CLOUDFLARE

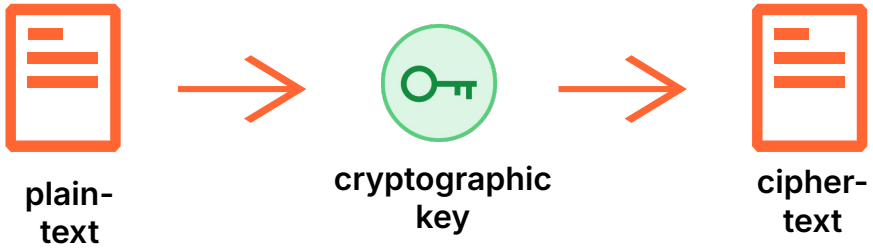
What if We Ask Linux to Do Cryptography for Us?

Oxana Kharitonova
Systems Engineer

Agenda

1. The Basics of Cryptography
2. Memory bugs
3. Mitigation against memory bugs
4. Linux Kernel Key Retention Service & Golang, Rust
5. Linux Crypto API & Golang, Rust
6. Questions

Cryptography



Cryptographic key:

- Random bytes of data
- Needs to be protected
- Accessed constantly

Incorrect memory accesses can leak cryptographic key

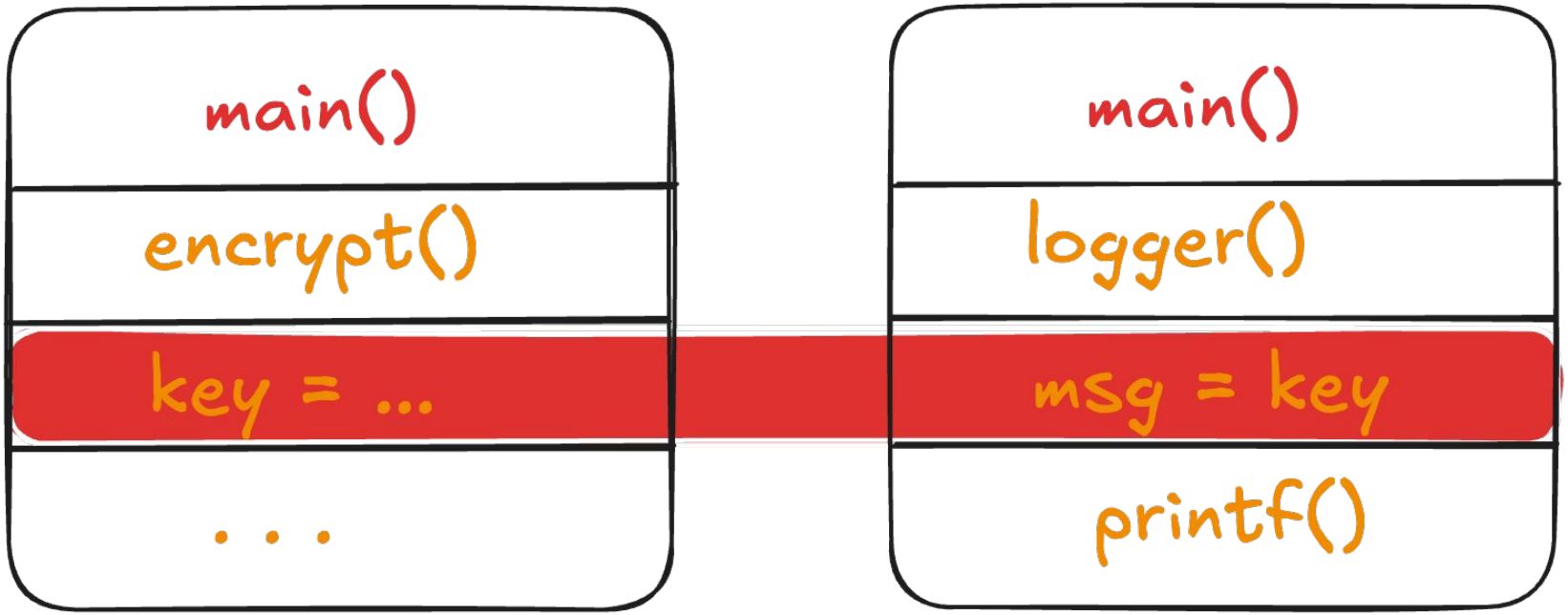
```
static void encrypt(void)
{
    uint8_t key[] = "Iamkey!";
}
static void logger(void)
{
    char msg[8];
    printf("the message is: %s\n", msg);
}
int main(void)
{
    encrypt();
    logger();
    return 0;
}
```

```
$ gcc -o main main.c
$ ./main
the message is: Iamkey!
```

