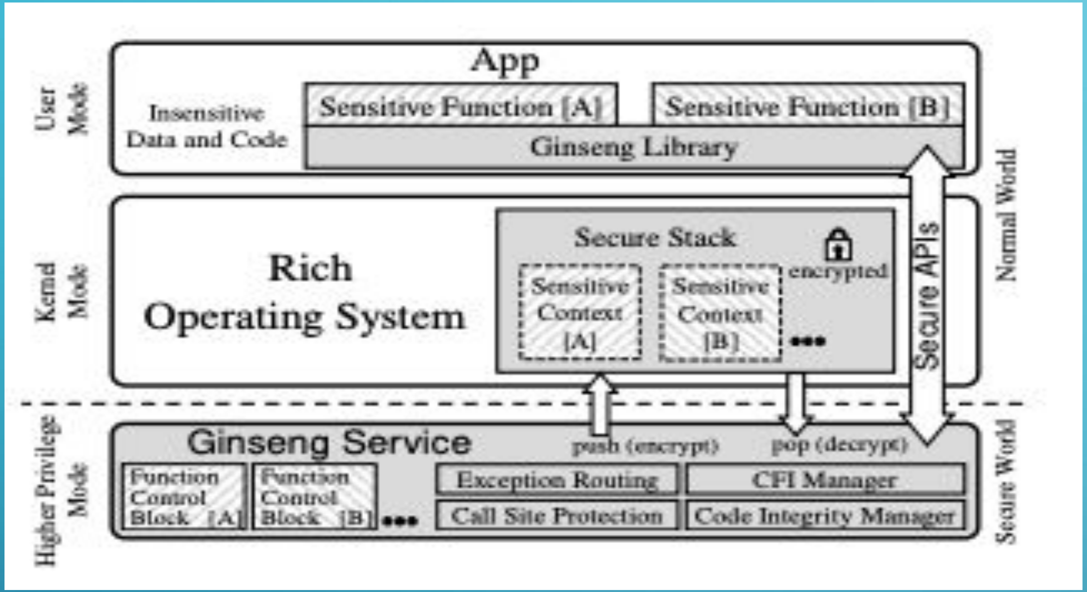
□ **protects only sensitive data and not complete data(complete application)**

- Ginseng is a system that aims to protect sensitive data in an app, such as passwords or personal information, from being accessed by an attacker who has taken control of the operating system (OS). To achieve this goal, Ginseng keeps sensitive data only in registers, instead of storing it in memory where the attacker can access it. When the sensitive data needs to be saved temporarily, Ginseng stores it in an encrypted form in a special memory area called the "secure stack". The encrypted data is also hashed and the hash is kept in a trusted environment (TEE) so that the attacker cannot modify the data.

CHALLENGES

- Code integrity
- Data confidentiality
- Control-flow integrity

ARCHITECTURAL REQUIREMENTS



Ginseng has three architectural requirements:

• A higher privilege mode than that of the untrusted OS to run Gservice.

• A direct call from an app to the higher privilege mode to bypass the OS.

• A way to trap writes to virtual memory control registers into the higher privilege mode.

Functions is the basic unit of protection and declaring **sensitive** variables and parameters within the function. The Ginseng compiler then ensures the confidentiality and integrity of the sensitive data by saving and restoring it to an encrypted secure stack, instead of to the normal memory, when the execution context changes

Ginseng has two main parts to its protection:

- Static protection in the compiler
- Runtime protection through the GService software

STATIC PROTECTION

- The Ginseng system helps protect sensitive information in computer programs by marking specific variables as sensitive and then ensuring that these variables are stored securely in protected places, such as registers. The compiler generates instructions to ask the GService to encrypt and store the sensitive data in a secure location, and to restore the encrypted data when the function returns. The register allocator in the compiler allocates sensitive registers before others to make sure they are always available and to avoid any performance overhead. When a function is called in a sensitive environment, the compiler saves any sensitive values in a secure stack that the untrusted operating system cannot access. The GService ensures the code's integrity before protecting the sensitive values during the function call.

GSERVICE

- GService is a piece of software that provides protection for sensitive data in an application. It doesn't have any application specific logic but instead provides APIs for the application's compiler to use in order to properly track and secure sensitive data.
- GService keeps track of the execution of each sensitive function through a data structure called the Function Control Block (fCB). This structure contains information about the physical memory address, code measurement, and a list of sensitive registers for each function. When a sensitive function is executed, the compiler-inserted code calls GService at the beginning and end of the function to trace its execution.
- The developers used a type of software called Trusted Firmware (ATF) and wrote most of the code in a programming language called Rust. Instead of using the usual way of allocating memory, they used a special way of allocating memory called linked_list_allocator. There are also some lines of assembly code used to access specific parts of the system, like the secure configuration register

RUNTIME PROTECTION

- This address the challenges code integrity, data confidentiality and control-flow integrity.

