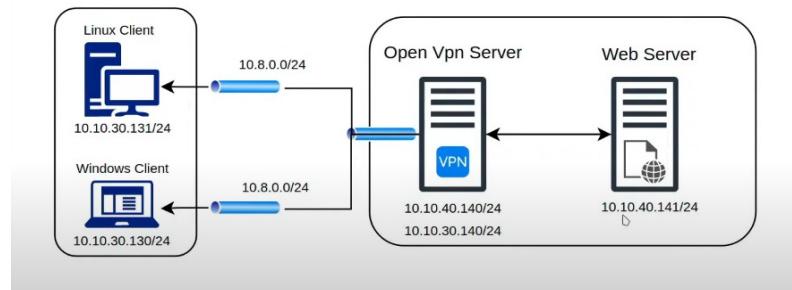


SIH 2023 PROJECT: PS ID- SIH1453 [BITHEADS]

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

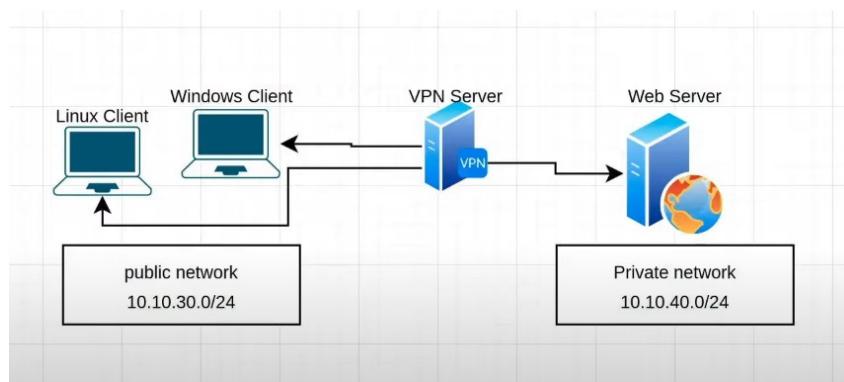
Goal 4: Identification of known and unknown vulnerabilities of crypto libraries in OpenVPN (IPSEC) in IPv6

Goal 5: Investigate and report vulnerabilities configure leading to

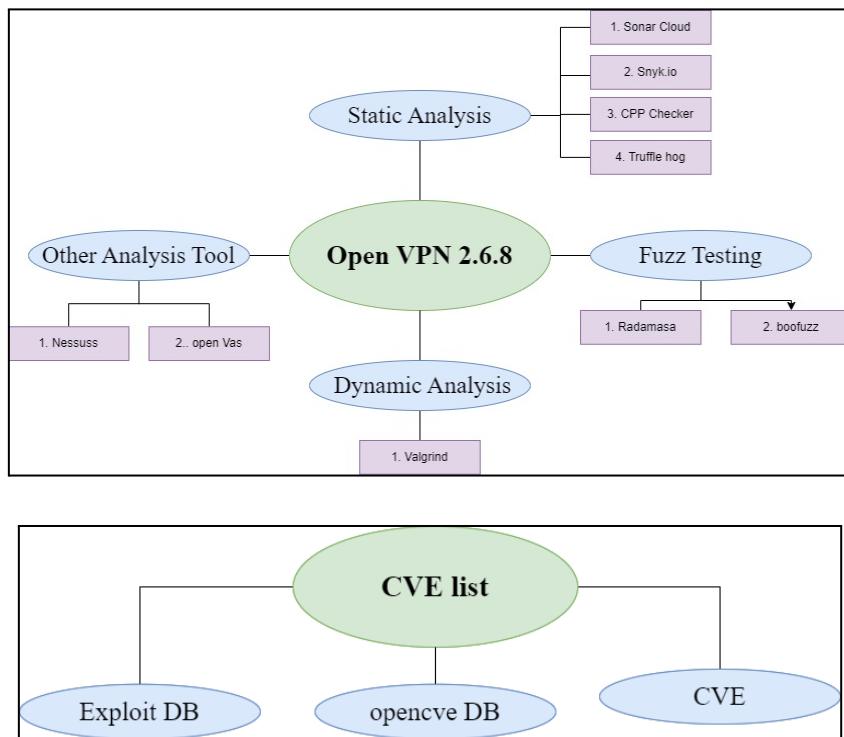
- Exploitation vector
- Compromise of data confidentiality