Incorrect memory accesses can leak cryptographic key

```
static void encrypt(void)
{
    uint8_t key[] = "Iamkey!";
}
static void logger(void)
{
    char msg[8];
    printf("the message is: %s\n", msg);
}
int main(void)
{
    encrypt();
    logger();
    return 0;
}
```

Stack within the same process



Memory safety bugs

https://heartbleed.com/

Memory safe languages

Python : Vulnerability Statistics

Products (37) Vulnerabilities (231) Search products CVSS Report Metasploit Modules

Vulnerability Trends Over Time

Year	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	File Inclusion	CSRF	XXE	SSRF	Open Redirect	Input Validation
2014	2	0	0	0	0	0	0	1	0	0	2
2015	1	0	0	0	0	0	0	0	0	0	1
2016	8	1	0	0	0	0	0	0	0	0	1
2017	2	1	0	0	0	0	0	2	0	0	0
2018	4	1	0	0	0	0	0	0	0	0	2
2019	0	2	0	2	1	0	0	0	0	1	2
2020	6	0	0	0	1	1	0	0	0	0	0
2021	4	3	0	0	1	0	0	0	0	0	4
2022	2	2	0	0	0	1	0	0	0	0	0
2023	0	2	0	0	0	1	0	1	0	1	2
2024	1	0	0	0	0	0	0	0	0	0	0
Total	30	12		2	3	3		4		2	14

Memory safe languages

Rust-lang » Rust : Product details, threats and statistics

Versions | Vulnerabilities (23) | Product Dashboard | CVSS Report | Metasploit Modul

Log in to view product risk score details

Vulnerabilities by types/categories

Year	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	File Inclusion
2018	2	0	0	0	0	0
2019	0	1	0	0	0	0
2021	6	3	0	0	0	0
2022	0	0	0	0	0	0
2023	0	0	0	1	0	0
2024	0	0	0	0	0	0
Total	8	4		1		

Unsafe Rust

"All the code we've discussed so far has had Rust's memory safety guarantees enforced at compile time. However, Rust has a second language hidden inside it that doesn't enforce these memory safety guarantees: it's called unsafe Rust and works just like regular Rust, but gives us extra superpowers."

Memory safe languages

Golang : Vulnerability Statistics

Products (13) Vulnerabilities (157) Search products CVSS Report Met

Vulnerability Trends Over Time

Year	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal
2014	0	0	0	0	0
2016	0	0	0	0	0
2017	0	0	0	0	0
2018	2	0	0	0	1
2019	0	0	0	0	0
2020	0	0	0	1	0
2021	2	0	0	1	0
2022	1	0	0	0	3
2023	1	0	0	3	2
2024	0	0	0	0	0
Total	6			5	6

Unsafe Golang

*“Package **unsafe** contains operations that step around the type safety of Go programs.“*

CLOUDFLARE

Do you trust your code?

