**SSHTunneler Framework**

1. **Introducción:**

*Debido al aumento de mecanismos de monitorización de incidentes de ciberseguridad y de detección de intrusos cada vez es más importante reducir el tiempo de un test de intrusión para evitar ser detectado, de la misma manera, utilizar herramientas nativas del sistema para exfiltrar datos o realizar movimientos laterales son la mejor alternativa para pasar desapercibido.*

*Por estas razones SSHTunneler ve la luz, para usar comandos legítimos que combinados pueden comprometer un sistema, exfiltrar datos y en general, poner a prueba la seguridad de una empresa u organismo de manera automatizada y permitiendo la colaboración de un equipo de personas.*

1. **Resumen:**

*SSHTunneler es un framework diseñado para facilitar y automatizar las tareas propias de un pentester, una vez comprometido un equipo inicial y teniendo credenciales de acceso SSH.*

*La filosofía subyacente de la aplicación es la de poder ejecutar comandos de sistema tal como lo haría un administrador legítimo, pero permitiendo la exfiltración de dichos datos y la automatización de tareas comunes sin necesidad de ir escribiendo cada comando y con la posibilidad de almacenar el proceso para su reutilización en posteriores pentestings.*

*Al ser multisistema y con acceso web, es posible publicarla tanto en Docker como en Cloud y que diferentes profesionales la usen y compartan automatizaciones y comandos de una forma ágil.*

*Como beneficio adicional, la herramienta puede ser usada por administradores de sistemas para poder gestionar equipos remotos de manera automatizada.*

1. **Requerimientos:**
2. **Funciones principales:**

*La herramienta debe ser capaz de conectar a cualquier equipo del que se posean credenciales, en forma de usuario y contraseña o usuario y certificado.*

*La herramienta debe ser capaz de ejecutar cualquier comando que pudiera ejecutar un usuario de forma física.*

*La herramienta debe permitir al usuario crear una batería de pruebas y almacenarla para su posterior ejecución.*

*La herramienta debe permitir al usuario configurar un timeout específico para cada acción.*

*La herramienta debe ser capaz de lanzar una batería de acciones específicas permitiendo al ejecutor guardar el resultado de dichas acciones para:*

1. *Efectuar acciones sobre los datos obtenidos.*
2. *Documentar el progreso del pentesting y las debilidades apreciadas.*
3. *Examinar los resultados a posteriori después de lanzar un ataque de manera desatendida.*

*La herramienta debe ser capaz de funcionar en cualquier sistema operativo, ya sea Linux, Windows, Mac, sobre Docker o en plataformas Cloud.*

*La configuración de los comandos debe ser simple y la adición de nuevos comandos debe ser inmediata.*

*La herramienta debe ser capaz de mantener múltiples conexiones con diferentes objetivos y de poder cambiar entre ellos de forma dinámica.*

*Los módulos de la herramienta pueden ser automatizados mediante scripting sin necesidad de usar la UI existente.*

1. **Funciones adicionales:**

*La herramienta debe tener la capacidad de crear un mapa de red en base a la información descubierta.*

*La herramienta debe tener la capacidad de ejecutar un programa externo de rotura de hashes para vulnerar contraseñas.*

*La herramienta debe ser capaz de redireccionar puertos al servidor local o a cualquier equipo intermedio de una manera sencilla.*

*La herramienta debe permitir conexiones a través de puertos redireccionados manteniendo la información sobre el objetivo.*