CODE INTEGRITY:

- Ginseng is a technology that ensures the integrity of sensitive functions in an operating system. This is done by hiding the code pages of these functions from the operating system, making it difficult for the OS to modify them. Ginseng uses two methods to achieve this:

- By making the kernel page table read-only and modifying the kernel so that it sends a request to GService instead of modifying the page table directly. This means that GService is in control of any modifications to the page table, and can ensure that sensitive functions are not modified.
- By trapping any attempts by the kernel to swap its page table base register. This prevents the kernel from using a compromised page table that it could modify.

RUNTIME PROTECTION

- when a sensitive function is called for the first time, GService checks its code integrity by walking the kernel page table to ensure no mapping to the function's code pages, then hashing the function code and comparing it to a hash provided by the compiler. If both checks pass, GService allocates and initializes an instance for the function. This process only happens once for each function, so the overhead is minimal

Data confidentiality:

- Intervene before the OS handles an exception by using a technique called dynamic trapping. This means that when an exception occurs, the system will modify the kernel code in memory to ensure that the exception is handled by the system, instead of the OS.
- Save the sensitive registers to the secure stack and return control to the OS to handle the exception. However, the system only knows which registers are potentially sensitive based on the function's information, so it saves all potentially sensitive registers.

RUNTIME PROTECTION

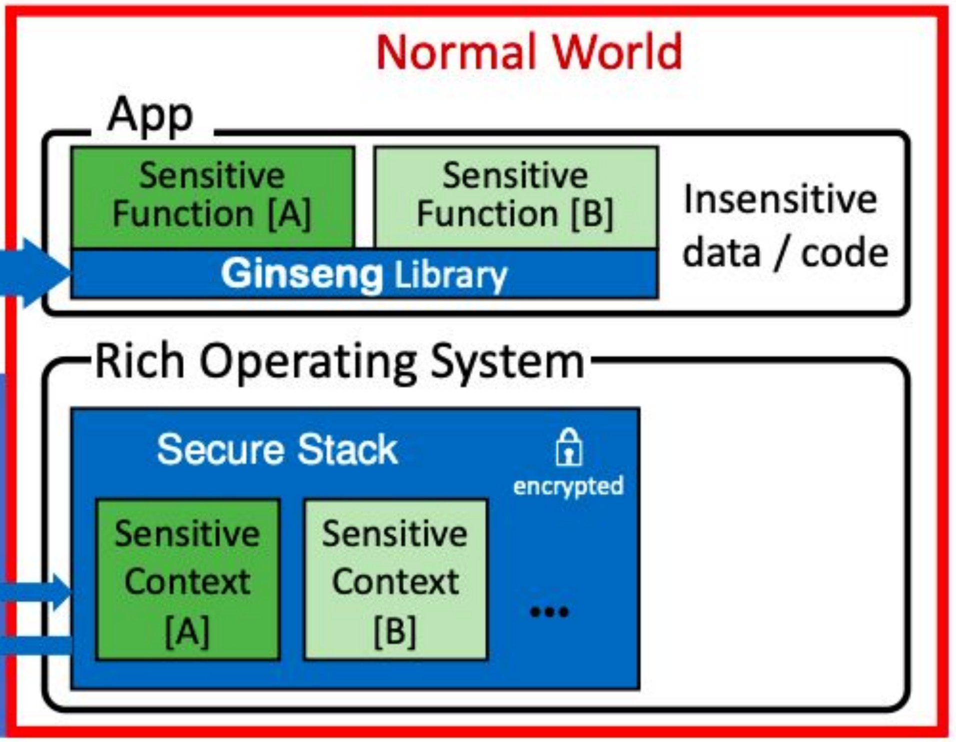
- Ginseng's call site protection helps keep sensitive data secure when making function calls. The Ginseng compiler does most of the work, but the service still needs to make sure sensitive data is restored to the right call site, especially if the same function is called multiple times in a recursive or concurrent manner. To do this, the service uses two techniques. First, it identifies a call site by its unique identifier and stack pointer, which is different for each thread. Second, it uses a last-in, first-out (LIFO) storage to support recursive calls, so that the most recently saved sensitive data is restored first.