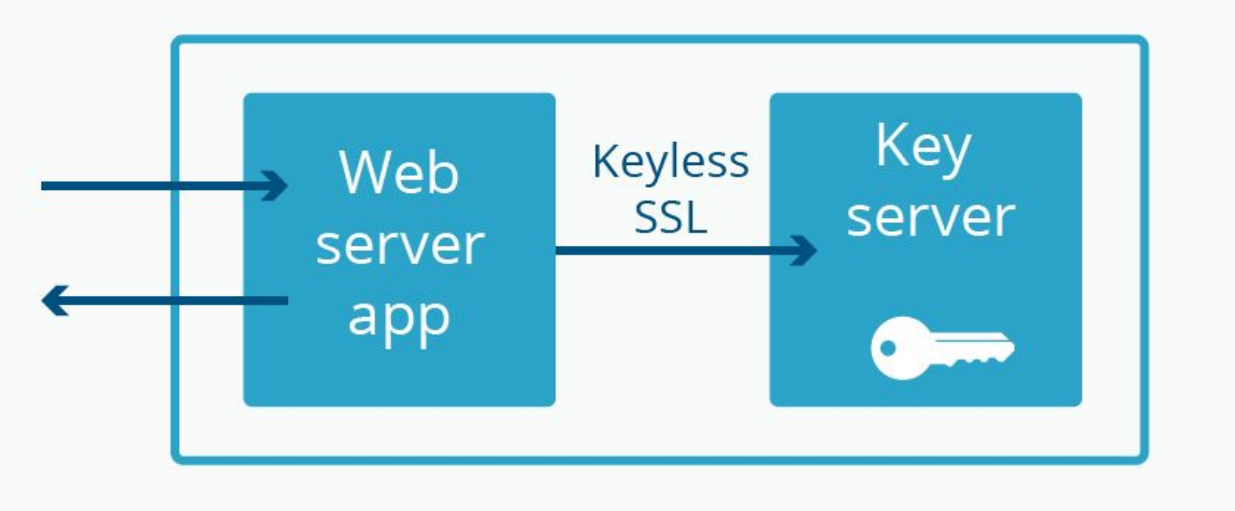
CLOUDFLARE

Do you trust the code of open source libraries?

CLOUDFLARE

Do you trust the Linux kernel?

Process memory address space



Two different processes don’t share memory by default.

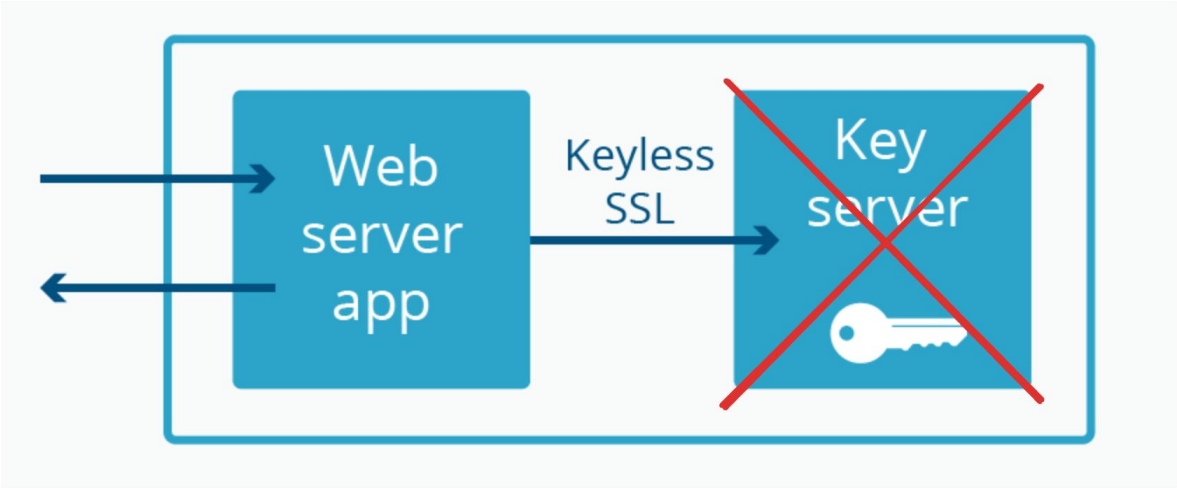
The key never leaves the Key server.

ssh-agent

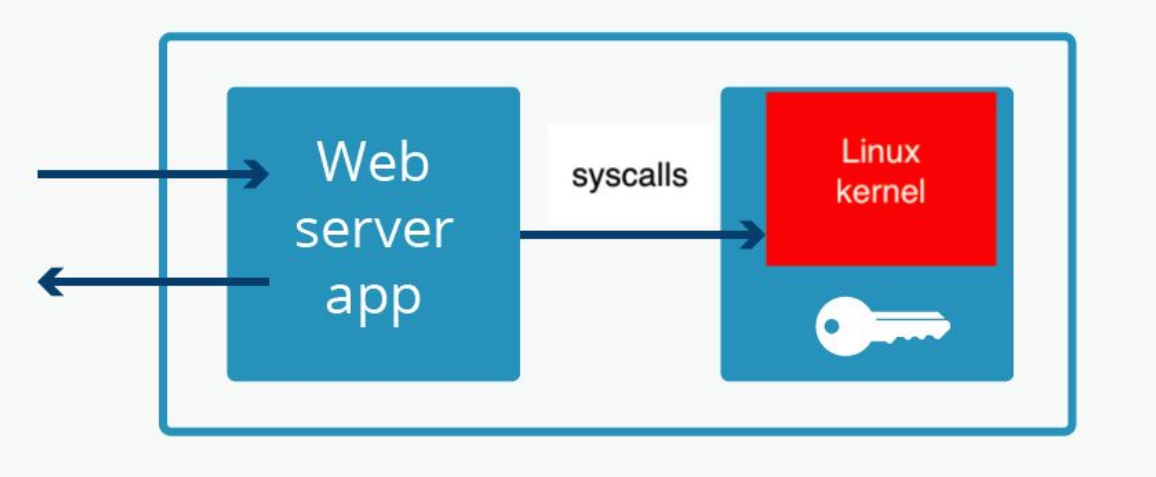
> *"The agent will never send a private key over its request channel. Instead, operations that require a private key will be performed by the agent, and the result will be returned to the requester. This way, private keys are not exposed to clients using the agent."*