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Python 2 Source Code (2 instances): Use of outdated and unsupported Python 2.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.


```

==4416== by 0x4400B474: __open_at(0x7f3f8000, 0x100) [dl-open.c:234]
==4416== by 0x44011085: _dl_catch_exception (dl-catch.c:237)
==4416== by 0x40008485: dl_open_worker (dl-open.c:782)
==4416== by 0x40014B88: _dl_catch_exception (dl-catch.c:237)
==4416== by 0x400088A7: _dl_open (dl-open.c:884)
==4416== by 0x40014B88: _dl_catch_exception (dl-catch.c:237)
==4416== by 0x40014B88: _dl_catch_exception (dl-catch.c:237)
==4416== by 0x40015A5E: _dl_catch_error (dl-catch.c:256)
==4416== 
==4416== 9,600 bytes in 1 blocks are still reachable in loss record 11 of 13
==4416== at 0x48400808: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==4416== by 0x136C74: log_history_obj_init (manage.c:4027)
==4416== by 0x138278: log_history_init (manage.c:4036)
==4416== by 0x138278: nan_persist_init (manage.c:2400)
==4416== by 0x138278: nan_persist_init (manage.c:2404)
==4416== by 0x13828F: nan_persist_init (manage.c:2415)
==4416== by 0x1382BF: management_init (manage.c:2644)
==4416== by 0x13348B: init_management (init.c:4292)
==4416== by 0x15108A: openvpn_main (openvpn.c:208)
==4416== by 0x441610C9: (below main) (libc_start_call_main.h:58)
==4416== 
==4416== 9,600 bytes in 1 blocks are still reachable in loss record 12 of 13
==4416== at 0x48400808: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==4416== by 0x136C74: log_history_obj_init (manage.c:4027)
==4416== by 0x138278: log_history_init (manage.c:4036)
==4416== by 0x138278: nan_persist_init (manage.c:2400)
==4416== by 0x138278: nan_persist_init (manage.c:2404)
==4416== by 0x13828F: nan_persist_init (manage.c:2415)
==4416== by 0x1382BF: management_init (manage.c:2644)
==4416== by 0x13348B: init_management (init.c:4292)
==4416== by 0x15108A: openvpn_main (openvpn.c:208)
==4416== by 0x441610C9: (below main) (libc_start_call_main.h:58)
==4416== 
==4416== LEAK SUMMARY:
==4416== definitely lost: 0 bytes in 0 blocks
==4416== indirectly lost: 0 bytes in 0 blocks
==4416== possibly lost: 0 bytes in 0 blocks
==4416== still reachable: 11,539 bytes in 13 blocks
==4416== suppressed: 0 bytes in 0 blocks
==4416== 