1. **Detalles de la solución:**

**Pila de tecnologías:**

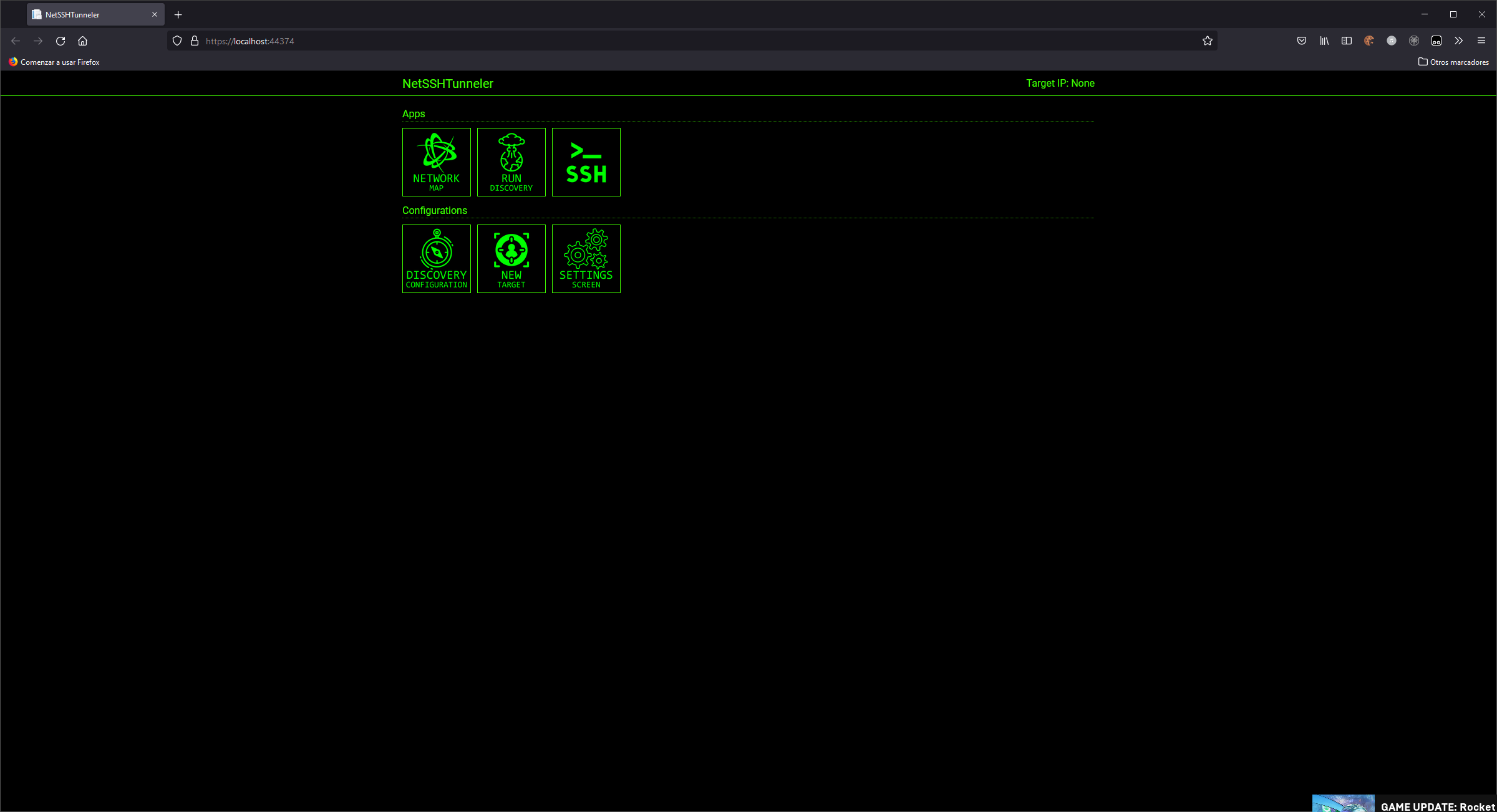
*Microsoft .NET 5.0 para el Back-End.*

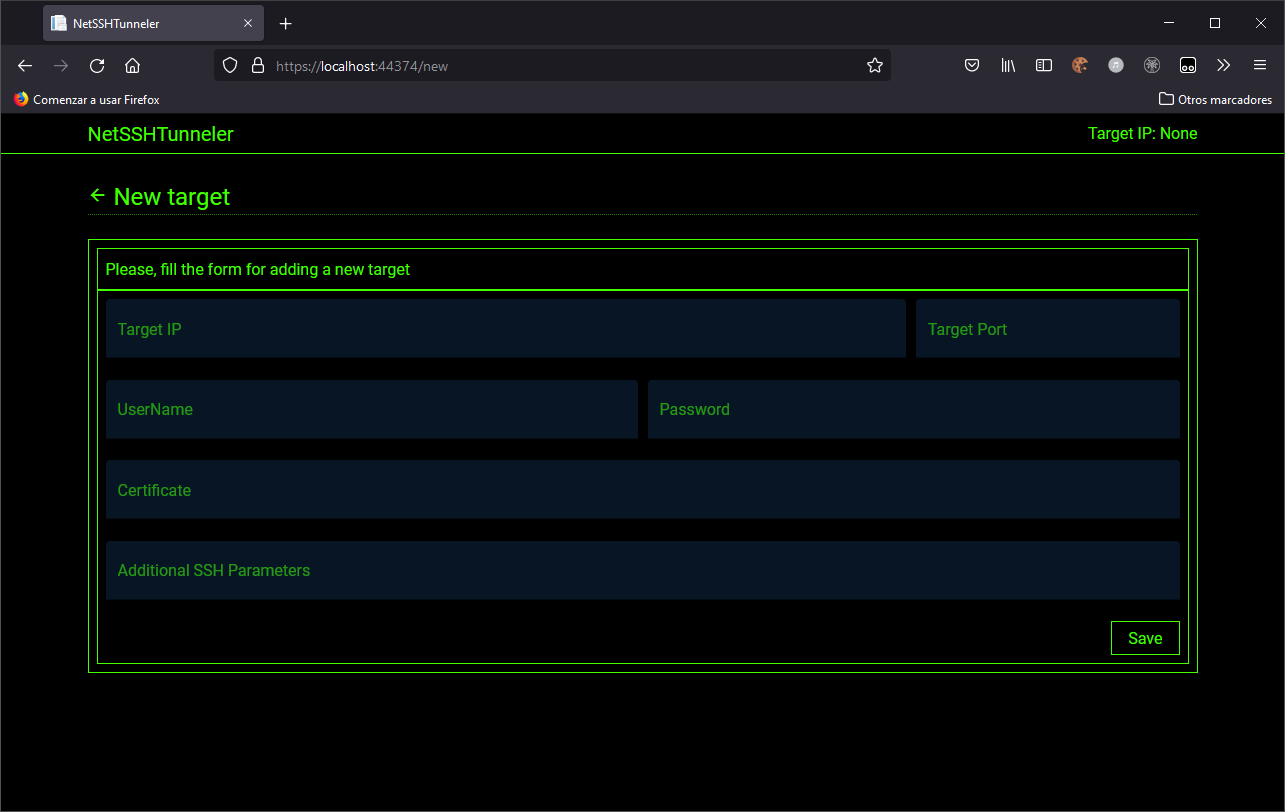
*AngularJS para el Front-End.*

*Se utilizará la librería Renci SSH.NET para la comunicación SSH con los equipos.*

*Se dará la opción al usuario de elegir entre JTR y Hashcat para la configuración de crackeo de contraseñas.*

1. **Funciones e Implementación:**
2. ***Modulo New Target:***

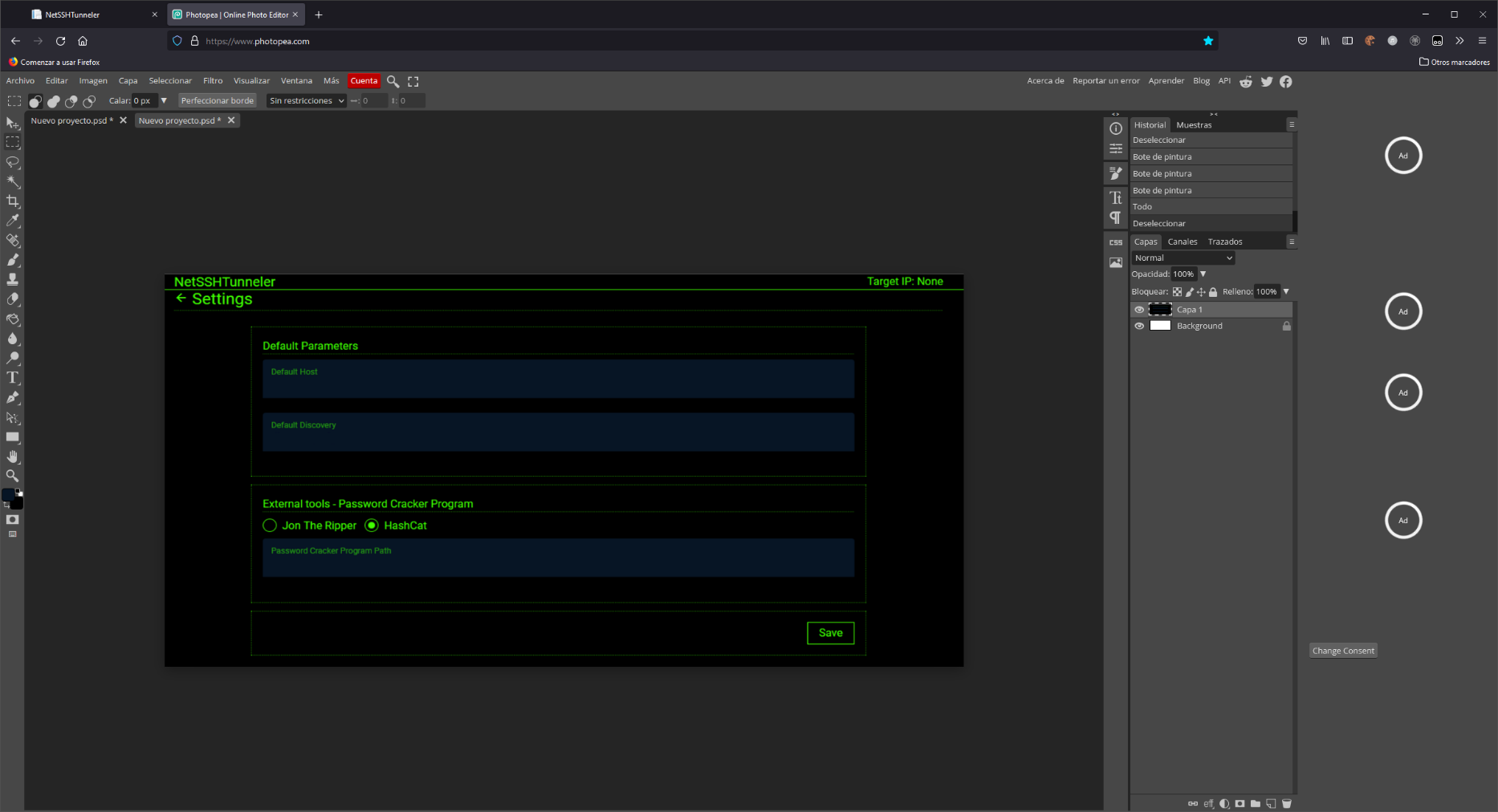
 *En esta pantalla el usuario configura los parámetros para un nuevo objetivo, se especifica y se almacena para su uso posterior.*

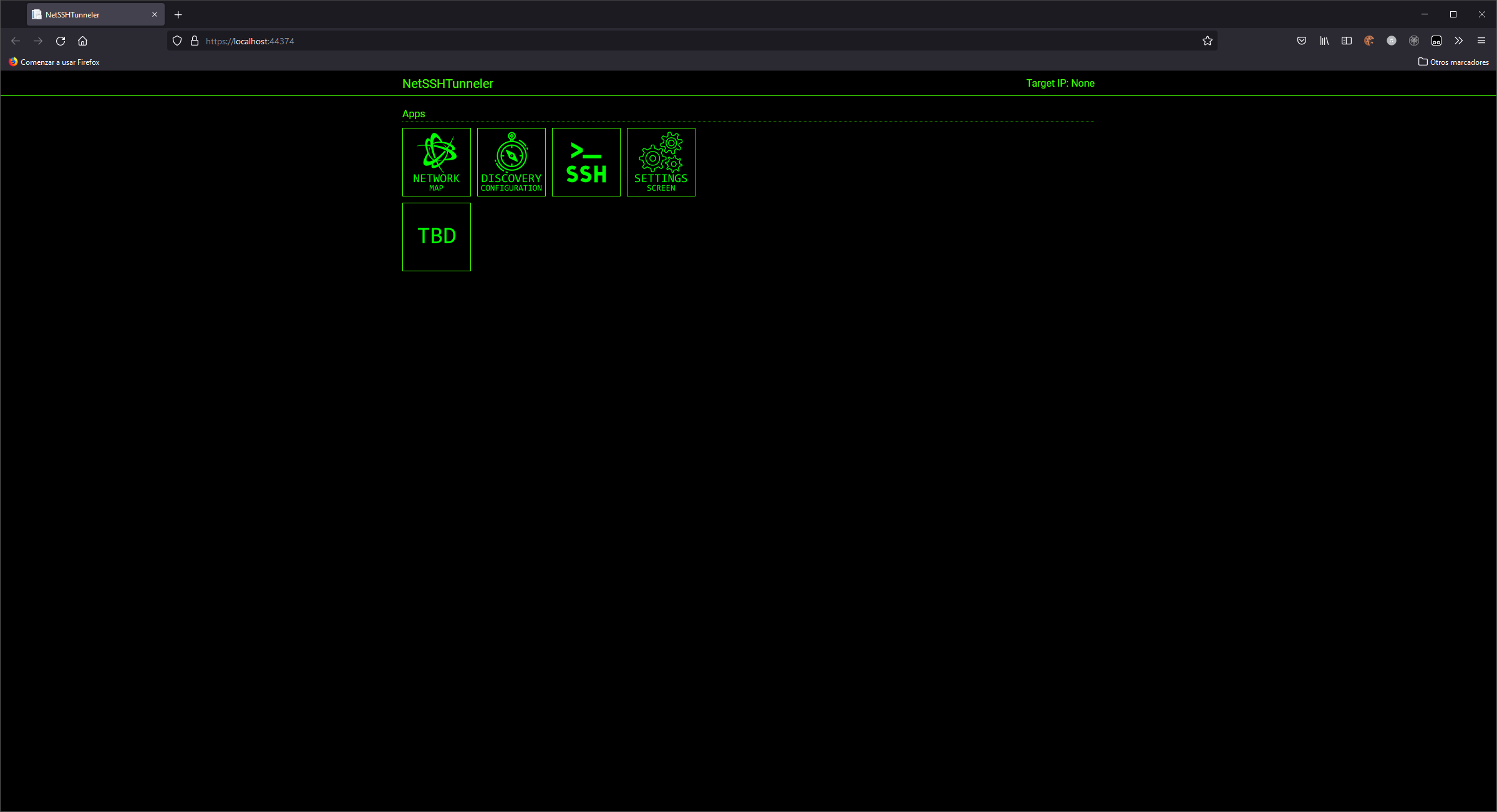
 *El valor del campo certificate debe corresponder al nombre del certificado que se vaya a utilizar para la conexión y debe estar en la carpeta certificates de la aplicación.*