CLOUDFLARE

Process memory address space



Process memory address space



The key moved from user space to kernel space

CLOUDFLARE

Linux Kernel Key Retention Service (aka Keystore)

https://www.kernel.org/doc/html/latest/security/keys/core.html

Linux Kernel Key Retention Service

Two types of entities: **keys** and **keyrings**. Analogous to files and directories respectively.

Linux Kernel Key Retention Service

- **Keys** can hold the actual cryptographic material
- The **key type** determines which operations can be performed over the keys (e.g. "***user***" - the payload may be read / updated by user-space applications; "***logon***" - the key payload is never visible from user space; "**asymmetric key**" used for public-key cryptography)

- **Keyrings** determine key lifetime and shared access:
  - **User keyrings** bound to the existence of a particular user, shared between all the processes of the same UID
  - **Process keyrings** bound to some process, thread or session
  - **Persistent keyrings** automatically cleaned up after an expiration time
  - **Special keyrings** owned by the kernel

Linux Kernel Key Retention Service

How we can interact with it:

- Keyctl - Linux Key Management Utilities
- Any programming languages which support system calls: C, Golang, Rust, Python, etc.

CLOUDFLARE

KEYCTL(1) Linux Key Management Utilities KEYCTL(1)

NAME top

keyctl − key management facility control

SYNOPSIS top

```
keyctl ––version
keyctl supports [<cap> | ––raw]
keyctl id [<keyring>]
keyctl show [−x] [<keyring>]
keyctl add [−x] <type> <desc> <data> <keyring>
keyctl padd [−x] <type> <desc> <keyring>
keyctl request <type> <desc> [<dest_keyring>]
keyctl request2 <type> <desc> <info> [<dest_keyring>]
keyctl prequest2 <type> <desc> [<dest_keyring>]
keyctl update [−x] <key> <data>
keyctl pupdate [−x] <key>
keyctl newring <name> <keyring>
keyctl revoke <key>
keyctl clear <keyring>
keyctl link <key> <keyring>
keyctl unlink <key> [<keyring>]
keyctl move [−f] <key> <from_keyring> <to_keyring>
keyctl search <keyring> <type> <desc> [<dest_keyring>]
keyctl restrict_keyring <keyring> [<type> [<restriction>]]
keyctl read <key>
keyctl pipe <key>
keyctl print <key>
keyctl list <keyring>
keyctl rlist <keyring>
keyctl describe <keyring>
keyctl rdescribe <keyring> [sep]
keyctl chown <key> <uid>
keyctl chgrp <key> <gid>
keyctl setperm <key> <mask>
keyctl new_session [<name>]
keyctl session
keyctl session − [<prog> <arg1> <arg2> ...]
keyctl session <name> [<prog> <arg1> <arg2> ...]
keyctl instantiate [−x] <key> <data> <keyring>
keyctl pinstantiate [−x] <key> <keyring>
keyctl negate <key> <timeout> <keyring>
keyctl reject <key> <timeout> <error> <keyring>
keyctl timeout <key> <timeout>
keyctl security <key>
keyctl reap [−v]
keyctl purge <type>
keyctl purge [−i] [−p] <type> <desc>
keyctl purge −s <type> <desc>
keyctl get_persistent <keyring> [<uid>]
keyctl dh_compute <private> <prime> <base>
keyctl dh_compute_kdf <private> <prime> <base> <output_length>
<hash_type>
```

https://man7.org/linux/man-pages/man1/keyctl.1.html

Linux Kernel Key Retention Service

For provisioning key:

- Request-Key Callback Service

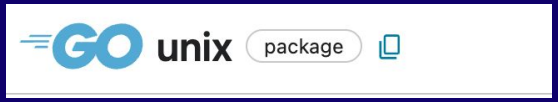
CLOUDFLARE

Linux Kernel Key Retention Service & Golang

Golang - add key

```
key_name := "user_key"
key_payload := make([]byte, 16)
rand.Read(key_payload)

func AddKey(key_name string, data []byte) int {
      serial, _ := unix.AddKey(
            "user",
            key_name,
            data,
            unix.KEY_SPEC_PROCESS_KEYRING,
      )
      return serial
}
```



Golang - read key

```
func ReadKey(serial int) ([]byte, error) {
	var buffer []byte
	for {
		// Get payload length
		length, _ := unix.KeyctlBuffer(unix.KEYCTL_READ, serial, buffer, 0)
		// Check if the data was written
		if length <= len(buffer) {
			return buffer, nil
		}
		// Make a bigger buffer if needed
		buffer = make([]byte, length)
	}
}
```

Golang - read key

```
func ReadKey(serial int) ([]byte, error) {
	var buffer []byte
	for {
		// Get payload length
		length, _ := unix.KeyctlBuffer(unix.KEYCTL_READ, serial, buffer, 0)
		// Check if the data was written
		if length <= len(buffer) {
			return buffer, nil
		}
		// Make a bigger buffer if needed
		buffer = make([]byte, length)
	}
}

serial, _ := unix.RequestKey("user", key_name, "", unix.KEY_SPEC_PROCESS_KEYRING)
```

CLOUDFLARE

Linux Kernel Key Retention Service & Rust

Rust - add key

```
let key = &b"be238e779b1e3cef051517efd60ea847"[..];
let key_name = "user_key\0";

fn add_key(key_name: &str, payload: &[u8]) -> libc::c_long {
    unsafe {
        libc::syscall(
            libc::SYS_add_key,
            "user\0".as_ptr(),
            key_name.as_ptr(),
            payload.as_ptr() as *const libc::c_void,
            payload.len(),
            libc::KEY_SPEC_PROCESS_KEYRING,
        )
    }
}
```

