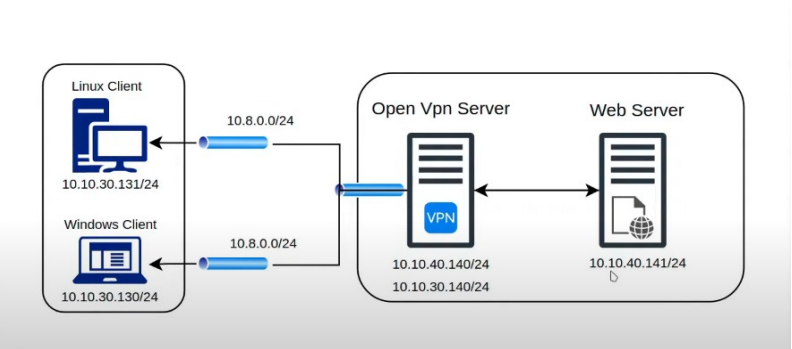
SIH 2023 PROJECT: PS ID- SIH1453 [BITHEADS]

By scrutinizing OpenVPN's crypto libraries, our team is actively identifying and resolving vulnerabilities through a meticulous process of static and dynamic analyses, complemented by ethical exploitation for comprehensive improvements.



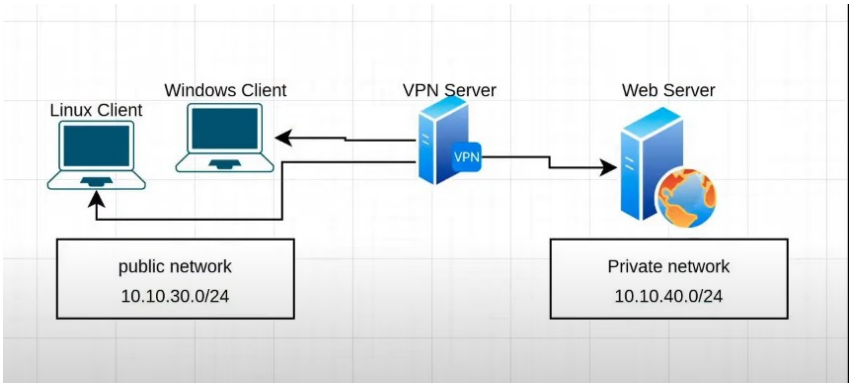
GOALS

Goal 1: Understand the basic concept of OpenVPN
Goal 2: Find all the vulnerability tools in the implementation of crypto libraries in OpenVPN (IPSEC) in IPv6
Goal 3: Perform static and dynamic analysis of relevant code to discover any unknown bug
Goal 4: Identification of known and unknown vulnerabilities of crypto libraries in OpenVPN (IPSEC) in IPv6
Goal5: Investigate and report vulnerabilities configure leading to

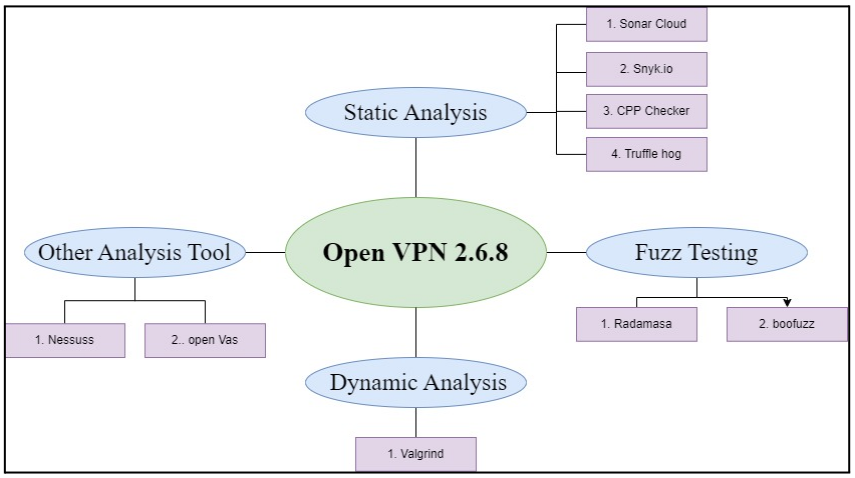
- Exploitation vector
- Compromise of data confidentiality