1. ***Modulo Settings:***

*En esta pantalla el usuario configura los parámetros por defecto para el objetivo inicial el bloque de comandos que se quiere por defecto y la herramienta externa para el crackeo de hashes, actualmente la herramienta soporta tanto* [*Hashcat*](https://hashcat.net/hashcat/) *como* [*John the Ripper*](https://www.openwall.com/john/) *en modos fuerza bruta y diccionario.*

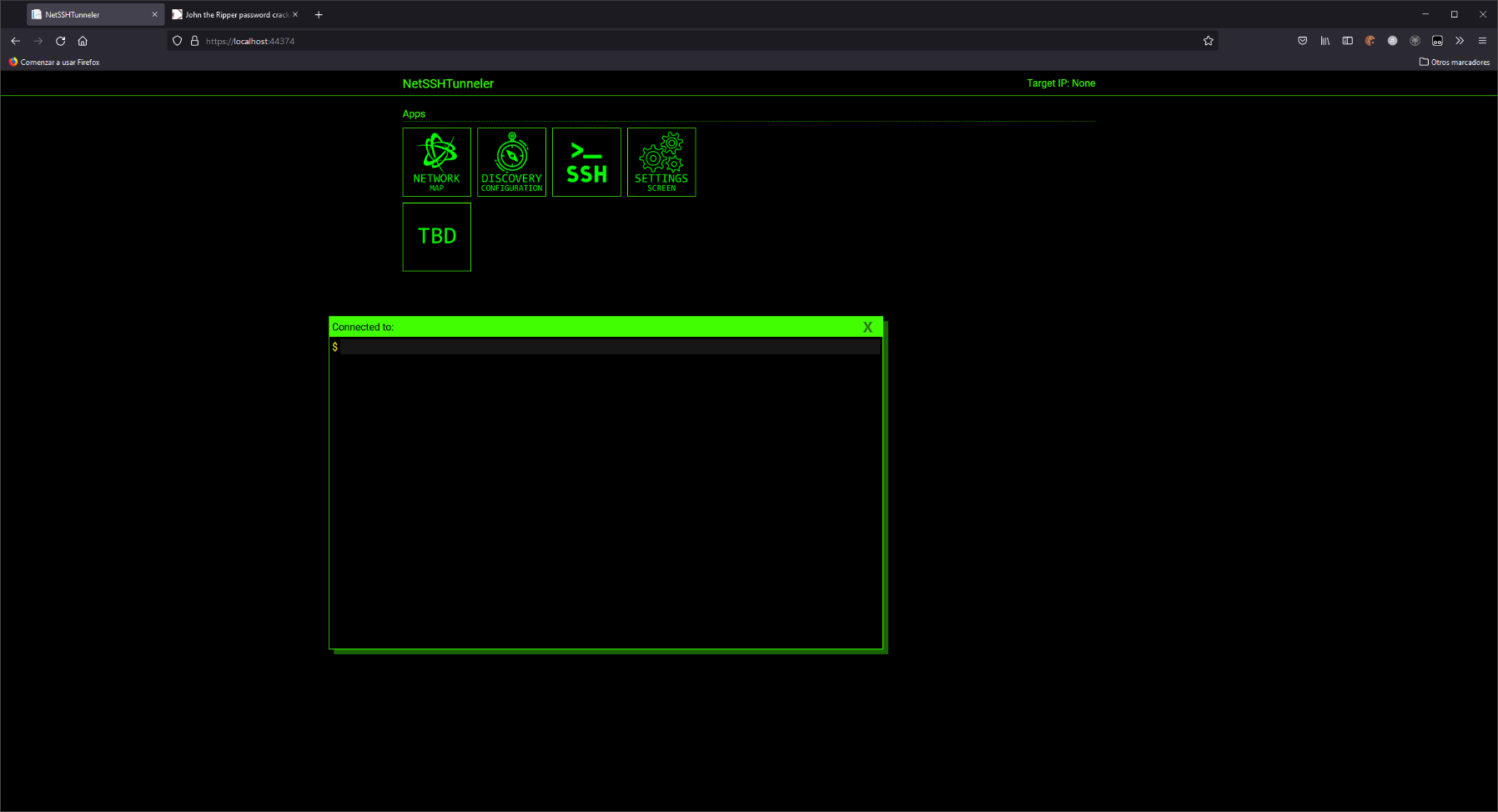
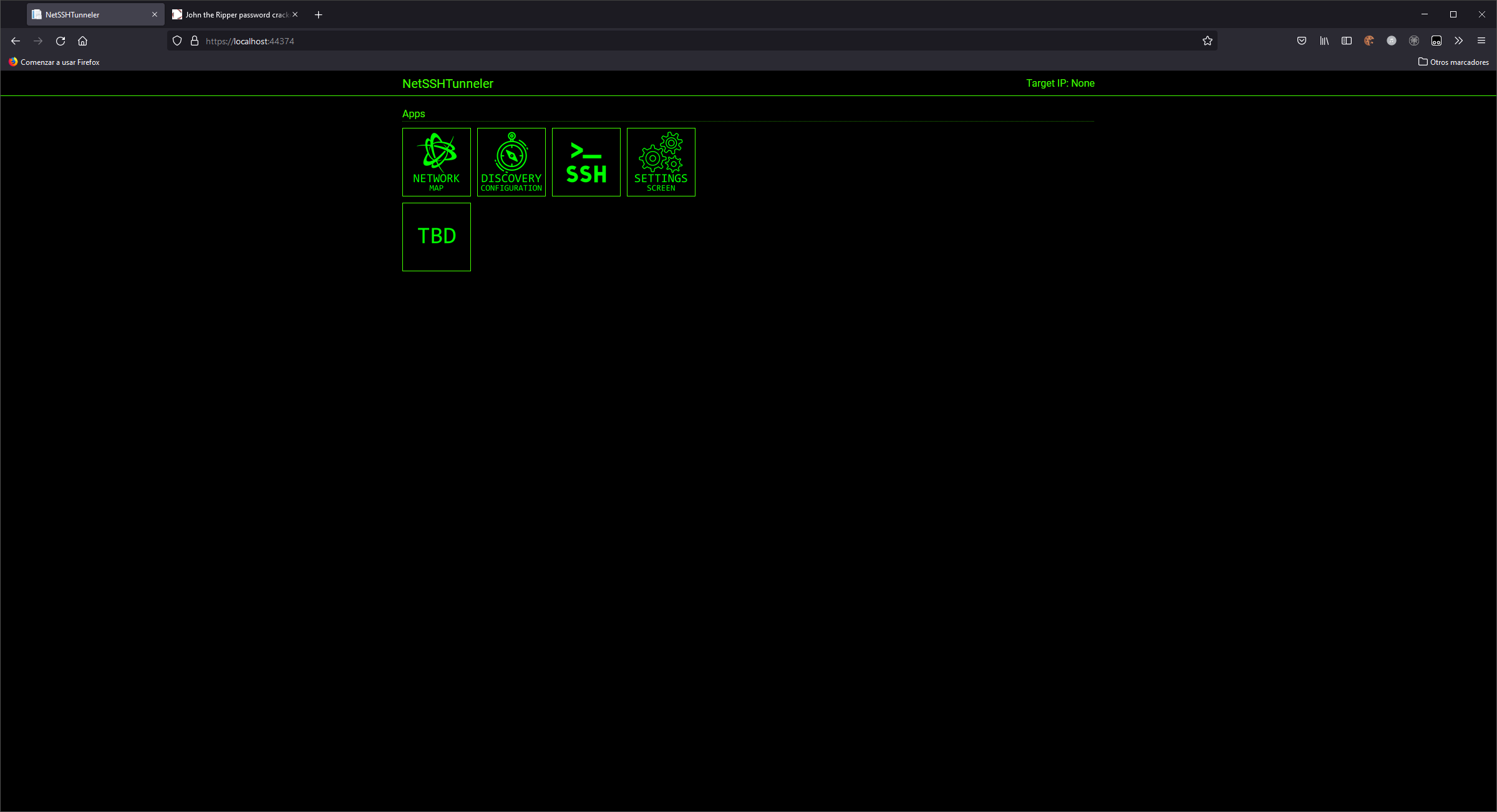
*En la casilla de “Password Cracker Program Path” debemos poner la ruta absoluta al ejecutable de la herramienta seleccionada.*