Crate libc

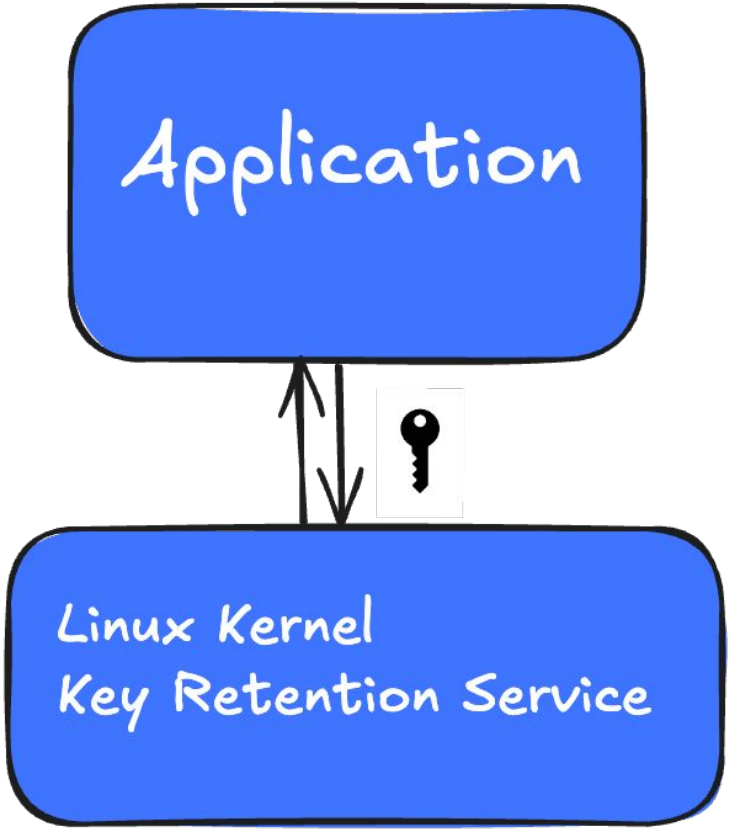
source

Rust - read key

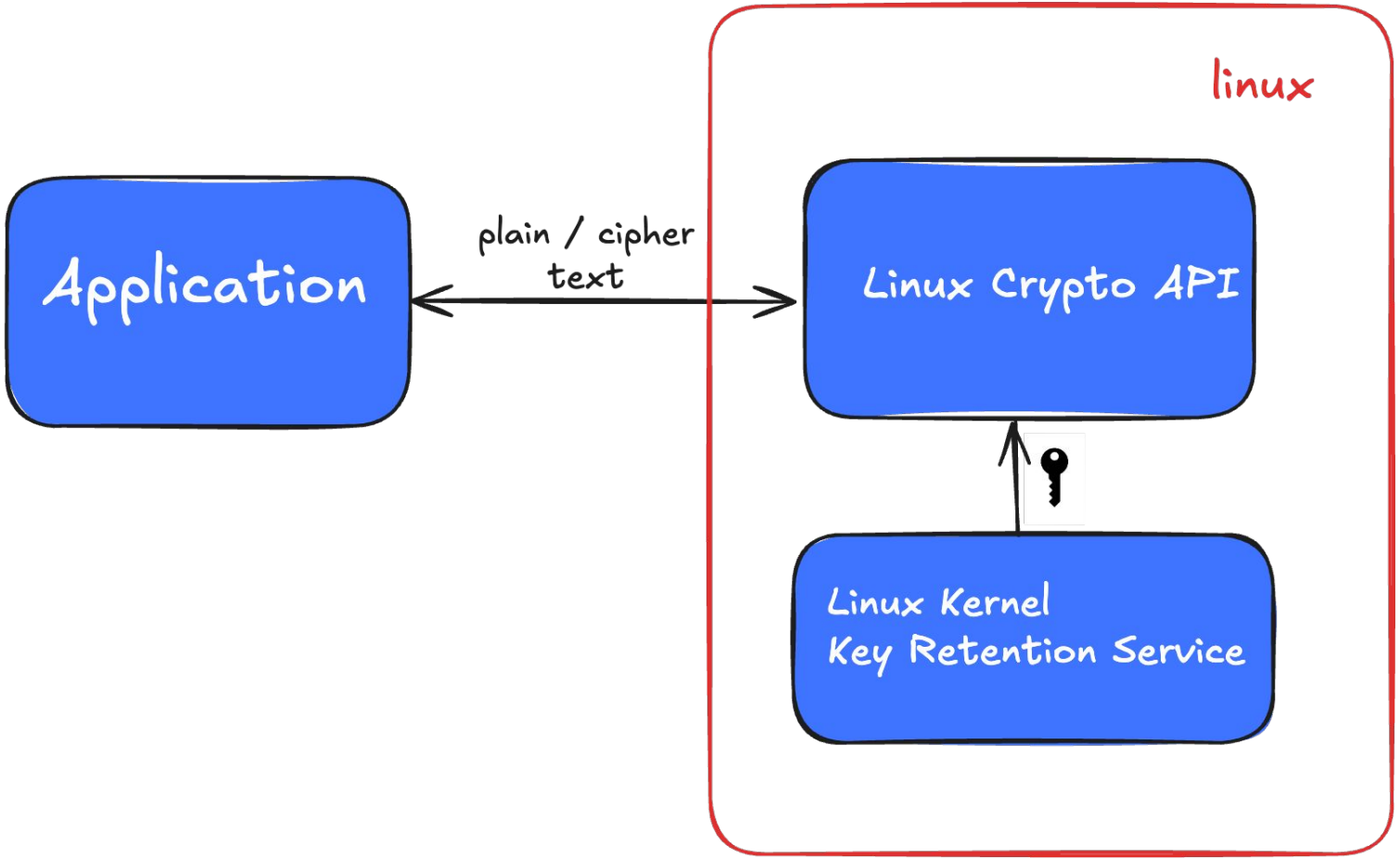
```
fn read_key(serial: i64) -> Vec<u8> {
    // The first request is made with an empty buffer to get a key length,
    // https://elixir.bootlin.com/linux/v5.15.46/source/security/keys/keyctl.c#L903
    let mut sz = usize::try_from(unsafe {
        libc::syscall(
           libc::SYS_keyctl,
           libc::KEYCTL_READ,
           serial,
           std::ptr::null() as *const Vec<u8>,
           0,
        )
    }).unwrap();
    let mut payload: Vec<u8> = vec![0; sz];
. . .
```