Control-flow integrity:

- It does this by checking the integrity of the functions being called and the data being passed as parameters. When a sensitive function is called through a function pointer, the system checks the integrity of the pointed function before executing it. The same is done when a sensitive value is returned from a function. To ensure this protection, the Ginseng compiler inserts specific code into the sensitive functions, and there is also a service (GService) that works at runtime to enforce the protection. The service uses a secure stack to store sensitive data, and employs techniques to ensure the correct data is restored to the correct call site in cases of recursive or concurrent calls

ARCHITECTURE

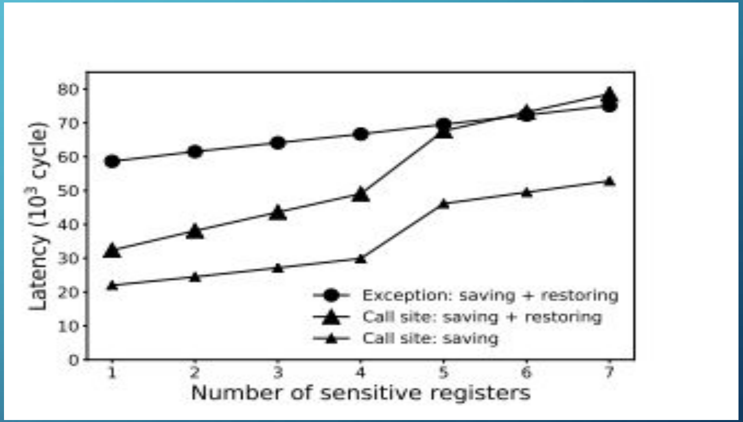
- Secure word: generic service
- Normal world: execution environment



EVALUATION:

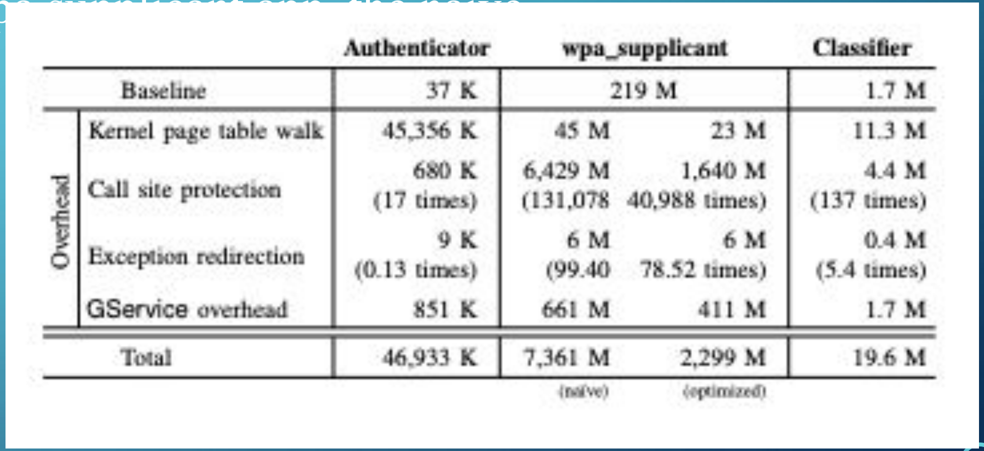
- **How much overhead does Ginseng impose to protect sensitive data? How much does each design component contribute?**

Overhead for Accessing the Secure Stack: the main source of the overhead for the secure stack at a call site is to encrypt and decrypt sensitive data and allocate a storage for them. On an exception, the four-time world switching, encryption, and decryption imposes the additional overheads. The standard deviations for a call site and exception are 0.66 K cycles and 1.03 K cycles at maximum



EVALUATION:

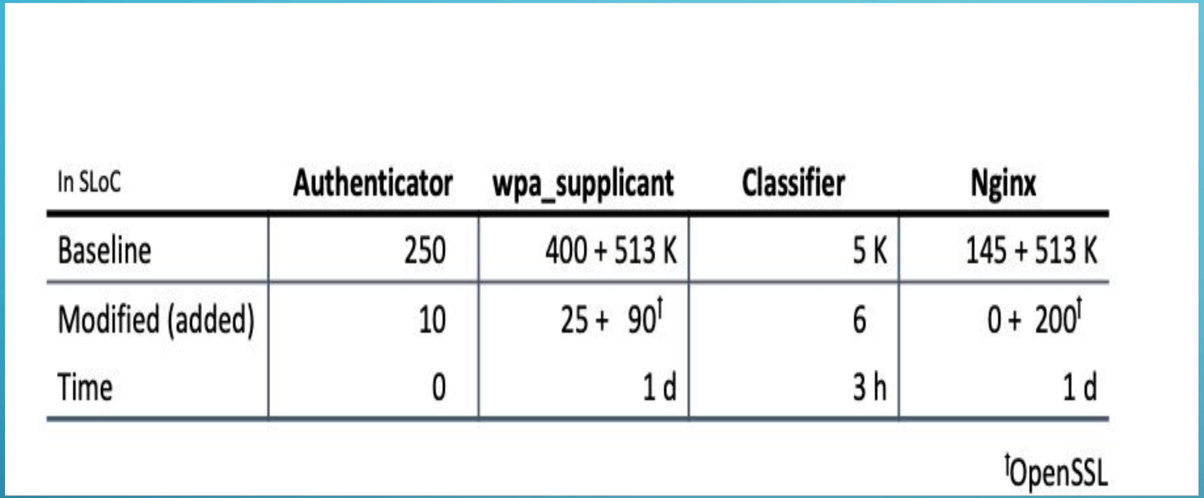
- **What is end-to-end performance overhead in practical applications? How much does each design component contribute to the overhead?**
- The number of function calls inside a sensitive function affects the overhead (additional processing time) from the "call site protection." As an example, in the wp
implementation increased the execution time by 4 seconds.
- It is found that 90% of the overhead comes from multiple function calls repeated within loops. To reduce this overhead, the number of function calls within sensitive functions was reduced by combining small functions, such as "memset()" and "memcpy()," into the main function. This reduced the overhead by 75%, from 6.4 B cycles to 1.6 B cycles, as shown in the table.