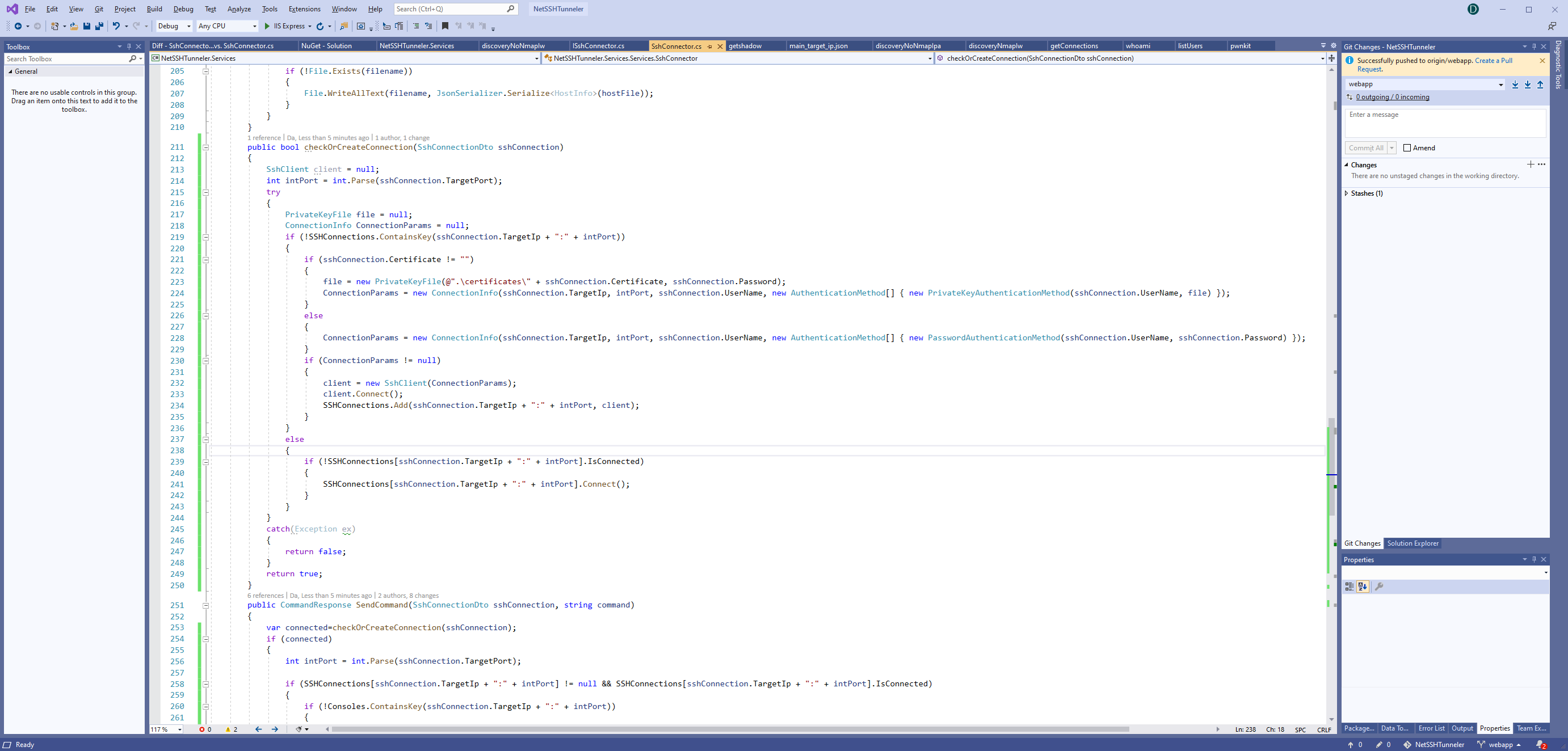
*Esta configuración se almacena en formato JSON dentro de la carpeta configuración de la solución.*

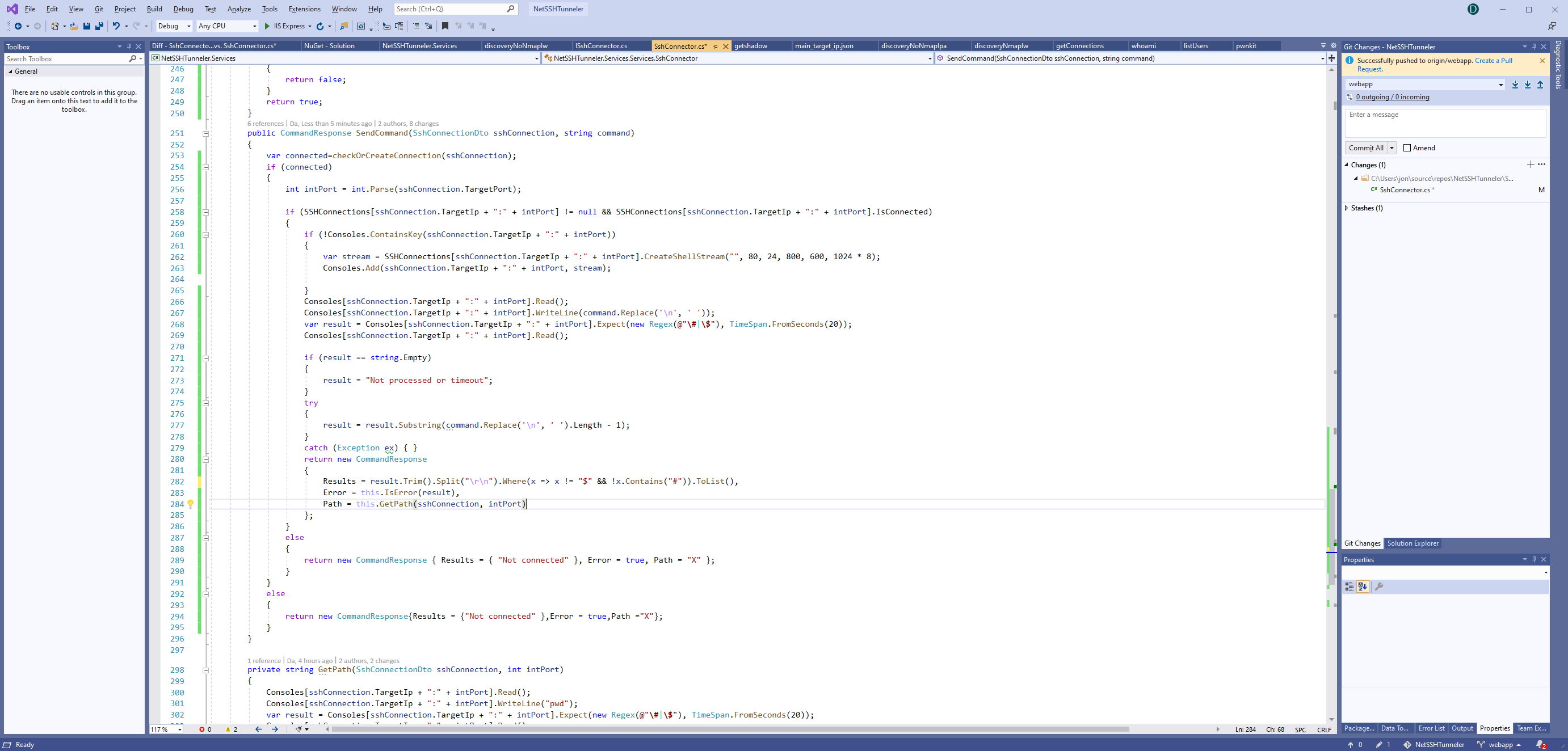
1. ***Módulo Terminal:***

*Esta pantalla permite la ejecución de una Shell SSH sobre el objetivo seleccionado, actualmente cualquier comando que no requiera interacción puede ser ejecutado y su respuesta capturada en la terminal.*