==4416== For lists of detected and suppressed errors, rerun with: -s
==4416== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Valgrind

- Findings of Nessus:

CVSS	VRI	Name	Family	Count
5.3		SMB Signing not required	Misc.	1
---	---	SMB (Multiple issues)	Windows	7
---	---	Microsoft Windows (Multiple issues)	Windows	2
---		DCE Services Enumeration	Windows	8
---		Authenticated Check - OS Name and Installed Package Enumeration	Settings	1
---		Common Platform Enumeration (CPE)	General	1
---		Device Type	General	1
---		Host Fully Qualified Domain Name (FQDN) Resolution	General	1
---		Nessus Scan Information	Settings	1
---		Nessus Windows Scan Not Performed with Admin Privileges	Settings	1
---		OS Identification	General	1
---		OS Identification and Installed Software Enumeration over SSH v2 (Ubuntu)	Misc.	1
---		OS Security-Patch Assessment Not Available	Settings	1
---		Target Credential Status by Authentication Protocol - No Credentials P...	Settings	1

Nessus

- Findings of OpenVAS:

Insight
The Timestamp Reply is an ICMP message which replies to a Timestamp message. It consists of the originating timestamp sent by the sender of the Timestamp as well as a receive timestamp and a transmit timestamp.

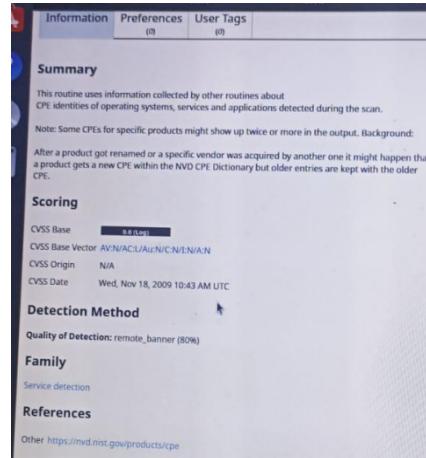
Detection Method
Sends an ICMP Timestamp (Type 13) request and checks if a Timestamp Reply (Type 14) is received.
Quality of Detection: remote_banner (80%)

Impact
This information could theoretically be used to exploit weak time-based random number generators in other services.

Solution
Solution Type: Mitigation
Various mitigations are possible:
- Disable the support for ICMP timestamp on the remote host completely
- Protect the remote host by a firewall, and block ICMP packets passing through the firewall in either direction (either completely or only for untrusted networks)

Family
General

Scans
Information
CVSS Base: 0.0 (Info)
CVSS Base Vector: AV:N/UF/N/C/N/I/N/A/N
CVSS Origin: N/A
CVSS Date: Thu Jul 8, 2010 5:27 PM UTC
Summary
Collect information about the network route and network distance between the scanner host and the target host.
Scoring
CVSS Base: 0.0 (Info)
CVSS Base Vector: AV:N/UF/N/C/N/I/N/A/N
CVSS Origin: N/A
CVSS Date: Thu Jul 8, 2010 5:27 PM UTC
Insight