Rust - read key

```
    . . .
        loop {
            sz = usize::try_from(unsafe { libc::syscall(
                    libc::SYS_keyctl,
                    libc::KEYCTL_READ,
                    serial,
                    payload.as_ptr(),
                    payload.len(),
            )}).unwrap();
            if sz <= payload.capacity() {
                break;
            }
            payload.resize(sz, 0);
        }
        payload.truncate(sz);
        payload
    }
```



Process2


linux
Application
plain / cipher text
Linux Crypto API
Linux Kernel
Key Retention Service

CLOUDFLARE

Linux Crypto API

https://www.kernel.org/doc/html/latest/crypto/index.html

$ cat /proc/crypto

```
oxana@dev:~$ cat /proc/crypto
name         : ctr(aes)
driver       : ctr-aes-aesni
module       : aesni_intel
priority     : 400
refcnt       : 1
selftest     : passed
internal     : no
type         : skcipher
async        : yes
blocksize    : 1
min keysize  : 16
max keysize  : 32
ivsize       : 16
chunksize    : 16
walksize     : 16
```

```
name         : sha3-512
driver       : sha3-512-generic
module       : kernel
priority     : 0
refcnt       : 1
selftest     : passed
internal     : no
type         : shash
blocksize    : 72
digestsize   : 64

. . .
```

$ cat /proc/crypto

```
oxana@dev:~$ cat /proc/crypto
name         : ctr(aes)
driver       : ctr-aes-aesni
module       : aesni_intel
priority     : 400
refcnt       : 1
selftest     : passed
internal     : no
type         : skcipher
async        : yes
blocksize    : 1
min keysize  : 16
max keysize  : 32
ivsize       : 16
chunksize    : 16
walksize     : 16
```

```
name         : sha3-512
driver       : sha3-512-generic
module       : kernel
priority     : 0
refcnt       : 1
selftest     : passed
internal     : no
type         : shash
blocksize    : 72
digestsize   : 64

. . .
```

CLOUDFLARE

AES-CTR-128

$ cat /proc/crypto

CLOUDFLARE

```
oxana@dev:~$ cat /proc/crypto
name         : ctr(aes)
driver       : ctr-aes-aesni
module       : aesni_intel
priority     : 400
refcnt       : 1
selftest     : passed
internal     : no
type         : skcipher
async        : yes
blocksize    : 1
min keysize  : 16
max keysize  : 32
ivsize       : 16
chunksize    : 16
walksize     : 16
```

```
name         : sha3-512
driver       : sha3-512-generic
module       : kernel
priority     : 0
refcnt       : 1
selftest     : passed
internal     : no
type         : shash
blocksize    : 72
digestsize   : 64

. . .
```

CLOUDFLARE

Linux Crypto API & Golang

https://www.kernel.org/doc/html/latest/crypto/index.html

Golang - encrypt by provided key

```
const IV_LEN = 16

type af_alg_iv struct {
      ivlen uint32
      iv    [IV_LEN]byte
}
```

Golang - encrypt

```
func crypto(key_payload, iv, plaintext []byte) {
      fd, _ := unix.Socket(unix.AF_ALG, unix.SOCK_SEQPACKET, 0)
      addr := &unix.SockaddrALG{
            Type: "skcipher",
            Name: "ctr(aes)",
      }
      unix.Bind(fd, addr)
      syscall.SetsockoptString(
            fd,
            unix.SOL_ALG,
            unix.ALG_SET_KEY,
            string(key_payload),
      )
      aes_ctr_fd, _, _ := unix.Syscall(unix.SYS_ACCEPT,
uintptr(fd), 0, 0)
      unix.Close(fd)
. . .
```

https://www.kernel.org/doc/html/latest/crypto/userspace-if.html:

"1. Create a socket of type AF_ALG with the struct sockaddr_alg parameter specified below for the different cipher types.

2. Invoke bind with the socket descriptor

3. Invoke accept with the socket descriptor. The accept system call returns a new file descriptor that is to be used to interact with the particular cipher instance. When invoking send/write or recv/read system calls to send data to the kernel or obtain data from the kernel, the file descriptor returned by accept must be used."