*Se implementa la clase SshConnector que permite gestionar las conexiones y generar tantas SSHShells como sean necesarias para la ejecución de los comandos de terminal.*

*Se utilizan ShellStreams para poder dar persistencia a los comandos y se implementa una devolución de las respuestas por línea para ser tratadas en el Front-End se crean también unas listas persistentes en forma de IP:Puerto->ClienteSSH e IP:Puerto->ShellStream para poder manejar conexiones continuas.*

 *Se utilizan las funciones checkOrCreateConnection y SendCommand para todas las operaciones de terminal de la manera detallada a continuación:*

*El método sendCommand está accesible a través de un servicio rest en el endpoint* [*http://server:port/command/send*](http://server:port/command/send) *y acepta un JSON con el formato*

{

    "TargetIp": "ConnectionID",

    "Command": "Command"

}

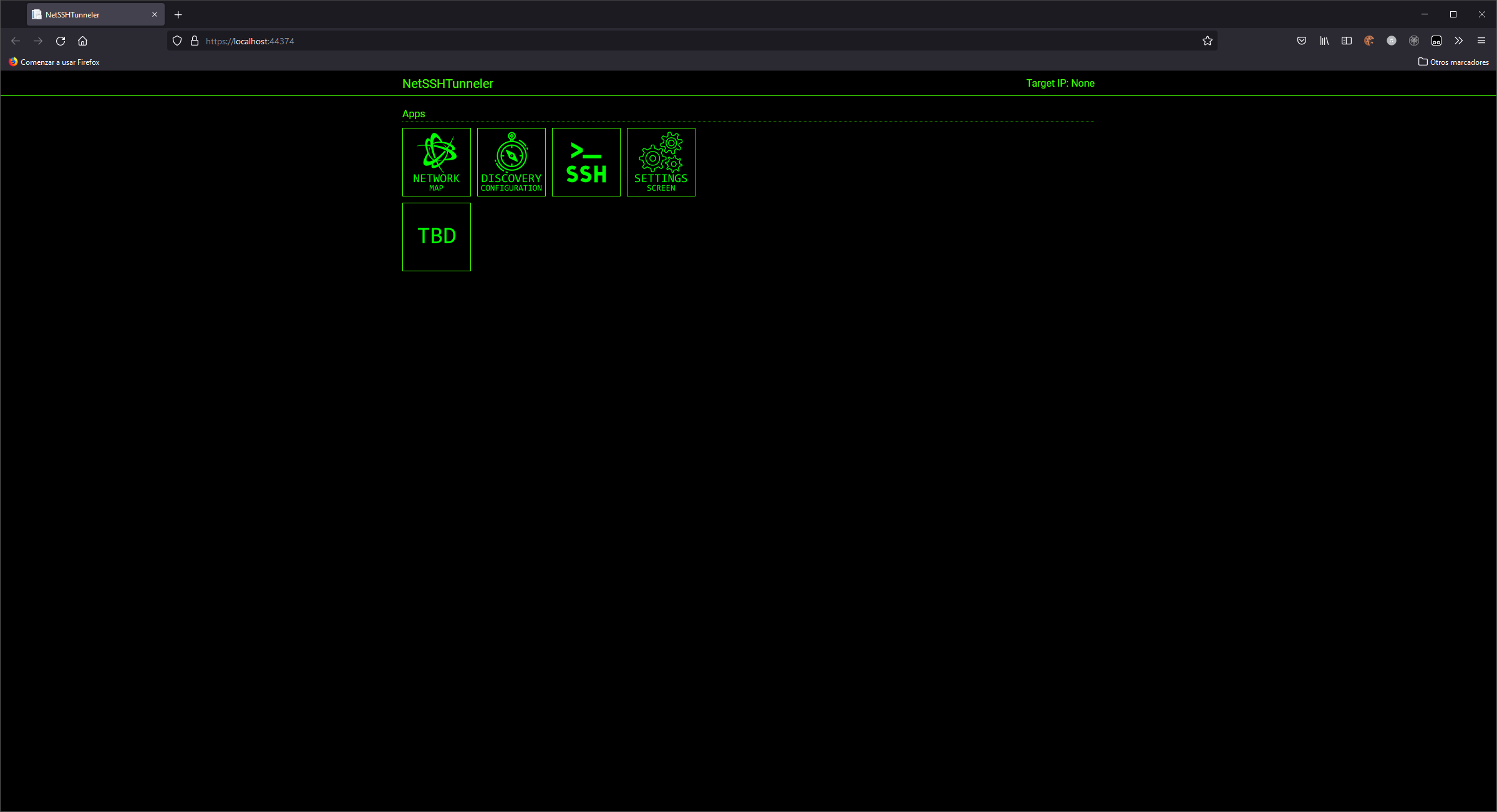
1. ***Módulo Configuración Discovery:***

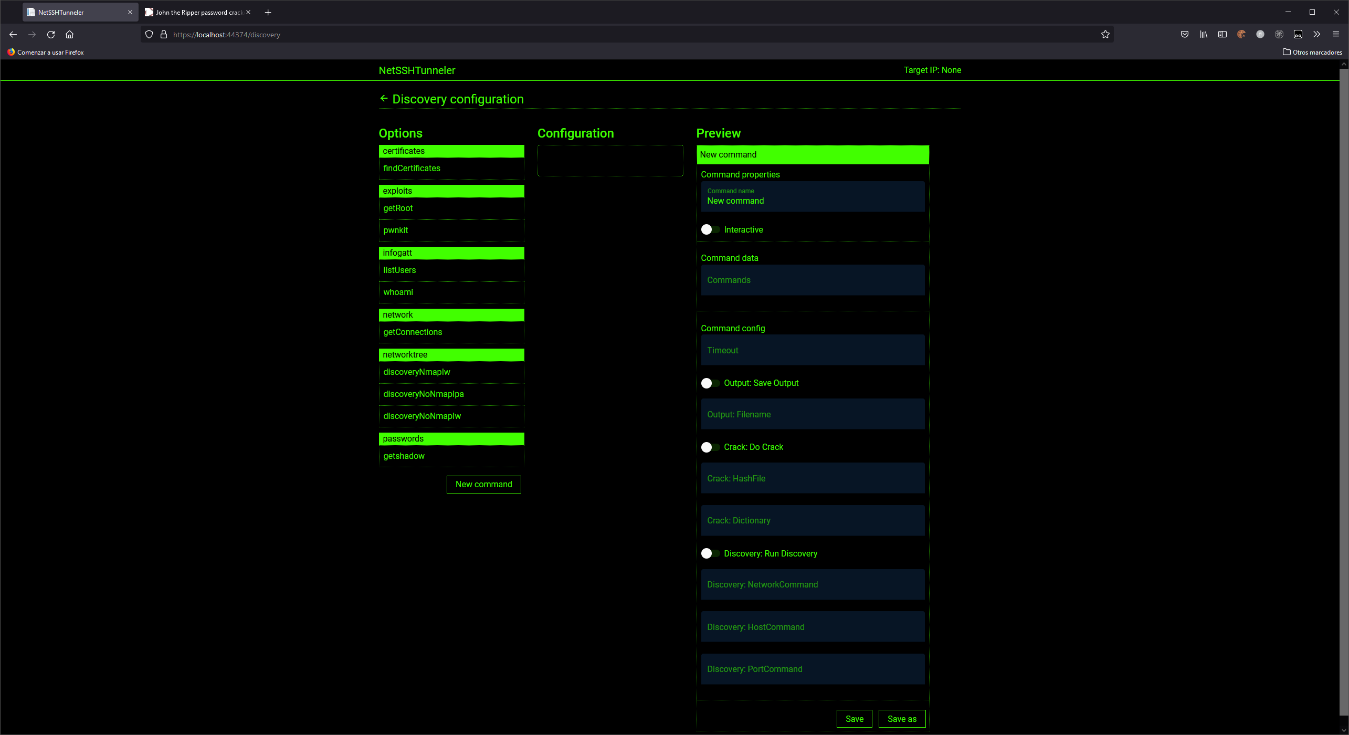
*En esta pantalla el usuario configura los parámetros para el ataque automatizado, se permite grabar el contenido de la acción a ejecutar en forma de comando y organizar cada uno de ellos para fijar el orden de ejecución. Estas automatizaciones pueden ser guardadas y recuperadas para ser ejecutadas posteriormente. De manera que se pueden preparar automatismos de recolección de información, de explotación, etc.*

