PROBLEM STATEMENT 14:

Protection of Passwords using SGX

Category: System Software, Security

Participants: 5th-8th Semester Students

Team Size: 2 or 3 Member Team

Scope: Developing an authorization module in Linux with and without SGX, and observe different OS level security protections.

Pre-requisite: Understanding of OS principles related to memory management and page tables in Linux. Programming in any language suited for System Software like C, C++, Python, etc.

Infrastructure Requirements:

Hardware:

- Intel XEON 3rd Gen (Icelake) or later generation CPU, with SGX enabled SKU.
- Xeon Platform with the suggested CPU above.

  Note:

  - ✓ NO separate requirements on Storage or RAM.
  - ✓ SGX is not supported on the Intel Core Class CPUs – Xeon is must.

Software (all open source and freely available as apt packages):

- Ubuntu 22.04 LTS or 23.04 LTS
- SGX SW
  - SGX SW packages to run a SGX application (Page 10 - https://download.01.org/intel-sgx/latest/linux-latest/docs/Intel_SGX_SW_Installation_Guide_for_Linux.pdf)
  - SGX SW packages to build a SGX application (developer) using Gramine – Refer to the steps here: https://gramine.readthedocs.io/projects/gsc/en/latest/#software-packages
- Docker - Docker packages are installed as part of previous steps.

Description:

Develop a simple user authorization application that is packaged as a container (like: Docker) and does the following.

1. Gets the username (max upto 20 users) and if the username first time registration ask the user to create one with password. If the username is already existing directly ask them to enter password and authorize and do allow/deny.

2. Generate an Encryption key (256 bit) a.k.a Password Encryption Key for password storage. Use existing Linux libraries.

3. Encrypt (using AES-256) the password with the above Password Encryption key and store them in simple text file like /etc/passwd in the root space (Hashing is optional).

Perform the following attack scenarios (Without SGX):

1. Run the above application.

2. Using Root privileged access dump out the memory of the application and scrub out Password Encryption Key from the collected memory dump.

3. Using this Password Encryption key decrypt all the stored user passwords from the stored text password file and publish the hacked passwords as a plain text file.

Perform the following mitigation (With SGX):

1. Prepare the machine to run SGX.

2. Convert the above application to run inside a SGX Enclave (using Gramine and GSC).

3. Run the application inside SGX Enclave.

4. Perform the same attack scenario as above (without SGX).

5. Observe the memory dump of the application is now encrypted, and hence even being a root user (privileged), we are not able to retrieve the user passwords.

Project Outputs:

1. Application workflow.
2. High-level algorithm.
3. Type of open source and System routines used for various tasks.
4. Test plan for testing various simple and corner cases.
5. Actual Source Code with appropriate comments archived in GitHub.
6. Demonstration of attack and mitigation scenarios.

Learning Outcome:

1. Partitioning the high-level problem statement into workflow and smaller independent tasks.

2. Understand and appreciate different levels of privileges like Kernel, User, etc.
3. Understand Linux process memory space (heap, stack, kernel memories, etc).
4. Understand user credential management.
5. Learn to hack applications using Root privileges.
6. How confidential computing technology in this case Intel SGX provides protection.

[PS: All the above are possible only if you don't use any ready-made ChatGPT/Generative AI tool]