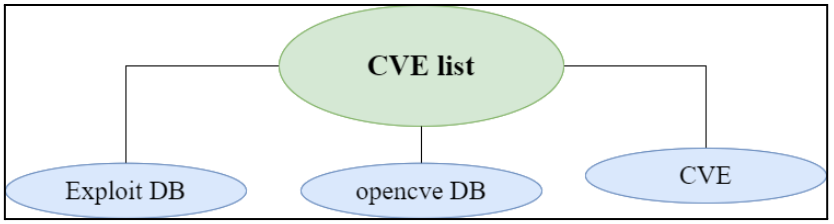
SOLUTION APPROACH

Understanding the basic workings of OpenVPN



Tools and Resources Used





- Findings of Synk.io:

  Path Traversal (12 instances): Allows unauthorized file access.
  Potential Buffer Overflow (11 instances): Caused by unsafe strcpy function usage.
  Insecure Password Hashing (7 instances): Use of weak EVP_md5_sha1 hash.
  Double Free (5 instances): Attempt to free the same memory block twice.
  Improper Null Termination (5 instances): Risk of information disclosure or buffer overflows.
  Use After Free (4 instances): Incorrect dynamic memory management.
  Dereference of NULL Pointer (3 instances): Accessing invalid memory addresses.
  Server-Side Request Forgery (SSRF) (2 instances): Potential for unauthorized server requests.
  Python 2 Source Code (2 instances): Use of outdated and unsupported Python 2.
  Memory Allocation of String Length (1 instance): Potential for memory-related issues.
  Authentication Bypass by Spoofing (1 instance): Weakness in identity verification.
  Integer Overflow (1 instance): Vulnerability due to numerical calculation errors.

- Findings of CPP Checker:

  The focus area of CPP Checker was identifying cryptographic vulnerabilities, memory management issues, input validation errors, incorrect usage of cryptographic functions, insecure configuration options and compliance with coding standards.

The outcomes it gave were some basic warnings and errors e.g., missing return statements, lack of initialization, etc.

Checking E:\Open\openvpn\src\openvpn\error.c: ENABLE_CRYPTO_MBEDTLS;ENABLE_CRYPTO_OPENSSL...

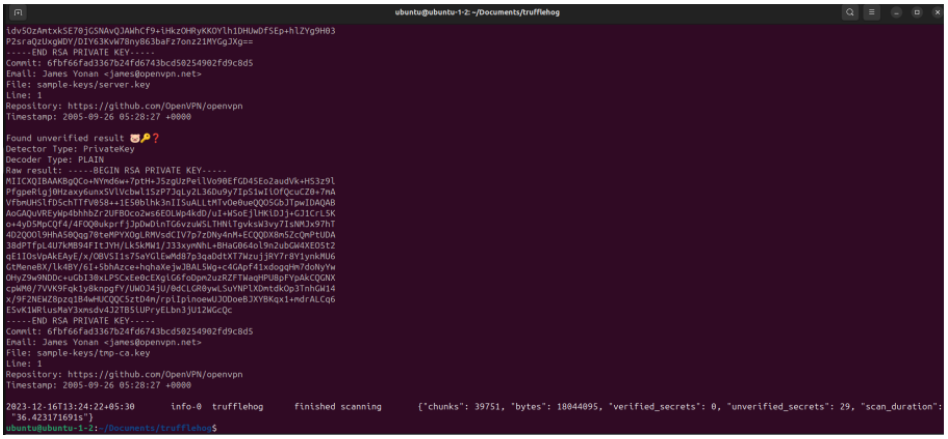
Checking E:\Open\openvpn\src\openvpn\error.c: ENABLE_CRYPTO_OPENSSL...

Checking E:\Open\openvpn\src\openvpn\error.c: ENABLE_CRYPTO_OPENSSL;ENABLE_CRYPTO_WOLFSSL;_WIN32...