*Cada comando permite ser personalizado con el timeout correspondiente, puede ser capaz de grabar el resultado en una carpeta local y puede activar la función Crack llamando a la herramienta de cracking configurada.*

*Existe un tipo de comando especial que realiza las funciones de descubrimiento de red, en el caso de que se use este comando es necesario introducir tres comandos, uno para descubrir qué redes son accesibles desde el objetivo, el segundo para descubrir los hosts visibles y el tercero para escanear los puertos de dichos hosts.*

*Los comandos que requieran interacción no son compatibles con el proceso de Discovery en esta versión.*





*Cada comando se almacena en un fichero de comando con contenido JSON en este formato:*

{

"Commands": [

"command 1 -X -Y","command 2 -W -X"

],

"Interactive": boolean,

"CommandConfig": {

"Timeout": integer,

"Output": {

"SaveOutput": boolean,

"Filename": ""

},

"Crack": {

"DoCrack": boolean,

"HashFile": "",

"Dictionary": ""

},

"Discovery": {

"RunDiscovery": boolean,

"HostCommand": "",

"NetworkCommand": "",

"PortCommand": ""

}

}

}

*Al recuperar la lista de comandos, la aplicación recorre la carpeta Discovery y genera una categoría por cada carpeta encontrada de manera que podemos organizar los comandos de la manera que sea más cómoda para el usuario. De igual modo, si la aplicación se utiliza de manera colaborativa, tan pronto como un usuario añade un comando, este pasa a estar disponible para el resto de los usuarios.*

*Opciones de los comandos:*

*Commands: cada comando está compuesto por uno o más comandos de sistema, cada comando se procesa con la lógica configurada. Para configurar múltiples comandos en una acción los comandos deben separarse con el símbolo “^”*

*SaveOutput: Si el comando tiene la opción de SaveOutput marcada, se generará un fichero con el contenido de comando, de esta manera podemos volcar por ejemplo certificados PEM, ficheros shadow o cualquier otra información sensible que se encuentre en el servidor sin activar ninguna alerta de copia.*

*Crack: Si sabemos que el fichero de output contendrá hashes, podemos activar la función crack apuntando al fichero generado y esto lanzará Hashcat o John contra el fichero seleccionado en segundo plano usando el diccionario especificado o con fuerza bruta si no se especifica y seguirá ejecutando el resto de instrucciones.*

*El modificador crack solo funciona en sistemas Windows por el momento*

*Discovery: Para poder lanzar el discovery de manera correcta, los comandos deben presentar un output siguiendo estas reglas:*

*“NetworkCommand”: Debe contener un formato de red IPv4 (XXX.XXX.XXX.XXX/X)*

*“HostCommand”: Debe contener un formato de host IPv4 (XXX.XXX.XXX.XXX) y un literal {network} que será reemplazado por las redes descubiertas en el comando previo*

*“PortCommand”: Debe contener los puertos abiertos por línea en formato “open XXXX” y un literal {host} que será reemplazado por los hosts descubiertos previamente.*

*Para cada host que encuentre en este proceso, el programa genera un fichero de configuración con la información necesaria para ser añadido al mapa de red.*

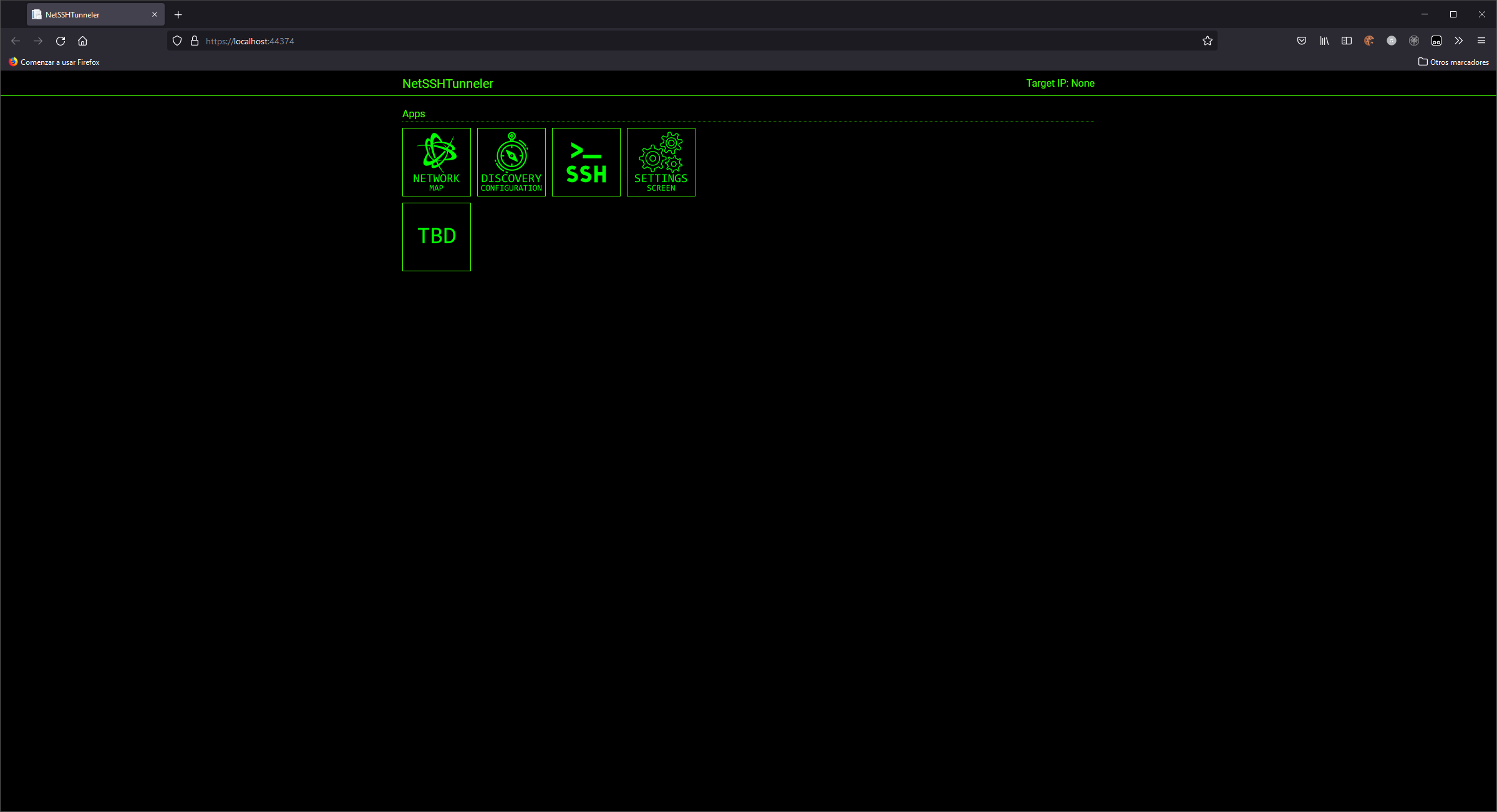
1. ***Módulo Network Map:***

*En esta pantalla el usuario puede ver el mapa de todas las redes, equipos y puertos descubiertos.*

*Para ayudar visualmente al usuario, se usa un código de iconos y colores para identificar el tipo de elemento mostrado y, en el caso de que sea un equipo, si este tiene servicios expuestos se mostrará en verde y si no los tiene se mostrará en rojo.*

*Adicionalmente, si el usuario pulsa sobre un equipo verde, se nos muestra la configuración de conexión para rellenarla y una vez rellena, ese equipo puede ser seleccionado como objetivo.*

*Si en lugar de un equipo, el usuario selecciona un puerto, el programa le da la opción de redireccionarlo a un puerto local, lo que permite avanzar en el movimiento lateral a través de las distintas redes.*



*Para poder representar el mapa se utiliza el propio fichero de configuración del objetivo, donde a parte de la información de conexión se almacena la red a la que pertenece, los puertos abiertos detectados y la información de la máquina a través de la cual ha sido descubierto. La estructura de los ficheros de configuración se almacena en forma de JSON de esta manera:*