		Authenticator	wpa_supplicant		Classifier
Baseline		37 K	219 M		1.7 M
Overhead	Kernel page table walk	45,356 K	45 M	23 M	11.3 M
	Call site protection	680 K (17 times)	6,429 M (131,078	1,640 M 40,988 times)	4.4 M (137 times)
	Exception redirection	9 K (0.13 times)	6 M (99.40	6 M 78.52 times)	0.4 M (5.4 times)
	GService overhead	851 K	661 M	411 M	1.7 M
Total		46,933 K	7,361 M	2,299 M	19.6 M
			(naïve)	(optimized)	

EVALUATION:

- **How hard is it to apply Ginseng?**



In SLoC	Authenticator	wpa_supplicant	Classifier	Nginx
Baseline	250	400 + 513 K	5 K	145 + 513 K
Modified (added)	10	25 + 90[†]	6	0 + 200[†]
Time	0	1 d	3 h	1 d

[†]OpenSSL

Code changing is depending upon the data we are inserting. Which may vary from scenario to Scenario. This depends on the limitation of complier Prototype, present complier supports only Primitive types, if the sensitive data is in the format of array or any complex structure we need to manually divided this data into small primitive types

Although having prototype Ginseng on an ARM-based device, they believe that implementing Ginseng on a different architecture could simplify the implementation and lead to less overhead. For example, on x86 architecture with a hypervisor, the secure APIs could be implemented using hyper calls and exceptions could be intercepted directly by the hypervisor, avoiding the need for dynamic exception trapping.

STRENGTHS

- Efficient protection of sensitive data: Ginseng protects sensitive data by allocating them to registers at compile time and encrypting them at runtime before they enter the memory, which reduces the attack surface.
- Low engineering effort: Ginseng does not require significant development effort and only requires minor markups to support existing apps.
- Low overhead: The overhead incurred by Ginseng is proportional to how often sensitive data needs to be encrypted and decrypted, and it is low for small sensitive data like passwords or social security numbers.
- Flexibility: Ginseng can be applied to different architectures, making the implementation simpler and leading to less overhead.
- Practical applications: Ginseng can be applied to practical applications with reasonable efforts and overhead. The example of Nginx web server demonstrates this.

WEAKNESSES:

- Limitations for large sensitive data: Ginseng may not be suited for protecting large amounts of sensitive data as the overhead incurred would be high in such cases.
- Dependent on architecture: The performance and efficiency of Ginseng are dependent on the architecture it is implemented on.
- Limitations of the prototype: The prototype of Ginseng is based on ARM TrustZone, and its performance and efficiency on other architectures is not proven yet.
- Limited security: The security of Ginseng is limited to protecting sensitive data from the operating system and may not provide protection against other types of attacks

FUTURE WORK:

- further optimization and improvement of the implementation to reduce overhead, expanding the range of supported architectures, adding more advanced security features, and integrating Ginseng into more real-world applications to test its effectiveness in protecting sensitive data. Additionally, researchers could investigate potential weaknesses or vulnerabilities in the design of Ginseng and work on improving its security against attacks.

CONCLUSION

- Ginseng is a system that helps protect sensitive data on a computer. It does this by using either ARM TrustZone or x86 hypervisor mode, which meet the requirements for the protection. To reduce any unnecessary slowdowns, Ginseng only keeps sensitive data in memory when it is being used and, when it needs to be saved, it uses a secure area of memory that the operating system cannot access. The researchers have tested Ginseng on an ARM-based device, and have found that it can be used with practical applications with only a small amount of extra effort and overhead. They also think that using a different type of computer architecture could make Ginseng even simpler and faster.

QUESTIONS ?

•

THANK YOU