Golang - encrypt

```
. . .
        /*
                type Cmsghdr struct {
                        Len   uint64
                        Level int32
                        Type  int32
                }
        */
        b := make([]byte, syscall.CmsgSpace(4)+syscall.CmsgSpace(20))
        h := (*syscall.Cmsghdr)(unsafe.Pointer(&b[0]))

        h.Level = unix.SOL_ALG
        h.Type = unix.ALG_SET_OP
        h.SetLen(syscall.CmsgLen(4))
        p := (*uint32)(unsafe.Pointer(uintptr(unsafe.Pointer(h)) + syscall.SizeofCmsghdr))
        *p = unix.ALG_OP_ENCRYPT
. . .
```

Golang - encrypt

```
. . .
        h = (*syscall.Cmsghdr)(unsafe.Pointer(&b[syscall.CmsgSpace(4)]))

        h.Level = unix.SOL_ALG
        h.Type = unix.ALG_SET_IV
        h.SetLen(syscall.CmsgLen(20))
        aiv := (*af_alg_iv)(unsafe.Pointer(uintptr(unsafe.Pointer(h)) + syscall.SizeofCmsghdr))
        aiv.ivlen = uint32(IV_LEN)
        copy(aiv.iv[:], iv)
```

Golang - encrypt

```
. . .
        syscall.Sendmsg(int(aes_ctr_fd), plaintext, b, nil, 0)
        aes_ctr := os.NewFile(aes_ctr_fd, "aes_ctr")
        ciphertext := make([]byte, len(plaintext))
        aes_ctr.Read(ciphertext)
}
```

Golang - encrypt by key from the Linux keystore

☹ `syscall.SetsockoptString(fd, unix.SOL_ALG, unix.ALG_SET_KEY, string(key_payload))`

☺ `syscall.SetsockoptInt(fd, unix.SOL_ALG, unix.ALG_SET_KEY_BY_KEY_SERIAL, key_serial)`

CLOUDFLARE

Linux Crypto API & Rust

https://www.kernel.org/doc/html/latest/crypto/index.html

https://blog.cloudflare.com/the-linux-crypto-api-for-user-applications/

Rust - encrypt

```
let fd = unsafe{ libc::socket(libc::AF_ALG, libc::SOCK_SEQPACKET, 0)};

let mut addr: libc::sockaddr_alg = unsafe { std::mem::zeroed() };
addr.salg_family = libc::AF_ALG as u16;
addr.salg_type[..9].copy_from_slice("skcipher\0".as_bytes());
addr.salg_name[..14].copy_from_slice("ctr-aes-aesni\0".as_bytes());
addr.salg_feat = 0;
addr.salg_mask = 0;

unsafe { libc::bind( fd, &addr as *const libc::sockaddr_alg as *const libc::sockaddr,
     std::mem::size_of::<libc::sockaddr_alg>() as libc::socklen_t)};
unsafe { libc::setsockopt( fd, libc::SOL_ALG, libc::ALG_SET_KEY_BY_KEY_SERIAL,
            &key_serial as *const _ as *const libc::c_void, 4) };
let aes_ctr_fd = unsafe { libc::accept(fd, std::ptr::null_mut(), std::ptr::null_mut()) };
. . .
```

Linux Crypto API - various

- Not all the crypto algorithms are supported by Crypto API, check /proc/crypto
- It might be slower as it adds additional overhead for system calls and copying plain/ciphertext from user space to kernel space:
  - Instead of copying data, zero-copy interface can be used, but it's restricted up to 64KB
  - We compared with OpenSSL - the crypto part is almost the same, but because of overhead on syscalls it's almost 2x slower. With Golang, Rust might be different outcome depends on which user space crypto library is used
- It's more secure than any user space library

Links

- https://www.kernel.org/doc/html/latest/security/keys/core.html
- https://blog.cloudflare.com/the-linux-kernel-key-retention-service-and-why-you-should-use-it-in-your-next-application/
- https://www.youtube.com/watch?v=OpPVX8gMe3I
- https://www.kernel.org/doc/html/latest/crypto/architecture.html
- https://blog.cloudflare.com/the-linux-crypto-api-for-user-applications/
- https://man7.org/linux/man-pages/man3/cmsg.3.html
- https://man7.org/linux/man-pages/man7/keyrings.7.html

Thank you

Questions?