{

"Network": "",

"Parent": "",

"Ports": [],

"conectionInfo": {

"TargetIp": "",

"TargetPort": "",

"UserName": "",

"Password": "",

"Certificate": "",

"AdditionalSshParameters": ""

}

}

*Para redireccionar los puertos, utilizamos la propia función de SSH.NET de portforwarding, y montamos el nuevo puerto en una lista de puertos utilizables y permanentes.*

*El código para realizar esta redirección es el siguiente:*

public bool RedirectPort(SshConnectionDto sshConnection, string originHost, uint originPort, uint destinationPort,string destinationHost)

{

var connected = checkOrCreateConnection(sshConnection);

int intPort = int.Parse(sshConnection.TargetPort);

if (connected)

{

if (!ForwardedPorts.ContainsKey(originHost + ":" + originPort.ToString() + "=>" + destinationHost + ":" + destinationPort.ToString()))

{

var fwport = new ForwardedPortLocal(destinationHost, destinationPort, originHost, originPort);

SSHConnections[sshConnection.TargetIp + ":" + intPort].AddForwardedPort(fwport);

fwport.Start();

ForwardedPorts.Add(originHost + ":" + originPort.ToString() + "=>" + destinationHost + ":" + destinationPort.ToString(), SSHConnections[sshConnection.TargetIp + ":" + intPort]);

}

return true;

}

else

{

return false;

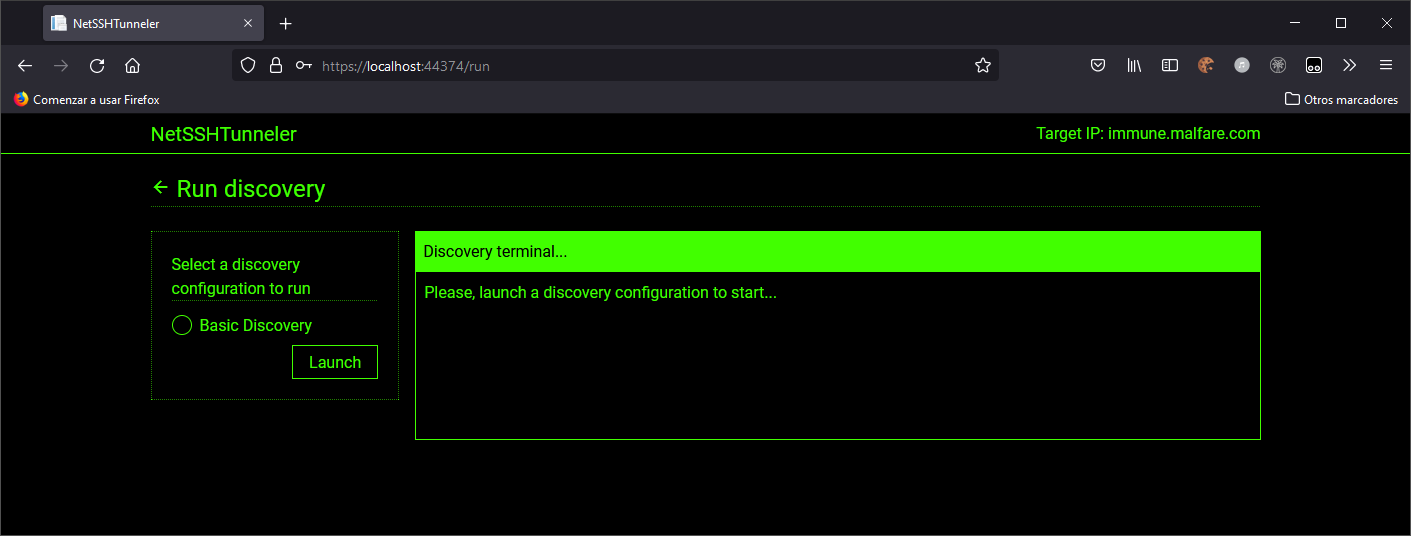
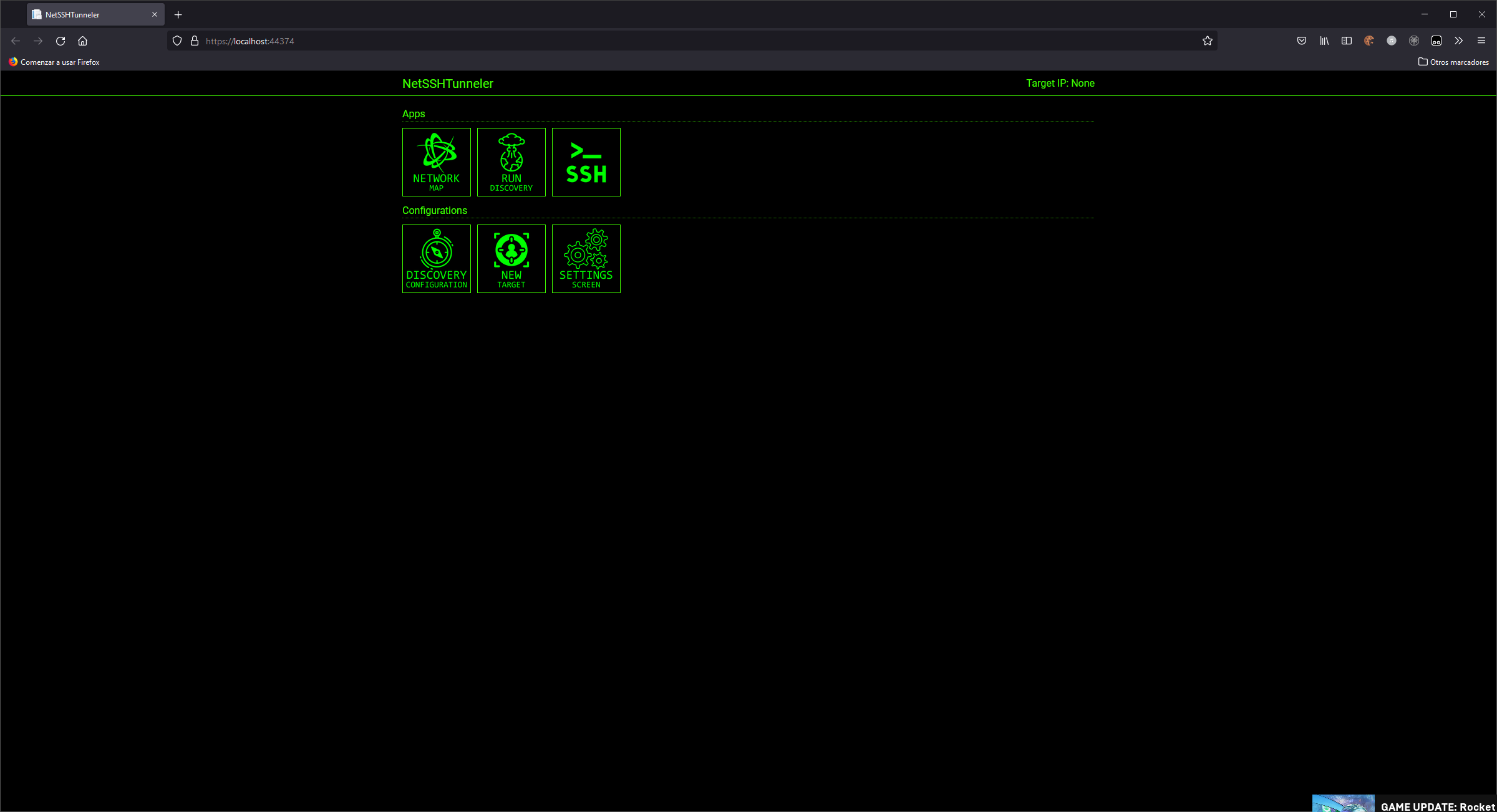
}

}

*Todos los puertos redireccionados se guardan con su relación origen-destino para poder ser identificados posteriormente. P.e: 192.168.56.99:80=>127.0.0.1:1485 nos indicará que el puerto 80 del host 192.168.56.99 se ha redireccionado al puerto local 1485.*

1. ***Módulo Run Discovery:***

*Desde aquí es desde donde se ejecuta el ataque automatizado, en esta pantalla el usuario puede ver el listado de todas las configuraciones Discovery guardas previamente y seleccionar cual quiere ejecutar una vez seleccionada, los comandos se ejecutarán en el orden configurado y al acabar nos mostrará el reporte del resultado.*



*Aquí es donde se tratan las configuraciones avanzadas de los comandos, a continuación, se muestran las partes del código involucradas:*

public DiscoveryResults ProcessDiscovery(SshConnectionDto sshConnection, List<string> files)