CPP Checker

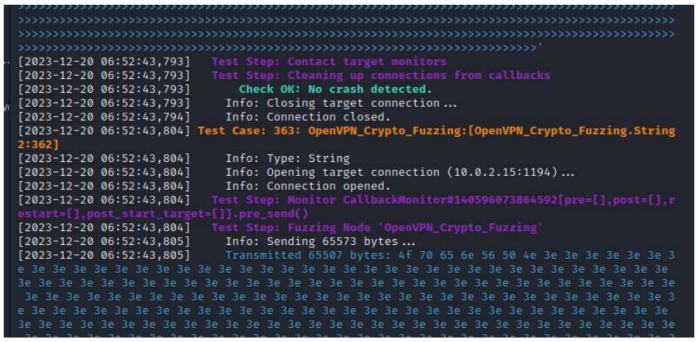
- Findings of Truffle Hog:

  Gave the information about public, and private keys and other credentials present in the GitHub repository.



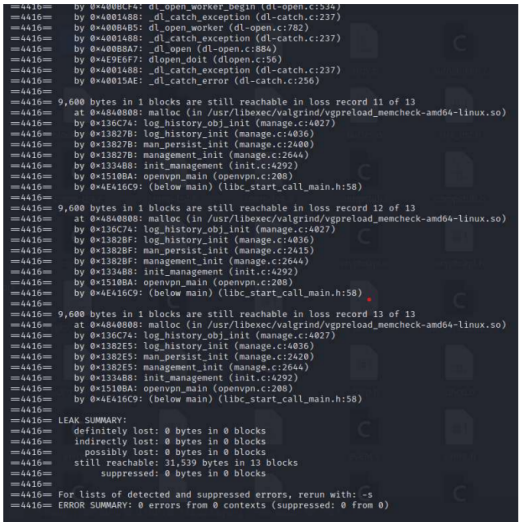
Truffle Hog

- Findings of Boofuz:



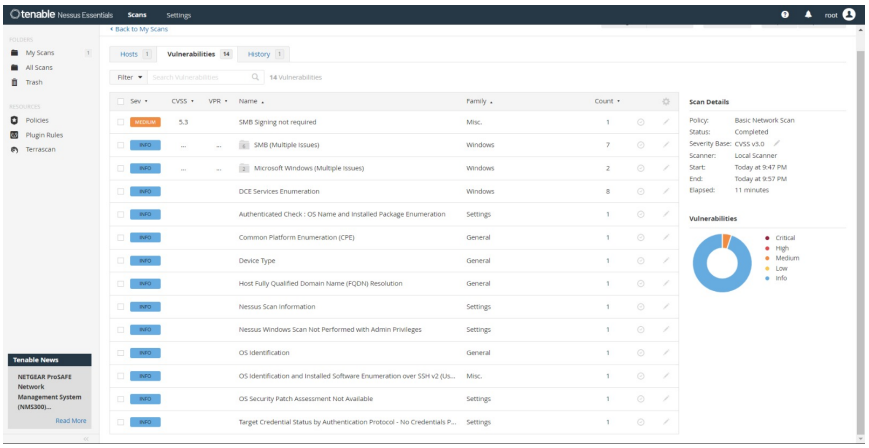
Boofuz

- Findings of Valgrind:



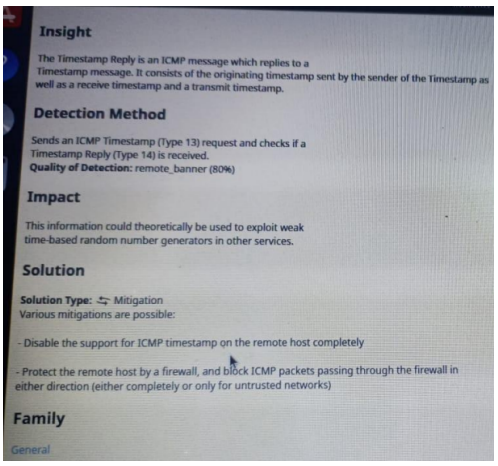
Valgrind

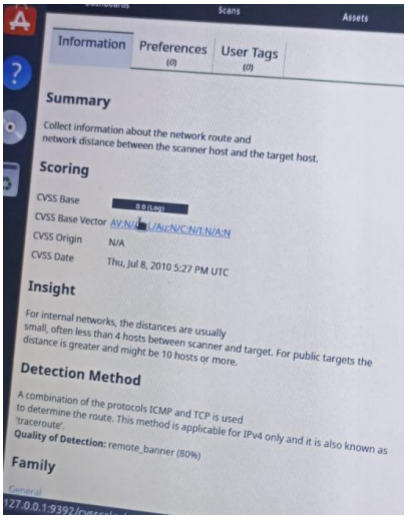
- Findings of Nessus:

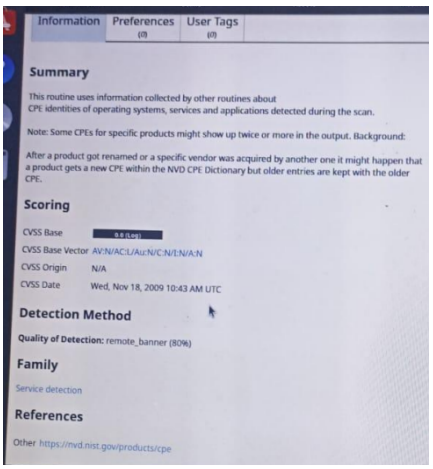


Nessus

- Findings of OpenVAS:







OpenVAS

- Findings of CVE List:

  CVE list gave the known vulnerabilities along with the patched and unpatched vulnerabilities.

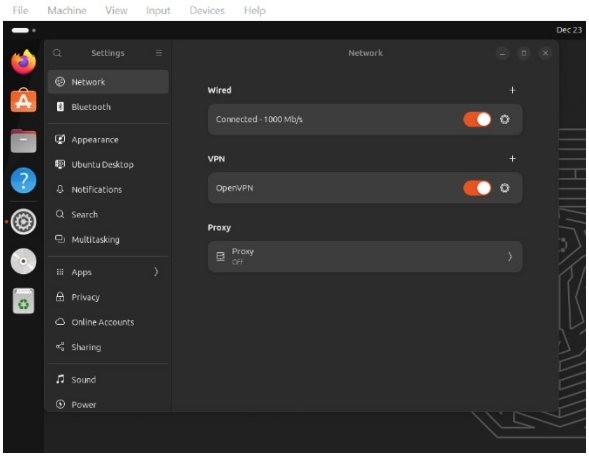
CLIENT AND SERVER SETUP

- Server:

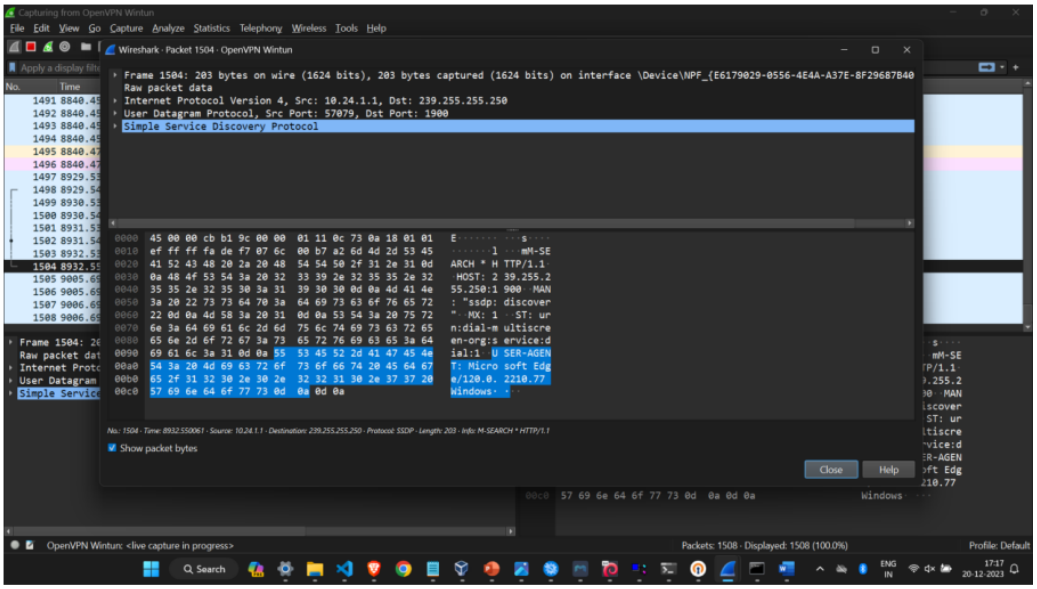
```
● openvpn-server@server.service - OpenVPN service for server
     Loaded: loaded (/lib/systemd/system/openvpn-server@.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2023-12-23 12:39:20 UTC; 35ms ago
       Docs: man:openvpn(8)
             https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage
             https://community.openvpn.net/openvpn/wiki/HOWTO
   Main PID: 7076 (openvpn)
     Status: "Pre-connection initialization successful"
      Tasks: 1 (limit: 18301)
     Memory: 1.0M
        CPU: 28ms
     CGroup: /system.slice/system-openvpn\x2dserver.slice/openvpn-server@server.service
             └─7076 /usr/sbin/openvpn --status /run/openvpn-server/status-server.log --status-version 2 --suppress-timestamps --config server.conf

Dec 23 12:39:20 vini-Vostro-5415 openvpn[7076]: TUN/TAP device tun0 opened
Dec 23 12:39:20 vini-Vostro-5415 openvpn[7076]: net_iface_mtu_set: mtu 1500 for tun0
Dec 23 12:39:20 vini-Vostro-5415 openvpn[7076]: net_iface_up: set tun0 up
Dec 23 12:39:20 vini-Vostro-5415 openvpn[7076]: net_addr_v4_add: 10.8.0.1/24 dev tun0
Dec 23 12:39:20 vini-Vostro-5415 openvpn[7076]: Could not determine IPv4/IPv6 protocol. Using AF_INET
Dec 23 12:39:20 vini-Vostro-5415 openvpn[7076]: Socket Buffers: R=[212992->212992] S=[212992->212992]
Dec 23 12:39:20 vini-Vostro-5415 openvpn[7076]: TCP/UDP: Socket bind failed on local address [AF_INET]10.70.83.216:1194: Cannot assign requested address (errno=99)
Dec 23 12:39:20 vini-Vostro-5415 openvpn[7076]: Exiting due to fatal error
Dec 23 12:39:20 vini-Vostro-5415 openvpn[7076]: Closing TUN/TAP interface
Dec 23 12:39:20 vini-Vostro-5415 openvpn[7076]: net_addr_v4_del: 10.8.0.1 dev tun0
```

- Client:

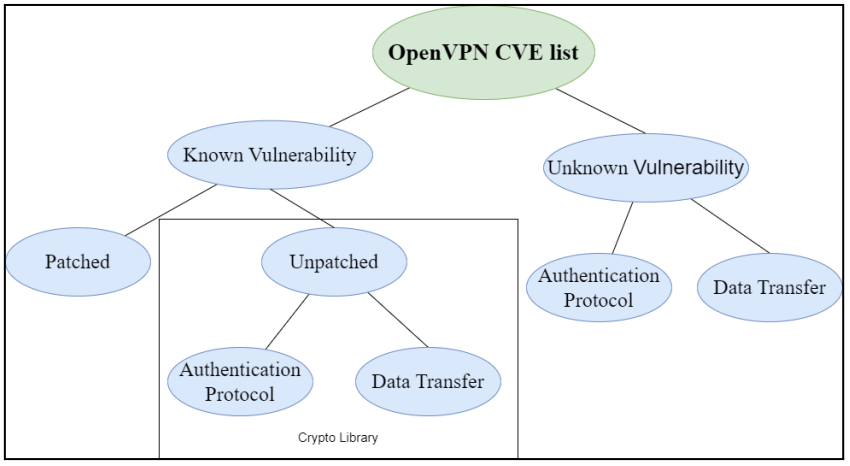


- Wireshark analysis on openvpn client and server connection:



Wireshark

INNOVATION



After following this approach, the vulnerabilities that we narrowed down are:

1. Dereference of a NULL Pointer in file ssl_mbedtls.c
   https://github.com/Atharvakarekar/openvpn/blob/ca84ea53679f97b2957f3cc65ce6041a7fc3a99f/src/openvpn/ssl_mbedtls.c#L702

   Exploitation:

```c
#define ASSERT(cond) \
    do { \
        if (!(cond)) { \
            printf("Assertion failed: %s\n", #cond); \
            return -1; \
        } \
    } while (0)

typedef struct {
    // Define your structure members here
    // ctx, sign_ctx, signature_length, etc.
} CryptoContext;

int sign_data(CryptoContext *ctx, const uint8_t *hash, size_t hashlen) {
    int rv = -1;
    uint8_t *to_sign = NULL;
    size_t asn_len = 0;
    // Simulating potential scenarios where hashlen might be invalid
    // or hash is NULL
    if (hash == NULL || hashlen <= 0 || hashlen > 1000) {
        // Invalid hash or hashlen
        printf("Invalid hash or hashlen!\n");
        return -1;
    }
```

```c
  // Simulating allocation failure
  to_sign = malloc(asn_len + hashlen);
  if (to_sign == NULL) {
      printf("Allocation failed!\n");
      return -1;
  }

  uint8_t *p = to_sign;
  if (ctx->md_alg != MBEDTLS_MD_NONE) {
      // Your ASN.1 encoding logic here
      // ...

      /* Double-check ASN length */
      ASSERT(asn_len == p - to_sign);
  }

  /* Copy the hash to be signed */
  memcpy(p, hash, hashlen);

  /* Simulating ctx->sign or related function failure */
  if (!ctx->sign(ctx->sign_ctx, to_sign, asn_len + hashlen, NULL, ctx->signature_length)) {
      rv = -1; // Signature function failure
      goto done;
  }

  rv = 0; // Successful signing

one:
  free(to_sign);
  return rv;
```

```c
int main() {
    // Simulating usage with invalid parameters
    CryptoContext ctx;
    // Initialize ctx and related parameters

    uint8_t *hash = NULL; // Simulating NULL hash
    size_t hashlen = 1200; // Simulating an invalid hash length

    int result = sign_data(&ctx, hash, hashlen);
    if (result != 0) {
        printf("Signing failed!\n");
        // Handle failure appropriately
    } else {
        printf("Signing successful!\n");
        // Handle success appropriately
    }

    return 0;
}
```

This includes simulations for potential scenarios where hash might be NULL, or hashlen could be invalid (too large or too small). Additionally, it includes a simulation of allocation failure for to_sign. These scenarios are detected and handled within the sign_data function, simulating potential issues in the code flow.

2. Buffer Overflow in file win32.c

https://github.com/Atharvakarekar/openvpn/blob/8656b85c7324fc9ae7f10a9f37227a58766aae33/src/openvpn/win32.c#L975

Exploitation:

```
work = gc_malloc(maxlen + 1, false, gc);
check_malloc_return(work);
buf = alloc_buf_gc(nchars, gc);

for (i = 0; i < a->argc; ++i)
{
    const char *arg = a->argv[i];
    size_t arg_len = strlen(arg);

    if (arg_len > maxlen) {
        // Handle arguments longer than maxlen + 1 here, for example:
        printf("Argument %d is too long: %zu characters\n", i, arg_len);
        continue; // Skip this argument and proceed with the next one
    }

    strcpy(work, arg);
    string_mod(work, CC_PRINT, CC_DOUBLE_QUOTE|CC_CRLF, '_');
    if (i)
    {
        buf_printf(&buf, " ");
    }
    if (string_class(work, CC_ANY, CC_SPACE))
    {
        buf_printf(&buf, "%s", work);
    }
    else
    {
        buf_printf(&buf, "\"%s\"", work);
    }
}
```

Modify arg Length: Provide an argument longer than maxlen + 1 characters.
Observe Behavior: The code might crash or behave unexpectedly due to overwritten memory.

Both these vulnerabilities are currently unpatched in the latest OpenVPN version i.e., version 2.6.8,