For internal networks, the distances are usually small, often less than 4 hosts between scanner and target. For public targets the distance is greater and might be 10 hosts or more.
Detection Method
A combination of the protocols ICMP and TCP is used to determine the route. This method is applicable for IPv4 only and it is also known as 'traceroute'.
Quality of Detection: remote_banner (80%)



OpenVAS

- Findings of CVE List:

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

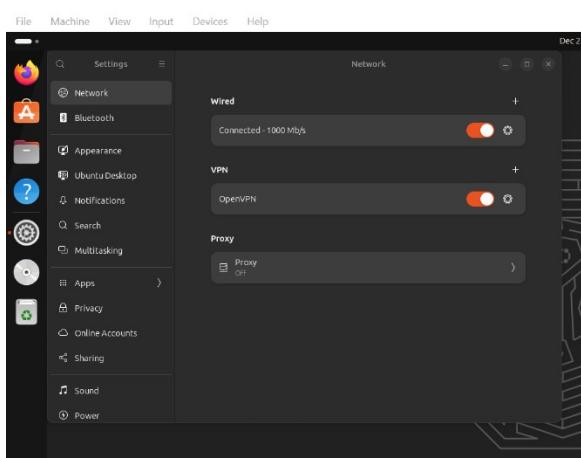
CLIENT AND SERVER SETUP

- Server:

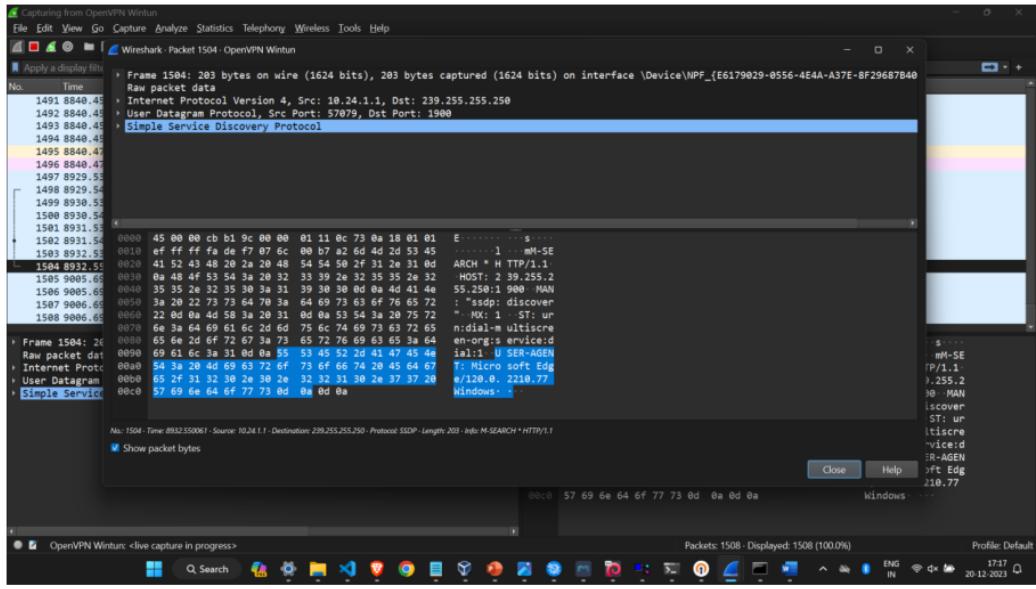
```
● openvpn-server@server.service - OpenVPN service for server
   Loaded: loaded (/lib/systemd/system/openvpn-server@.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2023-12-23 12:39:20 UTC; 35ms ago
     Docs: man:openvpn(8)
           https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage
           https://community.openvpn.net/openvpn/wiki/HOWTO
 Main PID: 7076 (openvpn)
 Status: "Pre-connection initialization successful"
   Tasks: 1 (lmt: 18301)
 Memory: 1.8M
      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 vinci-Vostro-5415 openvpn[7076]: TUN/TAP device tun0 opened
Dec 23 12:39:20 vinci-Vostro-5415 openvpn[7076]: net_iface_mtu_set: mtu 1500 for tun0
Dec 23 12:39:20 vinci-Vostro-5415 openvpn[7076]: net_iface_up: set tun0 up
Dec 23 12:39:20 vinci-Vostro-5415 openvpn[7076]: net_addr_v4_add: 10.8.0.1/24 dev tun0
Dec 23 12:39:20 vinci-Vostro-5415 openvpn[7076]: Could not determine IPv4/IPv6 protocol. Using AF_INET
Dec 23 12:39:20 vinci-Vostro-5415 openvpn[7076]: openvpn[7076]: [if=eth0, linklayer=52:12:99:92] S=[212992-5212992]
Dec 23 12:39:20 vinci-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 vinci-Vostro-5415 openvpn[7076]: Exiting due to fatal error
Dec 23 12:39:20 vinci-Vostro-5415 openvpn[7076]: Closing TUN/TAP Interface
Dec 23 12:39:20 vinci-Vostro-5415 openvpn[7076]: net_addr_v4_del: 10.8.0.1 dev tun0
```

- Client:

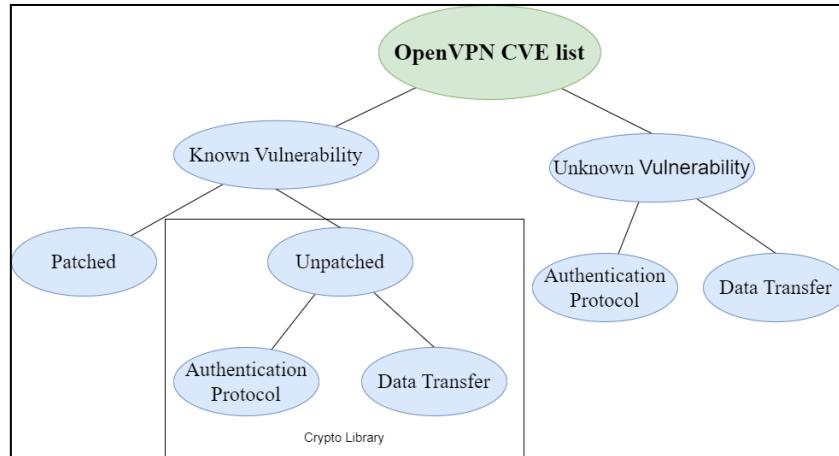


- Wireshark analysis on openvpn client and server connection:



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:

```

#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;
    }

    // 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
done:
    free(to_sign);
    return rv;
}

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,