{

DiscoveryResults result =new DiscoveryResults();

foreach (string file in files)

{

string content=File.ReadAllText(file);

CommandContainer command = JsonSerializer.Deserialize<CommandContainer>(content);

if (string.IsNullOrEmpty(command.CommandConfig.Discovery.NetworkCommand))

{

foreach (string execution in command.Commands)

{

var response = SendCommand(sshConnection, execution);

result.commandResponses.Add(response);

if (command.CommandConfig.Output.SaveOutput)

{

File.WriteAllLines(command.CommandConfig.Output.Filename, response.Results);

}

if (command.CommandConfig.Crack.DoCrack)

{

Crack(command);

}

}

}

else

{

result.commandResponses.AddRange(discoverNetwork(command, sshConnection).commandResponses);

}

}

return result;

}

*En este bloque se detecta si el tipo es un comando normal o uno de Discovery, en el caso de comando normal se envía de la misma manera que si lo hiciésemos desde el terminal y se comprueba:*

*En caso de necesitar copia local se graba el fichero en disco.*

*En caso de necesitar cómputo de hashes se manda a la función crack:*

private void Crack(CommandContainer command)

{

var configuration= GetGlobalConfig();

string hashtype = "500";

string attackmode = "3";

string dictionary = "";

FileInfo fi = new FileInfo(command.CommandConfig.Crack.HashFile);

var hashfile = fi.FullName;

string filetocrack = File.ReadAllText(hashfile);

if (configuration.cracker == 0) //john the Ripper

{

attackmode = "--incremental";

hashtype = "";

if (!string.IsNullOrEmpty(command.CommandConfig.Crack.Dictionary))

{

attackmode = "";

dictionary = "--wordlist="+command.CommandConfig.Crack.Dictionary;

}

else

{

attackmode = "--incremental";

}

string modifiers = "";

string outputfile = "--pot=" + hashfile;

LaunchProcess(hashtype, attackmode, dictionary, hashfile, outputfile,modifiers);

}

if (configuration.cracker == 1) //hashcat

{

if (filetocrack.Contains("$5$"))

{

hashtype = "-m 7400";

}

if (filetocrack.Contains("$6$"))

{

hashtype = "-m 1800";

}

if (!string.IsNullOrEmpty(command.CommandConfig.Crack.Dictionary))

{

attackmode = "-a 0";

dictionary = command.CommandConfig.Crack.Dictionary;

}

hashfile = "-o " + hashfile;

string modifiers = "-O -d 1";

LaunchProcess(hashtype, attackmode, dictionary, hashfile, hashfile, modifiers);

}

}

*En esta función se determinan los parámetros de ejecución para Hashcat o John según configuración general y se establece el diccionario especificado en el comando o la fuerza bruta si este no está presente. En el caso de Hashcat se intenta también identificar el tipo de hash.*

*Una vez fijados los parámetros se invoca al método LaunchProcess que lanza el proceso de crackeo en segundo plano y que una vez acabe dejará la contraseña en claro como nombredelfichero\_cracked.txt*

private void LaunchProcess(string hashtype, string attackmode, string dictionary, string hashfile, string outputfile, string modifiers)

{

var process = new Process();

process.StartInfo.FileName = "cmd";

process.StartInfo.WorkingDirectory = new FileInfo($"{GetGlobalConfig().crackerPath}").DirectoryName;

process.StartInfo.Arguments = $"/c start {GetGlobalConfig().crackerPath} {attackmode} {hashtype} {modifiers} {outputfile}\_cracked.txt {hashfile} {dictionary}";

process.StartInfo.CreateNoWindow = false;

process.StartInfo.UseShellExecute = true;

process.StartInfo.RedirectStandardOutput = false;

process.StartInfo.RedirectStandardError = false;

process.Start();

}

*Si el comando es un comando Discovery, se ignora cualquier cosa que haya en el texto del comando y se pasa directamente al proceso de descubrimiento de red:*

private DiscoveryResults discoverNetwork(CommandContainer command, SshConnectionDto stream)

{

DiscoveryResults result = new DiscoveryResults();

var response = SendCommand(stream, command.CommandConfig.Discovery.NetworkCommand);

result.commandResponses.Add(response);

foreach (string network in response.Results)

{

result.commandResponses.AddRange(HostScan(command, stream,network).commandResponses);

}

return result;

}

*En este primer bloque se ejecuta el comando de descubrimiento de red (p.e: ip a, iwconfig, ifconfig, …) que devuelva las redes disponibles en el objetivo y se pasa de forma ciega el contenido línea a línea al siguiente bloque:*

private DiscoveryResults HostScan(CommandContainer command, SshConnectionDto stream, string network)

{

DiscoveryResults result = new DiscoveryResults();

if (Regex.IsMatch(network, "([0-9]{1,3}[\\.]){3}[0-9]{1,3}/[0-9]{1,2}") && !network.Contains("127.0.0.1"))

{

var net = network.Substring(0, network.LastIndexOf(".") + 1);

var response = SendCommand(stream, command.CommandConfig.Discovery.HostCommand.Replace("{network}", net));

result.commandResponses.Add(response);

foreach (string host in response.Results)

{

result.commandResponses.AddRange(PortScan(command, stream, host, network).commandResponses);

}

}

return result;

}

*En este segundo bloque se filtran las redes y el bucle local y se ejecuta el comando de hosts reemplazando la etiqueta {network}por cada una de las redes válidas detectadas, una vez identificados los hosts activos se pasa al tercer bloque:*

private DiscoveryResults PortScan(CommandContainer command, SshConnectionDto stream, string host, string network)

{

DiscoveryResults result = new DiscoveryResults();

if (Regex.IsMatch(host, "([0-9]{1,3}[\\.]){3}[0-9]{1,3}") && !network.Contains("127.0.0.1"))

{

HostInfo hostFile = new HostInfo();

hostFile.Network = network;

hostFile.Parent = stream.TargetIp;

var response = SendCommand(stream, command.CommandConfig.Discovery.PortCommand.Replace("{host}", host));

result.commandResponses.Add(response);

foreach (string port in response.Results)

{

if (port.Contains("open") && !port.Contains("/"))

{

hostFile.Ports.Add(int.Parse(port.Substring(0, port.IndexOf(" "))));

}

}

GenerateHostConfig(host, hostFile);

}

return result;

}

*En el tercer bloque se filtran las IPs válidas y se lanza el escaneo de puertos host a host reemplazando la etiqueta {host} del comando por cada uno de los hosts descubiertos, una vez tenemos la cadena Red-host-puertos la almacenamos como nueva configuración de host.*

1. **Conclusiones & Distribución:**

*La herramienta está pensada para crecer con la comunidad y para poder ser utilizada de manera individual o por equipos completos.*

*Por esta razón, el software está pensado para distribuirse con licencia libre y en caso de querer monetizarse utilizaríamos las siguientes vías:*

*Botón “Pay us a coffee” para donaciones voluntarias.*

*Una versión premium donde se liberen parches y mejoras 6 meses antes que en el repositorio público y se incluya un soporte al usuario.*

*Paquetes de servicios extra cobrados de manera puntual por el propio paquete, por ejemplo:*

*Acceso a un repositorio de comandos ya creados y/o cadenas de Discovery configuradas ad-hoc.*

*Acceso a la aplicación como SAS utilizando una plataforma de nube para alojar y ejecutar los servicios de forma temporal.*

1. **Siguientes pasos:**

***Tareas pendientes****:*

1. *Limpiar el repositorio actual de datos de prueba y crear un repositorio nuevo con el código limpio.*
2. *Añadir un listado de puertos redirigidos para mayor claridad del usuario.*
3. *Mejorar la funcionalidad de crack para multi sistema operativo.*
4. *Incluir soporte para comandos interactivos.*
5. *Mejorar el Discovery para reportar avance del automatismo en tiempo real.*
6. *Añadir un listado de conexiones activas.*
7. *Permitir modificar la configuración de la estructura de directorios.*
8. *Añadir una pantalla para poder enviar/traer ficheros mediante Scp (La lógica ya está implementada)*
9. *Mejorar la detección de tipos de hash para Hashcat.*
10. *Implementar la extracción de hashes automatizada de ficheros zip/pdf/Word y sus parámetros de crack.*
11. *Implementar la función invítanos a café para monetizar el tiempo invertido.*

***Tareas a futuro****:*

1. *Implementar funciones de scripting remoto para sistemas Windows a través de remote powershell.*
2. *Implementar swagger para el descubrimiento y el uso de los servicios actuales.*
3. *Habilitar un botón para poder subir/descargar nuevos comandos y discoveries a/desde un servidor remoto.*